GEOMETRIC SET COVER AND RELATED GEOMETRIC OPTIMIZATION PROBLEMS

BY

QIZHENG HE

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2023

Urbana, Illinois

ABSTRACT

Geometric set cover is a classical problem in computational geometry, with a long history and many applications. Given a set X of points in $\mathbb{R}^d$ and a set S of geometric objects, the problem asks to find a smallest subset of objects from S that covers all points in X. In this thesis, we study algorithms, data structures and hardness results for geometric set cover and other related geometric optimization problems.

In the first part, we focus on static geometric set cover, where all points and objects are given in advance. As the problem is NP-hard for many classes of geometric objects, we are interested in designing $O(1)$-approximation algorithms, in particular, efficient algorithms that run in near-linear time. For the unweighted problem, we present near-optimal deterministic and randomized algorithms for 2D disks and 3D halfspaces, which are further improved to optimal $O(n \log n)$ time in the next part. We then extend our approach to solve the weighted problem for the same types of ranges, in also near-linear time.

In the second part, we explore geometric set cover problems in dynamic settings, allowing insertions and deletions of both points and objects. The goal is to efficiently maintain a set cover solution (satisfying certain quality requirement) for the dynamic problem instance. We give a plethora of new dynamic geometric set cover data structures for various geometric ranges in 1D, 2D and 3D, which significantly improve and extend the previous results. We also give the first sublinear results for the weighted version of the problem.

In the third part, we investigate the fine-grained complexity of the discrete k-center problem and related (exact) geometric set cover problems when k or the size of the cover is small. We give the first subquadratic algorithms for unweighted and weighted size-3 set cover for rectangles in $\mathbb{R}^2$, and also prove conditional lower bounds for these problems in constant dimensions.

In the fourth part, we develop approximation algorithms for another problem related to geometric set cover, namely, enclosing points with geometric objects. We solve this problem by adapting techniques for approximating geometric set cover.

In the last part, we inspect the FPT status of the monotone convex chain cover problem, where the problem asks for finding the minimum number of x-monotone convex chains $\kappa(P)$ that can together cover a point set P. We show that deciding whether $\kappa(P) \leq k$ is NP-hard and does not have a polynomial kernel, unless NP $\subseteq$ coNP/poly.

"To my parents, Shiqing and Zhiping, for their unconditional love and support."

ACKNOWLEDGMENTS

First of all, I'm grateful to be supervised by my advisor, Timothy M. Chan. As a research supervisor, Timothy broadened my vision in research, helped to improve my writing and presentation skills, and guided me to finish research projects that I will be proud of for the rest of my life. As a course instructor, he not only taught me rich knowledge about algorithms and computational geometry, but is also a great role model for showing care to the students. Moreover, his enthusiasm and rigorous attitude towards research guided my life.

In 2017 I visited University of Michigan, which is where my research path started. I thank Seth Pettie and Yi-Jun Chang, who helped me to build my faith in doing research in theoretical computer science, and determined my mind to start the journey as a Ph.D. student.

Two other UIUC CS faculties that I would like to thank are Sariel Har-Peled and Chandra Chekuri. Sariel taught me a lot in research, and a number of my papers benefited from our discussion. Specifically, Chapter 6 of this thesis about monotone convex chain cover came out of a project from his course, and another paper about change-making is inspired by a homework problem of him. Chandra not only taught me great courses, but also provided valuable suggestions on my research and life.

Next, I want to thank all committee members (again), for your valuable time on helping me improving this thesis. I especially thank Pankaj K. Agarwal for being the external committee member and providing valuable comments.

I thank my coauthors, Calvin Beideman, Pingan Cheng, Chi Han, Zhengcheng Huang, Yakov Nekrich, Subhash Suri, Zhean Xu, Jie Xue, and Yuancheng Yu. It was an enjoyable experience to collaborate with all of you; together we have produced amazing results.

I am glad to be a member of the UIUC theory group. Many thanks to the former and current theory group students. They are friendly and supportive, and together we had a lot of inspiring research discussions.

I also want to thank my roommates and friends. Maintaining an optimistic mindset was not easy during the special time of COVID, but with their kindness and support, we were able to overcome all the difficulties.

Lastly, I would like to express my gratitude to my parents. Thanks to their love and support throughout all these years, this thesis is dedicated to them.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

The main theme of this thesis is to study a geometric version of the well-known set cover problem. In the general set cover problem, we are given a finite set X of elements and a collection S of subsets of X, and the problem asks for finding a subset of S with minimum size whose union is X. This problem is known to be NP-hard [124]. The well-known greedy algorithm [96] gives a $(\ln n + 1)$-approximation[1] in polynomial time, which is nearly the best approximation ratio one can get, since it is proved that one cannot achieve $(1-\varepsilon)\ln n$-approximation unless P$\neq$NP [102, 213].

The geometric version of the problem, *geometric set cover*, is a classical problem in computational geometry, with a long history and many applications [13, 14, 46, 48, 68, 69, 71, 75, 93, 117, 177, 179, 204]. Here the elements in X are points in $\mathbb{R}^d$, and the sets in S, often called ranges, are defined by a simple class of geometric objects. For instance, the sets may be defined by 1D intervals, disks, squares, or halfspaces. The goal is to find the smallest subset of objects from S to cover all points in X. An example is shown in Fig. 1.1.

Problem 1.1 (Geometric set cover)**.** Given a set X of points in $\mathbb{R}^d$ and a set S of geometric objects, find the smallest subset of objects from S to cover all points in X.

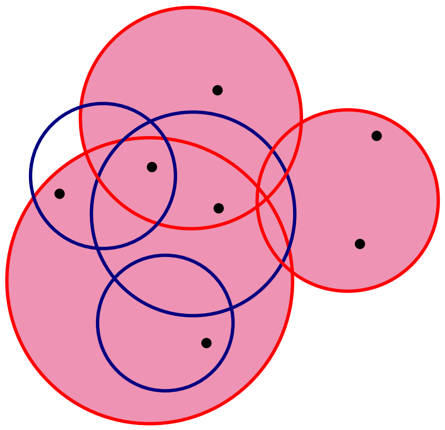


Figure 1.1: An example of geometric set cover: covering points in $\mathbb{R}^2$ by disks. The optimal solution is marked in red.

The geometric set cover problem is one of the most fundamental classes of geometric optimization problems. It has many applications, e.g., it is useful for sensor networks, VLSI design, databases and image processing. This problem has been extensively investigated

[1] An α-approximation algorithm is able to compute an approximate solution with size at most α times the optimum. α is called the *approximation ratio* of the algorithm ($\alpha \geq 1$ for minimization problems).

in the computational geometry literature. Finding an exact solution is NP-hard for many types of geometric objects, even for the simplest geometric families such as unit disks or unit-squares in $\mathbb{R}^2$ [120]. Therefore we are interested in designing efficient approximation algorithms, that is able to compute an approximate solution with size close to the optimal solution, and ideally runs in only near-linear time.

In the geometric setting, there are often efficient approximation algorithms with better (worst-case) performance than the general (combinatorial) set cover. Many classes of objects, such as squares and disks in 2D, objects with linear union complexity, and halfspaces in 3D, admit polynomial-time $O(1)$-approximation algorithms, by using ε-nets and linear programming (LP) rounding or the multiplicative weight update (MWU) method [46, 89, 93]. Some classes of objects, such as disks in 2D and halfspaces in 3D, even have polynomial-time approximation schemes (PTASes) [179] or quasi-PTASes [177], though with large polynomial or quasi-polynomial running time.

In this thesis, we investigate different versions of the geometric set cover problem. One distinction is unweighted vs weighted. In the weighted version of the problem, each object is associated with a non-negative weight, and we want to find a subset of the input objects with the minimum total weight that covers all points. The unweighted problem can be viewed as all objects having the same weight 1.

Problem 1.2 (Weighted geometric set cover)**.** Given a set X of points and a set S of geometric objects, where each object $s \in S$ has a non-negative weight w_s. Find a subset of objects $S^* \subseteq S$ with the minimum total weight $\sum_{s\in S^*} w_s$ that covers all points in X.

Another distinction is static vs dynamic. In the static problem (Problem 1.1), all points and objects are given in advance; while in the dynamic problem, both points and objects may be dynamically inserted or deleted, and we want to dynamically maintain an approximate solution after each update.

In addition, different geometric ranges may require different techniques to solve. In particular, we design approximation algorithms and data structures for 1D intervals, 2D unit-squares, 2D axis-aligned squares, 2D disks, 2D halfplanes, 3D halfspaces and more, using a number of different ideas that are tailored towards each of the particular ranges.

Finally, we will also adapt the set cover techniques to study several other problems related to geometric set cover, such as discrete k-centers of small size k and the problem of enclosing all points with geometric objects. We investigate the complexity of the monotone convex chain cover problem from the Fixed-Parameter Tractability (FPT) perspective, and prove a hardness result.

To summarize, the thesis will be organized around the following five themes:

- static geometric set cover,
- dynamic geometric set cover,
- exact algorithms and conditional lower bounds for set cover and discrete k-center with small size,
- a related problem about enclosing points with geometric objects,
- the FPT status of monotone convex chain cover.

1.1 NOTATIONS

Throughout the thesis, we use $n = |X| + |S|$ to denote the size of the problem instance (X, S), and use opt to denote the cardinality (or total weight) of the optimal solution. All squares, rectangles, hypercubes, and boxes are by default axis-aligned, and $\delta > 0$, $\varepsilon > 0$ denote arbitrarily small positive constants. The $\widetilde{O}$ notation hides polylogarithmic factors.

We use $[r]$ to denote the set $\{1, \ldots, r\}$. We denote by $A \sqcup B$ the disjoint union of two multi-sets A and B.

1.2 STATIC GEOMETRIC SET COVER

Unweighted static geometric set cover. In the first part of the thesis, we study one of the most fundamental classes of geometric optimization problems: (static) geometric set cover (Problem 1.1). In the dual set system, the problem corresponds to *geometric hitting set*, which asks for finding the smallest subset of points $H \subseteq X$, such that H intersects each object in S (i.e., all objects are "hit" by the chosen set of points H).

Since the problem is NP-hard in most scenarios, attention is turned towards approximation algorithms. Typically, approximation algorithms for static geometric set cover fall into the following categories:

1. Simple heuristics, e.g., greedy algorithms.
2. Approaches based on solving the linear programming (LP) relaxation (i.e., fractional set cover) and rounding the LP solution.
3. Polynomial-time approximation schemes (PTASs), e.g., via local search, shifted grids/quadtrees (sometimes with dynamic programming), or separator-based divide-and-conquer.

Generally, greedy algorithms achieve only logarithmic approximation factors (there are some easy cases where they give $O(1)$ approximation factors, e.g., hitting set for fat objects such as disks/balls in the "continuous" setting with $X = \mathbb{R}^d$ [105]). The LP-based approaches give better approximation factors in many cases, e.g., $O(1)$ approximation for set cover and hitting set for disks in 2D and halfspaces in 3D, set cover for objects in 2D with linear "union complexity", and hitting set for pseudodisks in 2D [23, 46, 93, 187, 203]. Subsequently, local-search PTASs have been found by Mustafa and Ray [179] in some cases, including set cover and hitting set for disks in 2D and halfspaces in 3D (earlier, PTASs were known for hitting set only in the continuous setting for unit disks/balls [139], and for arbitrary disks/balls and fat objects [59]).

Historically, the focus has been on obtaining good approximation factors. Here, we are interested in obtaining approximation algorithms with good—ideally, near linear—running time. Concerning the efficiency of known approximation algorithms:

1. Certain simple heuristics can lead to fast $O(1)$-approximation algorithms in some easy cases (e.g., continuous hitting set for unit disks or unit balls by using grids), but generally, even simple greedy algorithms may be difficult to implement in near linear time (as they may require nontrivial dynamic geometric data structures).

2. LP-based approaches initially may not seem to be the most efficient, because of the need to solve an LP. However, a general-purpose LP solver can be avoided. The set-cover LP can alternatively be solved (approximately) by the *multiplicative weight update* (MWU) method. In the computational geometry literature, the technique has been called *iterative reweighting*, and its use in geometric set cover was explored by Brönnimann and Goodrich [46] (one specific application appeared in an earlier work by Clarkson [89]), although the technique was known even earlier outside of geometry. On the other hand, the LP-rounding part corresponds to the well-known geometric problem of constructing *ε-nets*, for which efficient algorithms are known [78, 176].

3. PTAS approaches generally have large polynomial running time, even when specialized to specific approximation factors. For example, see [47] for efforts in improving the degree of the polynomial.

In this part we design faster approximation algorithms for geometric set cover via the LP/MWU-based approaches. There has been a series of work on speeding up MWU methods for covering or packing LPs (e.g., see [84, 153, 215]). In geometric settings, we would like more efficient algorithms (as generating the entire LP explicitly would already require quadratic time), by somehow exploiting geometric data structures. The main previous work was by Agarwal and Pan [14] from SoCG'14, who showed how to compute an $O(1)$-approximation for set cover for 2D disks or 3D halfspaces, in $O(n \log^4 n)$ randomized time. (Later in the journal version [10], they improved the running time of their randomized algorithm to $O(n \log^3 n)$.)

Agarwal and Pan actually proposed two MWU-based algorithms: The first is a simple variant of the standard MWU algorithm of Brönnimann and Goodrich, which proceeds in logarithmically many rounds. The second views the problem as a 2-player zero-sum game, works quite differently (with weight updates to both points and objects), and uses randomization; the analysis is more complicated. Because the first algorithm requires stronger data structures—notably, for approximate weighted range counting with dynamic changes to the weights—Agarwal and Pan chose to implement their second algorithm instead, to get their $O(n \log^4 n)$ result for 3D halfspaces.

Weighted static geometric set cover. The weighted geometric set cover problem (Problem 1.2) has also received considerable attention: Varadarajan [204] and Chan et al. [69] used the LP-based approach to obtain $O(1)$-approximation algorithms for weighted set cover for 3D halfspaces (or for objects in 2D with linear union complexity); the difficult part is in constructing ε-nets with small weights, which they solved by the *quasi-random sampling* technique. Later, Mustafa, Raman, and Ray [177] discovered a quasi-PTAS for 3D halfspaces by using geometric separators; the running time is very high ($n^{\log^{O(1)} n}$).

In SODA'20, Chekuri, Har-Peled, and Quanrud [82] described new randomized MWU methods which can efficiently solve the LP corresponding to various generalizations of geometric set cover, by using appropriate geometric data structures. In particular, for weighted set cover for 3D halfspaces, they obtained a randomized $O(n \log^{O(1)} n)$-time algorithm to solve the LP but with an unspecified number of logarithmic factors. They did not address the LP-rounding part, i.e., construction of an ε-net of small weight—a direct implementation of the quasi-uniform sampling technique would not lead to a near-linear time bound.

1.3 DYNAMIC GEOMETRIC SET COVER

Approximation algorithms for NP-hard problems and dynamic data structures are two of the major themes studied by the algorithms community. For dynamic data structures, there are many previous works, for example various dynamic graph algorithms [113, 121, 140, 149, 180]. In particular, there are a lot of dynamic data structures in computational geometry, e.g., there are results for dynamic convex hull [44], dynamic 2D nearest neighbor [65], dynamic bichromatic closest pair and diameter [65], dynamic width

[58], dynamic geometric connectivity [77], dynamic largest empty circle [65], and more. On the topic of approximation algorithms for NP-hard problems, there are works for set cover and hitting set [14, 23, 46, 93, 178, 187], independent set [4, 5, 70, 171], Euclidean TSP [25], and etc.

Recently, problems at the intersection of these two threads have gained much attention, and researchers in computational geometry have also started to systematically explore such problems. For example, two papers appeared at SoCG'20, one on dynamic geometric set cover by Agarwal, Chang, Suri, Xiao, and Xue [10], and another on dynamic geometric independent set by Henzinger, Neumann, and Wiese [138]. In this part of the thesis, we continue the study by Agarwal et al. [10] and investigate dynamic data structures for approximating the minimum set cover in natural geometric instances.

Dynamic geometric set cover. After studying approximation algorithms for the static geometric set cover problem in the previous part, it is natural to further consider the *dynamic* setting of the geometric set cover problem. Here, we want to design data structures that support insertions and deletions of points in X *as well as* insertions and deletions of objects in S, while maintaining an approximate solution.

Note that the solution may have linear size in the worst case. In the simplest version of the problem, we may just want to output the size of the solution. More strongly, we may want some representation of the solution itself, so that afterwards, the objects in the solution can be reported in constant time per element when needed.

Problem 1.3 (Dynamic geometric set cover)**.** Design a data structure that supports insertions and deletions of points as well as geometric objects, while maintaining an approximate set cover, such that the size of the maintained solution can be queried in $O(1)$ time, and the actual solution can be efficiently reported (ideally, the maintained solution can be reported in constant time per element).

Recently, an exciting line of research was launched by Agarwal et al. [10] on dynamic geometric set cover with the introduction of *sublinear* time data structures for *fully dynamic* maintenance of approximate set covers for intervals in one dimension and unit squares in two dimensions. They showed that for intervals in 1D, a $(1+\varepsilon)$-approximation can be maintained in $O((1/\varepsilon)n^{\delta})$ time per insertions and deletions of points and intervals. In 2D, they had only one main result: a fully dynamic $O(1)$-approximation algorithm for unit squares, with $O(n^{1/2+\delta})$ update time. This is the only previous work that achieves nontrivial results for the fully dynamic geometric set cover problem.

Dynamic general set cover. In contrast to the few known results for the geometric version, there are a number of previous works on designing approximation algorithms for *dynamic general (combinatorial) set cover*. However, one distinction is they mainly focus on the partially dynamic setting, where only the elements can be inserted or deleted:

Definition 1.1 (Dynamic general set cover)**.** Given a set system $(X, \mathcal{S})$, where the universe X is a set of elements, and $\mathcal{S}$ is a collection of subsets of X. Each set $S \in \mathcal{S}$ is associated with a weight $c_S > 0$ (for the unweighted case, all sets in $\mathcal{S}$ have the same weight). The goal is to maintain an approximate set cover solution that supports dynamic updates to the elements. For an insertion update, a new input element x is inserted into X, and we get to know which sets in $\mathcal{S}$ contain x. For a deletion update, the input element x is deleted from X.

Bhattacharya et al. [35] initiated the study on (weighted) dynamic general set cover, where they obtained $O(f \log n)$ amortized update time for $O(f^2)$-approximation, via the primal-dual method, where f denote the maximum frequency of the elements (i.e., the maximum number of sets containing a single element). A series of works followed. Gupta et al. [133] designed two algorithms: one with $O(f \log n)$ amortized update time for $O(\log n)$-approximation (which is the first result that obtained an approximation ratio independent of f), using greedy-like techniques; another with $O(f^2)$ amortized update time for $O(f^3)$-approximation, using a primal-dual framework. Abboud et al. [1] achieved $O(f^2 \log n/\varepsilon^5)$ amortized update time for $(1+\varepsilon)f$-approximation, but their algorithm was only able to solve the unweighted problem, and needed to assume an oblivious adversary. (The problem formulation that they considered was slightly different: in their model, the set system $(X, \mathcal{S})$ is known and fixed, but the subset of "active" elements $\hat{X} \subseteq X$ that need to be covered dynamically changes.)

If allowing randomization, Assadi and Solomo [28] obtained $O(f^2)$ amortized update time for f-approximation, which is the first algorithm that achieves an approximation ratio exactly f (instead of almost f), and is the best possible under the unique games conjecture for any fixed f [148]. Bhattacharya et al. [36] designed a deterministic algorithm with $(1+\varepsilon)f$-approximation and $O(f \log(Cn)/\varepsilon^2)$ amortized update time, where C denote the maximum ratio among the cost of the ranges. Bhattacharya et al. [37] improved their result in the low-frequency range, achieving $O((f^2/\varepsilon^3)+(f/\varepsilon^2)\log C)$ amortized update time for deterministic $(1+\varepsilon)f$-approximation, and also obtained the first worst-case result with $O(f \log^2(Cn)/\varepsilon^3)$ update time for the same $(1+\varepsilon)f$ approximation ratio.

As a side comment, the dynamic general set cover problem has also been studied in the online settings [133]. Here instead of optimizing the update time, the goal is to achieve bounded recourse (i.e., limit the number of changes to the past decisions at each step) while also maintaining a good competitive ratio.

The sublinear bounds for dynamic geometric set cover are in sharp contrast with the $\Omega(f)$ update time bottleneck faced by the general set cover problem in dynamic setting [28, 35, 36, 37, 133], because in their model, to insert an element x one needs to specify all its membership relations, which at a minimum requires updating all the sets that contain it. (Even for the problem formulation by Abboud et al. [1], where the $\Omega(f)$ bottleneck is less obvious since the membership relations in the set system $(X, \mathcal{S})$ is fixed in advance, they proved an $\Omega(n^{\delta})$ lower bound for some constant $\delta > 0$ on the approximation ratio, for any algorithm that has update time $O(f^{1-\varepsilon})$, under SETH.) For our geometric set cover problem, the maximum frequency f can be as large as $\Omega(n)$, so directly applying those algorithms for dynamic general set cover will not immediately give us a sublinear update time bound. The *implicit* form of sets in geometric set covering—an interval or a disk, for instance, takes only $O(1)$ pieces of information to add or delete—provides a natural yet challenging problem setting in which to explore the possibility of truly sublinear (possibly polylogarithmic) updates of both the elements and the sets.

In spite of these recent developments, the state of the art for dynamic geometric set covering is far from satisfactory even for the simplest of the set systems: covering points on the line by intervals or covering points in the plane by axis-aligned squares. For instance, the best update bound for the former is $O(n^{\delta}/\varepsilon)$ for a $(1+\varepsilon)$-approximation, and for the latter no previous result was known. More importantly, none of these schemes are able to handle the case of *weighted* dynamic geometric set covers. In this thesis, we make substantial progress on these fronts.

Remark on dynamic geometric independent set. Dynamic geometric independent set is closely related to dynamic geometric set cover, where the problem asks to dynamically maintain an approximate independent set solution, under insertion and deletion of geometric objects. Opposed to the few previous results on dynamic geometric set cover, more results were known for this related problem.

We first survey the previous works under the unweighted setting. For 1D intervals, there's a broad literature for obtaining $(1+\varepsilon)$-approximation efficiently. Henzinger et al. [138] presented an algorithm with $O_{\varepsilon}(1)\log^2 n \log^2 N$ update time (where N denote the range of the coordinates), but their method require the geometric objects have bounded size ratios. For geometric objects with arbitrary size, Bhore et al. [38] presented the first dynamic algorithm for 1D intervals independent set with polylogarithmic update time, namely, worst-case $O(\frac{\log n}{\varepsilon^2})$. Later, Compton et al. [94] improved the running time to worst-case $O(\frac{\log n}{\varepsilon})$ for update and $O(\log n)$ for query. If no interval is fully contained in another interval (e.g., unit intervals), Gavruskin et al. [125] gave an exact algorithm with $O(\log n)$ update time, based on dynamic tree.

For unit squares, Bhore et al. [39] obtained the first deterministic algorithm for dynamic independent set with approximation factor 4 and $O(1)$ update time. They also have a trade-off between approximation factor $2 + \frac{2}{k}$ and update time $O(k^2 \log n)$. For arbitrary axis-aligned squares, Bhore et al. [38] designed an expected $O(1)$-approximation data structure with $O(\log^5 n)$ amortized update time. For d-dimensional hypercubes, the data structure by Henzinger et al. [138] can maintain a $(1+\varepsilon)2^d$-approximation in $\mathrm{poly}(\log n, \log N)$ time. The approach of Bhore et al. [38] also generalizes to this case, getting $O(4^d)$-approximation with $O(2^d \log^{2d+1} n)$ update time. Cardinal et al. [52] achieved worst-case complexity for fat objects in d dimension.

The *weighted* setting is more challenging. For weighted 1D intervals, the approach by Henzinger et al. [138] can maintain an $(1+\varepsilon)$-approximation in $\mathrm{poly}(\log n, \log N, \log W)$ time, where W denote the range of the weights. Compton et al. [94] obtained $\mathrm{poly}(\log n, \log N, \log W, \frac{1}{\varepsilon})$ update time, which is an exponential improvement on the query time dependency on $1/\varepsilon$.

For weighted d-dimensional hypercubes, the approach by Henzinger et al. [138] can maintain an $O(2^d)$-approximation in $\mathrm{poly}(\log n, \log N, \log W)$ time. For weighted d-dimensional hyperrectangles, they are able to obtain a slightly worse approximation ratio $(1+\varepsilon)\log^{d-1} N$ with $\mathrm{poly}(\log n, \log N, \log W)$ update time.

1.4 GEOMETRIC SET COVER AND DISCRETE K-CENTER OF SMALL SIZE

The discrete k-center problem for small k. The *Euclidean k-center* problem is well-known in computational geometry and has a long history: given a set P of n points in $\mathbb{R}^d$ and a number k, we want to find k congruent balls covering S, while minimizing the radius. Euclidean 1-center can be solved in linear time for any constant dimension d by standard techniques for low-dimensional linear programming or LP-type problems [80, 90, 104, 169, 212]. In a celebrated paper from SoCG'96, Sharir [193] gave the first $\widetilde{O}(n)$-time algorithm for Euclidean 2-center in $\mathbb{R}^2$, which represented a significant improvement over previous near-quadratic algorithms (the hidden logarithmic factors have since been reduced in a series of subsequent works [55, 86, 111, 210]). The problem is more difficult in higher dimensions: the best time bound for Euclidean 2-center in $\mathbb{R}^d$ is about n^d (see [9, 11] for some results on the $\mathbb{R}^3$ case), and Cabello et al. [50] proved a conditional lower bound, ruling out $n^{o(d)}$-time algorithms, assuming the Exponential Time Hypothesis (ETH). We are not aware of any work specifically addressing the Euclidean 3-center problem.

The k-center problem has also been studied under different metrics. The most popular version after Euclidean is L_∞ or *rectilinear k-center*: here, we want to find k congruent hypercubes covering P, while minimizing the side length of the hypercubes. As expected, the rectilinear version is a little easier than the Euclidean. Sharir and Welzl in SoCG'96 [194] showed that rectilinear 3-center problem in $\mathbb{R}^2$ can be solved in linear time, and that rectilinear 4-center and 5-center in $\mathbb{R}^2$ can be solved in $\widetilde{O}(n)$ time (the logarithmic factors have been subsequently improved by Nussbaum [181]). Katz and Nielsen's work in SoCG'96 [145] implied near-linear-time algorithms for rectilinear 2-center in any constant dimension d, while Cabello et al. in SODA'08 [50] gave an $O(n \log n)$-time algorithm for rectilinear 3-center in any constant dimension d. Cabello et al. also proved a conditional lower bound for rectilinear 4-center, ruling out $n^{o(\sqrt{d})}$-time algorithms under ETH.

In this part of the thesis, we focus on a natural variant of the problem called *discrete k-center*, which has also received considerable attention:

Problem 1.4 (discrete k-center)**.** Given a set P of n points in $\mathbb{R}^d$ and a number k, we want to find k congruent balls covering P, while minimizing the radius, with the extra constraint that the centers of the chosen balls are from P.[2]

The Euclidean discrete 1-center problem can be solved in $O(n \log n)$ time in $\mathbb{R}^2$ by a straightforward application of farthest-point Voronoi diagrams; it can also be solved in $O(n \log n)$ (randomized) time in $\mathbb{R}^3$ with more effort [54], and in subquadratic $\widetilde{O}(n^{2-2/(\lceil d/2 \rceil+1)})$ time for $d \geq 4$ by standard range searching techniques [12, 166]. Agarwal, Sharir, and Welzl in SoCG'97 [17] gave the first subquadratic algorithm for Euclidean discrete 2-center in $\mathbb{R}^2$, running in $\widetilde{O}(n^{4/3})$ time.

We can similarly investigate the rectilinear version of the discrete k-center problem, which is potentially easier. For example, the rectilinear discrete 2-center problem can be solved in $\widetilde{O}(n)$ time in any constant dimension d, by a straightforward application of orthogonal range searching, as reported in several papers [33, 34, 144]. The approach does not seem to work for the rectilinear discrete 3-center problem. Naively, rectilinear discrete 3-center can be reduced to n instances of (some version of) rectilinear discrete 2-center, and solved in $\widetilde{O}(n^2)$ time. However, no better results have been published, leading to the following questions:

[2] Some authors define the problem slightly more generally, calling it "k-supplier" [199], where the constraint is that the centers are from a second input set; in other words, the input consists of two sets of points ("demand points" and "supply points"). The results of this chapter will apply to both versions of the problem.

k	Euclidean	rectilinear	Euclidean discrete	rectilinear discrete
1	$O(n)$	$O(n)$	$O(n \log n)$	$O(n)$
2	$\widetilde{O}(n)$ [193]	$O(n)$ [194]	$\widetilde{O}(n^{4/3})$ [17]	$\widetilde{O}(n)$
3		$\widetilde{O}(n)$ [194]		$\widetilde{O}(n^{3/2})$ **(new)**

Table 1.1: Summary of results on k-center for small k in $\mathbb{R}^2$.

Question 1.1. Is there a subquadratic-time algorithm for the rectilinear discrete 3-center problem?

Question 1.2. Are there lower bounds to show that the rectilinear discrete 3-center problem does not have near-linear-time algorithm (and is thus strictly harder than rectilinear discrete 2-center, or rectilinear continuous 3-center)?

Similar questions may be asked about rectilinear discrete k-center for $k \geq 4$. Here, the complexity of the problem is upper-bounded by $\widetilde{O}(n^{\omega(\lfloor k/2 \rfloor, 1, \lceil k/2 \rceil)})$, where $\omega(a, b, c)$ denotes the exponent for multiplying an $n^a \times n^b$ and an $n^b \times n^c$ matrix: by binary search, the problem reduces to finding k hypercubes of a given edge length r with centers in S covering S, which is equivalent to finding a dominating set of size k in the graph with vertex set S where an edge pz exists iff the distance of p and z is more than r—the dominating set problem reduces to rectangular matrix multiplication with the time bound stated, as observed by Eisenbrand and Grandoni [110]. Note that the difference $\omega(\lfloor k/2 \rfloor, 1, \lceil k/2 \rceil) - k$ converges to 0 as $k \to \infty$ by known matrix multiplication bounds [95] (and is exactly 0 if $\omega = 2$).

As k gets larger compared to d, a better upper bound of $n^{O(dk^{1-1/d})}$ is known for both the continuous and discrete k-center problem under the Euclidean and rectilinear metric [15, 141, 142]. Recently, in SoCG'22, Chitnis and Saurabh [85] (extending earlier work by Marx [163] in the $\mathbb{R}^2$ case) proved a nearly matching conditional lower bound for discrete k-center in $\mathbb{R}^d$, ruling out $n^{o(k^{1-1/d})}$-time algorithms under ETH. However, these bounds do not answer our questions concerning very small k's. In contrast, the conditional lower bounds by Cabello et al. [50] that we have mentioned earlier are about very small k and so are more relevant, but are only for the continuous version of the k-center problem. (The continuous version behaves differently from the discrete version; see Tables 1.1–1.2.)

The geometric set cover problem with small size k. The decision version of the discrete k-center problem (deciding whether the minimum radius is at most a given value) reduces to a *geometric set cover* problem: given a set P of n points and a set S of n objects, find the smallest subset of objects in S that cover all points of P. Geometric set

k	Euclidean	rectilinear	Euclidean discrete	rectilinear discrete
1	$O(n)$	$O(n)$	$\widetilde{O}(n^{2-2/(\lceil d/2\rceil+1)})$	$O(n)$
2	$n^{O(d)}$ CLB: $n^{\Omega(d)}$ [50]	$\widetilde{O}(n)$ [145]	$\widetilde{O}(n^{\omega})$	$\widetilde{O}(n)$
3		$\widetilde{O}(n)$ [50]	$\widetilde{O}(n^{\omega(1,1,2)})$	$\widetilde{O}(n^2)$ CLB: $\Omega(n^{4/3-\delta})$ **(new)**
4		$n^{O(d)}$ CLB: $n^{\Omega(\sqrt{d})}$ [50]	$\widetilde{O}(n^{\omega(2,1,2)})$	$\widetilde{O}(n^3)$

Table 1.2: Summary of results on k-center for small k in $\mathbb{R}^d$ for an arbitrary constant d. (CLB stands for "conditional lower bound".)

cover has been extensively studied in the literature, particularly from the perspective of approximation algorithms (since for most types of geometric objects, set cover remains NP-hard); for example, see the references in [71]. Here, we are interested in *exact* algorithms for the case when the optimal size k is a small constant.

For the application to Euclidean/rectilinear k-center, the objects are congruent balls/hypercubes, or by rescaling, unit balls/hypercubes, but other types of objects may be considered, such as arbitrary rectangles or boxes.

We can also consider the *weighted* version of the problem: here, given a set P of n points, a set S of n weighted objects, and a small constant k, we want to find a subset of k objects in S that cover all points of P, while minimizing the total weight of the chosen objects.

A "dual" problem is *geometric hitting set*, which in the weighted case is the following: given a set P of n weighted points, a set S of n objects, and a small constant k, find a subset of k points in P that hit all objects of S, while minimizing the total weight of the chosen points. (The continuous unweighted version, where the chosen points may be anywhere, is often called the *piercing* problem.) In the case of unit balls/hypercubes, hitting set is equivalent to set cover due to self-duality.

For rectangles in $\mathbb{R}^2$ or boxes in $\mathbb{R}^d$, size-2 geometric set cover (unweighted or weighted) can be solved in $\widetilde{O}(n)$ time, like discrete rectilinear 2-center [33, 34, 144], by orthogonal range searching. Analogs to Questions 1.1–1.2 may be asked for size-3 geometric set cover for rectangles/boxes.

Surprisingly, the complexity of exact geometric set cover of small size k has not received as much attention, but very recently in SODA'23, Chan [67] initiated the study of similar questions for geometric independent set with small size k, for example, providing subquadratic algorithms and conditional lower bounds for size-4 independent set for boxes.

For larger k, hardness results by Marx and Pilipczuk [164] and Bringmann et al. [43] ruled out $n^{o(k)}$-time algorithms for size-k geometric set cover for rectangles in $\mathbb{R}^2$ and unit hypercubes (or orthants) in $\mathbb{R}^4$, and $n^{o(\sqrt{k})}$-time algorithms for unit cubes (or orthants) in $\mathbb{R}^3$ under ETH. But like the other fixed-parameter intractability results mentioned, these proofs do not appear to imply any nontrivial lower bound for very small k such as $k = 3$.

1.5 ENCLOSING POINTS WITH GEOMETRIC OBJECTS

After studying the geometric set cover problem in the previous parts, we further consider a problem that is closely related to geometric set cover: enclosing all points with geometric objects (Enclosing-All-Points). The problem is defined as follows:

Problem 1.5 (Enclosing-All-Points)**.** Given a set X of points and a set S of geometric objects (possibly overlapping) in $\mathbb{R}^2$, find the smallest subset $S^* \subseteq S$ to enclose all points in X. More precisely, for each point $p \in X$, any curve connecting p with infinity $(+\infty, 0)$ must intersect at least one object in S^*.

An illustrative example is shown in Fig. 1.2. This problem turns out to be related to geometric set cover, as we will see later.

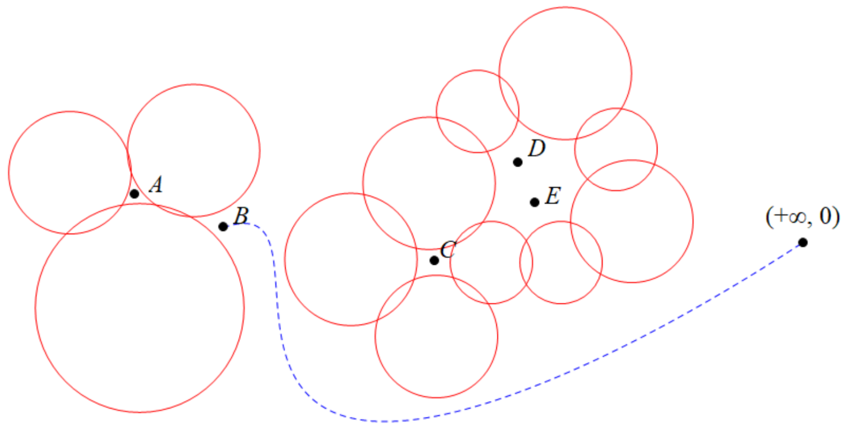


Figure 1.2: The set of red disks enclose points A, C, D and E, but does not enclose the point B.

We remark that the problem can also be more generally defined in higher dimensions; however, all previous related works only study the problem in 2D (since the initial motivation is to isolate points by wireless sensors, which are 2D disks or unit disks). Designing approximation algorithms for this problem in dimension three or higher appears to be much harder. Therefore, in this thesis we also only focus on the 2D case.

We start with surveying a number of similar problems related to the Enclosing-All-Points problem we study here.

The Points-Separation problem. Given a set S of geometric objects and a set X of points in $\mathbb{R}^2$, the Points-Separation problem asks to select the minimum number of input objects, such that all pairs of points in X are separated by the selected objects. We say a pair of points p and q are separated by a set S^* of objects, if any curve connecting p and q intersect at least one object in S^*. As a comparison, our enclosing problem asks to separate all points with a single point $(+\infty, 0)$ (the infinity), while the Points-Separation problem asks to separate all pairs of points.

The Points-Separation problem has applications in barrier coverage with wireless sensors. In real world, people use wireless sensors to guard or monitor buildings, estates, national borders etc, and the coverage region of the wireless sensors used can usually be modeled as unit disks or disks. If two points p and q are separated, then no intruder can get from one point to the other without getting noticed. Our enclosing problem is also naturally relevant to these applications, and this is one of the motivations for us to study this particular problem. Instead of covering the whole area with sensors, it is more economical to only monitor a boundary of the buildings, while still ensuring that the interior is guarded.

This line of research about separating points with objects was initiated by Gibson et al. [128] in ESA'11. They presented an $O(1)$-approximation algorithm for separating all points when the objects are unit disks or disks, by greedily finding the cheapest subset to separate some pair of input points, using this subset to partition the points into parts (each part of the points is contained in a face of the arrangement of the chosen objects), and then recursively separate in each remaining part. Their algorithm uses an $O(1)$-approximation algorithm for separating two points p and q as a subroutine. They also showed that it suffices to consider the case that no point in the input set is contained in any input disk. In an extended version of the paper, Gibson et al. [127] explicitly showed that their approximation factor for disks is $9 + \varepsilon$.

In a later work by Cabello and Giannopoulos [49], the first polynomial time exact algorithm with $O(n^3)$ running time is given for separating two points using arbitrary connected curves, assuming the curves have reasonable computational properties. (An earlier version of this paper appeared in a manuscript by Alt et al. [20], which only works for line segments.) They utilized a concept called the "3-path-condition" [198] (see also [173, Chapter 4]), connecting the Points-Separation problem to topology. They also showed NP-hardness of separating several input points using unit circles or horizontal and vertical segments, via a reduction from planar-3-SAT. In an independent work, Penninger and Vigan [184] also showed NP-hardness of the problem for unit disks, by reducing from planar-multiterminal-cut.

The Obstacle-Removal problem. Given a set S of geometric objects in $\mathbb{R}^2$ and two points p and q, the Obstacle-Removal problem asks to remove the minimum number of objects in order to connect the two points. This problem is also sometimes called computing the *barrier resilience* [53, 156], where a real-world application is to compute the minimum number of sensors that need to be deactivated, so that there is a path between p and q that does not intersect any of the active sensors. It also has applications in robotics [106, 116]. A related concept is called the *thickness* of the barrier, counting the minimum number of intersections with the sensors over all paths from p to q (a path may intersect a sensor several times). Easy to see the resilience is always a lower bound on the thickness.

The Obstacle-Removal problem was shown to be NP-hard for arbitrary line segments [20, 21], for unit segments [200, 201], and for certain types of fat regions with bounded ply (such as axis-aligned rectangles of aspect ratio $1 : 1+\varepsilon$ and $1+\varepsilon : 1$) [151]. For approximation results, Bereg and Kirkpatrick provided a 3-approximation algorithm for unit disks [32]. In particular, they showed any (Euclidean) shortest path from s to t that intersects a fixed number of distinct sensors, never intersects the same sensor more than three times. As a corollary, the thickness is at most three times the resilience. On the other hand, it is not known whether for unit disks the problem is still NP-hard.

Bandyapadhyay et al. [29] presented an $O(\sqrt{n})$-approximation algorithm for pseudodisks and rectilinear polygons. Kumar et al. [154] further designed an $O(1)$-approximation algorithm for any well-behaved objects, such as polygons or splines. Their arguments were later simplified by Kumar et al. [155].

FPT results. These problems were also considered in the FPT context (for backgrounds on this topic, see Sec. 1.6). For Obstacle-Removal, Korman et al. [152] showed that the problem is FPT for unit disks, and for similarly-sized β-fat regions with bounded ply and constant number of pairwise boundary intersections. Eiben and Kanj generalized their result to graphs satisfying the color-connectivity property [107, 108]. Eiben and Lokshtanov presented an FPT algorithm with $q^{O(q^3)}n^{O(1)}$ running time for general connected obtacles [109].

Kumar et al. [155] improved the running time for Obstacle-Removal to $2.3146^q n^{O(1)}$, and also gave a $2^{O(p)}n^{O(k)}$-time algorithm for Generalized Points-Separation, which receives n objects, a set A of k points, and p pairs of points from A as input, and asks to select the minimum number of objects to separate these pairs.

We remark that these seemingly similar problems may require different techniques to design approximation algorithms; and to the best of our knowledge, our particular version that requires enclosing all of the points (Problem 1.5) has not been studied before.

1.6 MONOTONE CONVEX CHAIN COVER

In the last part of the thesis, we turn our attention back to the geometric set cover problem, but here we study its *parameterized complexity*. This is a fundamental question related to clustering, and quite a bit of attention has been attracted in recent years [15, 43, 50, 141, 142, 163, 164].

Here we first introduce some basic definitions in parameterized complexity, to provide some backgrounds on this topic.

Fixed-Parameter Tractability. We say a problem is Fixed-Parameter Tractable (FPT), if there exists an algorithm with running time of the form $f(k)\cdot|x|^c$ for some function f, where x is a string encoding a given problem instance ($n = |x|$ is the size of the problem), k is the parameter, and c is a constant independent of both x and k. Typically, the goal in designing FPT algorithms is to make both the function $f(k)$ and the exponent c in the running time bound as small as possible. The problems that are FPT can be solved efficiently, if the fixed parameter k is small.

Kernelization. A *kernel* for a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm $\mathcal{A}$ that, given an instance (I, k) of Q, returns an equivalent instance (I', k') of Q in polynomial time such that $(I, k) \in Q$ iff $(I', k') \in Q$. The size of the kernel $|I'| + k'$ is required to be upper bounded by $f(k)$ for some computable function f [97]. We say the problem Q has a *polynomial kernel*, if there exists a polynomial such function f.

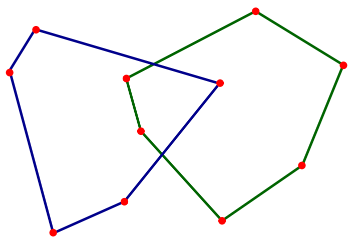
It is well-known that a parameterized problem is FPT if and only if it admits a kernelization algorithm. For more backgrounds on FPT and kernelization, see the books about parameterized algorithms [97, 119].

For simple geometric objects, Langerman and Morin [159] presented an FPT algorithm framework to solve abstract geometric covering problems, with running time depending exponentially on the combinatorial dimension of the problem. This abstract setting models a number of concrete problems, e.g., covering points in $\mathbb{R}^2$ by k lines, or more generally covering points by k hyperplanes in $\mathbb{R}^d$ for constant dimension d. In this case, the combinatorial dimension is equal to the geometric dimension d, and a simple bounded search tree and kernelization algorithm works [126, 131, 159, 211]. The problem can be solved in deterministic $O(k^{dk}n)$ time, or randomized $O(k^{d(k+1)} + 2^d k^{\lceil (d+1)/2 \rceil \lfloor (d+1)/2 \rfloor} n \log n)$ time.

The covering problem becomes more interesting for geometric objects with non-constant complexity, where fewer results were known in the literature. In this part of the thesis, we study the parameterized complexity of the problem of covering points in $\mathbb{R}^2$ by monotone convex chains.

The problem is defined as follows. Let $Q = (q_1, \ldots, q_m)$ be a sequence of points in the plane. The sequence Q is a *convex chain*, if $\forall\ 1 \leq i \leq m$, q_i is the i-th vertex of the convex hull $CH(Q)$ of Q. Furthermore, Q is an *x-monotone convex chain*, if $\forall\ 1 \leq i \leq m$, q_i is the i-th vertex of the lower hull $LH(Q)$ of Q (i.e., Q is a downward-convex point set, and the x-coordinates of the points q_i are monotonically increasing).

The *convex cover number* of a set of points P in $\mathbb{R}^2$, denoted as $\kappa_c(P)$, is the minimum number of convex chains using only points in P that together cover all points in P [22]. The *monotone convex cover number* $\kappa(P)$ is defined similarly, further requiring that the convex chains would also be x-monotone. See Fig. 1.3 for an example.


(a) P has convex cover number $\kappa_c(P) = 2$: P can be covered by two convex chains as shown, and easy to verify that $\kappa_c(P) > 1$, because $CH(P) \subsetneq P$.

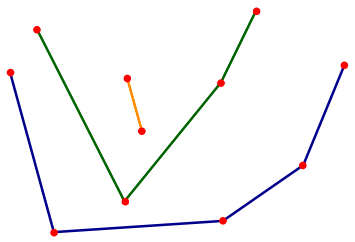

(b) The same point set P has monotone convex cover number $\kappa(P) = 3$.

Figure 1.3: The convex cover number and monotone convex cover number of a point set P.

Our goal is to determine the monotone convex cover number of a given point set P.

Problem 1.6 (Monotone Convex Chain Cover)**.** Given a set of points P in the plane and a parameter k, decide whether the monotone convex cover number $\kappa(P)$ of P is at most k.

Arkin et al. [22] proved that determining the convex cover number is NP-complete, and also presented an $O(\log n)$-approximation algorithm. In this work, we similarly prove that computing the monotone convex chain cover number is also NP-complete.

A somewhat related problem is *monotone subsequence cover*, which asks for deciding whether a permutation π can be partitioned into k monotone subsequences. If we require all subsequences to be monotonically increasing, then the problem can be solved in polynomial time by finding the longest antichain, using Dilworth's theorem [101]. The problem becomes NP-hard when each subsequence is allowed to be either (monotonically) increasing or decreasing [209]. Heggernes et al. showed that this problem is FPT, by giving an algorithm that solves the problem in $2^{O(k^2 \log k)} n^{O(1)}$ time [137]. They reduced the monotone subsequence cover problem to the cochromatic number problem, and provided an FPT algorithm to compute the cochromatic number on perfect graphs.

The convex cover number is also related to other mathematical concepts, such as the convex partition number $\kappa_p(P)$, which asks for the minimum number of convex chains covering P, where the convex chains are required to be pairwise-disjoint convex hulls [22]. Computing the convex partition number exactly is also NP-complete, and an $O(\log n)$-approximation algorithm is known [22]. Another related concept is the reflexivity $\rho(P)$, which is the minimum number of reflex vertices (i.e., having interior angle $> \pi$) in a simple polygonalization of P [3, 22]. One more similar problem is to compute a planar subdivision of the input point set P, where each bounded interior face is a convex polygon, and minimizing the number of such convex polygon faces. Polynomial-time constant factor approximation algorithm exists [150], but the best known exact algorithm runs in $n^{O(k)}$ time for deciding whether this number is at most k [118]. If we parameterize by the number of inner points k' in P, this problem is known to be FPT [130, 196].

Despite the extensive studies for related problems, there are only few previous works focusing on the complexity of monotone convex chain cover. A natural conjecture is that the monotone convex chain cover problem we study here is harder than the monotone subsequence cover problem, due to the additional convexity constraint (constraints on triples instead of pairs of points). One particular open question asked by Eppstein is whether convex chain cover is FPT [112, Open Problem 11.16], and the same question can also be asked for monotone convex chain cover. In Chapter 6, we provide a conditional hardness result showing that monotone convex chain cover does not have a polynomial kernel, unless NP $\subseteq$ coNP/poly, taking a step towards resolving the parameterized complexity of the problem.

Worst-case convex cover number. There are previous results on bounding the convex cover number in the worst case. Erdős and Szekeres proved that any point set with size n contains a convex subset with size $\Omega(\log n)$ [114, 115]. This is related to the famous happy ending problem [114], which had been recently (nearly) resolved by Suk [197]. As a consequence, Urabe showed that $\kappa_c(n) = \Theta(\frac{n}{\log n})$ [202], where $\kappa_c(n)$ denotes the worst-case convex cover number of a set of n points. The hidden constant in the Θ-notation is also of interest to some researchers, see e.g., [182].

For the monotone convex cover number, the trivial $\Theta(n)$ bound is tight: consider the input points forming a concave set as shown in Fig. 1.4, in this case the monotone convex cover number and the convex cover number are far apart. In the random case and the grid case, better bounds are known [99, 135].

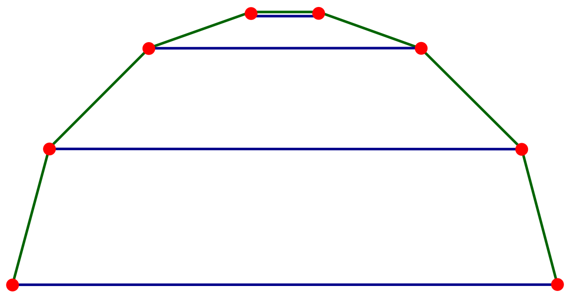


Figure 1.4: A concave point set P with even size n has convex cover number $\kappa_c(P) = 1$ (green chain) and monotone convex cover number $\kappa(P) = n/2$ (blue chains).

Approximation algorithms. It is not hard to get a polynomial time approximation algorithm for computing the convex cover number, with an $O(\log n)$-approximation factor [22]. The idea is to use the greedy algorithm for general set cover, where in each iteration we greedily choose the convex chain with the largest number of points in the remaining set of points. Note that we can compute the largest convex chain in polynomial time, by dynamic programming [172]. This algorithm readily leads to an $O(\log n)$-approximation for the monotone convex cover number. An open question is whether getting constant factor approximation is possible for either of the problems.

XP algorithm. XP (slicewise polynomial) is the class of parameterized problems that can be solved in time of the form $f(k) \cdot n^{g(k)}$ for some functions f and g, where n denote the size of the problem instance, and k is the parameter. If a problem is FPT, then it is also in XP.

A simple observation is the convex chain cover problem can be solved in $n^{O(k)}$ time, using dynamic programming [112, Section 11.5]. The same algorithm works for monotone convex chain cover, and for completeness we briefly redescribe it here. Suppose we want to decide whether $\kappa(P) \leq k$. In each state of dynamic programming, we maintain k x-monotone convex chains (may be empty), and keep track of the last two points of each monotone convex chain. Sort the input points according to their x-coordinates in increasing order, and attach them one by one to the tails of the convex chains, enumerating each combination. We can verify whether the current point can be added to a particular convex chain while maintaining convexity, using its relative position with the stored last two points of the chain.

There are $n^{O(k)}$ states, and computing the value of each state takes polynomial time, thus this problem is in XP. This is still the fastest algorithm known to the best of our knowledge.

1.7 OVERVIEW OF THE NEW RESULTS

In this thesis, we obtain a variety of new results on the problems that we have discussed in Sec. 1.2–Sec. 1.6. Below is a quick overview; detailed statements of all the results can be found at the beginning of each chapter.

In Chapter 2, we focus on static geometric set cover, where all points and objects are given in advance. As the problem is NP-hard for many classes of geometric objects, we are interested in designing $O(1)$-approximation algorithms, in particular, efficient algorithms that run in near-linear time. For the unweighted problem, we present near-optimal algorithms for disks in $\mathbb{R}^2$ and halfspaces in $\mathbb{R}^3$, in particular, $O(n \log^3 n \log\log n)$ deterministic and $O(n \log n (\log\log n)^{O(1)})$ randomized, which are further improved to optimal $O(n \log n)$ time in the next chapter. We then extend our approach to solve the weighted problem for the same types of ranges, in randomized $O(n \log^4 n \log\log n)$ time.

In Chapter 3, we turn our attention to geometric set cover problems in dynamic settings, allowing insertions and deletions of both points and objects. The goal is to efficiently maintain a set cover solution (satisfying certain quality requirement) for the dynamic problem instance. We give a plethora of new dynamic geometric set cover data structures for various geometric ranges, which significantly improve and extend the previous results. For example, we substantially improve *all* the main results of Agarwal et al. [10] on unweighted intervals in 1D (from $O(n^\delta)$ to $O(\log^3 n)$) and unweighted unit squares in 2D (from $O(n^{1/2+\delta})$ to $2^{O(\sqrt{\log n})}$). Our other results include axis-aligned squares, halfplanes in $\mathbb{R}^2$, disks in $\mathbb{R}^2$ and halfspaces in $\mathbb{R}^3$. We also give the first sublinear results for the weighted version of the problem. The detailed time bounds of our dynamic data structures are summarized in Table 3.1.

In Chapter 4, we investigate the fine-grained complexity of the discrete k-center problem and related (exact) geometric set cover problems when k or the size of the cover is small. Question 1.1 asks whether there is a subquadratic-time algorithm for the rectilinear discrete 3-center problem. We answer it in the affirmative for dimension $d = 2$, by presenting the first subquadratic algorithms for unweighted and weighted size-3 set cover for rectangles in $\mathbb{R}^2$. We also prove a number of conditional lower bounds for these problems in constant dimensions, including superlinear conditional lower bounds for size-3 set cover using weighted unit squares in $\mathbb{R}^2$, unweighted boxes in $\mathbb{R}^3$, or unit hypercubes in $\mathbb{R}^4$ (which is equivalent to rectilinear discrete 3-center in $\mathbb{R}^4$). We also prove a near-quadratic lower bound for weighted size-6 set cover for rectangles in $\mathbb{R}^2$.

In Chapter 5, we develop polynomial-time approximation algorithms for another problem related to geometric set cover, namely, enclosing points with geometric objects. For example, we get an $O(\alpha(n)\log n)$-approximation algorithm for enclosing all points using arbitrary line segments in $\mathbb{R}^2$ (where $\alpha(n)$ denote the inverse Ackermann function), and an $O(\log n)$-approximation algorithm for enclosing all points using disks in $\mathbb{R}^2$. We solve this problem by adapting techniques for approximating geometric set cover. Our algorithms also more generally work for a colored version of the problem.

In Chapter 6, we inspect the FPT status of the monotone convex chain cover problem, where the problem asks for finding the minimum number of x-monotone convex chains $\kappa(P)$ that can together cover a point set P. We show that deciding whether $\kappa(P) \leq k$ is NP-hard and does not have a polynomial kernel, unless NP $\subseteq$ coNP/poly.

At the end of the thesis (Chapter 7), we summarize with a number of open questions to be further explored.

Paper	Chapter
Timothy M. Chan and Qizheng He. Faster approximation algorithms for geometric set cover. Proceedings of the 36th Annual Symposium on Computational Geometry (SoCG), vol. 164. pages 27:1–27:14, 2020. [71]	Chapter 2
Timothy M. Chan and Qizheng He. More dynamic data structures for geometric set cover with sublinear update time. Journal of Computational Geometry (JoCG), 2022. Originally appeared at SoCG'21. [72]	Chapter 3 (3.4–3.6)
Timothy M. Chan, Qizheng He, Subhash Suri, and Jie Xue. Dynamic geometric set cover, revisited. Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 3496–3528, 2022. [73]	Chapter 3 (3.7–3.12)
Timothy M. Chan, Qizheng He, and Yuancheng Yu. On the fine-grained complexity of small-size geometric set cover and discrete k-center for small k. Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP), vol. 261. pages 34:1–34:19, 2023. [74]	Chapter 4
Qizheng He. On the FPT status of monotone convex chain cover. Proceedings of the 35th Canadian Conference on Computational Geometry (CCCG), pages 307–312, 2023. [136]	Chapter 6

Table 1.3: The papers included and their location within this thesis.

The results of this thesis have appeared in five published papers: see Table 1.3 for the citations.

CHAPTER 2: STATIC GEOMETRIC SET COVER

In this chapter, we study the static geometric set cover problem: given a set X of points in $\mathbb{R}^d$ and a set S of geometric objects, find the smallest subset of objects from S to cover all points in X. We are interested in designing $O(1)$-approximation algorithms, in particular, efficient algorithms that run in near-linear time.

2.1 OUR RESULTS

We improve the running times of $O(1)$-approximation algorithms for the set cover problem in geometric settings, specifically, covering points by disks in the plane, or covering points by halfspaces in three dimensions. Our results are as follows:

- In the unweighted case, Agarwal and Pan [SoCG'14] gave a randomized $O(n \log^4 n)$-time, $O(1)$-approximation algorithm, by using variants of the multiplicative weight update (MWU) method combined with geometric data structures. We simplify the data structure requirement in one of their methods, and obtain a new *deterministic* $O(n \log^3 n \log\log n)$-time, $O(1)$-approximation algorithm for set cover for 2D disks and 3D halfspaces (Theorem 2.1), which besides eliminating randomization is also a little faster than Agarwal and Pan's.

- With further new ideas, we obtain a still faster randomized near-linear time $O(1)$-approximation algorithm for set cover for 2D disks and 3D halfspaces (Theorem 2.2). Its running time is $O(n \log n (\log\log n)^{O(1)})$, which is essentially optimal[3] ignoring minor $\log\log n$ factors.

- For the weighted problem, we also give a randomized $O(n \log^4 n \log\log n)$-time, $O(1)$-approximation algorithm, by simple modifications to the MWU method and the quasi-uniform sampling technique (Theorem 2.3).

Although generally shaving logarithmic factors may not be the most important endeavor, the problem is fundamental enough that we feel it's worthwhile to find the most efficient algorithm possible.

[3]Even for 1D intervals, just deciding whether a set cover solution exists requires $\Omega(n \log n)$ time in the algebraic decision-tree model, by Ben-Or's technique [30].

Techniques. Our approach interestingly is to go back to Agarwal and Pan's first MWU algorithm. We show that with one simple modification, the data structure requirement can actually be relaxed: namely, for the approximate counting structure, there is no need for weights, and the only update operation is insertion. By standard techniques, insertion-only data structures reduce to static data structures. This simple idea immediately yields our deterministic result. (Before, Bus et al. [48] also aimed to find variants of Agarwal and Pan's first algorithm with simpler data structures, but they did not achieve improved theoretical time bounds.) Our best randomized result requires a more sophisticated combination of several additional ideas. In particular, we incorporate random sampling in the MWU algorithm, and extensively use *shallow cuttings*, in both primal and dual space.

For the weighted problem, we observe that a simple direct modification of the standard MWU algorithm of Brönnimann and Goodrich, or Agarwal and Pan's first algorithm, can also solve the LP for weighted geometric set cover, with arguably simpler data structures than Chekuri et al.'s [82]. Secondly, we observe that an ε-net of small weight can be constructed in near-linear time, by using quasi-uniform sampling more carefully. This leads to a randomized $O(n\log^4 n\log\log n)$-time, $O(1)$-approximation algorithm for weighted set cover for 3D halfspaces (and thus for 2D disks).

Remark. We have stated our results for set cover for 3D halfspaces. This case is arguably the most central. It is equivalent to hitting set for 3D halfspaces, by duality, and also includes set cover and hitting set for 2D disks as special cases, by the standard lifting transformation. The case of 3D dominance ranges is another special case, by a known transformation [76, 183] (although for the dominance case, word-RAM techniques can speed up the algorithms further). The ideas here are likely useful also in the other cases considered in Agarwal and Pan's paper (e.g., hitting set for rectangles, set cover for fat triangles, etc.), but in the interest of keeping this chapter focused, we will not discuss these implications.

2.2 PRELIMINARIES

ε-nets. Let X be a set of points and S be a set of objects. For a point p, its *depth* in S refers to the number of objects in S containing p. A point p is said to be ε-*light* in S, if it has depth $\leq \varepsilon|S|$ in S; otherwise it is ε-*heavy*. A subset of objects $T \subseteq S$ is an ε-*net* of S if T covers all points that are ε-heavy in S.

It is known that there exists an ε-net with size $O(\frac{1}{\varepsilon})$ for any set of halfspaces in 3D or disks in 2D [168] (or more generally for objects in the plane with linear union complexity [93]).

2.2.1 The basic MWU algorithm

We first review the standard multiplicative weight[4] update (MWU) algorithm for geometric set cover, as described by Brönnimann and Goodrich [46] (which generalizes an earlier algorithm by Clarkson [89], and is also well known outside of computational geometry).

Let X be the set of input points and S be the set of input objects, with $n = |X| + |S|$. Let opt denote the size of the minimum set cover. We assume that a value $t = \Theta(\mathsf{opt})$ is known; this assumption will be removed later by a binary search for t. In the following pseudocode, we work with a multiset $\hat{S}$; in measuring size or counting depth, we include multiplicities (e.g., $|\hat{S}|$ is the sum of the multiplicities of all its elements).

```
1: Guess a value $t \in [\mathsf{opt}, 2\,\mathsf{opt}]$ and set $\varepsilon = \frac{1}{2t}$.
2: Define a multiset $\hat{S}$ where each object $i$ in $S$ initially has multiplicity $m_i = 1$.
3: while we can find a point $p \in X$ which is $\varepsilon$-light in $\hat{S}$ do
4:     for each object $i$ containing $p$ do          ▷ call lines 4–5 a multiplicity-doubling step
5:         Double its multiplicity $m_i$.
6: Return an $\varepsilon$-net of the multiset $\hat{S}$.
```

Figure 2.1: MWU (unoptimized version) for set cover.

Since at the end all points in X are ε-heavy in $\hat{S}$, the returned subset is a valid set cover of X. For halfspaces in 3D or disks in 2D, its size is $O(\frac{1}{\varepsilon}) = O(t) = O(\mathsf{opt})$.

A standard analysis shows that the algorithm always terminates after $O(t \log \frac{n}{t})$ multiplicity-doubling steps. We include a quick proof: Each multiplicity-doubling step increases $|\hat{S}|$ by a factor of at most $1 + \varepsilon$, due to the ε-lightness of p. Thus, after z doubling steps, $|\hat{S}| \le n(1+\varepsilon)^z \le ne^{\varepsilon z} = ne^{z/(2t)}$. On the other hand, consider a set cover T^* of size t. In each multiplicity-doubling step, at least one of the objects in T^* has its multiplicity doubled. So, after z multiplicity-doubling steps, the total multiplicity in T^* is at least $t2^{z/t}$. We conclude that $t2^{z/t} \le |\hat{S}| \le ne^{z/(2t)}$, implying that $z = O(t \log \frac{n}{t})$.

2.2.2 Agarwal and Pan's (first) MWU algorithm

Next, we review Agarwal and Pan's first variant of the MWU algorithm [14]. One issue in implementing the original algorithm lies in the test in line 3: searching for one light point

[4] In our algorithm description, we prefer to use the term "multiplicity" instead of "weight", to avoid confusion with the weighted set cover problem later.

by scanning all points in X from scratch every time seems inefficient. In Agarwal and Pan's refined approach, we proceed in a small number of rounds, where in each round, we examine the points in X in a fixed order and test for lightness in that order.

```
1: Guess a value $t \in [\mathsf{opt}, 2\,\mathsf{opt}]$ and set $\varepsilon = \frac{1}{2t}$.
2: Define a multiset $\hat{S}$ where each object $i$ in $S$ initially has multiplicity $m_i = 1$.
3: loop                                          ▷ call this the start of a new round
4:     for each point $p \in X$ in any fixed order do
5:         while $p$ is $\varepsilon$-light in $\hat{S}$ do
6:             for each object $i$ containing $p$ do  ▷ call lines 6–7 a multiplicity-doubling step
7:                 Double its multiplicity $m_i$.
8:             if the number of multiplicity-doubling steps in this round exceeds $t$ then
9:                 Go to line 3 and start a new round.
10:    Terminate and return an $\frac{\varepsilon}{2}$-net of the multiset $\hat{S}$.
11: end loop
```

Figure 2.2: Refined MWU for set cover.

To justify correctness, observe that since each round performs at most t multiplicity-doubling steps, $|\hat{S}|$ increases by a factor of at most $(1+\varepsilon)^t \leq e^{\varepsilon t} \leq e^{1/2} < 2$. Thus, a point p that is checked to be ε-heavy in $\hat{S}$ at any moment during the round will remain $\frac{\varepsilon}{2}$-heavy in $\hat{S}$ at the end of the round.

Since all but the last round performs t multiplicity-doubling steps and we have already shown that the total number of such steps is $O(t \log \frac{n}{t})$, the number of rounds is $O(\log \frac{n}{t})$.

2.3 "NEW" MWU ALGORITHM

Agarwal and Pan's algorithm still requires an efficient data structure to test whether a given point is light, and the data structure needs to support dynamic changes to the multiplicities. We propose a new variant that requires simpler data structures.

Our new algorithm is almost identical to Agarwal and Pan's, but with just one very simple change! Namely, after line 3, at the beginning of each round, we add the following line, to readjust all multiplicities:

```
3.5:   for each object $i$, reset its multiplicity $m_i \leftarrow \lceil m_i \frac{10n}{|\hat{S}|} \rceil$.
```

Figure 2.3: Readjusting all multiplicities at the beginning of each round.

To analyze the new algorithm, consider modifying the multiplicity m_i instead to $\lceil m_i \frac{10n}{|\hat{S}|} \rceil \cdot \frac{|\hat{S}|}{10n}$. The algorithm behaves identically (since the multiplicities are identical except for a common rescaling factor), but is more convenient to analyze. In this version, multiplicities are nondecreasing over time (though they may be non-integers). After the modified line 3.5, the new $|\hat{S}|$ is at most $\sum_i \left(m_i \frac{10n}{|\hat{S}|} + 1\right) \cdot \frac{|\hat{S}|}{10n} \leq 11n \cdot \frac{|\hat{S}|}{10n} = 1.1|\hat{S}|$. If the algorithm makes z multiplicity-doubling steps, then it performs line 3.5 at most z/t times and we now have $|\hat{S}| \leq n(1+\varepsilon)^z \cdot 1.1^{z/t} \leq ne^{z/(2t)} \cdot 1.1^{z/t}$. This is still sufficient to imply that $z = O(t \log \frac{n}{t})$, and so the number of rounds remains $O(\log \frac{n}{t})$.

Now, let's go back to line 3.5 as written. The advantage of this multiplicity readjustment step is that it decreases $|\hat{S}|$ to $\sum_i \left(m_i \frac{10n}{|\hat{S}|} + 1\right) = O(n)$. At the end of the round, $|\hat{S}|$ increases by a factor of at most $(1+\varepsilon)^t < 2$ and so remains $O(n)$. Thus, in line 7, instead of doubling the multiplicity of an object, we can just repeatedly increment the multiplicity (i.e., insert one copy of an object) to reach the desired value. The total number of increments per round is $O(n)$.

Note that in testing for ε-lightness in line 5, a constant-factor approximation of the depth is sufficient, with appropriate adjustments of constants in the algorithm. Also, although the algorithm as described may test the same point p for lightness several times in a round, this can be easily avoided: we just keep track of the increase D in the depth of the current point p; the new depth of p can be 2-approximated by the maximum of the old depth and D.

To summarize, an efficient implementation of each round of the new algorithm requires solving the following geometric data structure problems (Report for line 6, and Approx-Count-Decision for line 5):

Problem Report**:** Design a data structure to store a static set S of size $O(n)$ so that given a query point $p \in X$, we can report all objects in S containing the query point p. Here, the output size of a query is guaranteed to be at most $O(k)$ where $k := \frac{n}{t}$ (since ε-lightness of p implies that its depth is at most $\varepsilon|\hat{S}| = \Theta(\frac{n}{t})$ even including multiplicities).

Problem Approx-Count-Decision**:** Design a data structure to store a multiset $\hat{S}$ of size $O(n)$ so that given a query point $p \in X$, we can either declare that the number of objects in $\hat{S}$ containing p is less than a fixed threshold value k, or that the number is more than $\frac{k}{c}$, for some constant $c > 1$. Here, the threshold again is $k := \frac{n}{t}$ (since $\varepsilon|\hat{S}| = \Theta(\frac{n}{t})$). The data structure should support the following type of updates: insert one copy of an object to $\hat{S}$. (Deletions are not required.) Each point in X is queried once.

To bound the cost of the algorithm:

- Let T_{report} denote the total time for $O(t)$ queries in Problem Report.
- Let T_{count} denote the total time for $O(n)$ queries and $O(n)$ insertions in Problem Approx-Count-Decision. (Note that the initialization of $\hat{S}$ at the beginning of the round can be done by $O(n)$ insertions.)
- Let T_{net} denote the time for computing an ε-net of size $O(\frac{1}{\varepsilon})$ for a given multiset $\hat{S}$ of size $O(n)$.

The total running time over all $O(\log \frac{n}{t})$ rounds is

$$O\left((T_{\text{report}} + T_{\text{count}}) \log \tfrac{n}{t} \; + \; T_{\text{net}}\right). \tag{2.1}$$

2.4 IMPLEMENTATIONS OF MWU

In this section, we describe specific implementations of our MWU algorithm when the objects are halfspaces in 3D (which include disks in 2D as a special case by the standard lifting transformation). We first consider deterministic algorithms.

2.4.1 Deterministic version

Shallow cuttings. We begin by reviewing an important tool that we will use several times later. For a set of n planes in $\mathbb{R}^3$, a *k-shallow ε-cutting* is a collection of interior-disjoint polyhedral cells, such that each cell intersects at most εn planes, and the union of the cells cover all points of level at most k (the *level* of a point refers to the number of planes below it). The list of all planes intersecting a cell Δ is called the *conflict list* of Δ. Matoušek [166] proved the existence of a k-shallow $(\frac{ck}{n})$-cutting with $O(\frac{n}{k})$ cells for any constant c. Chan and Tsakalidis [78] gave an $O(n \log \frac{n}{k})$-time deterministic algorithm to construct such a cutting, along with all its conflict lists (an earlier randomized algorithm was given by Ramos [190]). If c is sufficiently large, the cells may be made "downward", i.e., they all contain $(0, 0, -\infty)$.

Constructing ε-nets. The best known deterministic algorithm for constructing ε-nets for 3D halfspaces is by Chan and Tsakalidis [78] and runs in $T_{\text{net}} = O(n \log \frac{1}{\varepsilon}) = O(n \log n)$ time.

The result follows directly from their shallow cutting algorithm (using a simple argument of Matoušek [166]): Without loss of generality, assume that all halfspaces are upper halfspaces, so depth corresponds to level with respect to the bounding planes (we can compute a net for lower halfspaces separately and take the union, with readjustment of ε by a factor of 2). We construct an (εn)-shallow $\frac{\varepsilon}{2}$-cutting with $O(\frac{1}{\varepsilon})$ cells, and for each cell, add a plane completely below the cell (if it exists) to the net. To see correctness, for a point p with level εn, consider the cell Δ containing p; at least $\varepsilon n - \frac{\varepsilon n}{2} > 0$ planes are completely below Δ, and so the net contains at least one plane below p.

Solving Problem Report. This problem corresponds to 3D *halfspace range reporting* in dual space, and by known data structures [7, 56, 78], the total time to answer $O(t)$ queries is $T_{\text{report}} = O(t \cdot (\log n + k)) = O(t \log n + n)$, assuming an initial preprocessing of $O(n \log n)$ time (which is done only once).

This result also follows directly from shallow cuttings (since space is not our concern, the solution is much simplified): Without loss of generality, assume that all halfspaces are upper halfspaces. We construct a k-shallow $O(\frac{k}{n})$-cutting with $O(\frac{n}{k})$ downward cells. Given a query point $p \in X$, we find the cell containing p, which can be done in $O(\log n)$ time by planar point location; we then do a linear search over its conflict list, which has size $O(k)$.

Note that the point location operations can be actually be done during preprocessing in $O(n \log n)$ time since X is known in advance. This lowers the time bound for $O(t)$ queries to $T_{\text{report}} = O(tk) = O(n)$.

Solving Problem Approx-Count-Decision. This problem corresponds to the decision version of 3D *halfspace approximate range counting* in dual space, and several deterministic and randomized data structures have already been given in the static case [6, 8], achieving $O(\log n)$ query time and $O(n \log n)$ preprocessing time.

This result also follows directly from shallow cuttings: Without loss of generality, assume that all halfspaces are upper halfspaces. We construct a $\frac{n}{b^i}$-shallow $O(\frac{1}{b^i})$-cutting with $O(b^i)$ downward cells for every $i = 1, \ldots, \log_b n$ for some constant b. Chan and Tsakalidis's algorithm can actually construct all $O(\log n)$ such cuttings in $O(n \log n)$ total time. With these cuttings, we can compute an $O(1)$-approximation to the depth/level of a query point p by simply finding the largest i such that p is contained in a cell of the $\frac{n}{b^i}$-shallow cutting (the level of p would then be $O(\frac{n}{b^i})$ and at least $\frac{n}{b^{i+1}}$). In Chan and Tsakalidis's construction, each cell in one cutting intersects $O(1)$ cells in the next cutting, and so we can locate the cells containing p in $O(1)$ time per i, for a total of $O(\log n)$ time.

To solve Problem Approx-Count-Decision, we still need to support insertion. Although the approximate decision problem is not decomposable, the above solution solves the approximate counting problem, which is decomposable, so we can apply the standard *logarithmic method* [31] to transform the static data structure into a semi-dynamic, insertion-only data structure. The transformation causes a logarithmic factor increase, yielding in our case $O(\log^2 n)$ query time and $O(\log^2 n)$ insertion time. Thus, the total time for $O(n)$ queries and insertions is $T_{\text{count}} = O(n \log^2 n)$.

Conclusion. By (2.1), the complete algorithm has running time $O((T_{\text{report}} + T_{\text{count}}) \log \frac{n}{t} + T_{\text{net}}) = O((n + n \log^2 n) \log \frac{n}{t} + n \log n) = O(n \log^3 n)$.

One final issue remains: we have assumed that a value $t \in [\mathsf{opt}, 2\,\mathsf{opt}]$ is given. In general, either the algorithm produces a solution of size $O(t)$, or (if it fails to complete within $O(\log \frac{n}{t})$ rounds) the algorithm may conclude that $\mathsf{opt} > t$. We can thus find an $O(1)$-approximation to opt by a binary search over t among the $O(\log n)$ possible powers of 2, with $O(\log \log n)$ calls to the algorithm. The final time bound is $O(n \log^3 n \log \log n)$.

Theorem 2.1. Given a set X of points and a set S of halfspaces in $\mathbb{R}^3$, where $|X| + |S| = n$, there exists an algorithm that can find a subset of halfspaces from S covering all points in X, of size within $O(1)$ factor of the minimum, in deterministic $O(n \log^3 n \log \log n)$ time.

2.4.2 Randomized version 1

We now describe a better solution to Problem Approx-Count-Decision, by using randomization and the fact that all query points (namely, X) are given in advance.

Reducing the number of insertions in Problem Approx-Count-Decision. In solving Problem Approx-Count-Decision, one simple way to speed up insertions is to work with a random sample R of $\hat{S}$. When we insert an object to $\hat{S}$, we independently decide to insert it to the sample R with probability $\rho := \frac{c_0 \log n}{k}$, or ignore it with probability $1 - \rho$, for a sufficiently large constant c_0. (Different copies of an object are treated as different objects here.) It suffices to solve the problem for the sample R with the new threshold around ρk.

To justify correctness, consider a fixed query point $p \in X$. Let $x_1, x_2, \ldots$ be the sequence of objects in $\hat{S}$ that contain p, in the order in which they are inserted (extend the sequence arbitrarily to make its length greater than k). Let $y_i = 1$ if object x_i is chosen to be in the sample R, or 0 otherwise. Note that the x_i's may not be independent (since the object we insert could depend on random choices made before); however, the y_i's are independent. By the Chernoff bound, $\sum_{i=1}^{k/c} y_i \le \frac{(1+\delta)\rho k}{c}$ and $\sum_{i=1}^{k} y_i \ge (1-\delta)\rho k$ with probability $1-e^{-\Omega(\rho k)} = 1-n^{-\Omega(c_0)}$ for any fixed constant $\delta > 0$. Thus, with high probability (w.h.p.), at any time, if the number of objects in R containing p is more than $\frac{(1+\delta)\rho k}{c}$, then the number of objects in $\hat{S}$ containing p is more than $\frac{k}{c}$; if the former number is less than $(1-\delta)\rho k$, then the later number is less than k. Since there are $O(n)$ possible query points, all queries are correct with high probability.

By this strategy, the number of insertions is reduced to $O(\rho n) = O(\frac{n}{k}\log n) = O(t \log n)$ with high probability.

Preprocessing step. Next we use a known preprocessing step to ensure that each object contains at most $\frac{n}{t}$ points, in the case of 3D halfspaces. This subproblem was addressed in Agarwal and Pan's paper [14] (where it was called "(P5)"—curiously, they used it to implement their second algorithm but not their first MWU-based algorithm.) We state a better running time:

Lemma 2.1. In $O(n \log t)$ time, we can find a subset $T_0 \subseteq S$ of $O(t)$ halfspaces, such that after removing all points in X covered by T_0, each halfspace of S contains at most $\frac{n}{t}$ points.

Proof. We may assume that all halfspaces are upper halfspaces. We work in dual space, where S is now a set of points and X is a set of planes. The goal is to find a subset $T_0 \subseteq S$ of $O(t)$ points such that after removing all planes of X that are below some points of T_0, each point of S has depth/level at most $\frac{n}{t}$.

We proceed in rounds. Let b be a constant. In the i-th round, assume that all points of S have level $\le \frac{n}{b^i}$. Compute a $\frac{n}{b^i}$-shallow $\frac{1}{b^{i+1}}$-cutting with $O(b^i)$ cells. In each cell, add an arbitrary point of S (if exists) to the set T_0. In total $O(b^i)$ points are added. Remove all planes that are below these added points from X.

Consider a point p of S. Let Δ be the cell containing p, and let q be the point in Δ that was added to T_0. Any plane that is below p but not removed (and thus above q) must intersect Δ, so there can be at most $\frac{n}{b^{i+1}}$ such planes. Thus, after the round, the level of p is at most $\frac{n}{b^{i+1}}$. We terminate when b^i reaches t. The total size of T_0 is $O(\sum_{i=1}^{\log_b t} b^i) = O(t)$.

Naively computing each shallow cutting from scratch by Chan and Tsakalidis's algorithm would require $O(n \log n \cdot \log n) = O(n \log^2 n)$ total time. But Chan and Tsakalidis's approach can compute multiple shallow cuttings more quickly: given a $\frac{n}{b^i}$-shallow cutting along with its conflict lists, we can compute the next $\frac{n}{b^{i+1}}$-shallow cutting along with its conflict lists in $O(n + b^i \log b^i)$ time. However, in our application, before computing the next cutting, we also remove some of the input planes. Fortunately, this type of scenario has been examined in a recent paper by Chan [66], who shows that the approach still works, provided that the next cutting is relaxed to cover only points covered by the previous cutting (see Lemma 8 in his paper); this is sufficient in our application. In our application, we also need to locate the cell containing each point of S. This can still be done in $O(n)$ time given the locations in the previous cutting. Thus, the total time is $O(\sum_{i=1}^{\log_b t}(n + b^i \log b^i)) = O(n \log t)$. QED.

At the end, we add T_0 back to the solution, which still has $O(t)$ total size.

Solving Problem Approx-Count-Decision. We now propose a very simple approach to solve Problem Approx-Count-Decision: just explicitly maintain the depth of all points in X. Each query then trivially takes $O(1)$ time. When inserting an object, we find all points contained in the object and increment their depths.

Due to the above preprocessing step, the number of points contained in the object is $O(\frac{n}{t})$. For the case of 3D halfspaces, we can find these points by halfspace range reporting; as explained before for Problem Report, this can be done in $O(\frac{n}{t})$ time by using shallow cuttings, after an initial preprocessing in $O(n \log n)$ time. Thus, each insertion takes $O(\frac{n}{t})$ time. Since the number of insertions has been reduced to $O(t \log n)$ by sampling, the total time for Problem Approx-Count-Decision is $T_{\text{count}} = O((t \log n) \cdot \frac{n}{t}) = O(n \log n)$.

Conclusion. By (2.1), the complete randomized algorithm has running time $O((T_{\text{report}} + T_{\text{count}}) \log \frac{n}{t} + T_{\text{net}}) = O((n + n \log n) \log \frac{n}{t} + n \log n) = O(n \log n \log \frac{n}{t}) = O(n \log^2 n)$ (even including the $O(n \log n)$-time preprocessing step). Including the binary search for t, the time bound is $O(n \log^2 n \log \log n)$.

2.4.3 Randomized version 2

Finally, we combine the ideas from both the deterministic and randomized implementations, to get our fastest randomized algorithm for 3D halfspaces.

Solving Problem Approx-Count-Decision. We may assume that all halfspaces are upper halfspaces. We work in dual space, where $\hat{S}$ is now a multiset of points and X is a set of planes. In a query, we want to approximately count the number of points in $\hat{S}$ that are above a query plane in X. By the sampling reduction from Section 2.4.2, we may assume that the number of insertions to $\hat{S}$ is $O(t \log n)$. By the preprocessing step from Section 2.4.2, we may assume that all points in $\hat{S}$ have level at most $\frac{n}{t}$.

Compute a $\frac{n}{t}$-shallow $O(\frac{1}{t})$-cutting with $O(t)$ downward cells, along with its conflict lists. For each point $p \in R$, locate the cell containing p. All this can be done during a (one-time) preprocessing in $O(n \log n)$ time.

For each cell Δ, we maintain $\hat{S} \cap \Delta$ in a semi-dynamic data structure for 3D approximate halfspace range counting. As described in Section 2.4.1, we get $O(\log^2 n_\Delta)$ query and insertion time, where $n_\Delta = |\hat{S} \cap \Delta|$.

In an insertion of a point p to $\hat{S}$, we look up the cell Δ containing p and insert the point to the approximate counting structure in Δ.

In a query for a plane $h \in X$, we look up the cells Δ whose conflict lists contain h, answer approximate counting queries in these cells, and sum the answers.

We bound the total time for all insertions and queries. For each cell Δ, the number of insertions in its approximate counting structure is n_Δ and the number of queries is $O(\frac{n}{t})$ (since each plane $h \in X$ is queried once). The total time is

$$O\left(\sum_{\Delta} \left(\frac{n}{t} + n_\Delta\right) \log^2 n_\Delta\right). \tag{2.2}$$

Since there are $O(t)$ terms and $\sum_\Delta n_\Delta = O(t \log n)$, we have $n_\Delta = O(\log n)$ "on average"; applying Jensen's inequality to the first term, we can bound the sum by $O(n \log^2 \log n + t \log^3 n)$. Thus, $T_{\text{count}} = O(n \log^2 \log n + t \log^3 n)$.

Conclusion. By (2.1), the complete randomized algorithm has running time $O((T_{\text{report}} + T_{\text{count}}) \log \frac{n}{t} + T_{\text{net}}) = O((n + n \log^2 \log n + t \log^3 n) \log \frac{n}{t} + n \log n) = O(n \log n \log^2 \log n + t \log^4 n)$. If $t \leq n/\log^3 n$, the first term dominates. On the other hand, if $t > n/\log^3 n$, our earlier randomized algorithm has running time $O(n \log n \log \frac{n}{t}) = O(n \log n \log \log n)$. In any case, the time bound is at most $O(n \log n \log^2 \log n)$. Including the binary search for t, the time bound is $O(n \log n \log^3 \log n)$.

Theorem 2.2. Given a set X of points and a set S of halfspaces in $\mathbb{R}^3$, where $|X| + |S| = n$, there exists an algorithm that can find a subset of halfspaces from S covering all points in X, of size within $O(1)$ factor of the minimum, in $O(n \log n \log^3 \log n)$ time by a randomized Monte-Carlo algorithm with error probability $O(n^{-c_0})$ for any constant $c_0 > 0$.

Remark. The number of the $\log \log n$ factors is improvable with still more effort, but we feel it is of minor significance; plus, removing all of the $\log \log n$ factors doesn't seem to be easy using the current methods. In the next chapter, we solve this problem from a very different perspective, showing that ideas from *dynamic* geometric set cover are surprisingly useful to get an optimal result with running time $O(n \log n)$ for the *static* problem. See Theorem 3.3 in Sec. 3.6.

2.5 WEIGHTED 3D HALFSPACES

In this final section, we consider the weighted set cover problem. We define ε-lightness and ε-nets as before, ignoring the weights. It is known that there exists an ε-net of S with total weight $O(\frac{1}{\varepsilon} \cdot \frac{w(S)}{|S|})$, for any set of 3D halfspaces or 2D disks (or objects in 2D with linear union complexity) [69]. Here, the weight $w(S)$ of a set S refers to the sum of the weights of the objects in S.

2.5.1 MWU algorithm in the weighted case

Let X be the set of input points and S be the set of weighted input objects, where object i has weight w_i, with $n = |X| + |S|$. Let opt be the weight of the minimum-weight set cover. We assume that a value $t \in [\mathsf{opt}, 2\,\mathsf{opt}]$ is given; this assumption can be removed by a binary search for t.

We may delete objects with weights $> t$. We may automatically include all objects with weights $< \frac{1}{n}t$ in the solution, and delete them and all points covered by them, since the total weight of the solution increases by only $O(n \cdot \frac{1}{n}t) = O(t)$. Thus, all remaining objects have weights in $[\frac{1}{n}t,\, t]$. By rescaling, we may now assume that all objects have weights in $[1, n]$ and that $t = \Theta(n)$.

In the following, for a multiset $\hat{S}$ where object i has multiplicity m_i, the weight of the multiset is defined as $w(\hat{S}) = \sum_i m_i w_i$.

We describe a simple variant of the basic MWU algorithm to solve the weighted set cover problem. (A more general, randomized MWU algorithm for geometric set cover was given recently by Chekuri, Har-Peled, and Quanrud [82], but our algorithm is simpler to describe and analyze.) The key innovation is to replace doubling with multiplication by a factor $1 + \frac{1}{w_i}$, where w_i is the weight of the concerned object i. (Note that multiplicities may now be non-integers.)

```
1: Guess a value $t \in [\mathsf{opt}, 2\,\mathsf{opt}]$.
2: Define a multiset $\hat{S}$ where each object $i$ in $S$ initially has multiplicity $m_i = 1$.
3: repeat
4:     Find a point $p$ which is $\varepsilon$-light in $\hat{S}$ with $\varepsilon = \frac{1}{2t} \cdot \frac{w(\hat{S})}{|\hat{S}|}$.
5:     for each object $i$ containing $p$ do        ▷ call lines 5–6 a "multiplicity-increasing step"
6:         Multiply its multiplicity $m_i$ by $1 + \frac{1}{w_i}$.
7: until all points are $\varepsilon$-heavy in $\hat{S}$.
8: Return an $\varepsilon$-net of the multiset $\hat{S}$.
```

Figure 2.4: MWU for weighted set cover.

Since at the end all points are ε-heavy in $\hat{S}$, the returned subset is a valid set cover of X. For halfspaces in 3D or disks in 2D, its weight is $O(\frac{1}{\varepsilon} \cdot \frac{w(\hat{S})}{|\hat{S}|}) = O(\mathsf{opt})$.

We now prove that the algorithm terminates in $O(t \log n) = O(n \log n)$ multiplicity-increasing steps.

In each multiplicity-increasing step, $w(\hat{S})$ increases by

$$\sum_{\text{object } i \text{ containing } p} m_i \cdot \tfrac{1}{w_i} \cdot w_i \;=\; \sum_{\text{object } i \text{ containing } p} m_i \;\le\; \tfrac{w(\hat{S})}{2t}, \tag{2.3}$$

i.e., $w(\hat{S})$ increases by a factor of at most $1 + \frac{1}{2t}$. Initially, $w(\hat{S}) \le n^2$. Thus, after z multiplicity-increasing steps, $w(\hat{S}) \le n^2(1 + \frac{1}{2t})^z \le n^2 e^{z/(2t)}$.

On the other hand, consider the optimal set cover T^*. Suppose that object i has its multiplicity increased z_i times. In each multiplicity-increasing step, at least one object in T^* has its multiplicity increased. So, after z multiplicity-increasing steps, $\sum_{i \in T^*} z_i \ge z$ and $\sum_{i \in T^*} w_i \le t$. In particular, $z_i/w_i \ge z/t$ for some $i \in T^*$. Therefore, $w(\hat{S}) \ge (1 + \frac{1}{w_i})^{z_i} w_i \ge (1 + \frac{1}{w_i})^{z_i} \ge 2^{z_i/w_i} \ge 2^{z/t}$ (since $w_i \ge 1$). We conclude that $2^{z/t} \le w(\hat{S}) \le n^2 e^{z/(2t)}$, implying that $z = O(t \log n)$.

Similar to Agarwal and Pan's first MWU algorithm, we can also divide the multiplicity-increasing steps into rounds, with each round performing up to t multiplicity-increasing steps. Within each round, the total weight $w(\hat{S})$ increases by at most $(1 + \frac{1}{2t})^t = O(1)$. Also if $|\hat{S}|$ increases by a constant factor, we immediately start a new round: because $|\hat{S}| \le w(\hat{S})$ and $w(\hat{S})$ may be doubled at most $O(\log n)$ times, this case can happen at most $O(\log n)$ times. This ensures that if a point is checked to be ε-heavy at any moment during a round, it will remain $\Omega(\varepsilon)$-heavy at the end of the round. There are only $O(\log n)$ rounds.

Additional ideas are needed to speed up implementation (in particular, our modified MWU algorithm with multiplicity-readjustment steps does not work as well now). First, we work with an approximation $\tilde{m}_i$ to the multiplicity m_i of each object i. By rounding, we may assume all weights w_i are powers of 2. In the original algorithm, $m_i = (1 + \frac{1}{w_i})^{z_i}$, where z_i is the number of points $p \in Z$ that are contained in object i, and Z be the multiset consisting of all points p that have undergone multiplicity-increasing steps so far. Note that since the total multiplicity is $n^{O(1)}$, we have $z_i = O(w_i \log n)$. Let $Y^{(w_i)}$ be a random sample of Z where each point $p \in Z$ is included independently with probability $\frac{\log^2 n}{w_i}$ (if $w_i = O(\log^2 n)$, we can just set $Y^{(w_i)} = Z$). Let y_i be the number of points $p \in Y^{(w_i)}$ that are contained in object i. By the Chernoff bound, since $\frac{\log^2 n}{w_i} z_i = O(\log^3 n)$, we have $|y_i - \frac{\log^2 n}{w_i} z_i| \le O(\log^2 n)$ with high probability. By letting $\tilde{m}_i = (1 + \frac{1}{w_i})^{y_i w_i / \log^2 n}$, it follows that $\tilde{m}_i$ and m_i are within a factor of $O(1)$ of each other, with high probability, at all times, for all i. Thus, our earlier analysis still holds when working with $\tilde{m}_i$ instead of m_i. Since $z_i = O(w_i \log n)$, we have $y_i = O(\log^3 n)$ with high probability. So, the total number of increments to all y_i and updates to all $\tilde{m}_i$ is $O(n \log^3 n)$. In lines 5–6, we flip a biased coin to decide whether p should be placed in the sample $Y^{(2^j)}$ (with probability $\frac{\log^2 n}{2^j}$) for each j, and if so, we use halfspace range reporting in the dual to find all objects i of weight 2^j containing p, and increment y_i and update $\tilde{m}_i$. Over all $O(n \log n)$ executions of lines 5–6 and all $O(\log n)$ indices j, the cost of these halfspace range reporting queries is $O(n \log n \cdot \log n \cdot \log n)$ plus the output size. As the total output size for the queries is $O(n \log^3 n)$, the total cost is $O(n \log^3 n)$.

We also need to redesign a data structure for lightness testing subject to multiplicity updates: For each j, we maintain a subset $S^{(j)}$ containing all objects i with multiplicity at least 2^j, in a data structure to support approximate depth (without multiplicity). The depth of a point p in $\hat{S}$ can be $O(1)$-approximated by $\sum_j 2^j \cdot (\text{depth of } p \text{ in } S^{(j)})$. Each subset $S^{(j)}$ undergoes insertion only, and the logarithmic method can be applied to each $S^{(j)}$. Since $|\hat{S}| \le w(\hat{S}) \le n^{O(1)}$, there are $O(\log n)$ values of j. This slows down lightness testing by a logarithmic factor, and so in the case of 3D halfspaces, the overall time bound is $O(n \log^4 n \log \log n)$, excluding the ε-net construction time.

2.5.2 Speeding up quasi-uniform sampling

Finally, we show how to efficiently construct an ε-net of the desired weight for 3D halfspaces. We will take advantage of the fact that we need ε-nets only in the discrete setting, with respect to a given set X of $O(n)$ points. Without loss of generality, assume that all halfspaces are upper halfspaces.

We begin by sketching (one interpretation of) the quasi-random sampling algorithm of

Varadarajan [204] and Chan et al. [69]:

```
1:  Let $k = \varepsilon|S|$.
2:  repeat
3:      Remove all points $p \in X$ with depth in $S$ less than $k$.
4:      Move each point $p \in X$ downward so that its depth in $S$ is $\Theta(k)$.
5:      Pick a random sample $R \subseteq S$ of size $|S|/2 + h$ for some appropriate choice of $h$.
6:      Let $S' = S$.
7:      repeat
8:          Find an object $i \in S'$ containing the fewest number of non-equivalent points in $X$.
9:          if object $i$ contains a bad point then add object $i$ to the output.
10:         Remove object $i$ from $S'$.
11:     until $S'$ is empty.
12:     Set $S \leftarrow R$ and $k \leftarrow k/2$.
13: until $k$ is below a constant.
14: Add $S$ to the output.
```

Figure 2.5: Algorithm for quasi-random sampling.

In line 4, we use the property that the objects are upper halfspaces. In line 8, two points p and q of X are considered *equivalent* iff the subset of objects from S' containing p is the same as the corresponding subset for q. In line 9, a point p is said to be *bad* iff its depth in S' is equal to k and its depth in $R \cap S'$ is less than $k/2$.

With appropriate choices of parameters, in the case of 3D halfspaces, Chan et al. [69] showed that the output is an ε-net, with the property that each object of S is in the output with probability $O(\frac{1}{\varepsilon|S|})$ (these events are not independent, so the output is only a "quasi-uniform" sample). This property immediately implies that the output has expected weight $O(\frac{1}{\varepsilon} \cdot \frac{w(S)}{|S|})$. We will not redescribe the proof here, as our interest lies in the running time.

Consider one iteration of the outer repeat loop. For the very first iteration, lines 3–4 can be done by answering $|X|$ halfspace range reporting queries in the dual (reporting up to $O(k)$ objects containing each query point $p \in X$), which takes $O(|S|\log|S| + |X|k)$ total time by using shallow cuttings (as described in Section 2.4.1). As a result, we also obtain a list of the objects containing each point; these lists have total size $O(|X|k)$. In each subsequent iteration, lines 3–4 take only $O(|X|k)$ time by scanning through these lists and selecting the k lowest bounding planes per list.

For each point p, we maintain its depth in S' and its depth in $R \cap S'$. Whenever an object is removed from S', we examine all points in the object, and if necessary, decrement these depth values; this takes $O(|X|k)$ total time (since there are $O(|X|k)$ object-point containment pairs). Then line 9 can be done by scanning through all points in the object; again, this takes $O(|X|k)$ total time.

Line 8 requires more care, as we need to keep track of equivalence classes of points. One way is to use hashing or fingerprinting [174]: for example, map each point p to

$$\left(\sum_{\text{object } i \in S' \text{ containing } p} x^i \right) \bmod u \tag{2.4}$$

for a random $x \in [u]$ and a fixed prime $u \in \Theta(n^c)$, where c is a sufficiently large constant. Then two points are equivalent iff they are hashed to the same value, with high probability. When we remove an object i from S', we examine all points contained in the object, recompute the hash values of these points (which takes $O(1)$ time each, given a table containing $x^i \bmod u$), and whenever we find two equivalent points with the same hash values, we remove one of them. This takes $O(|X|k)$ total time (since there are $O(|X|k)$ object-point containment pairs). To implement line 8, for each object, we maintain a count of the number of points it contains. Whenever we remove a point, we decrement the counts of objects containing it; again, this takes $O(|X|k)$ total time. The minimum count can be maintained in $O(1)$ time per operation without a heap, since the only update operations are decrements (for example, we can place objects in buckets indexed by their counts, and move an object from one bucket to another whenever we decrement).

To summarize, the first iteration of the outer repeat loop takes $O(|S| \log |S| + |X|k)$ time, and each subsequent iteration takes $O(|X|k)$ time. Since k is halved in each iteration, the total time over all iterations is $O(|S| \log |S| + |X|k_0)$ where $k_0 = \varepsilon|S|$.

In our application, we need to compute an ε-net of a *multiset* $\hat{S}$. Since the initial halfspace range reporting subproblem can be solved on the set S without multiplicities, the running time is still $O(|S| \log |S| + |X|k_0)$ but with $k_0 = \varepsilon|\hat{S}|$.

For $|X|, |S| = O(n)$, the time bound is $O(n \log n + nk_0)$, which is still too large. To reduce the running time, we use one additional simple idea: take a random sample $R \subseteq \hat{S}$ of size $\frac{c}{\varepsilon} \log n$ for a sufficiently large constant c. Then $\mathbb{E}[w(R)] = O(\frac{w(\hat{S})}{\varepsilon|\hat{S}|} \log n)$. For a fixed point p of depth $\geq \varepsilon|\hat{S}|$ in $\hat{S}$, the depth of p in R is $\Omega(\varepsilon|R|) = \Omega(\log n)$ with high probability, by the Chernoff bound. We then compute a $\Theta(\varepsilon)$-net of R, which gives us an ε-net of $\hat{S}$, with expected weight $O(\frac{1}{\varepsilon} \cdot \frac{\mathbb{E}[w(R)]}{|R|}) = O(\frac{1}{\varepsilon} \cdot \frac{w(\hat{S})}{|\hat{S}|})$. The net for R is easier to compute, since k_0 is reduced to $\varepsilon|R| = O(\log n)$. The final running time for the ε-net construction is

$O(|S|\log|S| + nk_0) = O(n\log n)$.

We can verify that the net's weight bound holds (and that all points of X are covered), and if not, repeat the algorithm for $O(1)$ expected number of trials.

We conclude:

Theorem 2.3. Given a set X of points and a set S of weighted halfspaces in $\mathbb{R}^3$, where $|X| + |S| = n$, there exists an algorithm that can find a subset of halfspaces from S covering all points in X, of total weight within $O(1)$ factor of the minimum, in $O(n\log^4 n\log\log n)$ expected time by a randomized Las Vegas algorithm.

2.6 LOWER BOUND FOR APPROXIMATE 1D INTERVALS

Our $O(n\log n)$ running time for static geometric set cover (Theorem 3.3) is tight, because deciding whether a feasible set cover exists requires $\Omega(n\log n)$ time even for 1D intervals, which is not difficult to prove by standard techniques in the algebraic decision-tree model [30]. But under the promise that the union of the sets cover all the points, it is not immediately obvious that finding an approximate solution also requires $\Omega(n\log n)$ time. Surprisingly, to the best of our knowledge, this problem has not been addressed in prior literature.

In this part, we prove a matching $\Omega(n\log n)$ lower bound in the comparison model, for any Las Vegas algorithm that computes an $O(1)$-approximate set cover for 1D intervals. The idea is based on an adversary argument for approximating the number of distinct elements, which is inspired by a lower bound proof from Chan [61].

Lemma 2.2. Any Las Vegas algorithm that can compute an α-approximate solution for the number of distinct elements in a set X of n real numbers requires $\Omega(n\log n)$ time in the comparison model, for any constant $\alpha \geq 1$.

Proof. The idea is to let the adversary maintain a candidate interval $\mathcal{I}_i = (\ell_i, r_i)$ for the i-th number $x_i \in X$ ($1 \leq i \leq n$), where one can set the actual value of x_i to be anywhere in the interval $\mathcal{I}_i$, without violating any of the previous comparison results that the algorithm has already performed.

To maintain the candidate intervals, suppose the algorithm performs a new comparison that compares the i-th element x_i with the j-th element x_j (w.l.o.g. assume $i < j$). Let $m_i = (\ell_i + r_i)/2$ (resp. $m_j = (\ell_j + r_j)/2$) denote the middle point of $\mathcal{I}_i$ (resp. $\mathcal{I}_j$). If $m_i \leq m_j$, then return "$x_i < x_j$" as the comparison result, update $\mathcal{I}_i$ to be (ℓ_i, m_i), and update $\mathcal{I}_j$ to be (m_j, r_j). It is easy to verify that any number in $\mathcal{I}_i$ is strictly less than any number in $\mathcal{I}_j$. Otherwise if $m_i > m_j$, then return "$x_i > x_j$", update $\mathcal{I}_i$ to be (m_i, r_i), and update $\mathcal{I}_j$ to be (ℓ_j, m_j).

We say a candidate interval $\mathcal{I}_i$ has depth d, if the element x_i has been compared d times. If $\mathcal{I}_i$ has depth d, then it must have the form $\mathcal{I}_i = (\frac{k}{2^d}, \frac{k+1}{2^d})$ for some integer $0 \le k \le 2^d - 1$.

Suppose the adversary only performs at most $\delta \cdot n \log n$ comparisons. Then there are at most $\frac{\delta}{d_0} \cdot n$ candidate intervals that have depth $\ge d_0 \log n$, where d_0 is a parameter to be set later. For the rest of the candidate intervals that have depth $< d_0 \log n$, there are only at most $\sum_{i=0}^{d_0-1} 2^{i \log n} \le n^{d_0}$ distinct types.

It is possible to assign a value for each number within their candidate intervals, such that all numbers are distinct. In this case, the number of distinct elements in X is n. On the other hand, if we assign the same value for all numbers that have the same candidate interval, then there are only at most $n^{d_0} + \frac{\delta}{d_0} \cdot n$ distinct numbers in X. The algorithm cannot distinguish between these two cases, and since Las Vegas algorithms cannot make any mistakes, it must have approximation factor $\ge \frac{n}{n^{d_0} + \frac{\delta}{d_0} \cdot n}$.

For any constant $\alpha > 1$, we can set $\delta = \frac{1}{4\alpha}$ and $d_0 = \frac{1}{2}$, such that the approximation factor of any Las Vegas algorithm that performs only at most $\delta \cdot n \log n$ comparisons has approximation factor at least $\frac{n}{n^{d_0} + \frac{\delta}{d_0} \cdot n} \ge \alpha$. QED.

As a corollary, we obtain the following lower bound for approximating static set cover for 1D intervals: notice that the problem of α-approximating the number of distinct elements reduces to computing an α-approximate static set cover for 1D intervals. Given a distinct elements instance X, create a static 1D intervals set cover instance (X', S') as follows: for each number $x \in X$, create a point x in X', and create an interval $[x - \delta, x + \delta]$ in S' for a sufficiently small $\delta > 0$, such that the interval only covers the points in X' that equals to x. One can verify that the number of distinct elements in X equals to the optimal set cover solution for (X', S').

Theorem 2.4. Any Las Vegas algorithm that can compute an α-approximate set cover solution for a set X of points and a set S of intervals in $\mathbb{R}$ (where $|X| + |S| = n$) requires $\Omega(n \log n)$ time in the comparison model, for any constant $\alpha \ge 1$.

As a special case, this lower bound holds for any deterministic algorithms. It also readily holds for static set cover for disks in $\mathbb{R}^2$ (and halfspaces in $\mathbb{R}^3$), by replacing each interval $[\ell, r]$ in S with a disk that has that interval as its diameter.

CHAPTER 3: DYNAMIC GEOMETRIC SET COVER

After designing approximation algorithms for *static* geometric set cover in the previous chapter, we now turn our attention to the *dynamic* geometric set cover problem, which asks for maintaining an approximate set cover solution while allowing insertions and deletions of both points and geometric objects.

3.1 OUR RESULTS

In this chapter, we give a plethora of new fully dynamic geometric set cover data structures in 1D, 2D and 3D, as summarized in Table 3.1, which substantially improve *all* the main results of Agarwal et al. [10] on unweighted intervals in 1D and unweighted unit squares in 2D, and also extend the previous results to more types of geometric ranges. In particular, our results include the following[5]:

1. For unweighted arbitrary squares in 2D, We present the first dynamic data structure that can maintain an $O(1)$-approximation in sublinear update time. More precisely, we obtain $O(n^{2/3+\delta})$ update time (with Monte Carlo randomization). Previously, a dynamic geometric set cover data structure with sublinear update time was known only for unit squares by Agarwal, Chang, Suri, Xiao, and Xue [SoCG'20]. Compared to the unit square case, the arbitrary square case is more challenging. (The unit square case reduces to the case of dominance ranges, i.e., quadrants, via a standard grid approach; since the union of such ranges forms a "staircase" sequence of vertices, the problem is in some sense "1.5-dimensional". In contrast, the arbitrary square problem requires truly "2-dimensional" ideas.) See Theorem 3.1.

   Using more ideas, our update time is further improved to $O(n^{1/2+\delta})$, matching the previous update time by Agarwal et al. for unit squares. See Theorem 3.6.

2. For 3D halfspaces, we obtain a dynamic data structure with $O(n^{12/13+\delta}) \leq O(n^{0.924})$ randomized update time. Our result here is slightly weaker: it only finds the size of an $O(1)$-approximate solution (which could be good enough in some applications). If a solution itself is required, we can still get sublinear update time as long as opt is sublinear (below $n^{1-\delta}$). This assumption seems reasonable, since sublinear reporting time is not possible otherwise. (However, we currently do not know how to obtain a

[5]Throughout this chapter, all the update bounds are amortized.

Ranges	Approx.	Previous update time	New update time	Notes
Unweighted 1D intervals	$1+\varepsilon$	n^δ [10]	$\log^3 n$	Theorem 3.4 [73]
Unweighted 2D unit squares	$O(1)$	$n^{1/2+\delta}$ [10]	$2^{O(\sqrt{\log n})}$	Theorem 3.5 [73]
Unweighted 2D arbitrary squares	$O(1)$	none	$n^{2/3+\delta}$ $(\star)$ $n^{1/2+\delta}$ $(\star)$	Theorem 3.1 [72] Theorem 3.6 [73]
Unweighted 2D halfplanes	$O(1)$	none	$n^{17/23+\delta}$ $(\star)$	Theorem 3.8 [73]
Unweighted 2D disks & 3D halfspaces	$O(1)$	none	$n^{12/13+\delta}$ $(\star, \dagger)$	Theorem 3.2 [72]
Weighted 1D intervals	$3+\varepsilon$	none	$2^{O(\sqrt{\log n \log\log n})}$	Theorem 3.9 [73]
Weighted 2D unit squares	$O(1)$	none	n^δ	Theorem 3.10 [73]

Table 3.1: Summary of previous and our new data structures for approximate dynamic geometric set cover. Here, hidden constant factors in the approximation factors and update bounds may depend on ε and δ. All the updated bounds are amortized. In the entries marked $(\star)$, the results are randomized. In the entry marked $(\dagger)$, the algorithm can only return the size of the solution, not the solution itself.

stronger time bound of the form $O(n^\alpha + \mathsf{opt})$ with $\alpha < 1$ to report a solution for 3D halfspaces.) See Theorem 3.2.

The 3D halfspaces case is fundamental, as set cover for 2D disks reduces to set cover for 3D halfspaces by the standard lifting transformation [100]. Also, by duality, hitting set for 3D halfspaces is equivalent to set cover for 3D halfspaces, and hitting set for 2D disks reduces to set cover for 3D halfspaces as well.

As a byproduct, our techniques for dynamic set cover also yield an optimal randomized $O(n \log n)$-time algorithm for static set cover for 2D disks and 3D halfspaces, improving our earlier $O(n \log n(\log\log n)^{O(1)})$ result in Chapter 2. See Theorem 3.3.

3. For unweighted intervals in 1D, we obtain the first dynamic data structure with polylogarithmic update time and constant approximation factor. We achieve $1+\varepsilon$ approximation with $O(\log^3 n/\varepsilon)$ update time, which improves Agarwal et al.'s previous update bound of $O(n^\delta/\varepsilon)$. (The dynamic hitting set data structure for 1D intervals in [10] does have polylogarithmic update time but not the set cover data structure.) See Theorem 3.4.

4. For unweighted unit squares in 2D, we obtain the first dynamic data structure with $n^{o(1)}$ update time and constant approximation factor. The precise update bound is $2^{O(\sqrt{\log n})}$, which significantly improves Agarwal et al.'s previous update bound of $O(n^{1/2+\delta})$. See Theorem 3.5.

5. For unweighted halfplanes in 2D, we obtain the first dynamic data structure with sublinear update time and constant approximation factor that can efficiently report an approximate solution (in time linear in the solution size). The (randomized) update bound is $O(n^{17/23+\delta}) = o(n^{0.74})$. Although our previous solution in Sec. 3.5 can more generally handle halfspaces in 3D, it has a larger (randomized) update bound of $O(n^{12/13+\delta})$ and can only output the size of an approximate solution. (Specializing that solution to halfplanes in 2D can lower the update time a bit, but it would still be worse than the new bound.) See Theorem 3.8.

Note that although for the static problem, PTASs were known for unweighted arbitrary squares and disks in 2D [179] (and exact polynomial-time algorithms were known for halfplanes in 2D [134]), the running times of these static algorithms are superquadratic. Thus, for any of the 2D problems above, constant approximation factor is the best one could hope for under the current state of the art if the goal is sublinear update time.

A second significant contribution of this chapter is to extend the dynamic set cover data structures to *weighted* instances, thus providing the *first* nontrivial results for dynamic weighted geometric set cover. (Although there were previous results on weighted independent set for 1D intervals and other ranges by Henzinger, Neumann, and Wiese [138], and Compton et al. [94], no results on dynamic weighted geometric set cover were known even in 1D. This is in spite of the considerable work on static weighted geometric set cover [69, 117, 134, 177, 204].) In particular, we present the following results:

6. For weighted intervals in 1D, we obtain a dynamic data structure with $n^{o(1)}$ update time and constant approximation factor. The precise update bound is $2^{O(\sqrt{\log n \log\log n})}$ and the approximation factor is $3 + o(1)$. See Theorem 3.9.

7. For weighted unit squares in 2D, we also obtain a dynamic data structure with $O(n^\delta)$ update time and constant approximation factor (where the constant depends on δ and weights are assumed to be polynomially bounded integers). Even when compared to Agarwal et al.'s unweighted $O(n^{1/2+\delta})$ result [10], our result is a substantial improvement, besides being more general. See Theorem 3.10.

For the cases of (unweighted or weighted) unit squares in 2D and unweighted halfplanes in 2D, the same results hold for the *hitting set* problem—given a set of points and a set of ranges, find the smallest (or minimum weight) subset of points that hit all the given ranges—because hitting set is equivalent to set cover for these types of ranges by duality.

3.2 OVERVIEW OF OUR TECHNIQUES

We give seven different methods to achieve these results. Many of these methods require significant new ideas that go beyond minor modifications of previous techniques:

1. For unweighted 2D arbitrary squares, our algorithms are obtained by handling two cases differently: when opt is small and when opt is large. Intuitively, the small opt case is easier since we are generating fewer objects, but the large opt case also seems potentially easier since we can tolerate a larger additive error when targeting an $O(1)$-factor approximation—so, we are in a "win-win" situation. For our algorithms for 3D halfspaces, we even find it necessary to handle an intermediate case when opt is medium (achieving sublinear time for sublinear opt).

   Our algorithms for the small opt case are based on the previous static MWU algorithms [14, 46, 71] (also see Chapter 2). The adaptation of these static algorithms is not straightforward, and requires using various known techniques in new ways (($\leq k$)-levels in arrangements, for our 2D squares algorithm in Section 3.4.1, and "augmented" partition trees, for our 3D halfspaces algorithm in Section 3.5.1). The medium case for 3D halfspaces (in Section 3.5.2) is technically even more challenging (where we use "shallow" partition trees and other ideas).

   Our algorithm for squares in the large opt case (in Section 3.4.2) is different (not based on MWU), and interestingly uses quadtrees in a non-obvious way. For 3D halfspaces, our algorithm in the large opt case (in Section 3.5.3) can compute only the value of an approximate solution, and is based on random sampling. However, the obvious way to use a random sample (just solving the problem on a random subset of points and objects) does not work. We use sampling in a nontrivial way, combining geometric cuttings with planar graph separators.

2. We further improve the update time for unweighted 2D arbitrary squares. In our previous method, the small opt algorithm was obtained by modifying a known static approximation algorithm based on multiplicative weight updates [14, 46, 71, 89], and achieved $\widetilde{O}(\mathsf{opt}^2)$ update time. The large opt algorithm employed quadtrees and achieved $\widetilde{O}(n^{1/2+\delta}+n/\mathsf{opt})$ update time. Combining the two algorithms yielded $\widetilde{O}(n^{2/3})$ update time, as the critical case occurs when opt is near $n^{1/3}$. We modify the large opt algorithm by incorporating some extra technical ideas (treating so-called "light" vs. "heavy" canonical rectangles differently, and carefully tuning parameters); this allows us to improve the update time to $O(n^{1/2+\delta})$ uniformly for all opt, pushing the approach to its natural limit.

3. For unweighted 1D intervals, Agarwal et al. [10] obtained their result with $O(n^\delta)$ update time by a "bootstrapping" approach, but extra factors accumulate in each round of bootstrapping. To obtain polylogarithmic update time, we refine their approach with a better recursion, whose analysis distinguishes between "one-sided" and "two-sided" intervals.

4. For unweighted 2D unit squares, it suffices to solve the problem for quadrants (i.e., 2-sided orthogonal ranges) due to a standard reduction. We adopt an interesting geometric divide-and-conquer approach (different from more common approaches like k-d trees or segment trees). Roughly, we form an $r \times r$ nonuniform grid, where each column/row has $O(n/r)$ points, and recursively build data structures for each grid cell and for each grid column and each grid row. Agarwal et al.'s previous data structure [10] also used an $r \times r$ grid but did not use recursion per column or row; the boundary of a quadrant intersects $O(r)$ out of the r^2 grid cells and so updating a quadrant causes $O(r)$ recursive calls, eventually leading to $O(n^{1/2+\delta})$ update time. With our new ideas, updating a quadrant requires recursive calls in only $O(1)$ grid columns/rows and grid cells, leading to $n^{o(1)}$ update time.

5. For unweighted 2D halfplanes, we handle the small opt case by adapting our previous method for unweighted 3D halfspaces in Sec. 3.5, but we present a new method for the large opt case. We propose a geometric divide-and-conquer approach based on the well-known Partition Theorem of Matoušek [165]. The Partition Theorem was originally formulated for the design of range searching data structures, but its applicability to decompose geometric set cover instances is less apparent. The key to the approximation factor analysis is a simple observation that the boundary of the union of the halfplanes in the optimal solution is a convex chain with $O(\mathsf{opt})$ edges, and so in a partition of the plane into b disjoint cells, the number of intersecting pairs of edges and cells is $O(\mathsf{opt} + b)$.

For weighted dynamic geometric set cover, none of the previous approaches generalizes. Essentially all previous approaches for the unweighted setting make use of the dichotomy of small vs. large opt: in the small opt case, we can generate a solution quickly from scratch; on the other hand, in the large opt case, we can tolerate a large additive error (in particular, this enables divide-and-conquer with a large number of parts). However, all this breaks down in the weighted setting because the cardinality of the optimal solution is no longer related to its value. A different way to bound approximation factors is required.

6. For weighted 1D intervals, our key new idea is to incorporate dynamic programming

(DP) into the divide-and-conquer. In addition, we use a common trick of grouping weights by powers of a constant, so that the number of distinct weight groups is logarithmic.

7. For weighted 2D unit squares, we again use a geometric divide-and-conquer based on the $r \times r$ grid, but the recursion gets even more interesting as we incorporate DP. (We also group weights by powers of a constant.) To keep the approximation factor $O(1)$, the number of levels of recursion needs to be $O(1)$, but we can still achieve $O(n^{\delta})$ update time.

Application to static geometric set cover. Although we did not intend to revisit the static problem, our techniques for dynamic 3D halfspaces set cover in Sec. 3.5 can lead to a randomized $O(1)$-approximation algorithm for set cover for 3D halfspaces running in $O(n \log n)$ time, which completely eliminates the extra $\log \log n$ factors in the result of our previous paper [71] in Chapter 2 and is optimal (in comparison-based models)! This bonus result is interesting in its own right, and is described in Section 3.6.

Remarks. In some of our algorithms, notably the small opt algorithms for squares and 3D halfspaces (in Sections 3.4.1 and 3.5.1) and the large opt algorithm for 3D halfspaces (in Section 3.5.3), the data structure part is "minimal": we just assume that points and objects are stored separately in standard range searching data structures. We describe sublinear-time algorithms to compute a solution from scratch, using range searching as oracles. Dynamization becomes trivial, since range searching data structures typically are already known to support insertions and deletions. This type of approach also easily enables other operations, such as answering "range-restricted" queries: e.g., given a query orthogonal range, compute a set cover solution for the subset of all points inside the range (since a range searching structure for the subset can be implicitly derived from a range searching structure for the entire point set).

The topic of *sublinear-time algorithms* has received considerable attention in the algorithms community, due to applications to big data (where we want to solve problems without examining the entire input). A similar model of sublinear-time algorithms where the input is augmented with range searching data structures was proposed by Czumaj et al. [98], who presented results on approximating the weight of the Euclidean minimum spanning tree in any constant dimension under this model.

Our results in Sec. 3.4, 3.9, 3.5 and 3.10 are randomized in the Monte Carlo sense: the computed solution may not always be correct, but it is correct with high probability, i.e., probability at least $1-1/n^c$ for an arbitrarily large constant c. The error probability bounds hold even when the user has knowledge of the random choices made by the algorithms. (We do not assume an "oblivious adversary"; in fact, the algorithms make a new set of random choices each time it computes a solution, and the data structures themselves are not randomized.)

The update times are better than stated in many cases, notably, when opt is small or when opt is large. More precise opt-sensitive bounds are given in lemmas and theorems throughout the chapter. (The $O(n^{2/3+\delta})$, $O(n^{1/2+\delta})$, $O(n^{12/13+\delta})$ and $O(n^{17/23+\delta})$ bounds are obtained by "balancing" two or more of these opt-sensitive bounds.)

We have assumed that we are required to compute a solution after every update. In some (but not all) cases, the actual update cost is smaller than the cost of computing a solution (so this would give better results if we need the solution only after performing several updates).

3.3 PRELIMINARIES

3.3.1 Notations

For dynamic geometric set cover, in a *size query*, we want to output an approximation to the size opt. In a *membership query*, we want to determine whether a given object is in the approximate solution maintained by the data structure. In a *reporting query*, we want to report all elements in the approximate solution (in time sensitive to the output size). As in the previous work [10, 72], in all of our results, the set cover solution we maintain is a *multi-set* of ranges (i.e., each range may have multiple duplicates).

3.3.2 Review of an MWU algorithm

To keep this chapter self-contained, we briefly review the static approximation algorithm for geometric set cover, based on the *multiplicative weight update (MWU)* method. Some of our dynamic algorithms will be built upon this algorithm.

Specifically, we consider our randomized algorithm in Chapter 2, which is a variant of a standard algorithm by Brönnimann and Goodrich [46] or Clarkson [89] (see also Agarwal and Pan [14]). Below, c_0 is a sufficiently large constant. The *depth* of a point p in a set S of objects is the number of objects of S containing p. A subset of objects $T \subseteq S$ is called an *ε-net* of S if all points with depth $\geq \varepsilon|S|$ in S are covered by T. The size $|\hat{S}|$ of a multiset $\hat{S}$ refers to the sum of the multiplicities of its elements $\sum_{i \in S} m_i$.

```
1: Guess a value $t \in [\mathsf{opt}, 2\,\mathsf{opt}]$.
2: Define a multiset $\hat{S}$ where each object $i$ in $S$ initially has multiplicity $m_i = 1$.
3: while true do                                    ▷ call this the start of a new round
4:     Fix $\rho := \frac{c_0 t \log n}{|\hat{S}|}$ and take a random sample $R$ of $\hat{S}$ with sampling probability $\rho$.
5:     while there exists a point $p \in X$ with depth in $R$ at most $\frac{c_0}{4}\log n$ do
6:         for each object $i$ containing $p$ do     ▷ call lines 6–8 a multiplicity-doubling step
7:             Double its multiplicity $m_i$, i.e., insert $m_i$ new copies of object $i$ into $\hat{S}$.
8:             For each copy, independently decide to insert it into $R$ with probability $\rho$.
9:         if the number of multiplicity-doubling steps in this round exceeds $t$ then
10:            Go to the next iteration of the outer while loop (line 3) and start a new round.
11:    Terminate and return a $\frac{1}{8t}$-net of $R$.
```

Figure 3.1: MWU for set cover on (X, S).

The correctness of the algorithm follows from the following observations, where were known in the previous works:

Claim 3.1. Assume that the guess t is at least opt.

(a) If a point p exists in line 5, then the depth of p in $\hat{S}$ is at most $\frac{1}{2t}|\hat{S}|$ w.h.p.

(b) The algorithm terminates in $O(\log \frac{n}{t})$ rounds and $O(t \log \frac{n}{t})$ multiplicity-doubling steps w.h.p. Furthermore, $|\hat{S}| \leq n^{O(1)}$ at all times w.h.p.

(c) Any $\frac{1}{8t}$-net of R in line 11 is a set cover of X w.h.p.

Proof.

(a) The depth of p in R is at most $\frac{c_0}{4}\log n \leq \frac{1}{4t}\rho|\hat{S}|$ (note that $|\hat{S}|$ can only increase during a round). It follows that the depth of p in $\hat{S}$ is at most $\frac{1}{2t}|\hat{S}|$ w.h.p. by a Chernoff bound, as shown in [71, Section 4.2].

(b) This part follows from standard analysis of the MWU method [14, 46, 71], which we include for the sake of completeness: Each multiplicity-doubling step increases $|\hat{S}|$ by a factor of at most $1+\frac{1}{2t}$, since line 5 guarantees the depth of p in $\hat{S}$ is at most $\frac{1}{2t}|\hat{S}|$ w.h.p. by (a). Thus, after z multiplicity-doubling steps, we have $|\hat{S}| \leq n(1+\frac{1}{2t})^z \leq ne^{z/(2t)}$. On the other hand, consider a set cover T^* of size $t \geq \mathsf{opt}$. In each multiplicity-doubling step, at least one of the objects in T^* that contain p has its multiplicity doubled. So, after z multiplicity-doubling steps, the total multiplicity in T^* is at least $t2^{z/t}$. We conclude that $t2^{z/t} \leq |\hat{S}| \leq ne^{z/(2t)}$, implying that $z = O(t \log \frac{n}{t})$. Furthermore, $|\hat{S}|$ increases by at most a factor of $(1 + \frac{1}{2t})^t \leq 2$ in each round w.h.p.; in particular, $|\hat{S}|$ is bounded by $n^{O(1)}$ at the end of $O(\log \frac{n}{t})$ rounds.

(c) When the algorithm reaches line 11, the depth in R of all points of X is at least $\frac{c_0}{4} \log n \geq \frac{|R|}{8t}$ w.h.p. (by a Chernoff bound, since the expected size of R is $c_0 t \log n$), and so X will be covered by a $\frac{1}{8t}$-net of R.

QED.

For objects that are axis-aligned squares in 2D, disks in 2D, or halfspaces in 3D, ε-nets of size $O(\frac{1}{\varepsilon})$ exist, and thus the above algorithm yields a set cover of size $O(t)$, i.e., a constant-factor approximation, assuming that the guess t is indeed between opt and $2\,\mathsf{opt}$. By known algorithms (e.g., [78]), the net for R in line 11 can be constructed in $\widetilde{O}(|R|) = \widetilde{O}(t)$ time.

Remarks. In the original version of MWU, the "lightness" condition in line 5 was whether the depth of p in $\hat{S}$ is at most $O(\frac{1}{t}|\hat{S}|)$. In the above randomized version, lightness is tested with respect to the sample R instead, which is computationally easier to work with. Also, in the original version, in the end the algorithm returns a $\Theta(\frac{1}{t})$-net of the multiset $\hat{S}$. In the above version, a net of R is returned instead, which is again easier to compute.

Several modifications to the MWU method have been explored in previous work. For example, in the first algorithm of Agarwal and Pan [14], each round examines all points of X in a fixed order and test for lightness of the points one by one (based on the observation that a point found to have large depth will still have large depth by the end of the round). In the previous chapter, we have also added a step at the beginning of each round, where the multiplicities are rescaled and rounded, so as to keep $|\hat{S}|$ bounded by $O(n)$. These modifications led to a number of different static implementations running in $\widetilde{O}(n)$ time.

3.4 UNWEIGHTED SQUARES

Our first result for dynamic set cover is for axis-aligned squares. Previously, a sublinear time algorithm for dynamic set cover was known only for unit squares [10]. Our method will be divided into two cases: when opt is small and when opt is large. Let t be a guess on opt. We will maintain our data structure for each possible $t = 2^i$ in parallel. When the guess is wrong, our algorithm will be able to tell whether t is approximately smaller or larger than opt w.h.p. In the end we take the one with minimum size among all returned solutions. The update time will increase only by a factor of $O(\log n)$.

3.4.1 Algorithm for small opt

Our algorithm for the small opt case will be based on the randomized MWU algorithm described in Alg. 3.1 of Section 3.3.2. The key is to realize that this algorithm can actually be implemented to run in *sublinear* time, assuming that the points and the objects have been preprocessed in standard range searching data structures. Since these structures are dynamizable, we can just re-run the MWU algorithm from scratch after every update.

Data structures. Our data structures are simple. We store the point set X in the standard 2D *range tree* [100]. For each square s with center (x, y) and side length $2z$, map s to a point $s^{\uparrow} = (x, y, x-y, x+y, x-z, x+z, y-z, y+z)$ in 8D. We also store the lifted point set $S^{\uparrow} = \{s^{\uparrow} : s \in S\}$ in an 8D range tree [12, 100]. Range trees support insertions and deletions in X and S in polylogarithmic (amortized) time.

Computing a solution. We now show how to compute an $O(1)$-approximate solution in sublinear time when opt is small by running Algorithm 3.1 using the above data structures. At first glance, linear time seems unavoidable: (i) the obvious way to find low-depth points in line 5 is to scan through all points of X in each round (as was done in previous algorithms [14, 71]), and (ii) explicitly maintaining the multiplicities of all points would also require linear time.

To overcome these obstacles, we observe that (i) we can use data structures to find the next low-depth point p without testing each point one by one (recall that there are only $\widetilde{O}(t)$ multiplicity-doubling steps), and (ii) multiplicities do not need to be maintained explicitly, so long as in line 8 we can generate a multiplicity-weighted random sample among the objects containing a given point p efficiently (recall that the sample R has only $\widetilde{O}(t)$ size). The subproblem in (ii) is a *weighted range sampling* problem, which is nontrivial when weights are implicitly represented.

Finding a low-depth point. Let $b := \frac{c_0}{2} \log n$. Each time we want to find a low-depth point in line 5, we compute (from scratch) $\mathcal{L}_{\leq b}(R)$, the $(\leq b)$-level of R, i.e., the collection of all cells in the arrangement of the squares of depth at most b. It is known [192] that $\mathcal{L}_{\leq b}(R)$ has $O(|R|b)$ combinatorial complexity and can be constructed in $\widetilde{O}(|R|b)$ time, which is $\widetilde{O}(t)$ (since $|R| = \widetilde{O}(t)$ and $b = \widetilde{O}(1)$). Each cell γ in $\mathcal{L}_{\leq b}(R)$ is a rectilinear polygon, and can be decomposed into $O(|\gamma|)$ rectangular subcells. The total number of subcells remains $\widetilde{O}(t)$. To find a point p of X that has depth in R at most b, we simply examine each rectangular subcell of $\mathcal{L}_{\le b}(R)$, and perform an orthogonal range query to test if the subcell contains a point of X. All this takes $\widetilde{O}(t)$ time.

As there are $\widetilde{O}(t)$ multiplicity-doubling steps, the total cost is $\widetilde{O}(t^2)$.

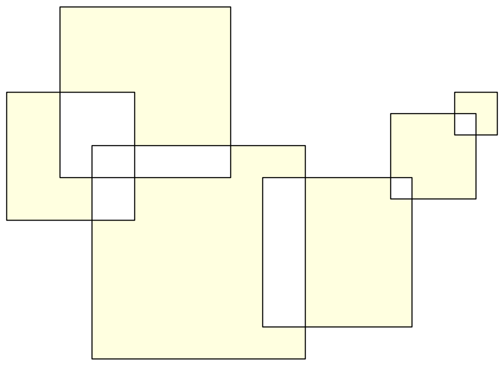


Figure 3.2: The 1-level (all cells of depth exactly 1) in an arrangement of squares.

Weighted range sampling. For each square s with center (x, y) and side length $2z$, define its *dual* point s^* to be (x, y, z) in 3D. For each point $p = (p_x, p_y)$, define its *dual* region p^* to be $\{(x, y, z) : z \ge \max\{|x - p_x|, |y - p_y|\}\}$ in 3D. Then a point p is in the square s iff the point s^* is in the region p^*.

Let Q be the set of all points p for which we have performed multiplicity-doubling steps thus far. Note that $|Q| = \widetilde{O}(t)$, because there are only at most $O(t \log \frac{n}{t})$ multiplicity-doubling steps. Let $b' = c_0' \log n$ for a sufficiently large constant c_0'. Each time we perform a multiplicity-doubling step, we compute (from scratch) $\mathcal{L}_{\le b'}(Q^*)$, the $(\le b')$-level of the dual regions $Q^* = \{p^* : p \in Q\}$. This structure corresponds to planar order-$(\le b')$ L_∞ Voronoi diagrams. By known results [57, 162, 192], $\mathcal{L}_{\le b'}(Q^*)$ has $O(|Q|(b')^2)$ combinatorial complexity and can be constructed in $\widetilde{O}(|Q|(b')^2)$ time, which is $\widetilde{O}(t)$ (since $|Q| = \widetilde{O}(t)$ and $b' = \widetilde{O}(1)$). Each cell γ of $\mathcal{L}_{\le b'}(Q^*)$ is a polyhedron, which is a region sandwiched between a lower envelope γ^+ and an upper envelope γ^- of surfaces of the form $z = \max\{|x - c|, |y - c'|\}$; note that the upper envelope γ^- has constant complexity. By taking the vertical decomposition of γ^+, we can divide each cell γ into $O(|\gamma|)$ subcells, where each subcell has constant complexity, and its faces are contained in planes of only 8 possible orientations, namely, planes of the form $x = c$, $y = c$, $y = \pm x + c$, $z = \pm x + c$, or $z = \pm y + c$. The total number of subcells remains $\widetilde{O}(t)$.

The multiplicity of a square $s \in S$ is equal to $2^{\text{depth of } s^* \text{ in } Q^*}$ (since each p^* in Q^* containing s^* causes the multiplicity of s to double). In particular, since multiplicities are bounded by

$n^{O(1)}$, the depth (i.e., level) of s^* must be logarithmically bounded. So, each s^* is covered by $\mathcal{L}_{\le b'}(Q^*)$, and the multiplicity of s is determined by the cell of $\mathcal{L}_{\le b'}(Q^*)$ containing the point s^* (since points in the same cell have the same depth).

To generate a multiplicity-weighted sample of the squares containing p for line 8, after p has been inserted to Q, we examine all subcells of $\mathcal{L}_{\le b'}(Q^*)$ that are contained in p^*. These subcells contain the dual points of all squares containing p. For each such subcell γ, we identify the squares $s \in S$ for which $s^* \in \gamma$. This reduces to an orthogonal range query in the 8D point set $S^\uparrow$, and the answer can be expressed as a disjoint union of $\widetilde{O}(1)$ canonical subsets in the range tree [12, 100]. Knowing the sizes and multiplicities of these canonical subsets for all such $\widetilde{O}(t)$ cells, we can then generate the weighted sample in time $\widetilde{O}(t)$ plus the size of the sample: define the weight of a canonical subset to be the total weight of all dual points contained in it. To sample a square containing p, just first sample a canonical subset according to the weights, then sample a dual point within the canonical subset in $O(1)$ time (since in the range tree, we keep a list of the dual points contained in each canonical subset).

Hence, the total cost of all $\widetilde{O}(t)$ multiplicity-doubling steps is $\widetilde{O}(t^2)$.

In addition, we need to generate a new weighted sample R (with a new sampling probability ρ) in line 4 at the beginning of each round, which can be done similarly as above: the multiplicity of each square s in S is determined by the subcell of $\mathcal{L}_{\le b'}(Q^*)$ containing it, and each subcell of $\mathcal{L}_{\le b'}(Q^*)$ maps to $\widetilde{O}(1)$ canonical subsets, for a total of $\widetilde{O}(t)$ canonical subsets. To sample a square in R, first sample a canonical subset according to the weights and then sample a dual point within the canonical subset. This takes $\widetilde{O}(t)$ time plus the size of the sample R (which is $\widetilde{O}(t)$), for each of the $O(\log n)$ rounds. As mentioned, the final net computation in line 11 takes $\widetilde{O}(t)$ time. We have thus obtained:

Lemma 3.1. There exists a data structure for the dynamic set cover problem for a set X of points and a set S of axis-aligned squares in $\mathbb{R}^2$ (where $|X| + |S| = O(n)$), that supports insertions and deletions in $\widetilde{O}(1)$ time and can find an $O(1)$-approximate solution w.h.p. to the set cover problem in $\widetilde{O}(\mathsf{opt}^2)$ amortized time.

3.4.2 Algorithm for large opt

To complement our solution for the small opt case, we now show that the problem also gets easier when opt is large, mainly because we can afford a large additive error. We describe a different, self-contained algorithm for this case (not based on modifying MWU), interestingly by using quadtrees in a novel way.

To allow for both multiplicative and additive error, we use the term (α, β)-*approximation* to refer to a solution with cost at most $\alpha\,\mathsf{opt} + \beta$.

For simplicity, we assume that all coordinates are integers bounded by $U = \text{poly}(n)$. At the end, we will comment on how to remove this assumption.

In the standard *quadtree*, we start with a bounding square cell and recursively divide a square cell into four square subcells. We define the size of a cell Γ to be the number of vertices in Γ among the squares of S, plus the number of points of X in Γ. We stop subdividing when a leaf cell has size at most b, where b is a parameter to be set later. This yields a subdivision into $O(\frac{n}{b})$ cells per level, and $O(\frac{n}{b}\log U)$ cells in total. An example is shown in Fig. 3.3. The quadtree decomposition can be easily made dynamic under insertions and deletions of points and squares.

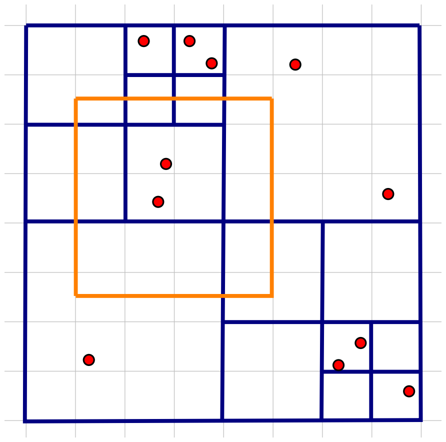


Figure 3.3: Example of a quadtree decomposition.

For each leaf cell Γ, a square in S intersecting Γ is called *short* if at least one of its vertices is in the cell, and *long* otherwise, as shown in Fig. 3.4. Note that a long square can have at most one side crossing the cell, because the quadtree cell Γ is also a square. The union of the long squares within the cell is defined by at most 4 long squares—call these the *maximal* long squares. (If there is a square containing Γ, we can designate one such square as the maximal long square.) For each leaf cell Γ, it suffices to approximate the optimal set cover for the input points in $X \cap \Gamma$ using only the short squares plus the at most 4 maximal long squares in Γ. By charging each square in the optimal solution to the cells containing its 4 vertices (i.e., the square is short in these cells), we see that the sum of the sizes of the optimal covers in the leaf cells is at most $4\,\mathsf{opt} + O(\frac{n}{b}\log U)$: let opt_Γ denote the size of the optimal solution in cell Γ, and opt'_Γ denote the size of the optimal solution that uses only short squares to cover the complement of the union of the maximal long squares in Γ; then we have

$$\begin{aligned}\sum_{\Gamma} \mathsf{opt}_\Gamma \;\le\; \sum_{\Gamma}(\mathsf{opt}'_\Gamma + 4) \;&\le\; \sum_{\Gamma} \mathsf{opt}'_\Gamma + 4\cdot(\#\text{ of leaf cells}) \\ &\le\; 4\cdot \mathsf{opt} + O(\tfrac{n}{b}\log U),\end{aligned} \tag{3.1}$$

which is indeed an $O(1)$-approximation if we choose $b \ge \frac{n\log n}{\mathsf{opt}}$.

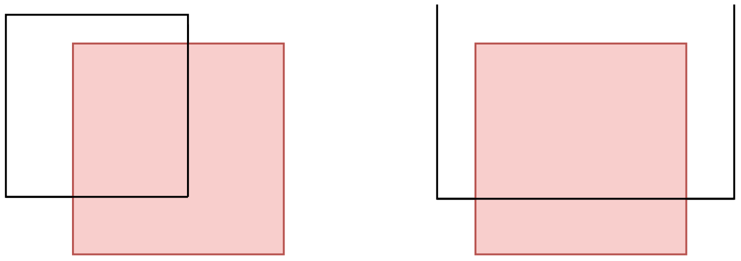


Figure 3.4: Short square (left) and long square (right). The quadtree cell is shaded.

Note that the complement of the union of the at most 4 maximal long squares in a cell Γ is a rectangle r_Γ. We will store the short squares in the cell Γ in a data structure $\mathcal{S}_\Gamma$ to answer the following type of query:

> Given any query rectangle r, compute an $O(1)$-approximation to the optimal set cover for the points in $X \cap r$ using only the short squares.

Assuming the availability of such data structures $\mathcal{S}_\Gamma$, we can solve the dynamic set cover problem as follows:

- An insertion/deletion of a square s in S requires updating 4 of these data structures $\mathcal{S}_\Gamma$ for the leaf cells Γ containing the 4 vertices of s, and also updating the maximal long squares for all leaf cells in $\widetilde{O}(\frac{n}{b})$ time.
- An insertion/deletion of a point p in X requires updating one data structure $\mathcal{S}_\Gamma$ for the leaf cell Γ containing p.
- Whenever we want to compute a set cover solution, we examine all $\widetilde{O}(\frac{n}{b})$ cells Γ and query the data structure $\mathcal{S}_\Gamma$ for r_Γ, and return the union of the answers.

First implementation of $\mathcal{S}_\Gamma$. A simple way to implement the data structure $\mathcal{S}_\Gamma$ is as follows: in an update, we just recompute an approximate solution from scratch for every possible query rectangle in Γ. Since there are only $O(b^4)$ combinatorially different query rectangles, and static approximate set cover on $O(b)$ squares and points takes $\widetilde{O}(b)$ time [14, 71], the update time is $\widetilde{O}(b^5)$, and the query time is trivially $O(1)$.

As a result, the cost of insertion/deletion in the overall method is $\widetilde{O}(b^5 + \frac{n}{b})$.

Improved implementation for $\mathcal{S}_\Gamma$. We further improve the update time for the data structure $\mathcal{S}_\Gamma$. Instead of recomputing solutions for all $O(b^4)$ rectangles, the idea is to recompute solutions for a smaller number of "canonical rectangles". More precisely, by using a 2D range tree [12, 100] for the $O(b)$ points in $X \cap \Gamma$, with branching factor a in each dimension, we can form a set of canonical rectangles with total size $O(a^{O(1)}b(\log_a b)^2)$, such that every query rectangle can be decomposed into $O((\log_a b)^2)$ canonical rectangles, ignoring portions that are empty of points. We set $a := b^\delta$ for an arbitrarily small constant $\delta > 0$.

For any rectangle r, call a square s intersecting it *short* if at least one of its vertices is in r, and *long* otherwise (similar to before). We prove that for each canonical rectangle r with size b_i, to find an optimal set cover for r, it suffices to consider only $O(b_i)$ candidate long squares among all long squares with respect to r: first consider the long squares that cut across r vertically. When we vertically project the points in r onto the x-axis, such long square covers a 1D interval on the x-axis (notice that now r is a rectangle rather than a square, so a long square may possibly have two edges that cut across r). So we can view the problem as a 1D interval set cover instance. Sort the projected points $p_1, \ldots, p_k$ ($k \leq b_i$) in increasing order. If in the optimal solution we pick a long square that projects to an interval $[l_0, r_0]$ where $l_0 \in [p_i, p_{i+1})$, then we can instead greedily pick the projected interval with left endpoint in $[p_i, p_{i+1})$ and has the largest right endpoint. Thus we only need to consider $O(b_i)$ such "maximal" intervals as candidates. A similar statement holds for the long squares that cut across r horizontally.

These "maximal" long squares can be found in $\widetilde{O}(b_i)$ time by standard orthogonal range searching. We can thus approximate the optimal set cover for the points in the canonical rectangle r, using the $O(b_i)$ short squares and maximal long squares with respect to r, in $\widetilde{O}(b_i)$ time by known static set cover algorithms [14, 71]. The total time over all canonical rectangles is $\widetilde{O}(a^{O(1)}b(\log_a b)^2) = \widetilde{O}(b^{1+O(\delta)})$.

Given a query rectangle, we can decompose it into $O((\log_a b)^2) = O(1)$ canonical rectangles and return the union of the optimal solutions in the canonical rectangles, which is an $O(1)$-approximation (more precisely, an $O(\delta^{-2})$-approximation).

As a result, the cost of insertion/deletion in the overall method is $\widetilde{O}(b^{1+O(\delta)} + \frac{n}{b})$.

Removing the dependency on U. When U may be large, we can reduce the tree depth from $O(\log U)$ to $O(\log n)$ by replacing the quadtree with the *BBD tree* of Arya et al. [27]. Each cell in the BBD tree is the set difference of two quadtree squares, one contained in the other. The size of each child cell is at most a fraction of the size of the parent cell. As before, we stop subdividing when a leaf cell has size at most b. Since any such leaf cell has size $\Theta(b)$ now, the number of leaf cells is $O(\frac{n}{b})$. The BBD tree can be maintained dynamically in polylogarithmic amortized time (for example, by periodically rebuilding when subtrees become unbalanced). Since a leaf cell Γ is the difference of two quadtree squares, it is not difficult to see that the number of maximal long squares in Γ remains $O(1)$. So, our previous analysis remains valid.

Lemma 3.2. Given a parameter b, there exists a data structure for the dynamic set cover problem for a set X of points and a set S of axis-aligned squares in $\mathbb{R}^2$ (where $|X| + |S| = O(n)$), that maintains an $(O(1), O(\frac{n}{b}))$-approximate solution with $\widetilde{O}(b^{1+O(\delta)} + \frac{n}{b})$ amortized insertion and deletion time.

Combining the algorithms. When $\mathsf{opt} \le n^{1/3}$, we use the algorithm for small opt; the running time is $\widetilde{O}(\mathsf{opt}^2) \le \widetilde{O}(n^{2/3})$. When $\mathsf{opt} > n^{1/3}$, we use the algorithm for large opt with $b = n^{2/3}$, so that an $(O(1), O(\frac{n}{b}))$-approximation is indeed an $O(1)$-approximation; the running time is $\widetilde{O}(b^{1+O(\delta)} + \frac{n}{b}) = O(n^{2/3+O(\delta)})$.

Reporting the solution. For the small opt algorithm, the final ε-net after MWU already gives the actual solution. For the large opt algorithm, the actual solution can be reported by taking the union of all solutions in the leaf cells of the quadtree (each of them consists of at most 4 maximal long squares and the solution in the complement region). So a reporting query takes $\widetilde{O}(\mathsf{opt})$ time.

Theorem 3.1. There exists a data structure for the dynamic set cover problem for a set X of points and a set S of axis-aligned squares in $\mathbb{R}^2$ (where $|X| + |S| = O(n)$), that maintains an $O(1)$-approximate solution w.h.p. with $O(n^{2/3+\delta})$ amortized insertion and deletion time for any constant $\delta > 0$. The actual solution can be reported in $\widetilde{O}(\mathsf{opt})$ time.

The case of fat rectangles (where the ratio of the side lengths is bounded by a constant) can be reduced to squares, since such rectangles can be replaced by $O(1)$ squares (increasing the approximation factor by only $O(1)$). Our approach can be modified to work more generally for homothets of a fixed fat convex polygon with a constant number of vertices.

3.5 UNWEIGHTED 3D HALFSPACES

In this section, we study dynamic geometric set cover for the more challenging case of 3D halfspaces. Using the standard lifting transformation [100], we can transform 2D disks to 3D upper halfspaces. For simplicity, we assume that all halfspaces are upper halfspaces; Section 3.5.4 discusses how to modify our algorithms when there are both upper and lower halfspaces. Our method will be divided into three cases: small, medium, and large opt.

3.5.1 Algorithm for small opt

Similar to the small opt algorithm for squares in Section 3.4.1, we describe a small opt algorithm for halfspaces based on the randomized MWU algorithm in Section 3.3.2. Although our earlier approach using levels in arrangements could be generalized, we describe a better approach based on augmenting partition trees with counters.

Data structures. We store the 3D point set X in Matoušek's *partition tree* [165]: The tree has height $O(\log n)$ and degree r for a sufficiently large constant r. Each node v stores a simplicial cell Γ_v and a "canonical subset" $X_v \subset \Gamma_v$, where $X_v = X$ at the root v, and Γ_v is contained in $\Gamma_{\text{parent}(v)}$, X_v is the disjoint union of $X_{v'}$ over all children v' or v, and X_v has constant size at each leaf v. Furthermore, any halfspace crosses $O(n^{2/3+\delta})$ cells of the tree for any arbitrarily small constant $\delta > 0$ (depending on r). Here a halfspace h *crosses* a cell Γ iff the boundary of h intersects Γ.

For each upper halfspace h, let h^* denote its dual point; for each point p, let p^* denote its dual upper halfspace. (The dual point of a halfspace $z \geq ax + by - c$ is defined as the point (a, b, c), and the dual upper halfspace of a point is defined vice versa, so that p is in h iff h^* is in p^* [100].)

We also store the 3D dual point set $S^* = \{h^* : h \in S\}$ in Matoušek's partition tree. Each node v stores a cell Γ_v and a canonical subset $S^*_v \subset \Gamma_v$ like above.

Matoušek's partition trees can be built in $\widetilde{O}(n)$ time and support insertions and deletions in X and S in polylogarithmic time. (In the static case, there are slightly improved partition trees reducing the n^δ factor in the crossing number bound [63, 165, 167], but these will not be important to us.)

Computing a solution. We now show how to compute an $O(1)$-approximate solution in sublinear time when opt is small by running Algorithm 3.1 using the above data structures. As in Section 3.4.1, the main subproblems are (i) finding a low-depth point with respect to R, and (ii) weighted range sampling, where the weights are the multiplicities (which are not explicitly stored).

Finding a low-depth point. We maintain two values at each node v of the partition tree of X:

- c_v is the number of halfspaces of R containing Γ_v but not containing $\Gamma_{\text{parent}(v)}$.
- d_v is the minimum depth among all points in X_v with respect to the halfspaces of R crossing Γ_v.

The overall minimum depth with respect to R is given by the value d_v at the root v. Whenever we insert a halfspace h to R, for each of the $O(n^{2/3+\delta})$ cells Γ_v crossed by h, we update the counters $c_{v'}$ for the children v' of the node v; afterwards, we update the value d_v *bottom-up* according to the formula $d_v = \min_{\text{child } v' \text{ of } v}(d_{v'} + c_{v'})$, i.e., we first recursively update $d_{v'}$ for all affected children v', and then update d_v using the formula. Thus, all values can be maintained in $O(n^{2/3+\delta})$ time per insertion to R.

As there are $\widetilde{O}(t)$ insertions to R, the total cost is $\widetilde{O}(tn^{2/3+\delta})$. This cost covers the resetting of counters at every round.

(We remark that the idea of augmenting nodes of partition trees with counters appeared before in at least one prior work on dynamic geometric data structures by Chan [60, Theorem 4.1].)

Weighted range sampling. Let Q be the set of all points p for which we have performed multiplicity-doubling steps thus far. Note that $|Q| = \widetilde{O}(t)$. The multiplicity of a halfspace $h \in S$ is $2^{\text{depth of } h^* \text{ in } Q^*}$. To implicitly represent the multiplicities and their sum, we maintain two values at each node v of the partition tree for S^*:

- c_v is the number of dual halfspaces of Q^* containing Γ_v but not containing $\Gamma_{\text{parent}(v)}$.
- m_v is the sum of $2^{\text{depth of } h^* \text{ among the halfspaces of } Q^* \text{ crossing } \Gamma_v}$ over all $h^* \in S^*_v$.

Whenever we insert a point p to Q, for each of the $O(n^{2/3+\delta})$ cells Γ_v crossed by the dual halfspace p^*, we update the counters $c_{v'}$ for the children v' of the nodes v; we also update the value m_v bottom-up according to the formula $m_v = \sum_{\text{child } v' \text{ of } v} 2^{c_{v'}} m_{v'}$. Thus, all values can be maintained in $O(n^{2/3+\delta})$ time per insertion to Q.

To generate a weighted sample of the halfspaces of S containing p for line 8, we find all $O(n^{2/3+\delta})$ cells Γ_v crossed by the dual halfspace p^*, and consider the canonical subsets $S^*_{v'}$ for the children v' of v with $\Gamma_{v'}$ contained in p^*. We can then sample from these canonical subsets, weighted by $m_{v'} 2^{\sum_u c_u}$, where the sum is over all ancestors u of v'. All this takes time $O(n^{2/3+\delta})$ plus the size of the sample.

Hence, the total cost of all $\widetilde{O}(t)$ multiplicity-doubling steps is $\widetilde{O}(tn^{2/3+\delta})$.

Lemma 3.3. There exists a data structure for the dynamic set cover problem for a set X of points and a set S of upper halfspaces in $\mathbb{R}^3$ (where $|X| + |S| = O(n)$), that supports insertions and deletions in $\widetilde{O}(1)$ time and can find an $O(1)$-approximate solution w.h.p. in $\widetilde{O}(\mathsf{opt} \cdot n^{2/3+\delta})$ time.

3.5.2 Algorithm for medium opt

The preceding algorithm works well only when opt is smaller than about $n^{1/3}$. We show that a more involved algorithm, also based on MWU, can achieve sublinear time even when opt approaches $n^{1-\delta}$. The basic approach is to use *shallow* versions of the partition trees [166].

Data structures. We begin with a lemma that was used before in some of the previous static algorithms by Agarwal and Pan [14] and ours [71]. With this lemma, we can effectively make every input halfspace shallow, i.e., contain at most $\widetilde{O}(\frac{n}{t})$ points. The extra condition in the second sentence of the lemma below is new, and is needed in order to make dynamization possible later (difficulty arises when halfspaces in T_0 get deleted). Because of this extra condition, we describe a construction which is different from the previous algorithms [14, 71].

Lemma 3.4. Given a set X_0 of n points, a set S_0 of n halfspaces in 3D and a parameter t, we can construct a subset of halfspaces $T_0 \subseteq S_0$ of size $O(t)$, and a subset of points $A_0 \subseteq X_0$, such that (i) each point in A_0 is covered by T_0, and (ii) each halfspace of $S_0 - T_0$ contains $\widetilde{O}(\frac{n}{t})$ points of $X_0 - A_0$.

Furthermore, we can decompose $A_0 = \bigcup_{h \in T_0} A_h$, where A_h has size $O(\frac{n}{t})$ for each $h \in T_0$, such that each point in A_h is covered by h. The construction takes $\widetilde{O}(n)$ time.

Proof. We use a simple "greedy" approach: We examine each halfspace $h \in S$ in an arbitrary order, and test whether h contains more than $\frac{n}{t}$ points of X_0. If so, we add h to T_0, pick some $\frac{n}{t}$ (but not more) points in $X_0 \cap h$ to add to A_h, and delete A_h from X_0. Clearly, the number of halfspaces added to T_0 is $O(t)$.

To bound the construction time, we maintain X_0 in Chan's dynamic 3D halfspace range reporting structure with polylogarithmic amortized update time [62]. The cost of all deletions is $\widetilde{O}(n)$. Testing each halfspace h requires a halfspace range counting query, but for the above purposes, it suffices to use an approximate count, with $\widetilde{O}(1)$ approximation factor. Given a query halfspace h containing k points, in $\widetilde{O}(1)$ time, one can find $O(\log n)$ lists of size $O(k)$ from Chan's data structure [62], so that the k points inside h are contained in the union of these lists (this consequence of Chan's data structure was explicitly stated in Theorem 3.1 of [64] in the 2D case, and Chan's data structure [62] works in 3D). The sum of the sizes of these lists is $O(k \log n)$ and $\Omega(k)$, yielding an $O(\log n)$-approximation of the count. QED.

We divide the update sequence into *phases* of g updates each, for a parameter $g \ll t$. Our data structure will be rebuilt periodically, after each phase. Let X_0 and S_0 be X and S at the beginning of the current phase. Let X_I and S_I be the current set of points and the set of halfspaces that have been inserted to X and S in the current phase. Let X_D and S_D be the current set of points and the set of halfspaces deleted from X and S in the current phase. At the start of each phase, we apply Lemma 3.4.

We store the 3D point set $X_0 - A_0$ in a partition tree, like before. However, we will need a bound on the crossing number that is sensitive to shallowness: namely,

- any halfspace containing $\widetilde{O}(\frac{n}{t})$ points of $X_0 - A_0$ crosses at most $O((\frac{n}{t})^{2/3+O(\delta)})$ cells of the tree;
- any other halfspace crosses at most $O(t \cdot (\frac{n}{t})^{2/3+\delta})$ cells.

This follows by combining Matoušek's *shallow* version of the partition tree [166] with the original version of the partition tree: using the shallow version of the partition tree, with $O(t)$ leaf cells containing $O(\frac{n}{t})$ points each, any halfspace that contains $\widetilde{O}(\frac{n}{t})$ points is known to cross at most $O(n^{\delta})$ leaf cells [166]. For each leaf cell with $O(\frac{n}{t})$ points, we build the original partition tree [165], which has crossing number bound $O((\frac{n}{t})^{2/3+\delta})$.

In addition, we maintain the following counter at each node v of the partition tree:

- $c_v^{\#}$ is the number of halfspaces of $T_0 \cup S_I - S_D$ containing Γ_v but not containing $\Gamma_{\text{parent}(v)}$.

In the beginning of a phase, we can compute each count $c_v^{\#}$ by $O(1)$ 3D simplex range counting query on the dual points of T_0^* (since halfspaces containing a simplex and not containing another simplex dualize to points in a polyhedral region of constant size); by known results [12, 165], $O(n)$ such queries on $O(t)$ points in 3D take $\widetilde{O}(n + (nt)^{3/4})$ time. The amortized cost is $\widetilde{O}(\frac{n+(nt)^{3/4}}{g})$.

Afterwards, during each insertion/deletion of a halfspace in S, we increment/decrement the counters at $O(t \cdot (\frac{n}{t})^{2/3+\delta})$ nodes of the tree (note that the halfspace may not be shallow). Thus, each update in S costs $O(t^{1/3} n^{2/3+\delta})$ time.

We also store the 3D dual point set $(S_0 - T_0 - S_D)^*$ in a partition tree, again with $O((\frac{n}{t})^{2/3+O(\delta)})$ crossing number bound with respect to halfspaces containing $\widetilde{O}(\frac{n}{t})$ points. Such a partition tree supports deletions in polylogarithmic time.

Computing a solution. We now show how to compute an $O(1)$-approximate solution in sublinear time when opt is sublinear using the above data structures. We first take a new unweighted random sample $R_{\text{extra}} \subset S_0 - T_0 - S_D$ of size t. We include $R_{\text{extra}} \cup T_0 \cup S_I - S_D$ in the solution. Since this set has size $O(t+g) = O(t)$, this increases the approximation factor only by $O(1)$. Let $E = \bigcup_{h \in T_0 \cap S_D} A_h \cup X_I - X_D$, which has $O(\frac{gn}{t})$ points. It remains to cover the points in $(X_0 - A_0 - X_D) \cup E$, excluding those already covered by $R_{\text{extra}} \cup T_0 \cup S_I - S_D$, using halfspaces in $S_0 - T_0 - S_D$. We will do so by running Algorithm 3.1 on these points and halfspaces using the above data structures. As before, the main subproblems are (i) finding a low-depth point, and (ii) weighted range sampling.

Finding a low-depth point. We can find the minimum depth of the points of $X_0 - A_0 - X_D$ (with respect to R) as in the small opt algorithm, by using counters in the partition tree for $X_0 - A_0$.

Since any halfspace in $S_0 - T_0 - S_D$ contains at most $\widetilde{O}(\frac{n}{t})$ points of $X_0 - A_0 - X_D$ by Lemma 3.4, the cost per insertion to R is reduced from $O(n^{2/3+\delta})$ to $O((\frac{n}{t})^{2/3+O(\delta)})$. The total cost over all $\widetilde{O}(t)$ insertions to R is $O(t(\frac{n}{t})^{2/3+O(\delta)}) = O(t^{1/3}n^{2/3+O(\delta)})$.

Because we have initially included $R_{\text{extra}} \cup T_0 \cup S_I - S_D$ in the solution, we should be excluding points already covered by $R_{\text{extra}} \cup T_0 \cup S_I - S_D$ when running Algorithm 3.1. To fix this, we add $c_0 \log n$ copies of $R_{\text{extra}} \cup T_0 \cup S_I - S_D$ to R at the beginning of each round. This way, points covered by $R_{\text{extra}} \cup T_0 \cup S_I - S_D$ would not be picked as low-depth points. Adding these copies of $T_0 \cup S_I - S_D$ requires no extra effort, since we can initialize c_v with the already computed $c_v^{\#}$ value, times $c_0 \log n$, for all nodes v encountered. The copies of the $\widetilde{O}(t)$ halfspaces of R_{extra} can be inserted one by one, in $\widetilde{O}((\frac{n}{t})^{2/3+O(\delta)})$ time each, as above. The extra cost for these $\widetilde{O}(t)$ insertions to R is bounded as above.

Also, because points in X_D are supposed to be deleted, we should be excluding the deleted points in X_D when defining the minimum depth values d_v. When we delete a point from X, we can update the d_v values along a path bottom-up in $\widetilde{O}(1)$ time.

We can find low-depth points of E (with respect to R) more naively, by examining the points of E, excluding those covered by $R_{\text{extra}} \cup T_0 \cup S_I - S_D$, and testing them one by one in each round (like in Agarwal and Pan's algorithm [14]). In line 5, it suffices to use an $O(1)$-approximation to the depth (after adjusting constants in the pseudocode), and as noted in our previous paper [71], we can apply known data structures for 3D halfspace approximate range counting [6] for the dual points R^*; queries and insertions to R take polylogarithmic time. A point that has been found to have depth larger than the threshold will remain having large depth during a round. The total cost over all rounds is $\widetilde{O}(|E|) = \widetilde{O}(\frac{gn}{t})$.

Weighted range sampling. During a multiplicity-doubling step for a point p, we can generate a multiplicity-weighted sample from the halfspaces containing p in the same way as in the small opt algorithm, by using counters in the partition tree for $(S_0 - T_0 - S_D)^*$. Observe that p has depth in $S_0 - T_0 - S_D$ at most $O(\frac{n}{t} \log n)$ w.h.p., because otherwise, p would be covered by the random sample R_{extra} and would have been excluded (in other words, a random sample R_{extra} of size t is a $\Theta(\frac{\log n}{t})$-net w.h.p.). Since the dual halfspace p^* contains at most $\widetilde{O}(\frac{n}{t})$ points of S^*, the cost per multiplicity-doubling step is reduced from $\widetilde{O}(n^{2/3+\delta})$ to $\widetilde{O}((\frac{n}{t})^{2/3+O(\delta)})$. The total cost over all $\widetilde{O}(t)$ multiplicity-doubling steps is $\widetilde{O}(t^{1/3}n^{2/3+O(\delta)})$.

In conclusion, the total time for running MWU is $\widetilde{O}(\frac{gn}{t} + t^{1/3}n^{2/3+O(\delta)})$. To balance this computation cost with the $\widetilde{O}(\frac{n+(nt)^{3/4}}{g})$ update cost, we set $g = \frac{t^{7/8}}{n^{1/8}} + \sqrt{t}$ and get a bound of $\widetilde{O}(\frac{n^{7/8}}{t^{1/8}} + \frac{n}{\sqrt{t}} + t^{1/3}n^{2/3+O(\delta)})$.

Lemma 3.5. There exists a data structure for the dynamic set cover problem for a set X of points and a set S of upper halfspaces in $\mathbb{R}^3$ (where $|X| + |S| = O(n)$), that maintains an $O(1)$-approximate solution w.h.p. with $\widetilde{O}(\frac{n^{7/8}}{\mathsf{opt}^{1/8}} + \frac{n}{\sqrt{\mathsf{opt}}} + \mathsf{opt}^{1/3}n^{2/3+\delta})$ amortized insertion and deletion time for any constant $\delta > 0$.

3.5.3 Algorithm for large opt

Lastly, we give an algorithm for the large opt case, which is very different from the algorithms in the previous subsections (and not based on modifying MWU). Here, we can only compute the size of the approximate set cover, not the cover itself. Like before, we will show that the problem gets easier for large opt, because we can afford a large additive error. The idea is to decompose the problem into subproblems via geometric sampling and planar separators, and then approximate the sum of the subproblems' answers by sampling again.

Data structures. We just store the dual point set S^* in a known 3D halfspace range reporting structure. The data structure by Chan [62] supports queries in $O((\log n + k)\log n)$ time for output size k, and insertions and deletions in S in polylogarithmic amortized time.

We store the xy-projection of the point set X in a known 2D triangle range searching structure [165] that supports queries in $O(\frac{n^{1/2+\delta}}{z^{1/2}} + k)$ time for output size k, and insertions and deletions in X in $\widetilde{O}(z)$ time for a given trade-off parameter $z \in [1, n]$.

Approximating the optimal value. Let b and g be parameters to be set later. Take a random sample R of the halfspaces S with size $\frac{n}{b}$. Imagine that R is included in the solution. The remaining uncovered space is the complement of the union of R, which is a 3D convex polyhedron. There are $O(|R|) = O(\frac{n}{b})$ cells in the vertical decomposition VD(R) of this polyhedron (formed by triangulating each face and drawing a vertical wall at each edge of the triangulation). Each cell is crossed by $O(b \log n)$ halfspaces w.h.p., by the standard Clarkson–Shor technique on geometric sampling [87, 88, 91, 175]. The decomposition VD(R) can be constructed in $\widetilde{O}(\frac{n}{b})$ time.

Our key idea is to use planar graph separators to divide into smaller subproblems. The following is a multi-cluster version of the standard planar separator theorem [161] (sometimes known as "r-divisions" [122]):

Lemma 3.6 (Planar Separator Theorem, Multi-Cluster Version)**.** Given a planar graph $G = (V, E)$ with n vertices, and a parameter g, we can partition V into $\frac{n}{g}$ subsets $V_1, \cdots, V_{n/g}$ of size $O(g)$ each, and an extra "boundary set" B of size $O(\frac{n}{\sqrt{g}})$, such that no two vertices from different subsets V_i and V_j are adjacent. The partition can be constructed in $\widetilde{O}(n)$ time.

(We remark that the general idea of combining cuttings/geometric sampling with planar graph separators appeared in some geometric approximation algorithms before, e.g., [4].)

We apply Lemma 3.6 to the dual graph of VD(R) (which has size $O(\frac{n}{b})$), yielding $O((n/b)/g)$ "clusters" of $O(g)$ cells each, and a set B of $O((n/b)/\sqrt{g})$ "boundary cells", in $\widetilde{O}(\frac{n}{b})$ time.

Let S_B be the subset of all halfspaces of S that cross boundary cells of B. Note that $|S_B| = O((n/b)/\sqrt{g} \cdot b \log n) = \widetilde{O}(\frac{n}{\sqrt{g}})$ w.h.p.

For each cluster γ, let X_γ denote the subset of all points of X whose xy-projections lie in the xy-projection of the cells of γ, and let S_γ denote the subset of all halfspaces of S that cross the cells of γ. Note that $|S_\gamma| = O(g \cdot b \log n) = \widetilde{O}(bg)$ w.h.p. Let opt_γ denote the optimal value for the set cover problem for the halfspaces of S_γ and the points of X_γ not covered by $R \cup S_B$.

Claim 3.2. $\sum_\gamma \mathsf{opt}_\gamma$ approximates opt with additive error $\widetilde{O}(\frac{n}{b} + \frac{n}{\sqrt{g}})$ w.h.p.

Proof. A feasible solution can be formed by taking the union of the solutions corresponding to opt_γ, together with R (to cover points not covered by VD(R)) and S_B (to cover points inside boundary cells). As $|R| = \frac{n}{b}$ and $|S_B| = \widetilde{O}(\frac{n}{\sqrt{g}})$ w.h.p., this proves that $\mathsf{opt} \leq \sum_\gamma \mathsf{opt}_\gamma + \widetilde{O}(\frac{n}{b} + \frac{n}{\sqrt{g}})$.

In the other direction, observe that if a halfspace h crosses two different clusters γ_i and γ_j, it must also cross some boundary cell in X by convexity: pick points $p \in h \cap \gamma_i$ and $q \in h \cap \gamma_j$; then the line segment $\overline{pq}$ must hit the wall of some boundary cell. So, after removing $R \cup S_B$ from the global optimal solution, we get disjoint local solutions in the clusters. This proves that $\mathsf{opt} \geq \sum_\gamma \mathsf{opt}_\gamma$. QED.

We use the following known fact about approximating a sum via random sampling (which is of course a standard trick):

Lemma 3.7. Suppose $a_1 + \cdots + a_m = T$ where $a_i \in [0, U]$. Take a random subset R of $r = \frac{(c_0/\varepsilon^2)mU\log n}{T}$ elements from $a_1, \ldots, a_m$, for a sufficiently large constant c_0. Then $\sum_{a_i \in R} a_i \cdot \frac{m}{r}$ is a $(1+\varepsilon)$-approximation to T w.h.p.

Proof. By rescaling the a_i's and T by a factor U, we may assume that $U = 1$. Define a random variable Y_i, which is the i-th sampled element that we include in R. Let $Y = \sum_{1 \le i \le r} Y_i$. Then $\mu \triangleq \mathbb{E}[Y] = r \cdot \frac{1}{m} \cdot \sum_i a_i = (c_0/\varepsilon^2)\log n$. The result follows from the standard Chernoff bound on the Y_i's:

$$\Pr\left[\left|\sum_{a_i \in R} a_i \cdot m/r - T\right| > \varepsilon T\right] = \Pr[|Y - \mu| > \varepsilon\mu] \le 2e^{-\varepsilon^2\mu/4} \le 2n^{-c_0/4}. \quad (3.2)$$

QED.

By applying the above lemma with $m = (n/b)/g$, $T = \Theta(t)$, and $U = \widetilde{O}(bg)$ (assuming opt is finite), we can $O(1)$-approximate opt by summing opt_γ over a random sample of $r = \widetilde{O}(\frac{mU}{T}) = \widetilde{O}(\frac{n}{t})$ clusters γ.

We can generate the set S_B by finding the halfspaces of S that contain the $O((n/b)/\sqrt{g})$ vertices of the cells in B—this corresponds to $O((n/b)/\sqrt{g})$ halfspace range reporting queries for the dual 3D point set S^*, each with output size $\widetilde{O}(b)$ w.h.p. and each taking $\widetilde{O}(b)$ time [62]. Thus, S_B can be found in $\widetilde{O}(\frac{n}{\sqrt{g}})$ time. We compute the union of $R \cup S_B$, which is the complement of an intersection of halfspaces, by the dual of 3D convex hull algorithm [100]. This takes $\widetilde{O}(\frac{n}{b} + \frac{n}{\sqrt{g}})$ time.

For each chosen cluster γ, we can generate S_γ similarly by $O(g)$ halfspace range reporting queries for S^*, each with output size $\widetilde{O}(b)$ w.h.p. Thus, S_γ can be found in $\widetilde{O}(bg)$ time. We can generate X_γ by performing $O(g)$ triangle range reporting queries for the 2D xy-projection of the point set X. Thus, X_γ can be found in $\widetilde{O}(g\frac{n^{1/2+\delta}}{z^{1/2}} + |X_\gamma|)$ time. We filter points of X_γ covered by $R \cup S_B$, by performing $|X_\gamma|$ planar point location queries [100] in the xy-projection of the boundary of the union of $R \cup S_B$. This takes $\widetilde{O}(|X_\gamma|)$ time. We can then compute an $O(1)$-approximation to opt_γ by running a known static set cover algorithm [14, 71] in $\widetilde{O}(bg + |X_\gamma|)$ time.

The expected sum of X_γ over all chosen clusters γ is $O(n \cdot \frac{r}{m}) = O(rbg)$. The total expected time over r clusters is $\widetilde{O}(rbg + rg\frac{n^{1/2+\delta}}{z^{1/2}}) = \widetilde{O}(\frac{bgn}{t} + \frac{gn^{3/2+\delta}}{tz^{1/2}})$. The overall expected running time is $\widetilde{O}(\frac{n}{b} + \frac{n}{\sqrt{g}} + \frac{bgn}{t} + \frac{gn^{3/2+\delta}}{tz^{1/2}})$, and we obtain an $(O(1), \widetilde{O}(\frac{n}{b} + \frac{n}{\sqrt{g}}))$-approximation. (The expected bound can be converted to worst-case by placing a time limit and re-running logarithmically many times.) Choosing $g = b^2$ yields the following result:

Lemma 3.8. Given parameters b and $z \in [1, n]$ and any constant $\varepsilon > 0$, there exists a data structure for the dynamic set cover problem for a set X of points and a set S of upper halfspaces in $\mathbb{R}^3$ (where $|X| + |S| = O(n)$), that supports insertions and deletions in $\widetilde{O}(z)$ amortized time and can find the value of an $(O(1), \widetilde{O}(\frac{n}{b}))$-approximation w.h.p. in $\widetilde{O}(\frac{n}{b} + \frac{b^3 n}{\mathsf{opt}} + \frac{b^2 n^{3/2+\delta}}{\mathsf{opt} \cdot z^{1/2}})$ time for any constant $\delta > 0$.

A minor technicality is that when applying Lemma 3.7, we have assumed that the optimal value is finite. The problem of checking whether a solution exists, i.e., whether a point set is covered by a set of halfspaces (or more generally, maintaining the lowest-depth point), subject to insertions and deletions of points and halfspaces, has already been solved before by Chan [60, Theorem 4.1], who gave a fully dynamic algorithm with $\widetilde{O}(n^{2/3})$ time per operation (based on augmenting partition trees with counters, similar to what we have done here).

Combining the algorithms. Finally, we combine all three algorithms:

1. When $\mathsf{opt} \le n^{2/9}$, we use the algorithm for small opt; the running time is $\widetilde{O}(\mathsf{opt} \cdot n^{2/3+\delta}) \le \widetilde{O}(n^{8/9+\delta})$.
2. When $n^{2/9} < \mathsf{opt} \le n^{10/13}$, we use the algorithm for medium opt; the running time is $\widetilde{O}(\frac{n^{7/8}}{\mathsf{opt}^{1/8}} + \frac{n}{\sqrt{\mathsf{opt}}} + \mathsf{opt}^{1/3} n^{2/3+O(\delta)}) \le O(n^{12/13+O(\delta)})$.
3. When $\mathsf{opt} > n^{10/13}$, we use the algorithm for large opt with $b = \tilde{\Theta}(n^{3/13})$ and $z = n^{7/13}$, so that an $(O(1), \widetilde{O}(\frac{n}{b}))$-approximation is indeed an $O(1)$-approximation; the running time is $\widetilde{O}(\frac{n}{b} + \frac{b^3 n}{\mathsf{opt}} + \frac{b^2 n^{3/2+\delta}}{\mathsf{opt} \cdot z^{1/2}}) \le O(n^{12/13+O(\delta)})$.

Lemma 3.9. There exists a data structure for the dynamic set cover problem for a set X of points and a set S of upper halfspaces in $\mathbb{R}^3$ (where $|X| + |S| = O(n)$), that maintains the value of an $O(1)$-approximate solution w.h.p. with $O(n^{12/13+\delta})$ amortized insertion and deletion time for any constant $\delta > 0$.

3.5.4 Upper and lower halfspaces

Small and medium opt. It is straightforward to modify the small and medium opt algorithms in Sections 3.5.1 and 3.5.2 to handle the case when there are both upper and lower halfspaces in S. For weighted range sampling, we can handle the upper halfspaces and the lower halfspaces separately; for example, we build the partition tree for the dual points of S separately for the upper and the lower halfspaces. To find low-depth points, we use just one partition tree for the points of X, where depth is defined relatively to the combined set of upper and lower halfspaces.

Large opt. In the large opt algorithm, we can no longer use the vertical decomposition. Instead, we pick a point p_0 inside the polyhedron and consider a "star" triangulation of the polyhedron where all tetrahedra have p_0 as a vertex. Because we can no longer use xy-projections, naively we would need to replace 2D triangle range searching with 3D simplex range searching, which would increase the update time slightly.

If p_0 is fixed, we can replace the orthogonal xy-projection with a perspective projection with respect to p_0, and we can still use 2D triangle range searching. We describe a way to find a point p_0 that stays fixed for a number of updates. (Note that p_0 need not be in X.)

Specifically, we divide the update sequence into phases with $\frac{n}{b}$ updates each. At the beginning of each phase, we set p_0 to be a point of minimum depth with respect to S, among all points in $\mathbb{R}^3$; an $O(1)$-approximation is fine and can be found in $\widetilde{O}(n)$ randomized time [6, 24].

If p_0 has depth at least $\frac{2n}{b}$ at the beginning of the phase, the minimum depth is at least $\frac{n}{b}$ during the entire phase, and a $(\frac{1}{b})$-net of size $\widetilde{O}(b)$ is a set cover and can be generated by random sampling; trivially, this gives an $(O(1), \widetilde{O}(\frac{n}{b}))$-approximation, assuming $b \leq \sqrt{n}$.

Otherwise, p_0 has depth $O(\frac{n}{b})$ during the entire phase. In the large opt algorithm, we let Z_0 be the set of $O(\frac{n}{b})$ halfspaces containing p_0 (which can be found by halfspace range reporting in the dual), include Z_0 in the solution, and remove Z_0 from S before taking the random sample R. As a result, the complement of the union of R indeed contains p_0. The rest of the algorithm is similar, using a perspective projection from p_0 instead of xy-projection. (Points covered by Z_0 should be excluded, and we can do so by adding Z_0 to S_B.) The additive error increases by $O(\frac{n}{b})$, and so is asymptotically unchanged.

The data structures have preprocessing time $\widetilde{O}(nz)$. Since we rebuild after every $\frac{n}{b}$ updates, the amortized update cost is $\widetilde{O}(bz)$. In our application with $b = \tilde{\Theta}(n^{3/13})$ and $z = n^{7/13}$, this cost does not dominate.

To summarize, Lemma 3.9 holds even when there are both upper and lower halfspaces.

Theorem 3.2. There exists a data structure for the dynamic set cover problem for a set X of points and a set S of halfspaces in $\mathbb{R}^3$ (where $|X| + |S| = O(n)$), that maintains the value of an $O(1)$-approximate solution w.h.p. with $O(n^{12/13+\delta})$ amortized insertion and deletion time for any constant $\delta > 0$.

3.6 IMPROVING STATIC SET COVER FOR 3D HALFSPACES

In this section, we show how the techniques we have developed for the dynamic geometric set cover problem can lead to a randomized algorithm for static set cover for 3D halfspaces running in $O(n \log n)$ time, which improves our previous $O(n \log n (\log \log n)^{O(1)})$ randomized algorithm [71], and is optimal (matching our lower bound result in Theorem 2.4). The new algorithm combines the medium opt algorithm in Section 3.5.2 and the large opt algorithm in Section 3.5.3.

Let N be a fixed parameter used to control the error probability.

Case 1: $\mathsf{opt} \le n^{5/6+\delta}$. Here, we modify the medium opt algorithm in Section 3.5.2. We first guess a value $t' \in [\mathsf{opt}/n^{\delta}, \mathsf{opt}]$; a constant ($O(1/\delta)$) number of guesses suffices. The preprocessing algorithm will use this parameter t'.

In the static setting, our old version of Lemma 3.4 for constructing T_0 suffices and takes $O(n \log n)$ time [71]. Specifically, we construct a set $T_0 \subseteq S$ of $O(t')$ halfspaces so that after removing the points covered by T_0, every halfspace of S contains at most $O(\frac{n}{t'})$ points. Since we are in the static setting, we can explicitly remove the points covered by T_0. The shallow versions of the partition trees [166] can be preprocessed in $O(n \log n)$ time. Parts of the algorithm can be simplified: there is no need to divide into phases, and no need for the extra counters $c_v^{\#}$ and the extra point set E during the MWU algorithm. The MWU algorithm then runs in $\widetilde{O}(t \cdot (\frac{n}{t'})^{2/3+O(\delta)}) = \widetilde{O}(t^{1/3} n^{2/3+O(\delta)}) = O(n^{17/18+O(\delta)})$ time. As we need to run the MWU algorithm for all guesses t that are powers of 2 (up to $n^{5/6+\delta}$), the running time of the MWU algorithm increases by a logarithmic factor but remains $O(n^{17/18+O(\delta)})$, excluding preprocessing.

To bound the error probability by $O(\frac{1}{N})$, we can re-run the MWU algorithm $O(\log N)$ times. The total running time including preprocessing is $O(n \log n + n^{17/18+O(\delta)} \log N)$.

Case 2: $\mathsf{opt} > n^{5/6+\delta}$. Here, we modify the large opt algorithm in Section 3.5.3. In the static setting, there is no need for the halfspace range reporting and triangle range searching structures. We first generate the conflict lists of the cells (the lists of halfspaces crossing the cells) in $\mathrm{VD}(R)$ in $O(n \log n)$ expected time [91]. We verify that indeed every conflict list has size $O(b \log n)$; if not, we restart, with $O(1)$ expected number of trials till success. For each point $p \in X$, we locate the cell in $\mathrm{VD}(R)$ which contains p in the xy-projection, by planar point location in $O(n \log n)$ time [100].

We refine $\mathrm{VD}(R)$ before applying the planar separator theorem: for each cell in $\mathrm{VD}(R)$, we subdivide into subcells each containing at most b points of X in the xy-projection. The number of extra cuts is $O(\frac{n}{b})$, and so the new decomposition still has $O(\frac{n}{b})$ cells.

(As noted before, when there are both upper and lower halfspaces, we replace the vertical decomposition with a "star" triangulation and replace orthogonal xy-projection with a perspective projection.)

The original large opt algorithm takes a random sample of the clusters to approximate the value of the optimal solution. To compute an actual solution, we instead use recursion in *every* cluster.

Recall that the number of clusters is $O(n/(bg))$. For each cluster γ, the number of halfspaces in S_γ is $\widetilde{O}(bg)$, and the number of points in X_γ is also $\widetilde{O}(bg)$, because of the above refinement of $\mathrm{VD}(R)$. Recall that after removing halfspaces in S_B, the S_γ's become disjoint; and after removing points covered by $R \cup S_B$, the sum of the optimal values in the subproblems opt_γ is upper-bounded by opt.

We set $b = n^{1/6}$ and $g = n^{1/3}$ so that the additive error is $\widetilde{O}(\frac{n}{b} + \frac{n}{\sqrt{g}}) = \widetilde{O}(n^{5/6}) \leq O(\mathsf{opt}/\log n)$.

Analysis. The worst-case expected running time of the overall algorithm for input size n satisfies the recurrence $T(n) \leq \sum_i T(n_i) + O(n \log n + n^{17/18+O(\delta)} \log N)$, for some n_i's with $\sum_i n_i \leq n$ and $\max_i n_i = \widetilde{O}(\sqrt{n})$. This recurrence solves to $T(n) = O(n \log n + n \log N)$. (The reason is that $\log n$ forms a geometric progression as we descend downward in the recursion tree, and the total instance sizes at each level is bounded by n.)

Let $E(n, x)$ be the worst-case additive error of the computed solution for an input of size n and optimal value x. In Case 1, the additive error is $O(x)$. In general, we have the recurrence $E(n, x) \leq \max\{O(x), \sum_i E(n_i, x_i) + O(\frac{x}{\log n})\}$, for some n_i's and x_i's with $\sum_i n_i \leq n$, and $\max_i n_i = \widetilde{O}(\sqrt{n})$, and $\sum_i x_i \leq x$. This recurrence solves to $E(n, x) = O(x)$. (The reason is that $\frac{1}{\log n}$ forms a geometric progression as we descend downward in the recursion tree.) Thus, the algorithm yields an $O(1)$-approximation.

The total error probability over the entire recursion is bounded by $O(\frac{n}{N})$. We set $N = n^c$ for the global input size n and an arbitrarily large constant c.

Theorem 3.3. Given a set X of points and a set S of halfspaces in $\mathbb{R}^3$, where $|X| + |S| = n$, there exists a randomized $O(1)$-approximation algorithm for finding the minimum subset of halfspaces from S to cover all points in X, which runs in $O(n \log n)$ expected time and is correct w.h.p.

It is possible to modify the analysis to get $O(n \log n)$ worst-case time instead of expected. In Case 2, the conflict lists have size $O(b \log n)$ w.h.p. and the construction time is actually $O(n \log n)$ w.h.p., at the root of the recursion. In subsequent levels of the recursion, we can apply a Chernoff bound to get a high-probability bound on the total running time.

3.7 UNWEIGHTED INTERVALS

Let $(X, \mathcal{I})$ be a dynamic (unweighted) interval set cover instance where X is the set of points in $\mathbb{R}$ and $\mathcal{I}$ is the set of intervals, and let $\varepsilon > 0$ denote the approximation factor. Our goal is to design a data structure $\mathcal{D}$ that maintains a $(1+\varepsilon)$-approximate set cover solution for the current instance $(X, \mathcal{I})$ and supports the desired queries (i.e., size, membership, and reporting queries) to the solution. Without loss of generality, we may assume that the *point range* of $(X, \mathcal{I})$ is $[0, 1]$, i.e., the points in X are always in the range $[0, 1]$.

Let r and $\alpha < 1$ be parameters to be determined. Consider the initial instance $(X, \mathcal{I})$ and let $n = |X| + |\mathcal{I}|$. We partition the range $[0, 1]$ into r connected portions (i.e., intervals) $J_1, \dots, J_r$ such that each portion J_i contains $O(n/r)$ points in X and $O(n/r)$ endpoints of intervals in $\mathcal{I}$. Define $X_i = X \cap J_i$ and $\mathcal{I}_i = \{I \in \mathcal{I} : I \cap J_i \neq \emptyset \text{ and } J_i \nsubseteq I\}$. When the instance $(X, \mathcal{I})$ changes, the portions $J_1, \dots, J_r$ remain unchanged while the X_i's and $\mathcal{I}_i$'s will change along with X and $\mathcal{I}$. Thus, we can view each $(X_i, \mathcal{I}_i)$ as a dynamic interval set cover instance with point range J_i. We then recursively build a dynamic interval set cover data structure $\mathcal{D}_i$ which maintains a $(1+\tilde{\varepsilon})$-approximate set cover solution for $(X_i, \mathcal{I}_i)$, where $\tilde{\varepsilon} = \alpha\varepsilon$. We call $(X_1, \mathcal{I}_1), \dots, (X_r, \mathcal{I}_r)$ *sub-instances* and call $\mathcal{D}_1, \dots, \mathcal{D}_r$ *sub-structures*. Besides the recursively built sub-structures, we also need three simple support data structures. The first one is the data structure $\mathcal{A}$ in the following lemma that can help to compute an optimal interval set cover in *output-sensitive* time.

Lemma 3.10 ([10])**.** One can store a dynamic (unweighted) interval set cover instance $(X, \mathcal{I})$ in a data structure $\mathcal{A}$ with $O(n \log n)$ construction time and $O(\log n)$ update time, such that at any point, an optimal solution for $(X, \mathcal{I})$ can be computed in $O(\mathsf{opt} \cdot \log n)$ time with the access to $\mathcal{A}$.

The second one is a dynamic data structure $\mathcal{B}$ built on $\mathcal{I}$ which can report, for a given query interval J, an interval $I \in \mathcal{I}$ that contains J (if such an interval exists); as shown in [10], there exists such a data structure with $O(\log n)$ update time, $O(\log n)$ query time, and $O(n \log n)$ construction time. The third one is a (static) data structure $\mathcal{L}$ which can report, for a given query point $q \in \mathbb{R}$, the portion J_i that contains q; for this one, we can simply use a binary search tree built on $J_1, \dots, J_r$ which has $O(\log r)$ query time. Our data structure $\mathcal{D}$ simply consists of the sub-structures $\mathcal{D}_1, \dots, \mathcal{D}_r$ and the support data structures.

It is easy to construct $\mathcal{D}$ in $O(n \log^2 n)$ time. To see this, we define $|(X, \mathcal{I})|$ as the total number of points in X and endpoints of intervals in $\mathcal{I}$ that are contained in the point range $[0, 1]$ of $(X, \mathcal{I})$. We have $|(X, \mathcal{I})| \leq \sum_{i=1}^{r} |(X_i, \mathcal{I}_i)|$ and $|(X_i, \mathcal{I}_i)| \leq |(X, \mathcal{I})|/2$ for all $i \in [r]$ (as r is sufficiently large). Now let $C(m)$ denote the time for constructing the data structure on an instance $(X, \mathcal{I})$ with $|(X, \mathcal{I})| = m$. We then have the recurrence $C(m) = \sum_{i=1}^{r} C(m_i) + O(m \log m)$, where $m \leq \sum_{i=1}^{r} m_i$ and $m_i \leq m/2$ for all $i \in [r]$. This recurrence solves to $C(m) = O(m \log^2 m)$. Since $|(X, \mathcal{I})| = O(n)$, $\mathcal{D}$ can be constructed in $O(n \log^2 n)$ time, i.e., in $\widetilde{O}(n)$ time.

Updating the sub-structures and reconstruction. Whenever the instance $(X, \mathcal{I})$ changes due to an insertion/deletion on X or $\mathcal{I}$, we first update the support data structures. After that, we recursively update the sub-structures $\mathcal{D}_i$ for $i \in [r]$ that $(X_i, \mathcal{I}_i)$ changes. An insertion/deletion on X only changes one X_i, and an insertion/deletion on $\mathcal{I}$ changes at most two $\mathcal{I}_i$'s (because an interval has two endpoints). Also, we observe that if the inserted/deleted interval I is "one-sided" in the sense that one endpoint of I is outside the point range $[0, 1]$, then that insertion/deletion only changes one $\mathcal{I}_i$. This observation is critical in the analysis of our data structure.

Besides the update, our data structure $\mathcal{D}$ will be periodically reconstructed. Specifically, the $(i+1)$-th reconstruction happens after processing n_i/r updates from the i-th reconstruction, where n_i denotes the size of $(X, \mathcal{I})$ at the point of the i-th reconstruction. (The 0-th reconstruction is just the initial construction of $\mathcal{D}$.)

Constructing a solution. We now describe how to construct an approximately optimal set cover $\mathcal{I}_{\text{appx}}$ for the current $(X, \mathcal{I})$ using our data structure $\mathcal{D}$. Denote by opt the size of an optimal set cover for the current $(X, \mathcal{I})$; we define $\mathsf{opt} = \infty$ if $(X, \mathcal{I})$ does not have a set cover. Set $\delta = \min\{n, c \cdot (r + \varepsilon r)/(\varepsilon - \alpha\varepsilon)\}$ for a sufficiently large constant c. If $\mathsf{opt} \leq \delta$, then we are able to use the algorithm of Lemma 3.10 to compute an optimal set cover for $(X, \mathcal{I})$ in $O(\delta \cdot \log n)$ time (with the help of the support data structure $\mathcal{A}$). Therefore, we simulate that algorithm within that amount of time. If the algorithm successfully computes a solution, we use it as our $\mathcal{I}_{\text{appx}}$. Otherwise, we construct $\mathcal{I}_{\text{appx}}$ as follows. For each $i \in [r]$, if J_i can be covered by an interval $I \in \mathcal{I}$, we define $\mathcal{I}_i^* = \{I\}$, otherwise let $\mathcal{I}_i^*$ be the $(1+\tilde{\varepsilon})$-approximate solution for $(X_i, \mathcal{I}_i)$ maintained in the sub-structure $\mathcal{D}_i$. (If for some $i \in [r]$, J_i cannot be covered by any interval in $\mathcal{I}$ and the sub-structure $\mathcal{D}_i$ tells us that the current $(X_i, \mathcal{I}_i)$ does not have a set cover, then we immediately decide that the current $(X, \mathcal{I})$ has no feasible set cover.) Then we define $\mathcal{I}_{\text{appx}} = \bigsqcup_{i=1}^{r} \mathcal{I}_i^*$, which is clearly a set cover of $(X, \mathcal{I})$. Note that for each $i \in [r]$, we can find in $O(\log n)$ time an interval $I \in \mathcal{I}$ that covers J_i using the support data structure $\mathcal{B}$ (if such an interval exists).

Answering queries to the solution. We show how to store the solution $\mathcal{I}_{\text{appx}}$ properly so that the desired queries for $\mathcal{I}_{\text{appx}}$ can be answered efficiently. If $\mathcal{I}_{\text{appx}}$ is computed by the algorithm of Lemma 3.10, then the size of $\mathcal{I}_{\text{appx}}$ is at most δ and we have all elements of $\mathcal{I}_{\text{appx}}$ in hand. In this case, we simply build a binary search tree on $\mathcal{I}_{\text{appx}}$ which can answer the desired queries with the required time costs. On the other hand, if $\mathcal{I}_{\text{appx}}$ is defined as $\mathcal{I}_{\text{appx}} = \bigsqcup_{i=1}^{r} \mathcal{I}_i^*$, the size of $\mathcal{I}_{\text{appx}}$ can be large and we are not able to retrieve all elements of $\mathcal{I}_{\text{appx}}$. However, in this case, each $\mathcal{I}_i^*$ either consists of a single interval that covers J_i or is the solution maintained in the sub-structure $\mathcal{D}_i$.

To support the size query, we only need to compute $|\mathcal{I}_i^*|$ (which can be done by recursively making size queries to the sub-structures) and calculate $|\mathcal{I}_{\text{appx}}| = \sum_{i=1}^{r} |\mathcal{I}^*|$; we then simply store this quantity so that a size query can be answered in $O(1)$ time. To support membership queries, we compute an index set $P \subseteq [r]$ consisting of the indices $i \in [r]$ such that $\mathcal{I}_i^*$ consists of a single interval covering J_i. Then we collect all intervals in the $\mathcal{I}_i^*$'s for $i \in P$, the number of which is at most r. We store these intervals in a binary search tree T which can answer membership queries in $O(\log r)$ time. To answer a membership query $I \in \mathcal{I}$, we first check if I is stored in T. After that, we find the (up to) two instances $\mathcal{I}_i$'s that contains I, and make membership queries to the sub-structures $\mathcal{D}_i$ to check whether $I \in \mathcal{I}_i^*$ (if $i \in [r]\backslash P$). Finally, to answer the reporting query, we first report the intervals stored in T and then for every $i \in [r]\backslash P$, we make recursively a reporting query to $\mathcal{D}_i$, which reports the intervals in $\mathcal{I}_i^*$.

Now we analyze the query time. If the solution $\mathcal{I}_{\text{appx}}$ is computed by the algorithm of Lemma 3.10, then it is stored in a binary search tree and we can answer a size query, a membership query, and a reporting query in $O(1)$ time, $O(\log |\mathcal{I}_{\text{appx}}|)$ time, and $O(|\mathcal{I}_{\text{appx}}|)$ time, respectively. So it suffices to consider the case where we construct the solution as $\mathcal{I}_{\text{appx}} = \bigsqcup_{i=1}^{r} \mathcal{I}_i^*$. In this case, answering a size query still takes $O(1)$, because we explicitly compute $|\mathcal{I}_{\text{appx}}|$.

To analyze the time cost for a membership query, we need to distinguish *one-sided* and *two-sided* queries. We use $Q_1(n)$ and $Q_2(n)$ to denote the time cost for a one-sided membership query (i.e., one endpoint of the query interval is outside the point range) and a two-sided membership query (i.e., both endpoints of the query interval are inside the point range), respectively, when the size of the current instance is n. Then for $Q_1(n)$, we have the recurrence

$$Q_1(n) \leq Q_1(O(n/r)) + O(\log r), \tag{3.3}$$

which solves to $Q_1(n) = O(\log n)$, as we only need to recursively query on one $\mathcal{D}_i$ (which is again a one-sided query). For $Q_2(n)$, we have the recurrence

$$Q_2(n) \leq \max\{Q_2(O(n/r)), 2Q_1(O(n/r))\} + O(\log r), \tag{3.4}$$

which also solves to $Q_2(n) = O(\log n)$, as we may need to have a recursive two-sided query on one $\mathcal{D}_i$ or have recursive one-sided queries on two $\mathcal{D}_i$'s. Therefore, a membership query can be answered in $O(\log n)$ time.

Finally, to answer a reporting query, we first report the intervals stored in T and recursively query the data structures $\mathcal{D}_i$ for all $i \in [r]\backslash P$ such that $\mathcal{I}_i^* \neq \emptyset$. Thus, in the recursion tree, the number of leaves is bounded by $|\mathcal{I}_{\text{appx}}|$ since at each leaf node we need to report at least one element. A naive analysis shows the overall time cost for a reporting query is $O(|\mathcal{I}_{\text{appx}}| \cdot \log n)$, because the height of the recursion tree is $O(\log_r n)$, and at each node of the recursion tree, the work can be done in $O(\log r)$ time plus $O(1)$ per outputted element. For a more careful analysis, our approximation factor analysis in the next paragraph shows that at each node of the recursion tree, we can allow an additive error of $O(r)$, while keeping the overall approximation factor $1 + \varepsilon$. So consider adding a "virtual leaf" below each internal node of the recursion tree, which simply outputs a virtual interval in the approximate solution. In this way, the total number of leaves is still bounded by $(1 + \varepsilon) \cdot |\mathcal{I}_{\text{appx}}|$, but now each internal node has degree at least 2, so the total number of nodes is only $O(|\mathcal{I}_{\text{appx}}|)$. As a result, we conclude the reporting query takes $O(|\mathcal{I}_{\text{appx}}|)$ time.

Correctness. First, we observe that $\mathcal{D}$ makes a no-solution decision iff the current instance $(X, \mathcal{I})$ has no set cover. Indeed, if we make a no-solution decision, then J_i is not covered by any interval in $\mathcal{I}$ and the sub-instance $(X_i, \mathcal{I}_i)$ has no set cover for some $i \in [r]$; in this case, $(X, \mathcal{I})$ has no set cover because the points in X_i can only be covered by the intervals in $\mathcal{I}_i$ or by an interval that covers J_i. On the other hand, if we do not make a no-solution decision, then the set $\mathcal{I}_{\text{appx}}$ we construct is a feasible solution for $(X, \mathcal{I})$.

Now it suffices to show that the solution $\mathcal{I}_{\text{appx}}$ is a $(1 + \varepsilon)$-approximation of an optimal set cover for $(X, \mathcal{I})$. Let $\mathcal{I}_{\text{opt}}$ be an optimal set cover for $(X, \mathcal{I})$. We have to show $|\mathcal{I}_{\text{appx}}| \leq (1 + \varepsilon) \cdot |\mathcal{I}_{\text{opt}}|$. If $\mathcal{I}_{\text{appx}}$ is computed by the algorithm of Lemma 3.10, then $|\mathcal{I}_{\text{appx}}| = |\mathcal{I}_{\text{opt}}|$. Otherwise, we know that $|\mathcal{I}_{\text{opt}}| > \delta$, which implies $|\mathcal{I}_{\text{opt}}| > c \cdot (r + \varepsilon r)/(\varepsilon - \alpha\varepsilon)$ for a sufficiently large constant c, because we cannot have $|\mathcal{I}_{\text{opt}}| > n$. In this case, we show the following.

Fact 3.1. $|\mathcal{I}_{\text{appx}}| \leq (1 + \tilde{\varepsilon}) \cdot |\mathcal{I}_{\text{opt}}| + O(r)$.

Proof. For $i \in [r]$, let opt_i be the size of an optimal set cover of $(X_i, \mathcal{I}_i)$ if $\mathcal{I}_i^*$ is the solution of $(X_i, \mathcal{I}_i)$ maintained by $\mathcal{D}_i$, and let $\mathsf{opt}_i = 1$ otherwise. Then for all $i \in [r]$, we have $|\mathcal{I}_i^*| \leq (1 + \tilde{\varepsilon}) \cdot \mathsf{opt}_i$. Since $|\mathcal{I}_{\text{appx}}| = \sum_{i=1}^r |\mathcal{I}_i^*|$, we have $|\mathcal{I}_{\text{appx}}| \leq (1 + \tilde{\varepsilon}) \cdot \sum_{i=1}^r \mathsf{opt}_i$. It suffices to show that $\sum_{i=1}^r \mathsf{opt}_i = |\mathcal{I}_{\text{opt}}| + O(r)$. Let n_i be the number of intervals in $\mathcal{I}_{\text{opt}}$ that are contained in J_i for $i \in [r]$. Clearly, $|\mathcal{I}_{\text{opt}}| \geq \sum_{i=1}^r n_i$. We claim that $\mathsf{opt}_i \leq n_i + 2$, which implies $\sum_{i=1}^r \mathsf{opt}_i = |\mathcal{I}_{\text{opt}}| + O(r)$. If J_i can be covered by some interval in $\mathcal{I}$, then

$\mathsf{opt}_i = 1 \leq n_i + 2$. Otherwise, we take all n_i intervals in $\mathcal{I}_{\text{opt}}$ that are contained in J_i and the (at most) two intervals in $\mathcal{I}_{\text{opt}}$ with one endpoint in J_i which have maximal intersections with J_i (i.e., the interval containing the left end of J_i with the rightmost right endpoint and the interval containing the right end of J_i with the leftmost left endpoint). These $n_i + 2$ intervals form a set cover of $(X_i, \mathcal{I}_i)$ and thus $\mathsf{opt}_i \leq n_i + 2$. QED.

Using the above observation and the fact $|\mathcal{I}_{\text{opt}}| > c \cdot (r + \varepsilon r)/(\varepsilon - \alpha\varepsilon) = c \cdot (r + \varepsilon r)/(\varepsilon - \tilde{\varepsilon})$, we conclude that $|\mathcal{I}_{\text{appx}}| \leq (1 + \varepsilon) \cdot |\mathcal{I}_{\text{opt}}|$.

Update time. To analyze the update time of our data structure $\mathcal{D}$, it suffices to consider the first period (including the first reconstruction). The first period consists of n_0/r operations, where n_0 is the size of the initial $(X, \mathcal{I})$. The size of $(X, \mathcal{I})$ during the first period is always in between $(1 - 1/r)n_0$ and $(1 + 1/r)n_0$ and is hence $\Theta(n_0)$, since r is a sufficiently large constant. We first observe that, excluding the recursive updates for the sub-structures, each update of $\mathcal{D}$ takes $O(r \log n/(\varepsilon - \alpha\varepsilon) + r \log^2 n)$ (amortized) time, where n is the size of the current instance $(X, \mathcal{I})$. Updating the support data structures takes $O(\log n)$ time. When constructing the solution $\mathcal{I}_{\text{appx}}$, we need to simulate the algorithm of Lemma 3.10 within $O(\delta \cdot \log n)$ time, i.e., $O(r \log n/(\varepsilon - \alpha\varepsilon))$ time. The time for storing the solution $\mathcal{I}_{\text{appx}}$ is also bounded by $O(r \log n/(\varepsilon - \alpha\varepsilon))$, because we only need to explicitly store $\mathcal{I}_{\text{appx}}$ when it is computed by the algorithm of Lemma 3.10, in which case its size is at most $\delta = O(r/(\varepsilon - \alpha\varepsilon))$. Finally, the reconstruction takes $O(r \log^2 n)$ amortized time, because the time cost of the (first) reconstruction is $O(n_1 \log^2 n_1) = O(n_0 \log^2 n_0)$ and the first period consists of n_0/r operations.

Next, we consider the recursive updates for the sub-structures. The depth of the recursion is $O(\log_r n)$. If we set $\alpha = 1 - 1/\log_r n$, the approximation parameter is $\Theta(\varepsilon)$ in any level of the recurrence. We distinguish three types of updates according to the current operation. The first type is caused by an insertion/deletion of a point in X (we call it *point update*). The second type is caused by an insertion/deletion of an interval in $\mathcal{I}$ whose one endpoint is outside the point range $[0, 1]$ of $(X, \mathcal{I})$ (we call it *one-sided interval update*). The third type is caused by an insertion/deletion of an interval in $\mathcal{I}$ whose both endpoints are inside the point range (we call it *two-sided interval update*).

In a point update, we only need to recursively update one sub-structure (which is again a point update), because an insertion/deletion on X only changes one X_i. Similarly, in a one-sided interval update, we only need to do a recursive one-sided interval update on one sub-structure, because the inserted/deleted interval belongs to one $\mathcal{I}_i$. Finally, in a two-sided interval update, we may need to do a recursive two-sided interval update on one sub-structure (when the two endpoints of the inserted/deleted interval belong to the same range J_i), or two recursive one-sided interval updates on two sub-structures (when the two endpoints belong to different J_i's).

Let $U(n)$, $U_1(n)$, $U_2(n)$ denote the time costs of a point update, a one-sided interval update, and a two-sided interval update, respectively, when the size of the current instance is n. Then for $U(n)$, we have the recurrence

$$U(n) \leq U(O(n/r)) + O(r \log^2 n/\varepsilon), \tag{3.5}$$

which solves to $U(n) = O(r \log_r n \log^2 n/\varepsilon)$. Similarly, for $U_1(n)$, we have the same recurrence, solving to $U_1(n) = O(r \log_r n \log^2 n/\varepsilon)$. Finally, the recurrence for $U_2(n)$ is

$$\begin{aligned} U_2(n) &\leq \max\{U_2(O(n/r)), 2U_1(O(n/r))\} + O(r \log^2 n/\varepsilon) \\ &= \max\{U_2(O(n/r)), O(r \log_r n \log^2 n/\varepsilon)\} + O(r \log^2 n/\varepsilon). \end{aligned} \tag{3.6}$$

A simple induction argument shows that $U_2(n) = O(r \log_r n \log^2 n/\varepsilon)$. Setting r to be a sufficiently large constant, our data structure $\mathcal{D}$ can be updated in $O(\log^3 n/\varepsilon)$ amortized time.

Theorem 3.4. There exists a dynamic data structure for $(1+\varepsilon)$-approximate unweighted interval set cover with $O(\log^3 n/\varepsilon)$ amortized update time and $O(n \log^2 n)$ construction time, which can answer size, membership, and reporting queries in $O(1)$, $O(\log n)$, and $O(\mathsf{opt})$ time, respectively, where n is the size of the instance and opt is the size of the optimal solution.

3.8 UNWEIGHTED UNIT SQUARES

It was shown in [10] that dynamic unit-square set cover can be reduced to dynamic *quadrant* set cover. Specifically, dynamic unit-square set cover can be solved with the same update time as dynamic quadrant set cover, by losing only a constant factor on the approximation ratio. Therefore, it suffices to consider dynamic quadrant set cover. Note that the problem is still challenging, as we need to simultaneously deal with all four types of quadrants.

Similar to interval set cover, quadrant set cover also admits an output-sensitive algorithm:

Lemma 3.11 ([10]). One can store a dynamic (unweighted) quadrant set cover instance $(X, \mathcal{Q})$ in a data structure $\mathcal{A}$ with $\widetilde{O}(n)$ construction time and $\widetilde{O}(1)$ update time such that at any point, a constant-factor approximate solution for $(X, \mathcal{Q})$ can be computed in $\widetilde{O}(\mathsf{opt})$ time with the access to $\mathcal{A}$.

Let $(X, \mathcal{Q})$ be a dynamic (unweighted) quadrant set cover instance where X is the set of points in $\mathbb{R}^2$ and $\mathcal{Q}$ is the set of quadrants. Suppose $\mu = O(1)$ is the approximation factor of the algorithm of Lemma 3.11. Our goal is to design a data structure $\mathcal{D}$ that maintains a $(\mu + \varepsilon)$-approximate set cover solution for the current instance $(X, \mathcal{Q})$ and supports the desired queries to the solution, for a given parameter $\varepsilon > 0$. Without loss of generality, we may assume that the *point range* of $(X, \mathcal{Q})$ is $[0, 1]^2$, i.e., the points in X are always in the range $[0, 1]^2$. We say a quadrant in $\mathcal{Q}$ is *trivial* (resp., *nontrivial*) if its vertex is outside (resp., inside) the point range $[0, 1]^2$. Note that a trivial quadrant is "equivalent" to a horizontal/vertical halfplane in terms of the coverage in $[0, 1]^2$.

Let r and $\alpha < 1$ be parameters to be determined. Consider the initial instance $(X, \mathcal{Q})$ and let $n = |X| + |\mathcal{Q}|$. We partition the point range $[0, 1]^2$ into $r \times r$ rectangular cells using $r - 1$ horizontal lines and $r - 1$ vertical lines such that each row (resp., column) of r cells contains $O(n/r)$ points in X and $O(n/r)$ vertices of the quadrants in $\mathcal{Q}$. Let $\Box_{i,j}$ be the cell in the i-th row and j-th column for $(i, j) \in [r]^2$. We denote by R_i the i-th row (i.e., $R_i = \bigcup_{j=1}^{r} \Box_{i,j}$) for $i \in [r]$ and by C_j the j-th column (i.e., $C_j = \bigcup_{i=1}^{r} \Box_{i,j}$) for $j \in [r]$. Define $X_{i,j} = X \cap \Box_{i,j}$, $X_{i,\bullet} = X \cap R_i$, and $X_{\bullet,j} = X \cap C_j$, for $i, j \in [r]$.

Next, we decompose $\mathcal{Q}$ into small subsets as follows. We say a quadrant Q *left intersects* a rectangle R if $R \nsubseteq Q$ and Q contains the left boundary of R. Among a set of quadrants that left intersect a rectangle R, the *maximal* one refers to the quadrant whose vertex is the rightmost, or equivalently, whose intersection with R is maximal. Similarly, we can define the notions of "right intersect", "top intersect", and "bottom intersect". For $i, j \in [r]$, we define $\mathcal{Q}_{i,j} \subseteq \mathcal{Q}$ be the subset consisting of all *nontrivial* quadrants whose vertices lie in $\Box_{i,j}$ and the (up to) four *nontrivial* maximal quadrants that left, right, top, bottom intersect $\Box_{i,j}$; we call the latter the four *special* quadrants in $\mathcal{Q}_{i,j}$. Similarly, for $i \in [r]$ (resp., $j \in [r]$), we define $\mathcal{Q}_{i,\bullet} \subseteq \mathcal{Q}$ (resp., $\mathcal{Q}_{\bullet,j} \subseteq \mathcal{Q}$) be the subset consisting of all nontrivial quadrants whose vertices lie in R_i (resp., C_j) and the four nontrivial maximal quadrants that left, right, top, bottom intersect R_i (resp., C_j); we call the latter the four *special* quadrants in $\mathcal{Q}_{i,\bullet}$ (resp., $\mathcal{Q}_{\bullet,j}$).

When the instance $(X, \mathcal{Q})$ changes, the cells $\Box_{i,j}$ (as well as the rows R_i and columns C_j) remain unchanged while the sets $X_{i,j}$, $X_{i,\bullet}$, $X_{\bullet,j}$ (resp., $\mathcal{Q}_{i,j}$, $\mathcal{Q}_{i,\bullet}$, $\mathcal{Q}_{\bullet,j}$) will change along with X (resp., $\mathcal{Q}$). We view each $(X_{i,j}, \mathcal{Q}_{i,j})$ as a dynamic quadrant set cover instance with point range $\Box_{i,j}$, and recursively build a sub-structure $\mathcal{D}_{i,j}$ that maintains a $(\mu + \tilde{\varepsilon})$-approximate set cover solution for $(X_{i,j}, \mathcal{Q}_{i,j})$, where $\tilde{\varepsilon} = \alpha\varepsilon$. Similarly, we view each $(X_{i,\bullet}, \mathcal{Q}_{i,\bullet})$ (resp.,

$(X_{\bullet,j}, \mathcal{Q}_{\bullet,j})$) as a dynamic quadrant set cover instance with point range R_i (resp., C_j), and recursively build a sub-structure $\mathcal{D}_{i,\bullet}$ (resp., $\mathcal{D}_{\bullet,j}$) that maintains a $(\mu+\tilde{\varepsilon})$-approximate set cover solution for $(X_{i,\bullet}, \mathcal{Q}_{i,\bullet})$ (resp., $(X_{\bullet,j}, \mathcal{Q}_{\bullet,j})$). For convenience, we call $(X_{i,j}, \mathcal{Q}_{i,j})$ the *cell sub-instances*, $(X_{i,\bullet}, \mathcal{Q}_{i,\bullet})$ the *row sub-instances*, and $(X_{\bullet,j}, \mathcal{Q}_{\bullet,j})$ the *column sub-instances*.

Besides the data structures recursively built on the sub-instances, we also need some simple support data structures. The first one is the data structure $\mathcal{A}$ required for the output-sensitive algorithm for quadrant set cover (Lemma 3.11). The second one is a dynamic data structure $\mathcal{B}$ built on $\mathcal{Q}$, which can report, for a given query rectangle R, the maximal quadrant in $\mathcal{Q}$ that left/right/top/bottom intersects R. The third one is a dynamic data structure $\mathcal{C}$ built on $\mathcal{Q}$, which can report, for a given query rectangle R, a quadrant in $\mathcal{Q}$ that contains R (if such a quadrant exists). The fourth one is a plane point-location data structure $\mathcal{L}$, which can report, for a given query point $q \in [0,1]^2$, the cell $\square_{i,j}$ that contains q. As shown in [10], all these support data structures can be built in $\widetilde{O}(n)$ time and updated in $\widetilde{O}(1)$ time.

Our data structure $\mathcal{D}$ consists of the recursively built sub-structures $\mathcal{D}_{i,j}$, $\mathcal{D}_{i,\bullet}$, $\mathcal{D}_{\bullet,j}$ and the support data structures $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{L}$. It is easy to construct $\mathcal{D}$ in $\widetilde{O}(2^{O(\log_r n)} \cdot n)$ time. To see this, notice that the size of each sub-instance is of size $O(n/r)$. Also, the total size of all (row, column, cell) sub-instances is bounded by $O(n)$. Therefore, if we denote by $\mathscr{C}(n)$ the construction time of the data structure when the size of the instance is n, we have the recurrence

$$\mathscr{C}(n) = \sum_{i=1}^{r}\sum_{j=1}^{r} \mathscr{C}(n_{i,j}) + \sum_{i=1}^{r} \mathscr{C}(n_{i,\bullet}) + \sum_{j=1}^{r} \mathscr{C}(n_{\bullet,j}) + \widetilde{O}(n) \tag{3.7}$$

for some $n_{i,j}, n_{i,\bullet}, n_{\bullet,j}$ satisfying $\sum_{i=1}^{r}\sum_{j=1}^{r} n_{i,j} + \sum_{i=1}^{r} n_{i,\bullet} + \sum_{j=1}^{r} n_{\bullet,j} = O(n)$ and $n_{i,j} = O(n/r)$, $n_{i,\bullet} = O(n/r)$, $n_{\bullet,j} = O(n/r)$ for all $i,j \in [r]$. The total size of the data structures grows by a constant factor at each level, and the recursion depth is $O(\log_r n)$, so the recurrence solves to $\mathscr{C}(n) = \widetilde{O}(2^{O(\log_r n)} \cdot n)$.

Update of the sub-structures and reconstruction. Whenever the instance $(X, \mathcal{Q})$ changes due to an insertion/deletion on X or $\mathcal{Q}$, we first update the support data structures. After that, we recursively update the sub-structures $\mathcal{D}_{i,j}$, $\mathcal{D}_{i,\bullet}$, $\mathcal{D}_{\bullet,j}$ for which the underlying sub-instances change. Observe that an insertion/deletion on X only changes one $X_{i,j}$, one $X_{i,\bullet}$, and one $X_{\bullet,j}$ (so at most three sub-instances). An insertion/deletion of a trivial quadrant does not change any sub-instances, while an insertion/deletion of a nontrivial quadrant changes at most $O(r)$ sub-instances (see Fig. 3.5).

Besides the update, our data structure $\mathcal{D}$ will be periodically reconstructed. Specifically, the $(i+1)$-th reconstruction happens after processing n_i/r updates from the i-th reconstruction, where n_i denotes the size of $(X, \mathcal{Q})$ at the point of the i-th reconstruction. (The 0-th reconstruction is just the initial construction of $\mathcal{D}$.)

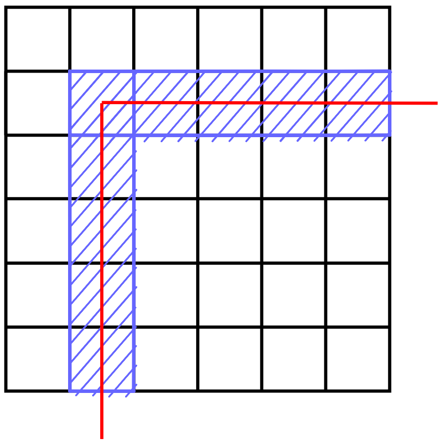


Figure 3.5: Updating a quadrant q recursively causes $O(1)$ nontrivial updates (within the cell, the row and the column containing the vertex of q), and $O(r)$ trivial updates (within the cells that the two edges of q cross).

Constructing a solution. We now describe how to construct an approximately optimal set cover $\mathcal{Q}_{\text{appx}}$ for the current $(X, \mathcal{Q})$ using our data structure $\mathcal{D}$. Denote by opt the size of an optimal set cover for the current $(X, \mathcal{Q})$; we define $\mathsf{opt} = \infty$ if $(X, \mathcal{Q})$ does not have a set cover. Set $\delta = \min\{n, c(r^2 + \varepsilon r^2)/(\varepsilon - \alpha\varepsilon)\}$, where c is a sufficiently large constant. If $\mathsf{opt} \leq \delta$, then we are able to use the algorithm of Lemma 3.11 to compute a μ-approximate set cover solution for $(X, \mathcal{Q})$ in $\widetilde{O}(\delta)$ time. Therefore, we simulate that algorithm within that amount of time. If the algorithm successfully computes a solution, we use it as our $\mathcal{Q}_{\text{appx}}$. Otherwise, we know that $\mathsf{opt} > \delta$. In this case, we construct $\mathcal{Q}_{\text{appx}}$ by combining the solutions maintained by the sub-structures as follows.

Consider the trivial quadrants in $\mathcal{Q}$. There are (up to) four maximal trivial quadrants that left, right, top, bottom intersect the point range $[0,1]^2$, which we denote by $Q_\leftarrow, Q_\rightarrow, Q_\uparrow, Q_\downarrow$, respectively. Let $i^- \in [r]$ (resp., $j^- \in [r]$) be the smallest index such that $R_{i^-} \nsubseteq Q_\uparrow$ (resp., $C_{j^-} \nsubseteq Q_\leftarrow$), and $i^+ \in [r]$ (resp., $j^+ \in [r]$) be the largest index such that $R_{i^+} \nsubseteq Q_\downarrow$ (resp., $C_{j^+} \nsubseteq Q_\rightarrow$). Note that $i^- \leq i^+$, because otherwise $X \subseteq [0,1]^2 \subseteq Q_\uparrow \cup Q_\downarrow$ and thus $\mathsf{opt} \leq 2$ (which contradicts with the fact $\mathsf{opt} > \delta$). For the same reason, $j^- \leq j^+$. We include $Q_\leftarrow, Q_\rightarrow, Q_\uparrow, Q_\downarrow$ in our solution $\mathcal{Q}_{\text{appx}}$. By doing this, all points in R_i (resp., C_j) for $i < i^-$ or $i > i^+$ (resp., $j < j^-$ or $j > j^+$) are covered. The remaining task is to cover the points in the complement A of $Q_\leftarrow \cup Q_\rightarrow \cup Q_\uparrow \cup Q_\downarrow$ in $[0,1]^2$; these points lie in the cells $\square_{i,j}$ for $i^- \leq i \leq i^+$ and $j^- \leq j \leq j^+$.

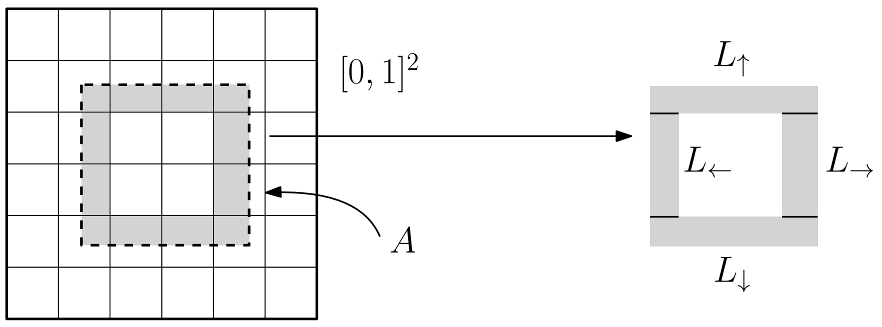


Figure 3.6: The rectangular annulus (the grey area) are partitioned into four rectangles.

We cover the points in A using two collections of quadrants. The first collection covers all points in the cells $\square_{i,j}$ contained in A, i.e., the cells $\square_{i,j}$ for $i^- < i < i^+$ and $j^- < j < j^+$. Specifically, if the cell $\square_{i,j}$ can be covered by a single quadrant $Q \in \mathcal{Q}$, we define $\mathcal{Q}^*_{i,j} = \{Q\}$, otherwise we define $\mathcal{Q}^*_{i,j} \subseteq \mathcal{Q}_{i,j}$ as the $(\mu + \tilde{\varepsilon})$-approximate set cover solution for the sub-instance $(X_{i,j}, \mathcal{Q}_{i,j})$ maintained by $\mathcal{D}_{i,j}$. (If there exists a cell $\square_{i,j}$ for $i^- < i < i^+$ and $j^- < j < j^+$ that is not covered by any single quadrant $Q \in \mathcal{Q}$ and the sub-structure $\mathcal{D}_{i,j}$ tells us that the sub-instance $(X_{i,j}, \mathcal{Q}_{i,j})$ has no solution, then we make a no-solution decision for $(X, \mathcal{Q})$.) We include in our solution $\mathcal{Q}_{\text{appx}}$ all quadrants in $\bigcup_{i=i^-+1}^{i^+-1} \bigcup_{j=j^-+1}^{j^+-1} \mathcal{Q}^*_{i,j}$, which cover the points in $\square_{i,j}$ for $i^- < i < i^+$ and $j^- < j < j^+$.

Now the only points uncovered are those lie in the rectangular annulus, which is the complement of the union of the cells $\square_{i,j} \subseteq A$ in A (see Figure 3.6). We partition this rectangular annulus into four rectangles $L_\uparrow, L_\downarrow, L_\leftarrow, L_\rightarrow$ (again see Figure 3.6), which are contained in $R_{i^-}, R_{i^+}, C_{j^-}, C_{j^+}$, respectively. We obtain a set cover for the points in each of $L_\uparrow, L_\downarrow, L_\leftarrow, L_\rightarrow$ using the corresponding row/column sub-structure as follows. Consider $L_\uparrow$. We temporarily insert the three *virtual* quadrants $Q_\uparrow, Q_\leftarrow, Q_\rightarrow$ to the sub-instance $(X_{i^-,\bullet}, \mathcal{Q}_{i^-,\bullet})$ (these quadrants will be deleted afterwards) and update the sub-structure $\mathcal{D}_{i^-,\bullet}$ so that $\mathcal{D}_{i^-,\bullet}$ now maintains a solution for $(X_{i^-,\bullet}, \mathcal{Q}_{i^-,\bullet} \cup \{Q_\uparrow, Q_\leftarrow, Q_\rightarrow\})$. This solution covers all points in $X_{i^-,\bullet}$. We then remove the quadrants $Q_\uparrow, Q_\leftarrow, Q_\rightarrow$ from the solution (if any of them are used), and the set $\mathcal{Q}^*_\uparrow$ of the remaining quadrants should cover all points in $L_\uparrow$. In a similar way, we can construct sets $\mathcal{Q}^*_\downarrow, \mathcal{Q}^*_\leftarrow, \mathcal{Q}^*_\rightarrow$ that cover the points in $L_\downarrow, L_\leftarrow, L_\rightarrow$, respectively, by using the sub-structures $\mathcal{D}_{i^+,\bullet}, \mathcal{D}_{\bullet,j^-}, \mathcal{D}_{\bullet,j^+}$. (If any of those sub-structures tells us the corresponding sub-instance has no solution, then we make a no-solution decision for $(X, \mathcal{Q})$.) We include in $\mathcal{Q}_{\text{appx}}$ all quadrants in $\mathcal{Q}^* = \mathcal{Q}^*_\uparrow \cup \mathcal{Q}^*_\downarrow \cup \mathcal{Q}^*_\leftarrow \cup \mathcal{Q}^*_\rightarrow$. This completes the construction of $\mathcal{Q}_{\text{appx}}$.

To summarize, we define

$$\mathcal{Q}_{\text{appx}} = \{Q_\uparrow, Q_\downarrow, Q_\leftarrow, Q_\rightarrow\} \sqcup \mathcal{Q}^* \sqcup \left(\bigsqcup_{(i,j)\in P} \mathcal{Q}^*_{i,j} \right), \tag{3.8}$$

where $P = \{(i,j) : i^- < i < i^+, j^- < j < j^+\}$. From the construction, it is easy to verify that $\mathcal{Q}_{\text{appx}}$ is a set cover for $(X, \mathcal{Q})$.

Answering queries to the solution. We show how to store the solution $\mathcal{Q}_{\text{appx}}$ properly so that the desired queries for $\mathcal{Q}_{\text{appx}}$ can be answered efficiently. If $\mathcal{Q}_{\text{appx}}$ is computed using the output-sensitive algorithm of Lemma 3.11, then $|\mathcal{Q}_{\text{appx}}| \leq \delta$ and we have all elements of $\mathcal{Q}_{\text{appx}}$. In this case, we simply build a binary search tree on $\mathcal{Q}_{\text{appx}}$, which can answer the desired queries with the required time costs. On the other hand, if $\mathcal{Q}_{\text{appx}}$ is defined using Equation 3.8, we cannot compute $\mathcal{Q}_{\text{appx}}$ explicitly. Instead, we simply compute the size of $\mathcal{Q}_{\text{appx}}$. We have $|\mathcal{Q}_{\text{appx}}| = 4 + |\mathcal{Q}^*| + \sum_{(i,j)\in P} |\mathcal{Q}^*_{i,j}|$, where $|\mathcal{Q}^*|$ and $|\mathcal{Q}^*_{i,j}|$ can be obtained by querying the sub-structures $\mathcal{D}_{i^-,\bullet}, \mathcal{D}_{i^+,\bullet}, \mathcal{D}_{\bullet,j^-}, \mathcal{D}_{\bullet,j^+}$ and $\mathcal{D}_{i,j}$'s. By storing $|\mathcal{Q}_{\text{appx}}|$, we can answer the size query in $O(1)$ time.

In order to answer membership queries, we need some extra work. The main difficulty is that one quadrant $Q \in \mathcal{Q}^*$ may belong to many $\mathcal{Q}^*_{i,j}$'s, but we cannot afford recursively querying all sub-structures $\mathcal{D}_{i,j}$. To overcome this difficulty, the idea is to store the special quadrants in $\mathcal{Q}^*_{i,j}$'s separately. Recall that $\mathcal{Q}_{i,j}$ consists of all nontrivial quadrants in $\mathcal{Q}$ whose vertices are in $\square_{i,j}$ and four special quadrants $Q_\leftarrow^{\square_{i,j}}, Q_\rightarrow^{\square_{i,j}}, Q_\uparrow^{\square_{i,j}}, Q_\downarrow^{\square_{i,j}}$. We collect all special quadrants in $\mathcal{Q}^*_{i,j}$ for $(i,j) \in P$, the number of which is at most $4|P| = O(r^2)$. We then store these special quadrants in a binary search tree T which can support membership queries. To answer a membership query $Q \in \mathcal{Q}$, we first compute its multiplicity in $\{Q_\uparrow, Q_\downarrow, Q_\leftarrow, Q_\rightarrow\}$ in $O(1)$ time. Then it suffices to compute the multiplicity of Q in $\mathcal{Q}^* \sqcup \bigsqcup_{(i,j)\in P} \mathcal{Q}^*_{i,j}$. Note that although there can be many parts among $\mathcal{Q}^*_\uparrow$, $\mathcal{Q}^*_\downarrow$, $\mathcal{Q}^*_\leftarrow$, $\mathcal{Q}^*_\rightarrow$, or $\mathcal{Q}^*_{i,j}$'s that contain Q, all of them contain Q as a special quadrant except the one corresponding to the region that contains the vertex of Q. So we only need to query T to obtain the multiplicity of Q contained in $\mathcal{Q}^*_{i,j}$'s as a special quadrant, similarly use orthogonal range searching to check whether Q is contained in $L_\uparrow, L_\downarrow, L_\leftarrow$, or $L_\rightarrow$ as a special quadrant in $O(\log n)$ time, and then perform a single recursive query in the sub-structure $\mathcal{D}_{i^-,\bullet}, \mathcal{D}_{i^+,\bullet}, \mathcal{D}_{\bullet,j^-}, \mathcal{D}_{\bullet,j^+}$ or $\mathcal{D}_{i,j}$ for the part that contains the vertex of Q.

Handling reporting queries is easy and is similar to that for dynamic interval set cover presented in Section 3.7. We first report the four quadrants $Q_\uparrow, Q_\downarrow, Q_\leftarrow, Q_\rightarrow$, and then report the quadrants in $\mathcal{Q}^*$ and $\mathcal{Q}^*_{i,j}$'s by recursively querying the sub-structures which maintain nonempty solutions.

Now we analyze the query time. If the solution $\mathcal{Q}_{\text{appx}}$ is computed by the algorithm of Lemma 3.11, then it is stored in a binary search tree and we can answer a size query, a membership query, and a reporting query in $O(1)$ time, $O(\log |\mathcal{Q}_{\text{appx}}|)$ time, and $O(|\mathcal{Q}_{\text{appx}}|)$ time, respectively. So it suffices to consider the case where we construct $\mathcal{Q}_{\text{appx}}$ using Equation 3.8. In this case, answering a size query still takes $O(1)$, because we explicitly compute $|\mathcal{Q}_{\text{appx}}|$. To answer a membership query $Q \in \mathcal{Q}$, we need to do a single recursive query on one sub-structure (the rectangle among $L_\uparrow, L_\downarrow, L_\leftarrow, L_\rightarrow$ or the cell $\square_{i,j}$ containing the vertex of Q can be found in $O(\log r)$ time using the point location data structure $\mathcal{L}$). Besides, we need to query the binary search tree T or the orthogonal range searching structure that checks special quadrants, which takes $O(\log n)$ time. All the other work takes $O(1)$ time. Note that the instances maintained in the sub-structures $\mathcal{D}_{i,\bullet}, \mathcal{D}_{\bullet,j}, \mathcal{D}_{i,j}$ have size $O(n/r)$. So if we use $Q_m(n)$ to denote the time cost for a membership query when the size of the instance is n, we have the recurrence $Q_m(n) = Q_m(O(n/r)) + O(\log n)$, which solves to $Q_m(n) = O(\log_r n \cdot \log n)$. Finally, to answer a reporting query, we first report the elements of $\{Q_\uparrow, Q_\downarrow, Q_\leftarrow, Q_\rightarrow\}$ and recursively query the relevant sub-structures which maintain nonempty solutions. Thus, in the recursion tree, the number of leaves is bounded by $|\mathcal{Q}_{\text{appx}}|$ since at each leaf node we need to report at least one element. Since the height of the recursion tree is $O(\log_r n)$ and at each node of the recursion tree the work can be done in $O(1)$ time, the overall time cost for a reporting query is $O(|\mathcal{Q}_{\text{appx}}| \cdot \log_r n)$.

Correctness. First, we show that $\mathcal{D}$ makes a no-solution decision iff the current instance $(X, \mathcal{Q})$ does not have a feasible set cover. The "if" part is clear because the set $\mathcal{Q}_{\text{appx}}$ we construct is always a set cover for $(X, \mathcal{Q})$. To see the "only if" part, we notice there are two cases that $\mathcal{D}$ can make a no-solution decision. The first case is when the sub-instance $(X_{i,j}, \mathcal{Q}_{i,j})$ has no set cover for some cell $\square_{i,j}$ for $i^- < i < i^+$ and $j^- < j < j^+$ that is not covered by any single quadrant $Q \in \mathcal{Q}$. In this case, there is some point $a \in X_{i,j}$ that cannot be covered by any quadrant in $\mathcal{Q}_{i,j}$. Note that $\mathcal{Q}_{i,j}$ contains all nontrivial quadrants in $\mathcal{Q}$ which partially intersect $\square_{i,j}$ and the intersection is maximal (among all quadrants in $\mathcal{Q}$ that partially intersect $\square_{i,j}$). Therefore, a cannot be covered by any nontrivial quadrant in $\mathcal{Q}$ which partially intersects $\square_{i,j}$ (and all nontrivial quadrants in $\mathcal{Q}$ intersecting $\square_{i,j}$ must intersect $\square_{i,j}$ partially). Also, a cannot be covered by any trivial quadrant in $\mathcal{Q}$ because $a \in \square_{i,j} \subseteq A$. So the no-solution decision made here is correct. The second case is when constructing $\mathcal{Q}^*$. It is easy to see that if any of the sub-structures $\mathcal{D}_{i^-,\bullet}, \mathcal{D}_{i^+,\bullet}, \mathcal{D}_{\bullet,j^-}, \mathcal{D}_{\bullet,j^+}$ reports "no solution", then $(X, \mathcal{Q})$ has no set cover. For example, if $\mathcal{D}_{i^-,\bullet}$ reports "no solution", then the points in $X_{i^-,\bullet}$ cannot be covered by the quadrants in

$\mathcal{Q}_{i^-,\bullet} \cup \{Q_\uparrow, Q_\leftarrow, Q_\rightarrow\}$ and thus the points in $L_\uparrow$ cannot be covered by the quadrants in $\mathcal{Q}$. Therefore, the no-solution decision made here is correct.

Now it suffices to show $\mathcal{Q}_{\text{appx}}$ is a $(\mu+\varepsilon)$-approximate solution for $(X, \mathcal{Q})$. If $\mathcal{Q}_{\text{appx}}$ is constructed by the algorithm of Lemma 3.11, then it is a μ-approximate solution. So suppose $\mathcal{Q}_{\text{appx}}$ is constructed using Equation 3.8. Let opt be the size of an optimal set cover for $(X, \mathcal{Q})$. Our key observation is the following.

Lemma 3.12. $|\mathcal{Q}_{\text{appx}}| \leq (\mu + \tilde{\varepsilon}) \cdot \mathsf{opt} + O(r^2)$.

Proof. Let $\mathcal{Q}_{\text{opt}}$ be an optimal set cover of $(X, \mathcal{Q})$. Define $n_\uparrow, n_\downarrow, n_\leftarrow, n_\rightarrow$ as the number of quadrants in $\mathcal{Q}_{\text{opt}}$ whose vertices are in $L_\uparrow, L_\downarrow, L_\leftarrow, L_\rightarrow$, respectively. Also, define $n_{i,j}$ as the number of quadrants in $\mathcal{Q}_{\text{opt}}$ whose vertices are in $\square_{i,j}$. We have $\mathsf{opt} = |\mathcal{Q}_{\text{opt}}| \geq n_\uparrow + n_\downarrow + n_\leftarrow + n_\rightarrow + \sum_{i=i^-}^{i^+} \sum_{j=j^-}^{j^+} n_{i,j}$. On the other hand, by Equation 3.8, we have

$$\begin{aligned} |\mathcal{Q}_{\text{appx}}| &= 4 + |\mathcal{Q}^*| + \sum_{i=i^-}^{i^+} \sum_{j=j^-}^{j^+} |\mathcal{Q}^*_{i,j}| \\ &= 4 + |\mathcal{Q}^*_\uparrow| + |\mathcal{Q}^*_\downarrow| + |\mathcal{Q}^*_\leftarrow| + |\mathcal{Q}^*_\rightarrow| + \sum_{i=i^-}^{i^+} \sum_{j=j^-}^{j^+} |\mathcal{Q}^*_{i,j}|. \end{aligned} \tag{3.9}$$

We show that $|\mathcal{Q}^*_\uparrow| \leq (\mu + \tilde{\varepsilon}) \cdot (n_\uparrow + 7)$, $|\mathcal{Q}^*_\downarrow| \leq (\mu + \tilde{\varepsilon}) \cdot (n_\downarrow + 7)$, $|\mathcal{Q}^*_\leftarrow| \leq (\mu + \tilde{\varepsilon}) \cdot (n_\leftarrow + 7)$, $|\mathcal{Q}^*_\rightarrow| \leq (\mu + \tilde{\varepsilon}) \cdot (n_\rightarrow + 7)$, and $|\mathcal{Q}^*_{i,j}| \leq (\mu + \tilde{\varepsilon}) \cdot (n_{i,j} + 4)$ for $i^- < i < i^+$ and $j^- < j < j^+$, which implies the inequality in the lemma. All these inequalities are proved similarly, so we only show $|\mathcal{Q}^*_\uparrow| \leq (\mu + \tilde{\varepsilon}) \cdot (n_\uparrow + 7)$ here. Recall that $\mathcal{Q}^*_\uparrow$ is the solution maintained by $\mathcal{D}_{i^-,\bullet}$ for $(X_{i^-,\bullet}, \mathcal{Q}_{i^-,\bullet} \cup \{Q_\uparrow, Q_\leftarrow, Q_\rightarrow\})$, excluding the quadrants $Q_\uparrow, Q_\leftarrow, Q_\rightarrow$. Now we create a set of at most $n_\uparrow + 7$ quadrants in $\mathcal{Q}_{i^-,\bullet} \cup \{Q_\uparrow, Q_\leftarrow, Q_\rightarrow\}$, which consists of all quadrants in $\mathcal{Q}_{\text{opt}}$ whose vertices are in $L_\uparrow$, the (up to) four maximal nontrivial quadrants in $\mathcal{Q}$ that top, bottom, left, right intersect R_{i^-}, and the three quadrants $Q_\uparrow, Q_\leftarrow, Q_\rightarrow$. These $n_\uparrow + 7$ quadrants cover all points in $X_{i^-,\bullet}$, because $\mathcal{Q}_{\text{opt}}$ is a set cover of $(X, \mathcal{Q})$. Since $\mathcal{D}_{i^-,\bullet}$ maintains a $(\mu+\varepsilon)$-approximate solution for $(X_{i^-,\bullet}, \mathcal{Q}_{i^-,\bullet} \cup \{Q_\uparrow, Q_\leftarrow, Q_\rightarrow\})$, we have $|\mathcal{Q}^*_\uparrow| \leq (\mu + \tilde{\varepsilon}) \cdot (n_\uparrow + 7)$. QED.

Recall that we construct $\mathcal{Q}_{\text{appx}}$ using Equation 3.8 only when $\mathsf{opt} \geq \delta \geq c(r^2 + \varepsilon r^2)/(\varepsilon - \alpha\varepsilon)$. By the above lemma and the fact that c is sufficiently large, we have

$$|\mathcal{Q}_{\text{appx}}| - (\mu + \tilde{\varepsilon}) \cdot \mathsf{opt} \leq c' \cdot r^2 \leq (\varepsilon - \alpha\varepsilon)\mathsf{opt} = (\varepsilon - \tilde{\varepsilon})\mathsf{opt}, \tag{3.10}$$

which implies $|\mathcal{Q}_{\text{appx}}| \leq (\mu + \varepsilon) \cdot \mathsf{opt}$.

Update time. To analyze the update time of our data structure $\mathcal{D}$, it suffices to consider the first period (including the first reconstruction). The first period consists of n_0/r operations, where n_0 is the size of the initial $(X, \mathcal{Q})$. The size of $(X, \mathcal{Q})$ during the first period is always in between $(1 - 1/r)n_0$ and $(1 + 1/r)n_0$ and is hence $\Theta(n_0)$ (later we shall choose a super-constant r). We first observe that, excluding the recursive updates for the sub-structures, each update of $\mathcal{D}$ takes $\widetilde{O}(r^2/(\varepsilon - \alpha\varepsilon))$ (amortized) time. Updating the support data structures can be done in $\widetilde{O}(1)$ time. When constructing the solution $\mathcal{Q}_{\text{appx}}$, we need to simulate the algorithm of Lemma 3.11 within $\widetilde{O}(\delta) = \widetilde{O}(r^2/(\varepsilon - \alpha\varepsilon))$ time. If $\mathcal{Q}_{\text{appx}}$ is defined using Equation 3.8, we need to do some extra work. First, we need to obtain the quadrants $Q_\uparrow, Q_\downarrow, Q_\leftarrow, Q_\rightarrow$, which can be done in $\widetilde{O}(1)$ time using the support data structure $\mathcal{B}$. Then we need to compute $|\mathcal{Q}^*|$, which involves $O(1)$ size queries to the sub-structures $\mathcal{D}_{i^-,\bullet}, \mathcal{D}_{i^+,\bullet}, \mathcal{D}_{\bullet,j^-}, \mathcal{D}_{\bullet,j^+}$ and hence takes $O(1)$ time. Finally, we need to compute $|\mathcal{Q}^*_{i,j}|$ and retrieve the special quadrants in $\mathcal{Q}^*_{i,j}$ for all $(i, j) \in P$ (and build the query structure for these special quadrants). Checking whether a cell $\square_{i,j}$ is coverable can be done in $\widetilde{O}(1)$ time using the support data structure $\mathcal{C}$. After knowing whether each cell is coverable, computing $|\mathcal{Q}^*_{i,j}|$ and retrieving the special quadrants can be done via $O(r^2)$ size and orthogonal range searching queries to the sub-structures $\mathcal{D}_{i,j}$, which takes $\widetilde{O}(r^2)$ time.

The reconstruction of $\mathcal{D}$ takes $\widetilde{O}(2^{O(\log_r n)} \cdot r)$ amortized time, because the time cost of the (first) reconstruction is $\widetilde{O}(2^{O(\log_r n_1)} \cdot n_1)$, i.e., $\widetilde{O}(2^{O(\log_r n_0)} \cdot n_0)$, while the first period consists of $O(n_0/r)$ operations.

Next, we consider the recursive updates of the sub-structures. Similar to the analysis of our dynamic interval set cover data structure, we distinguish three types of updates according to the current operation. The first type is *point update*, which is caused by the insertion/deletion of a point in X. The second type is *trivial quadrant update* (or *trivial update* for short), which is caused by the insertion/deletion of a trivial quadrant in $\mathcal{Q}$ (recall that a quadrant is trivial if its vertex is outside the point range $[0, 1]^2$ of $(X, \mathcal{Q})$). The third type is *nontrivial quadrant update* (or *nontrivial update* for short), which is caused by the insertion/deletion of a nontrivial quadrant in $\mathcal{Q}$.

We first consider the recursive updates required for all three types of updates. Recall that when constructing the solution $\mathcal{Q}_{\text{appx}}$ using Equation 3.8, we need to temporarily insert some virtual quadrants to $\mathcal{Q}_{i^-,\bullet}, \mathcal{Q}_{i^+,\bullet}, \mathcal{Q}_{\bullet,j^-}, \mathcal{Q}_{\bullet,j^+}$ (and delete them afterwards). This involves a constant number of recursive updates, which are all trivial updates because the virtual quadrants inserted are all trivial. Besides these recursive updates, a point update requires three recursive (point) updates, because the insertion/deletion of a point in X changes one cell sub-instance, one row sub-instance, and one column sub-instance. A trivial update does not require any other recursive updates, because the insertion/deletion of trivial quadrant in $\mathcal{Q}$ does not change any sub-instance.

Finally, we consider a nontrivial update, where our algorithm is more involved. Let Q be the nontrivial quadrant inserted/deleted and suppose the vertex of Q is contained in the cell $\square_{i,j}$. Then we may need to update the cell sub-structures $\mathcal{D}_{i,1}, \dots, \mathcal{D}_{i,r}$ and $\mathcal{D}_{1,j}, \dots, \mathcal{D}_{r,j}$, in which the update of $\mathcal{D}_{i,j}$ is a nontrivial update and the others are all trivial updates since the vertex of Q is outside the point ranges of all cell sub-instances except $(X_{i,j}, \mathcal{Q}_{i,j})$. Also, we need to update the row (resp., column) sub-structures, in which the update of $\mathcal{D}_{i,\bullet}$ (resp., $\mathcal{D}_{\bullet,j}$) are nontrivial updates and the others are all trivial updates.

To summarize, a point update requires $O(1)$ recursive point updates and $O(1)$ recursive trivial updates, a trivial update requires $O(1)$ recursive trivial updates, and a nontrivial update requires $O(1)$ recursive nontrivial updates and $O(r)$ recursive trivial updates (as shown in Fig. 3.5).

The depth of the recursion is $O(\log_r n)$. If we set $\alpha = 1 - 1/\log_r n$, the approximation factor parameter is $\Theta(\varepsilon)$ in any level of the recurrence. Let $U(n)$, $U_1(n)$, $U_2(n)$ denote the time costs of a point update, a trivial update, and a nontrivial update, respectively, when the size of the current instance is n. Then we have the recurrences

$$\begin{aligned} U(n) &= O(1) \cdot U(O(n/r)) + O(1) \cdot U_1(O(n/r)) + \widetilde{O}(r^2/\varepsilon), \\ U_1(n) &= O(1) \cdot U_1(O(n/r)) + \widetilde{O}(r^2/\varepsilon), \\ U_2(n) &= O(1) \cdot U_2(O(n/r)) + O(r) \cdot U_1(O(n/r)) + \widetilde{O}(r^2/\varepsilon). \end{aligned} \tag{3.11}$$

The recurrence for $U_1(n)$ solves to $U_1(n) = \widetilde{O}((r^2/\varepsilon) \cdot 2^{O(\log_r n)})$. Based on this, we further solve the recurrences for $U(n)$ and $U_2(n)$, and obtain $U(n) = \widetilde{O}((r^2/\varepsilon) \cdot 2^{O(\log_r n)})$ and $U_2(n) = \widetilde{O}((r^3/\varepsilon) \cdot 2^{O(\log_r n)})$. Setting $r = 2^{O(\sqrt{\log n})}$, the amortized update time of $\mathcal{D}$ is then $\widetilde{O}((r^3/\varepsilon) \cdot 2^{O(\log_r n)}) = 2^{O(\sqrt{\log n})}$.

The time cost of a membership query is $O(\log_r n \cdot \log n) = O(\log^{3/2} n)$, while the time cost of a reporting query is $O(|\mathcal{Q}_{\text{appx}}| \cdot \log_r n) = O(|\mathcal{Q}_{\text{appx}}| \cdot \sqrt{\log n}) = O(\mathsf{opt}\sqrt{\log n})$. Also, the construction time is $\widetilde{O}(2^{O(\log_r n)} \cdot n) = 2^{O(\sqrt{\log n})} \cdot n$. We conclude the following.

Theorem 3.5. There exists a dynamic data structure for $O(1)$-approximate unweighted unit-square set cover with $2^{O(\sqrt{\log n})}$ amortized update time and $2^{O(\sqrt{\log n})} \cdot n$ construction time, which can answer size, membership, and reporting queries in $O(1)$, $O(\log^{3/2} n)$, and $O(\mathsf{opt}\sqrt{\log n})$ time, respectively, where n is the size of the instance and opt is the size of the optimal solution.

3.9 IMPROVING UNWEIGHTED SQUARES

In this section, we present a more efficient data structure for $O(1)$-approximate dynamic square set cover, improving our previous near-$O(n^{2/3})$-time result in Sec. 3.4.

Let $(X, \mathcal{S})$ be a dynamic (unweighted) square set cover instance where X is the set of points in $\mathbb{R}^2$ and $\mathcal{S}$ is the set of squares. Let $n = |X| + |\mathcal{S}|$ denote the total number of points and squares.

3.9.1 Algorithm for small opt

Based on the randomized multiplicative weight update (MWU) method [14, 46, 89], we have provided an $O(1)$-approximation algorithm with $\widetilde{O}(\mathsf{opt}^2)$ query time and $\widetilde{O}(1)$ update time in Lemma 3.1, assuming the points and objects have been preprocessed in standard range searching data structures. When opt is small, this algorithm runs in sublinear time. We will use this algorithm as a subroutine later.

3.9.2 Algorithm for large opt

When opt is large, we can afford a larger additive error. Our previous approach in Sec. 3.4 utilized this observation and used a quadtree to partition the problem into subproblems, paying $O(1)$ additive error per subproblem when combining the solutions. We refine that approach and further improve the update time to near $O(n^{1/2})$.

Previous data structures. Our data structure is based on the previous data structure in Sec. 3.4, so we will first briefly redescribe that approach, and then introduce our new ideas. For simplicity, assume all coordinates are integers bounded by $U = \text{poly}(n)$. This assumption can be removed, namely by using the BBD tree as we have explained in Sec. 3.4. We also assume an $O(1)$-approximation t of opt is known, by running our algorithm for all possible guesses $t = 2^i$ in parallel (our algorithm is able to detect whether the guess is wrong).

The key idea in our previous method is to construct a standard quadtree, starting with a bounding square cell and recursively divide into four square cells. We stop subdividing when a leaf cell Γ has size at most b, for a parameter b to be set later, where the size of Γ is defined as the total number of points in X and vertices of squares in $\mathcal{S}$ that are inside Γ. This yields $O(\frac{n}{b})$ cells per level, and thus $O(\frac{n}{b} \log U)$ cells in total.

Since the quadtree cells are also squares, there are only two types of intersections between a quadtree cell Γ and an input square in $\mathcal{S}$. Call a square s *short* in the cell Γ, if at least one of its vertices is in Γ, otherwise *long*, as shown in Fig. 3.4. A key observation is that it suffices to keep (at most) 4 "maximal" long squares in each cell, since their union covers the union of all long squares of the cell. We use $\mathcal{M}_\Gamma$ to denote the set of maximal long squares in the cell Γ.

Now we only need to maintain a data structure $\mathcal{D}_\Gamma$ for each leaf cell Γ, that supports the following type of query:

> Given any query rectangle r in Γ, compute an $O(1)$-approximate set cover solution for the points in $X \cap r$, using only the short squares in Γ.

To compute an approximate set cover solution for Γ, it suffices to first include the 4 maximal long squares $\mathcal{M}_\Gamma$ of Γ in the approximate solution (thus paying $O(1)$ additive error), and then query for an $O(1)$-approximate solution $\mathcal{S}_{\text{appx}_\Gamma}$ in the complement region $\square_\Gamma$ of $\mathcal{M}_\Gamma$, which is a rectangle as shown in Figure 3.7, using only the short squares. $\mathcal{D}_\Gamma$ is implemented using 2D range trees [12, 100] with branching factor $a = b^\delta$ built on the points within Γ, where δ is a sufficiently small constant. In this way, we form a set of canonical rectangles with total size $O(a^{O(1)}b(\log_a b)^2) = O(b^{1+O(\delta)})$, such that any query rectangle in the cell Γ can be decomposed into $O((\log_a b)^2) = (\frac{1}{\delta})^{O(1)} = O(1)$ canonical rectangles. Then $\mathcal{S}_{\text{appx}_\Gamma}$ is obtained by taking the union of the $O(1)$-approximate solutions in the $O(1)$ canonical rectangles that $\square_\Gamma$ decomposes to, and doing this only loses a constant factor.

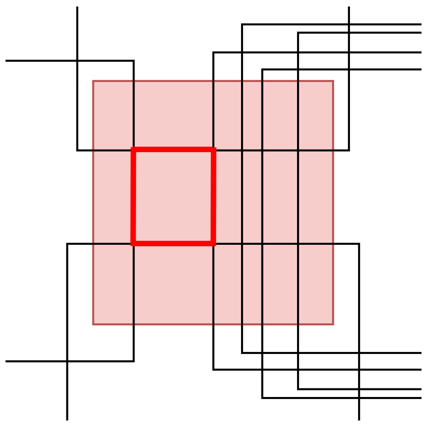


Figure 3.7: The complement region of the (at most) 4 maximal long squares of a cell.

The global approximate solution $\mathcal{S}_{\text{appx}}$ is formed by taking the union of the approximate solutions in each leaf cell Γ, which is $\mathcal{S}_{\text{appx}_\Gamma}$ plus the at most 4 maximal long squares of Γ, i.e., we let $\mathcal{S}_{\text{appx}} = \bigsqcup_\Gamma(\mathcal{S}_{\text{appx}_\Gamma} \sqcup \mathcal{M}_\Gamma)$.

To analyze the approximation factor, let $\mathcal{S}_{\mathsf{opt}_\Gamma}$ contain the squares in the optimal solution that are short in the cell Γ. We have $\sum_\Gamma |\mathcal{S}_{\mathsf{opt}_\Gamma}| \leq 4 \cdot \mathsf{opt}$, since an input square is short in at most 4 leaf cells containing its 4 vertices. The size of our approximate solution $\mathcal{S}_{\text{appx}}$ can be upper-bounded as follows:

$$|\mathcal{S}_{\text{appx}}| \leq \sum_{\Gamma} \left(|\mathcal{S}_{\text{appx}_\Gamma}| + |\mathcal{M}_\Gamma|\right) \leq \sum_{\Gamma} \left(O\left(|\mathcal{S}_{\mathsf{opt}_\Gamma}|\right) + 4\right) \leq O(\mathsf{opt}) + \widetilde{O}\left(\frac{n}{b}\right). \tag{3.12}$$

As long as we set $b = \tilde{\Omega}(\frac{n}{\mathsf{opt}})$, this is an $O(1)$-approximation.

New approach. Now we describe the parts that we will change. In our previous algorithm, an $O(1)$-approximate solution within each canonical rectangle in each cell is maintained, using the static $O(1)$-approximate set cover algorithm with near-linear running time [14, 71]. Our previous observation was that for a canonical rectangle r with size b_i, although there may exist a lot of squares that cut across r, it suffices to keep only $O(b_i)$ "maximal" long squares with respect to r among them, which can be found in $\widetilde{O}(b_i)$ time using range searching. So the running time for the static algorithm is $\widetilde{O}(b_i)$.

To further improve the update time, our new idea is to classify the canonical rectangles into two categories, based on their sizes. Call a canonical rectangle *heavy*, if its size exceeds g, otherwise *light*, where g is a parameter to be set later.

For each heavy canonical rectangle r with size $g' \geq g$, we maintain another level of subquadtree, using the previous algorithm as stated before, which subdivides it into $\widetilde{O}(\lambda)$ subcells each with size $O(\frac{g'}{\lambda})$, where λ is a parameter to be set later. For each subcell Λ in the subquadtree, maintain the set $\mathcal{M}_\Lambda$ of the at most 4 maximal long squares, and an $O(1)$-approximate set cover solution $\mathcal{S}_{\text{appx}_\Lambda}$ for the complement region.

For the light canonical rectangles, we don't maintain the approximate solution, but rather choose to compute it from scratch during the query.

For each cell Γ, the data structure $\mathcal{D}_\Gamma$ can be constructed in $O(b^{1+O(\delta)})$ time, since the total size of the heavy canonical rectangles in the cell is at most $O(b^{1+O(\delta)})$, and the static approximate set cover algorithm runs in near-linear time.

Update. When we insert or delete a square s, for each leaf cell Γ that contains a vertex of s, we update the subquadtree for each heavy canonical rectangle in Γ. For a heavy canonical rectangle with size $g' \geq g$, this takes $O((\lambda + \frac{g'}{\lambda}) \cdot g'^{O(\delta)})$ time (as analyzed in the previous paper). Summing over all heavy canonical rectangles, the total time is at most $O((\lambda \cdot \frac{b}{g} + \frac{b}{\lambda}) \cdot b^{O(\delta)})$, as the total size of the (heavy) canonical rectangles is bounded by $O(b^{1+O(\delta)})$. We also update the set of maximal long squares $\mathcal{M}_\Gamma$ for each leaf cell Γ, in $\widetilde{O}(\frac{n}{b})$ time.

When we insert or delete a point p, we update the data structure $\mathcal{D}_\Gamma$ for the leaf cell Γ containing p. For each canonical rectangle r in Γ that contains p, we may need to update $O(1)$ maximal long squares that cut across r, which takes $O((\lambda + \frac{b}{\lambda}) \cdot b^{O(\delta)})$ time for the heavy ones, and $O(b^{O(\delta)})$ time for the light ones.

Query. When we perform a query, we need to compute the approximate solution $\mathcal{S}_{\text{appx}_\Gamma}$ in the complement region $\square_\Gamma$ of the maximal long squares for each cell Γ. The region $\square_\Gamma$ can be decomposed into $O(1)$ canonical rectangles. For each light canonical rectangle r, let opt_i denote the size of the optimal solution, we compute an $O(1)$-approximate solution from scratch, using either the small opt algorithm as described earlier in Sec. 3.4.1 in $\widetilde{O}(\mathsf{opt}_i^2)$ time, or the static algorithm [14, 71] in $\widetilde{O}(g)$ time (since the size of r is at most g), whichever is faster (by running them in parallel). For each heavy canonical rectangle, the size of the precomputed approximate solution (which is implicitly represented by the subquadtree) can be retrieved in $O(1)$ time, but it has additive error $O(\lambda)$. So if the result is $O(\lambda)$, we need to recompute an $O(1)$-approximate solution from scratch, using the small opt algorithm in $\widetilde{O}(\mathsf{opt}_i^2)$ time, which is bounded by $\widetilde{O}(\mathsf{opt}_i \cdot \lambda)$ since $\mathsf{opt}_i = O(\lambda)$.

The total query time is

$$\widetilde{O}\left(\sum_i \min\{\mathsf{opt}_i^2, g\} + \sum_i \mathsf{opt}_i \cdot \lambda + \frac{n}{b}\right), \tag{3.13}$$

which is at most $\widetilde{O}(\mathsf{opt} \cdot \sqrt{g} + \mathsf{opt} \cdot \lambda + \frac{n}{b})$, since $\sum_i \mathsf{opt}_i = \mathsf{opt}$.

To balance the query and update times, when $\mathsf{opt} \le \sqrt{n}$, set $b = \frac{n}{\mathsf{opt}}$, $g = \frac{n}{\mathsf{opt}^2}$ and $\lambda = \frac{\sqrt{n}}{\mathsf{opt}}$ (note that the requirements $b \ge g \ge \lambda \ge 1$ and $b = \tilde{\Omega}(\frac{n}{\mathsf{opt}})$ are satisfied); both the query and update time are $O(n^{1/2+\delta})$—interesting, we get the same bound uniformly for all the terms. When $\mathsf{opt} > \sqrt{n}$, our previous algorithm already obtains $O(n^{1/2+\delta})$ query and update time, by setting $b = \sqrt{n}$.

The actual solution $\mathcal{S}_{\text{appx}}$ can be reported by taking the union of all solutions in the leaf cells of the quadtree, which takes $\widetilde{O}(\mathsf{opt})$ time.

Theorem 3.6. There exists a dynamic data structure for $O(1)$-approximate unweighted square set cover with $O(n^{1/2+\delta})$ query and update time and $O(n^{1+\delta})$ construction time w.h.p., for any constant $\delta > 0$, where n is the size of the instance.

3.10 UNWEIGHTED 2D HALFPLANES

In this section, we present a data structure for $O(1)$-approximate dynamic 2D halfplane set cover. Previously in Sec 3.5, we have provided a dynamic set cover structure for 3D halfspaces with $O(n^{12/13+\delta})$ update time, which clearly also holds for 2D halfplanes, but that scheme is unable to actually find a set cover—it only reports its size. This is because our previous idea for the "large opt" case is based on estimating the size of the solution by summing over a small random sample of the terms. Here we not only improve the update time, but can also find the approximate set cover solution.

Let $(X, \mathcal{H})$ be a dynamic (unweighted) 2D halfplanes set cover instance where X is a set of points in $\mathbb{R}^2$ and $\mathcal{H}$ is a set of halfplanes, with $m = |X|$ and $n = |\mathcal{H}|$. We use $N = n + m$ to denote the global upper bound on the instance size.

3.10.1 Algorithm for small opt

We first note that there is an algorithm that is efficient when opt is small, which will be used later as a subroutine. The idea is to modify the small opt algorithm for squares in Sec. 3.4.1, which is based on an efficient implementation of the randomized multiplicative weight update (MWU) method [14, 46, 71, 89], using ($\leq b$)-levels and various geometric data structures. Here we note the changes that we make in order to work for 2D halfplanes.

To efficiently implement the MWU algorithm on $(X, \mathcal{H})$ as shown in Alg. 3.1, we need to solve two subproblems: 1) finding a low-depth point p, and 2) weighted range sampling.

Finding a low-depth point. Let $b := \frac{c_0}{2}\log(n+m)$ where c_0 is a sufficiently large constant. To find a low-depth point in line 5, we compute (from scratch) the ($\leq b$)-level $\mathcal{L}_{\leq b}(R)$ of R, which is the collection of all cells in the arrangement of the halfplanes in R of depth at most b. It is known that $\mathcal{L}_{\leq b}(R)$ has $O(|R|b)$ cells (after triangulation) and can be constructed in $\widetilde{O}(|R|b)$ time [92], which is $\widetilde{O}(t)$.

To find a point $p \in X$ with depth in R at most b, we perform a triangle range query for each (triangulated) cell of $\mathcal{L}_{\leq b}(R)$, to test if the cell contains a point $p \in X$. It is known that we can construct a 2D triangle range searching structure $\tilde{\mathcal{D}}$ [165] on the point set X, with $O(\frac{m^{1/2+\delta}}{z^{1/2}})$ query time for emptiness/counting/sampling and $\widetilde{O}(z)$ insertion/deletion time, for a given trade-off parameter $z \in [1, m]$. As there are $\widetilde{O}(t)$ multiplicity-doubling steps, the total cost is $\widetilde{O}(t^2 \cdot \frac{m^{1/2+\delta}}{z^{1/2}})$.

Weighted range sampling. The algorithm is similar to the previous part, but this time we work in the dual. We use h^* to denote the dual point of a halfplane h, and p^* denote the dual halfplane of a point p.

Let Q be the set of all points p for which we have performed multiplicity-doubling steps thus far. Note that $|Q| = \widetilde{O}(t)$. Each time we perform a multiplicity-doubling step, we compute (from scratch) the $(\le b)$-level $\mathcal{L}_{\le b}(Q^*)$. The multiplicity of a halfplane $h \in \mathcal{H}$ is equal to $2^{\text{depth of } h^* \text{ in } Q^*}$, and all dual points h^* in a cell of $\mathcal{L}_{\le b}(Q^*)$ share the same multiplicity. It is known that the multiplicities are bounded by $(n+m)^{O(1)}$, so each h^* is covered by $\mathcal{L}_{\le b}(Q^*)$.

To generate a multiplicity-weighted sample of the halfplanes containing p for line 8, after p has been inserted to Q, we examine all cells of $\mathcal{L}_{\le b}(Q^*)$ contained in p^*. For each such cell γ, we use triangle range counting to compute its size, using a 2D triangle range searching structure $\tilde{\mathcal{D}}^*$ built on the dual point set $\mathcal{H}^*$ as described before. Knowing the sizes and multiplicities for all such $\widetilde{O}(t)$ cells, we can then generate the weighted sample in time $O(\frac{n^{1/2+\delta}}{z^{1/2}})$ times the size of the sample, again using the data structure $\tilde{\mathcal{D}}^*$. The random sample R in line 4 with size $\widetilde{O}(t)$ is generated similarly.

As we perform $\widetilde{O}(t^2)$ triangle range counting queries, and the total size of the samples is $\widetilde{O}(t)$, the total cost is $\widetilde{O}(t^2 \cdot \frac{n^{1/2+\delta}}{z^{1/2}} + t \cdot \frac{n^{1/2+\delta}}{z^{1/2}}) = \widetilde{O}(t^2 \cdot \frac{n^{1/2+\delta}}{z^{1/2}})$.

Lemma 3.13. There exists a data structure for the dynamic set cover problem for $O(m)$ points and $O(n)$ 2D halfplanes that supports updates in $\widetilde{O}(z)$ time and can find an $O(1)$-approximate solution w.h.p. in $\widetilde{O}(\mathsf{opt}^2 \cdot \frac{(n+m)^{1/2+\delta}}{z^{1/2}})$ time, for any constant $\delta > 0$ and trade-off parameter $z \in [1, n+m]$. The data structure can be constructed in $\widetilde{O}((n+m)\cdot z)$ time.

Alternative algorithm. As an alternative to implement the small opt algorithm, we can also slightly modify the small opt algorithm in Sec. 3.5.1 for halfspaces in 3D (which uses partition trees), and obtain an algorithm for 2D halfplanes. The only difference is that we use partition trees in 2D instead of 3D.

Lemma 3.14. There exists a data structure for the dynamic set cover problem for $O(m)$ points and $O(n)$ 2D halfplanes that supports updates in $\widetilde{O}(1)$ time and can find an $O(1)$-approximate solution w.h.p. in $\widetilde{O}(\mathsf{opt} \cdot (n+m)^{1/2+\delta})$ time, for any constant $\delta > 0$. The data structure can be constructed in $\widetilde{O}(n+m)$ time.

3.10.2 Main algorithm

Here we first present a solution that only supports halfplane insertions and deletions as well as point deletions; the ways to support point insertions are more technical, and will be explained in the last part.

Data structures. To construct the data structure, our idea is to recursively apply Matoušek's Partition Theorem [165] as stated below, and decompose the problem into subproblems.

Theorem 3.7 (Matoušek's Partition Theorem)**.** Given a set X of m points in $\mathbb{R}^2$, for any positive integer $b \leq m$, we can partition X into b subsets X_i each with size $O(\frac{m}{b})$ and find b disjoint triangular cells $\Delta_i \supset X_i$, where each cell is a triangle, such that any halfplane crosses (i.e., the boundary intersects) $O(\sqrt{b})$ cells. The partition can be constructed in $O(m^{1+\delta})$ time.

The original version of Matoušek's theorem does not guarantee disjointness of cells, but a later version by Chan [63] does.

Intuitively, with subproblems defined with this partition, any inserted/deleted halfplane h only affects a small fraction of the subproblems. More precisely, we use Theorem 3.7 to partition the set of points X into b disjoint cells $\Delta_1, \dots, \Delta_b$, each containing a subset X_i of $O(\frac{m}{b})$ points, where b is a parameter to be set later. Any halfplane h will cross $O(\sqrt{b})$ cells, so each cell is crossed by $O(\frac{n \cdot \sqrt{b}}{b}) = O(\frac{n}{\sqrt{b}})$ halfplanes in $\mathcal{H}$ on average. Call a cell *good* if it crosses $\leq g \cdot \frac{n}{\sqrt{b}}$ halfplanes in $\mathcal{H}$ (for a sufficiently large constant g), otherwise *bad*. The halfplanes in $\mathcal{H}$ cross $O(n \cdot \sqrt{b})$ cells in total, so the total number of bad cells is $O(\frac{n\sqrt{b}}{gn/\sqrt{b}}) = O(\frac{b}{g})$.

We construct the standard range searching data structures $\tilde{\mathcal{D}}$ and $\tilde{\mathcal{D}}^*$ required by the small opt algorithm in Section 3.10.1 on the problem instance $(X, \mathcal{H})$ in $\widetilde{O}((n+m) \cdot z)$ time, and maintain an (implicit) $O(1)$-approximate solution $\mathcal{H}_{\text{appx}}$ for $(X, \mathcal{H})$. For each good cell Δ_i, let $\mathcal{H}_i$ be the set of halfplanes crossing Δ_i, we recursively construct a data structure $\mathcal{D}_i$ for the subproblem $(X_i, \mathcal{H}_i)$. If there exists a halfplane that completely contains Δ_i, then record any one of them. We also construct a data structure $\mathcal{D}_{\text{bad}}$ for the subproblem $(X_{\text{bad}}, \mathcal{H})$ where $X_{\text{bad}} = \bigcup_{i\text{: cell } i \text{ bad}} X_i$, which contains all points in the union of the bad cells. In the worst case, the union of bad cells may cross all halfplanes in $\mathcal{H}$, so the number of halfplanes in the subproblem will not necessarily decrease. However the total number of points decreases by a constant fraction, since $|X_{\text{bad}}| = \sum_{i\text{: cell } i \text{ bad}} |X_i| \leq O(\frac{b}{g} \cdot \frac{m}{b}) = O(\frac{m}{g})$.

The construction time satisfies the recurrence

$$T\left(m,n\right) = b \cdot T\left(O\left(\frac{m}{b}\right), \frac{gn}{\sqrt{b}}\right) + T\left(O\left(\frac{m}{g}\right), n\right) + \widetilde{O}(m+nb) + \widetilde{O}((n+m)\cdot z). \quad (3.14)$$

Set $b = N^{\delta_0}$ where $\delta_0 > 0$ is a sufficiently small constant, and set $z = (n+m)^{1/3}$ to later balance the terms in the update time. For simplicity of analysis we assume that initially we have $m \le n$, and then this condition holds for all subproblems as we recurse. The running time is dominated by the costs at the lowest level, so the recurrence solves to $T(m,n) = \widetilde{O}(m \cdot (\frac{n}{\sqrt{m}})^{4/3}) = O(m^{1/3}n^{4/3} \cdot N^{O(\delta)})$.

Update. When we insert or delete a halfplane h, we recurse in the $O(\sqrt{b})$ good cells $\Delta_1, \ldots, \Delta_\tau$ crossed by h, in order to recompute the approximate solutions $\mathcal{H}_{\text{appx}}(X_i)$ for $1 \le i \le \tau$. We also recurse in the subproblem $(X_{\text{bad}}, \mathcal{H})$ containing the union of bad cells, and recompute the approximate solution $\mathcal{H}_{\text{appx}}(X_{\text{bad}})$. We update the range searching data structure $\tilde{\mathcal{D}}^*$ required by the small opt algorithm, using $\widetilde{O}(z)$ time.

To reconstruct the approximate solution $\mathcal{H}_{\text{appx}}$ we proceed as follows. If $\mathsf{opt} \triangleq |\mathcal{H}_{\mathsf{opt}}| \le c_2 \cdot b \log N$ for a sufficiently large constant c_2, we use the $\widetilde{O}(\mathsf{opt}^2 \cdot \frac{(n+m)^{1/2+\delta}}{z^{1/2}})$ time algorithm for small opt (Lemma 3.13) to recompute $\mathcal{H}_{\text{appx}}$ from scratch. The condition $\mathsf{opt} \overset{?}{\le} c_2 \cdot b \log N$ can be tested by the small opt algorithm itself. Otherwise $\mathsf{opt} > c_2 \cdot b \log N$, and we can afford a larger additive error, so we return the union of the $O(1)$-approximate solutions for the good cells X_i and the union of the bad cells X_{bad}, i.e., let $\mathcal{H}_{\text{appx}} = \bigsqcup_{i\text{: cell } i \text{ good}} \mathcal{H}_{\text{appx}}(X_i) \sqcup \mathcal{H}_{\text{appx}}(X_{\text{bad}})$, which is stored implicitly. (One special case is when there exists a halfplane h that contains the cell Δ_i, we include h instead of $\mathcal{H}_{\text{appx}}(X_i)$ in the solution $\mathcal{H}_{\text{appx}}$.)

We rebuild the entire data structure after every $g \cdot \frac{n}{\sqrt{b}}$ halfplane updates, so that good cells will not become bad. The amortized cost per update is $\frac{T(m,n)}{gn/\sqrt{b}} + \widetilde{O}(b^2 \cdot \frac{(n+m)^{1/2+\delta}}{z^{1/2}}) + \widetilde{O}(z) = O(m^{1/3}n^{1/3} \cdot N^{O(\delta)})$.

When we delete a point p, if p is contained in a good cell Δ_i, we recurse in the subproblem within Δ_i in order to recompute the approximate solution $\mathcal{H}_{\text{appx}}(X_i)$. Otherwise p is contained in a bad cell, so we recurse in the subproblem $(X_{\text{bad}}, \mathcal{H})$ and recompute the approximate solution $\mathcal{H}_{\text{appx}}(X_{\text{bad}})$. Then we recompute $\mathcal{H}_{\text{appx}}$, using the procedure described above. We update the range searching data structure $\tilde{\mathcal{D}}$ required by the small opt algorithm, using $\widetilde{O}(z)$ time.

Let $U(m,n)$ denote the update time for an instance with m points and n halfplanes. Since point deletions are easier, we mainly focus on halfplane updates. It satisfies the recurrence

$$U\left(m,n\right)=O\left(\sqrt{b}\right)\cdot U\left(O\left(\frac{m}{b}\right),\frac{gn}{\sqrt{b}}\right)+U\left(O\left(\frac{m}{g}\right),n\right)+O\left(m^{1/3}n^{1/3}\cdot N^{O(\delta)}\right),\quad(3.15)$$

which solves to $U(m,n)=O(m^{1/3}n^{1/3}\cdot N^{O(\delta)})$.

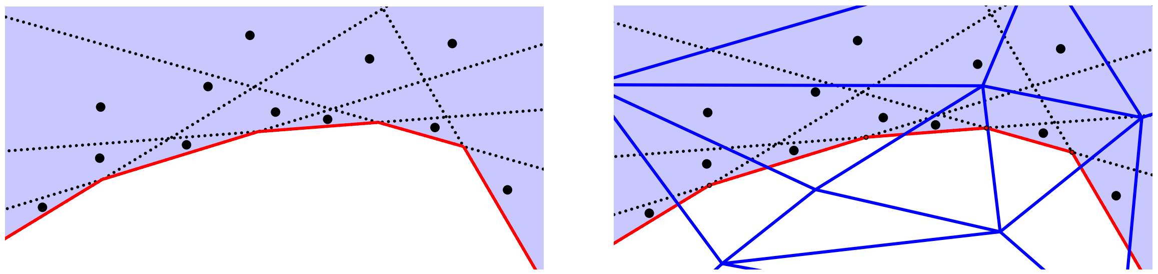


Figure 3.8: The boundary of the optimal solution must form a convex chain (see left). For simplicity we only include the upper halfplanes; the lower halfplanes are similar. Each triangular cell of the partition only intersects the convex chain $O(1)$ times (see right).

Approximation factor analysis. The key observation here is that the boundary of the optimal solution $\mathcal{H}_{\mathsf{opt}}$ for $(X,\mathcal{H})$ must form a convex chain, as shown in Figure 3.8. As each cell is a triangle, it can only intersect the optimal convex chain $O(1)$ times, thus the b disjoint cells $\Delta_1,\ldots,\Delta_b$ will partition $\mathcal{H}_{\mathsf{opt}}$ into $O(b)$ disjoint pieces. (Unfortunately, such nice property does not hold in 3D.) If we take the union of the optimal solutions in all the cells, the additive error is at most $O(b)$, i.e., we have

$$\sum_{1\le i\le b}|\mathcal{H}_{\mathsf{opt}}(X_i)|\le\sum_{1\le i\le b}(|\mathcal{H}_{\mathsf{opt}}\cap\Delta_i|+O(1))\le|\mathcal{H}_{\mathsf{opt}}|+O(b).\quad(3.16)$$

Similarly, we have

$$\sum_{i:\ \text{cell } i \text{ good}}|\mathcal{H}_{\mathsf{opt}}(X_i)|+|\mathcal{H}_{\mathsf{opt}}(X_{\text{bad}})|\ \le\ |\mathcal{H}_{\mathsf{opt}}|+c_1b\quad(3.17)$$

for some constant c_1.

Let c_0 denote the constant multiplicative error factor guaranteed by the small opt algorithm. Suppose that for some constant multiplicative error factor $c\ge\max\{c_0,c_1\}$, we have $|\mathcal{H}_{\text{appx}}(X_i)|\le c\cdot|\mathcal{H}_{\mathsf{opt}}(X_i)|$ for all good cells i, and also $|\mathcal{H}_{\text{appx}}(X_{\text{bad}})|\le c\cdot|\mathcal{H}_{\mathsf{opt}}(X_{\text{bad}})|$. Then if $\mathsf{opt}>c_2\cdot b\log N$, we set $\mathcal{H}_{\text{appx}}$ to be the union of the approximate solutions in the subproblems, and obtain

$$\begin{aligned} |\mathcal{H}_{\text{appx}}| &\leq \sum_{i:\text{ cell } i \text{ good}} |\mathcal{H}_{\text{appx}}(X_i)| + |\mathcal{H}_{\text{appx}}(X_{\text{bad}})| && (3.18) \\ &\leq c \cdot \left(\sum_{i:\text{ cell } i \text{ good}} |\mathcal{H}_{\mathsf{opt}}(X_i)| + |\mathcal{H}_{\mathsf{opt}}(X_{\text{bad}})| \right) && (3.19) \\ &\leq c \cdot (|\mathcal{H}_{\mathsf{opt}}| + c_1 b) && (3.20) \\ &\leq \left(1 + \tfrac{1}{\log N}\right) \cdot c \cdot |\mathcal{H}_{\mathsf{opt}}|. && (3.21) \end{aligned}$$

Otherwise we will use the small opt algorithm to compute $\mathcal{H}_{\text{appx}}$ from scratch, which guarantees that $|\mathcal{H}_{\text{appx}}| \leq c_0 \cdot |\mathcal{H}_{\mathsf{opt}}|$.

The recursion depth of the data structure is $O(\log_g N) = O(\log N)$. The multiplicative error factor multiplies by a factor of at most $1+\frac{1}{\log N}$ at each level, so the global multiplicative error is $c \cdot (1 + \frac{1}{\log N})^{O(\log N)} = O(1)$.

Reporting the actual solution. If the $O(1)$-approximate set cover solution $\mathcal{H}_{\text{appx}}$ is explicitly stored (i.e., computed by the small opt algorithm), then we can directly report the solution in $O(\mathsf{opt})$ time. Otherwise $\mathcal{H}_{\text{appx}}$ is implicitly stored as the union of the approximate solutions in the subproblems, and we recursively report the approximate solutions $\mathcal{H}_{\text{appx}}(X_i)$ for each good cell i, as well as $\mathcal{H}_{\text{appx}}(X_{\text{bad}})$ for the bad cells.

The number of leaf nodes of the recursion tree is bounded by $|\mathcal{H}_{\text{appx}}|$, since at each leaf node we need to report at least one element. Each internal node of the recursion tree has degree at least 2, so the total number of nodes is also bounded by $O(|\mathcal{H}_{\text{appx}}|)$. The additional work at each node of the recursion tree is $O(1)$, therefore the overall running time is proportional to the output size, i.e., $O(|\mathcal{H}_{\text{appx}}|) = O(\mathsf{opt})$.

Lemma 3.15. There exists a data structure for $O(1)$-approximate dynamic 2D halfplane set cover that supports halfplane insertion/deletions and point deletions in amortized $O(n^{2/3+\delta})$ time w.h.p. for any constant $\delta > 0$, and can answer size and reporting queries in $O(1)$ and $O(\mathsf{opt})$ time, respectively, where n is the size of the instance.

Point insertions. The issue with point insertions is we need to rebuild the whole structure every $O(\frac{m}{b})$ point insertions to ensure the cell sizes are bounded by $O(\frac{m}{b})$, which is costly when m is too small. We resolve this issue by reducing the fully dynamic problem to partially dynamic, using the *logarithmic method* [31] but with a larger base N^δ and a non-trivial base case, and finally obtain sublinear update time for all four types of updates.

Specifically, we use the logarithmic method to partition the set of points X into $O(\frac{1}{\delta})$ subsets X_i, where the i-th subset contains $m_i \in [\frac{N}{(N^\delta)^{i+1}}, \frac{N}{(N^\delta)^i})$ points ($i = 0, \ldots, f$) for some $f = O(\frac{1}{\delta})$, for a sufficiently small constant δ. We build a partially dynamic data structure $\mathcal{D}_i$ that supports halfplane insertion/deletions and point deletions, on the instance $(X_i, \mathcal{H})$; an exception is the last data structure $\mathcal{D}_f$, which will be fully dynamic. We obtain the global approximate set cover solution by taking the union of the approximate solutions for each of the subproblems, losing only a constant approximation factor since there are only $O(\frac{1}{\delta}) = O(1)$ subproblems. The logarithmic method guarantees that the subset X_i is rebuilt only after every $\Theta(\frac{m_i}{N^\delta})$ point insertions.

To implement the fully dynamic data structure $\mathcal{D}_f$, we use the small opt algorithm (Lemma 3.13), with $\widetilde{O}(m_f^2 \cdot \sqrt{\frac{n}{z}} + z) = \widetilde{O}(m_f^{4/3} n^{1/3})$ update time, by setting $z = m_f^{4/3} n^{1/3}$ (here and in the following we ignore the $N^{O(\delta)}$ factors by a slight abuse of the $\widetilde{O}$ notation).

To implement the partially dynamic data structure $\mathcal{D}_i$ ($i < f$) for the subproblem $(X_i, \mathcal{H})$, we suggest two different methods depending on the size m_i.

For $m_f < m_i \leq n^{2/3}$, we modify our partially dynamic solution which we introduced earlier. In particular, at each node of the recursion tree we build the data structure required by the second small opt algorithm (Lemma 3.14), instead of the first small opt algorithm. Another change is that we stop recursing when the current number of halfplanes n' becomes at most $\frac{n}{\sqrt{B_i}}$ (which also ensures the the current number of points $m' \leq \frac{m}{B_i}$), for a parameter B_i to be set later. Following a similar analysis, we obtain update time $\widetilde{O}(\sqrt{B_i} \cdot \frac{m}{B_i} \cdot \sqrt{\frac{n}{\sqrt{B_i}}}) = \widetilde{O}(\frac{m\sqrt{n}}{B_i^{3/4}})$ (the cost is dominated by the lowest level) and construction time $\widetilde{O}(B_i \cdot \frac{n}{\sqrt{B_i}}) = \widetilde{O}(n\sqrt{B_i})$. Recall that the whole data structure needs to be rebuilt after every $\Theta(\frac{m_i}{N^\delta})$ point insertions. By setting $B_i = \frac{m_i^{8/5}}{n^{2/5}} + 1$ (verify that $1 \leq B_i \leq m_i$), the amortized update time is $\widetilde{O}(\frac{m\sqrt{n}}{B_i^{3/4}} + \frac{n\sqrt{B_i}}{m_i}) = \widetilde{O}(\frac{n^{4/5}}{m_f^{1/5}} + \frac{n}{m_f})$.

For $m_i > n^{2/3}$, we use our partially dynamic solution, but this time using a common parameter z_i in the small opt algorithms at all nodes of the recursion tree. The construction time becomes $T(m', n') = \widetilde{O}(m' \cdot \frac{n}{\sqrt{m'}} \cdot z_i)$, and the update time becomes $U(m', n') = \widetilde{O}(\sqrt{m'} \cdot \sqrt{\frac{n'/\sqrt{m'}}{z_i}} + z_i + \sqrt{m'} \cdot z_i)$. Adding the amortized cost for rebuilding (after every $\Theta(\frac{m_i}{N^\delta})$ point insertions), the amortized update time is $\widetilde{O}(\frac{T(m_i,n)}{m_i} + U(m_i, n)) = \widetilde{O}(\frac{m_i^{1/4} n^{1/2}}{\sqrt{z_i}} + (\sqrt{m_i} + \frac{n}{\sqrt{m_i}}) \cdot z_i)$, which is $\widetilde{O}(n^{2/3})$ by setting $z_i = \frac{m_i^{1/2}}{n^{1/3}} \geq 1$.

Finally setting m_f near $n^{7/23}$ (up to N^δ factors) to balance the $\widetilde{O}(m_f^{4/3} n^{1/3})$ and $\widetilde{O}(\frac{n^{4/5}}{m_f^{1/5}} + \frac{n}{m_f})$ terms, we obtain amortized update time $O(n^{17/23+O(\delta)})$.

Theorem 3.8. There exists a data structure for $O(1)$-approximate dynamic 2D halfplane set cover with amortized $O(n^{17/23+\delta})$ update time w.h.p. for any constant $\delta > 0$, and can answer size and reporting queries in $O(1)$ and $O(\mathsf{opt})$ time, respectively, where n is the size of the instance.

The exponent $17/23 < 0.74$ is likely further improvable with more work. The main message is that sublinear update time is achievable for the fully dynamic problem while supporting efficient reporting queries.

3.11 WEIGHTED INTERVALS

In this section, we present the first dynamic data structure with sublinear update time and constant factor approximation for weighted interval set cover.

Let $(X, \mathcal{I})$ be a dynamic weighted interval set cover instance where X is the set of points in $\mathbb{R}$ and $\mathcal{I}$ is the set of weighted intervals. For each interval $I \in \mathcal{I}$, we use $w(I) \geq 0$ to denote the weight of I. Without loss of generality, we may assume that the *point range* of $(X, \mathcal{I})$ is $[0,1]$, i.e., the points in X are always in the range $[0,1]$. We say an interval $I \in \mathcal{I}$ is *two-sided* if both of the endpoints of I lie in the interior of the point range $[0,1]$, and *one-sided* if at least one endpoint of I is outside $[0,1]$.

We first observe that the approach we used for unweighted dynamic interval set cover (Section 3.7) can be easily extended to the weighted case to obtain an $n^{O(1)}$-approximation. Recall that in Section 3.7, we partitioned $[0,1]$ into r connected portions $J_1, \dots, J_r$, each of which contains $O(n/r)$ points in X and $O(n/r)$ endpoints of intervals in $\mathcal{I}$. Then we defined $X_i = X \cap J_i$ and $\mathcal{I}_i = \{I \in \mathcal{I} : I \cap J_i \neq \emptyset \text{ and } J_i \nsubseteq I\}$. Each $(X_i, \mathcal{I}_i)$ was viewed as a dynamic interval set cover instance (called a sub-instance) with point range J_i, and we recursively built a sub-structure $\mathcal{D}_i$ for $(X_i, \mathcal{I}_i)$. In Section 3.7, we construct an approximate set cover $\mathcal{I}_{\text{appx}}$ for $(X, \mathcal{I})$ by distinguishing two cases: when the optimum is small, we compute $\mathcal{I}_{\text{appx}}$ using the output-sensitive algorithm of Lemma 3.10; when the optimum is large, $\mathcal{I}_{\text{appx}}$ is constructed by (essentially) taking the union of the solution maintained in the $\mathcal{D}_i$'s. The output-sensitive algorithm of Lemma 3.10, unfortunately, does not work for the weighted case. Therefore, here we always construct $\mathcal{I}_{\text{appx}}$ in a way similar to that used for the large-optimum case. Specifically, for each J_i, we find a minimum-weight interval $I \in \mathcal{I}$ such that $J_i \subseteq I$ (if it exists) and let w_i be the cost of the set cover of $(X_i, \mathcal{I}_i)$ maintained by $\mathcal{D}_i$. If $w_i \leq w(I)$, we define $\mathcal{I}_i^*$ as the set cover of $(X_i, \mathcal{I}_i)$ maintained by $\mathcal{D}_i$, otherwise we define $\mathcal{I}_i^* = \{I\}$. We then define $\mathcal{I}_{\text{appx}} = \bigsqcup_{i=1}^{r} \mathcal{I}_i^*$. We observe the following fact.

Fact 3.2. If each sub-structure $\mathcal{D}_i$ maintains a t-approximate set cover of the sub-instance $(X_i, \mathcal{I}_i)$, then $\mathcal{I}_{\text{appx}}$ is an rt-approximate set cover of the instance $(X, \mathcal{I})$.

Proof. For $i \in [r]$, let opt_i denote the cost of an optimal set cover of $(X_i, \mathcal{I})$. Clearly, $\mathsf{opt}_i \leq \mathsf{opt}$ for all $i \in [r]$ and thus $\sum_{i=1}^{r} \mathsf{opt}_i \leq r \cdot \mathsf{opt}$. We then show that $\mathsf{cost}(\mathcal{I}_i^*) \leq t \cdot \mathsf{opt}_i$, which implies that $\mathsf{cost}(\mathcal{I}_{\text{appx}}) = \sum_{i=1}^{r} \mathsf{cost}(\mathcal{I}_i^*) \leq rt \cdot \mathsf{opt}$. If an optimal set cover of $(X_i, \mathcal{I})$ consists of a single interval in $\mathcal{I}$ that covers J_i, then we have $\mathsf{cost}(\mathcal{I}_i^*) = \mathsf{opt}_i$. Otherwise, an optimal set cover of $(X_i, \mathcal{I})$ is a set cover of $(X_i, \mathcal{I}_i)$, and hence $\mathsf{cost}(\mathcal{I}_i^*) \leq t \cdot \mathsf{opt}_i$. QED.

The above fact shows that the approximation ratio of our data structure satisfies the recurrence $A(n) = r \cdot A(O(n/r))$, which solves to $A(n) = n^{O(1)}$. Furthermore, as analyzed in Section 3.7, the data structure can be updated in $\widetilde{O}(r)$ amortized time. Setting r to be a constant, we get an $n^{O(1)}$-approximation data structure for dynamic weighted interval set cover with $\widetilde{O}(1)$ amortized update time. In particular, we can maintain an estimation $\mathsf{opt}^{\sim}$ of the optimum opt of $(X, \mathcal{I})$ in $\widetilde{O}(1)$ amortized update time, which satisfies $\mathsf{opt} \leq \mathsf{opt}^{\sim} \leq n^{O(1)}\mathsf{opt}$. With this observation, we now discuss our $(3+\varepsilon)$-approximation data structure. Since our data structure here is somehow involved (compared to the unweighted one in Section 3.7), we shall first (informally) describe the underlying basic ideas, then present the formal definitions and analysis.

3.11.1 Main ideas

The main reason why the data structure in Section 3.7 only achieves an $n^{O(1)}$-approximation is that it decomposes the entire problem into r sub-problems and combines the solutions of the sub-problems in a trivial way. However, these sub-problems are not independent: one interval in $\mathcal{I}$ can be used in all of the r sub-problems in the worst case. As such, each level of the recursion can possibly increase the approximation ratio by a factor of r. In order to handle this issue, our first key idea is to combine the solutions of the sub-problems using *dynamic programming*. To see why DP is helpful, let us assume at this point that each sub-structure $\mathcal{D}_i$ maintains an *optimal* solution for the sub-instance $(X_i, \mathcal{I}_i)$. Under this assumption, we show how DP can be applied to obtain a 3-approximate solution for the instance $(X, \mathcal{I})$.

Let $x_1, \dots, x_{r+1}$ be the endpoints of $J_1, \dots, J_r$ sorted from left to right (so the endpoints of J_i are x_i and x_{i+1}). Consider an interval $I \in \mathcal{I}$. If I contains at least one point in $\{x_1, \dots, x_{r+1}\}$, we "chop" I into at most three pieces as follows. Let i^- (resp., i^+) be the smallest (resp., largest) index such that $x_{i^-} \in I$ (resp., $x_{i^+} \in I$). Then x_{i^-} and x_{i^+} partition I into three pieces: the *left* piece (the part to the left of x_{i^-}), the *middle* piece (the part in between x_{i^-} and x_{i^+}), and the *right* piece (the part to the right of x_{i^+}). We give each piece a weight equal to $w(I)$. Let $\mathcal{I}'$ be the resulting set of intervals after chopping the intervals in $\mathcal{I}$, i.e., $\mathcal{I}'$ consists of all pieces of the chopped intervals in $\mathcal{I}$ and all unchopped intervals in $\mathcal{I}$. It is clear that the optimum of the instance $(X, \mathcal{I}')$ is within $[\mathsf{opt}, 3\mathsf{opt}]$, where opt is the optimum of $(X, \mathcal{I})$.

Now we observe a good property of the interval set $\mathcal{I}'$: each interval in $\mathcal{I}'$ is either contained in J_i for some $i \in [r]$ (e.g., the unchopped intervals and the left/right pieces) or is equal to $[x_{i^-}, x_{i^+}]$ for some $i^-, i^+ \in [r+1]$ (e.g., the middle pieces); we call the intervals of the first type *short intervals* and those of the second type *long intervals*. Let $\mathcal{I}'_{\text{long}} \subseteq \mathcal{I}'$ be the set of long intervals and $\mathcal{I}'_i \subseteq \mathcal{I}'$ be the set of short intervals contained in J_i. Then in any set cover of $(X, \mathcal{I}')$, for each $i \in [r]$, either J_i is covered by a long interval or the points in X_i are covered by short intervals in $\mathcal{I}'_i$. Furthermore, in an optimal set cover of $(X, \mathcal{I}')$, if the points in X_i are covered by short intervals in $\mathcal{I}'_i$, then those short intervals must be an optimal set cover of $(X_i, \mathcal{I}'_i)$. Note that the instance $(X_i, \mathcal{I}'_i)$ is in fact equivalent to the sub-instance $(X_i, \mathcal{I}_i)$, because $\mathcal{I}'_i = \{I \cap J_i : I \in \mathcal{I}_i\}$ (where the weight of $I \cap J_i$ is equal to the weight of I) and the points in X_i are all contained in J_i. Thus, by assumption, an optimal set cover of $(X_i, \mathcal{I}'_i)$ is already maintained in the sub-structure $\mathcal{D}_i$.

Based on this observation, we can use DP to compute an optimal set cover of $(X, \mathcal{I}')$ as follows. For a long interval $I = [x_{i^-}, x_{i^+}] \in \mathcal{I}'_{\text{long}}$, we write $\pi(I) = i^- - 1$. For each i from 1 to r, we compute an optimal set cover for $(\bigcup_{j=1}^{i} X_j, \mathcal{I}')$. To this end, we consider how the points in X_i are covered. Clearly, we can cover the points in X_i using a long interval $I \in \mathcal{I}'_{\text{long}}$ satisfying $J_i \subseteq I$. In this case, the best solution is the union of $\{I\}$ and an optimal set cover of $(\bigcup_{j=1}^{\pi(I)} X_j, \mathcal{I}')$ which has already been computed as $\pi(I) < i$. Alternatively, we can cover the points in X_i using the short intervals in $\mathcal{I}'_i$. In this case, the best solution is the union of an optimal set cover for $(\bigcup_{j=1}^{i-1} X_j, \mathcal{I}')$ and an optimal set cover for $(X_i, \mathcal{I}'_i)$, where the former has already been computed and the latter is maintained in the sub-structure $\mathcal{D}_i$. We try all these possibilities and take the best solution found, which is an optimal set cover for $(\bigcup_{j=1}^{i} X_j, \mathcal{I}')$. When the DP procedure completes, we get an optimal set cover of $(X, \mathcal{I}')$, which in turn gives us a 3-approximation of an optimal set cover of $(X, \mathcal{I})$.

Although the above approach seems promising, there are two issues we need to resolve. First, the above DP procedure takes $O(r \cdot |\mathcal{I}'_{\text{long}}|)$ time, but $|\mathcal{I}'_{\text{long}}| = \Omega(n)$ in the worst case. This issue can be easily handled by observing that there are only $O(r^2)$ different intervals in $\mathcal{I}'_{\text{long}}$. Indeed, every interval in $\mathcal{I}'_{\text{long}}$ is equal to $[x_{i^-}, x_{i^+}]$ for some $i^-, i^+ \in [r+1]$. Among a set of identical intervals in $\mathcal{I}'_{\text{long}}$, only the one with the minimum weight is useful. Therefore, we only need to keep $O(r^2)$ minimum-weight intervals in $\mathcal{I}'_{\text{long}}$. Furthermore, these minimum-weight intervals can be computed in $\widetilde{O}(r^2)$ time using a range-min data structure *without* computing $\mathcal{I}'_{\text{long}}$. Specifically, we identify each interval $I = [a, b] \in \mathcal{I}$ with a weighted point $(a, b) \in \mathbb{R}^2$ with weight $w(I)$. The minimum-weight $[x_{i^-}, x_{i^+}]$ in $\mathcal{I}'_{\text{long}}$ is just the middle piece of the minimum-weight interval whose left endpoint lies in $J_{i^- -1}$ and right point lies in $J_{i^+ +1}$, which corresponds to the minimum-weight point in the rectangular range $[x_{i^- -1}, x_{i^-}] \times [x_{i^+}, x_{i^+ +1}]$. Thus, if we maintain the corresponding weighted points of the intervals in $\mathcal{I}$ in a dynamic 2D range-min data structure, the minimum-weight intervals in $\mathcal{I}'_{\text{long}}$ can be computed in $\widetilde{O}(r^2)$ time and the DP procedure can be done in $\widetilde{O}(r^3)$ time.

The second issue is more serious. We assumed that each sub-structure $\mathcal{D}_i$ maintains an *optimal* solution for the sub-instance $(X_i, \mathcal{I}_i)$. Clearly, this is not the case, as the sub-structures are recursively built and hence can only maintain approximate solutions for the sub-instances. In this case, the approximation ratio may increase by a constant factor at each level of the recursion: an interval in $\mathcal{I}$ is chopped into three pieces and its left/right pieces can be further chopped by the sub-structures in lower levels. To handle this issue, we need to prevent the sub-structures from chopping the intervals that are already chopped in higher levels of the recursion. A key observation is the following: if an interval is chopped in the current level, then its left/right pieces are both *one-sided* intervals in the sub-instances. Therefore, if we only chop the two-sided intervals, we should be able to avoid the issue that an interval is chopped more than once. However, this strategy brings us a new difficulty, i.e., handling the one-sided intervals when constructing the set cover.

We overcome this difficulty as follows. We call a one-sided interval in $\mathcal{I}$ *left* (resp., *right*) one-sided interval if it covers the left (resp., right) end of the point range $[0, 1]$. First, observe that we need at most one left one-sided interval and one right one-sided interval in our solution, simply because the coverage of the left (resp., right) one-sided intervals is nested and thus only the rightmost (resp., leftmost) one in the solution is useful. So a naïve idea is to enumerate the left/right one-sided interval used in our solution. (Clearly, we cannot afford to do this because there might be $\Omega(n)$ one-sided intervals. But at this point let us ignore the issue about running time – we will take care of it later.)

If $L \in \mathcal{I}$ and $R \in \mathcal{I}$ are the left and right one-sided intervals in our solution, then the remaining task is to cover the points in $X \backslash (L \cup R)$. It turns out that we can still apply the DP approach above to compute a set cover for the points in $X \backslash (L \cup R)$ using the intervals in $\mathcal{I}'$. To see this, suppose the right endpoint of L lies in J_{i^-} and the left endpoint of R lies in J_{i^+}. Then the points to be covered are those lying in the portions $J_{i^-} \backslash L, J_{i^- +1}, \dots, J_{i^+ -1}, J_{i^+} \backslash R$. Same as before, in a set cover of $(X \backslash (L \cup R), \mathcal{I}')$, for each portion J_i where $i^- < i < i^+$, either J_i itself is covered by a long interval in $\mathcal{I}'_{\text{long}}$, or the points in X_i are covered by short intervals in $\mathcal{I}'_i$; in the latter case we can use the solution of $(X_i, \mathcal{I}_i)$ maintained in the sub-structure $\mathcal{D}_i$. The only difference occurs in the portions $J_{i^-} \backslash L$ and $J_{i^+} \backslash R$. We can either cover J_{i^-} (resp., J_{i^+}) using a long interval in $\mathcal{I}'_{\text{long}}$ or cover the points in $X_{i^-} \backslash L$ (resp., $X_{i^+} \backslash R$) using short intervals in $\mathcal{I}'_{i^-}$ (resp., $\mathcal{I}'_{i^+}$). However, we do not have a good set cover for $(X_{i^-}\backslash L, \mathcal{I}'_{i^-})$ (resp., $(X_{i^+}\backslash R, \mathcal{I}'_{i^+})$) in hand: the solution maintained in the sub-structure $\mathcal{D}_{i^-}$ (resp., $\mathcal{D}_{i^+}$) is for covering all points in X_{i^-} (resp., X_{i^+}) and hence might be much more expensive than an optimal solution of $(X_{i^-}\backslash L, \mathcal{I}'_{i^-})$ (resp., $(X_{i^+}\backslash R, \mathcal{I}'_{i^+})$). We resolve this by temporarily inserting the interval L (resp., R) with weight 0 to $\mathcal{I}'_{i^-}$ (resp., $\mathcal{I}'_{i^+}$) and update the sub-structure $\mathcal{D}_{i^-}$ (resp., $\mathcal{D}_{i^+}$). Note that with the weight-0 interval L (resp., R), the points in $X_{i^-} \cap L$ (resp., $X_{i^+} \cap R$) can be covered "for free" and thus the solution maintained in $\mathcal{D}_{i^-}$ (resp., $\mathcal{D}_{i^+}$) should be a good set cover of $(X_{i^-}\backslash L, \mathcal{I}'_{i^-})$ (resp., $(X_{i^+}\backslash R, \mathcal{I}'_{i^+})$). Once we have the set covers for the points in $J_{i^-}\backslash L, J_{i^-+1}, \dots, J_{i^+-1}, J_{i^+}\backslash R$ using short intervals, we can use the same DP as above to compute a set cover of $(X\backslash(L \cup R), \mathcal{I}')$, which together with L and R gives us a set cover solution of $(X, \mathcal{I})$.

One can verify that if the sub-structures $\mathcal{D}_1, \dots, \mathcal{D}_r$ are recursively built, then the set cover we obtain is a 3-approximate solution of $(X, \mathcal{I})$, essentially because when an interval is chopped (into up to three pieces) in the current level, its left/right pieces become one-sided intervals in the next level of recursion and can no longer cause any error.

Next, we discuss how to avoid enumerating all the left/right one-sided intervals. The key idea is that if we have a set of left (resp., right) one-sided intervals whose weights are similar, say in a range $[w, (1+\varepsilon)w]$, then we can simply keep the one that has the maximum coverage, i.e., the rightmost (resp., leftmost) one, and discard the others. Indeed, instead of using a left/right one-sided interval we discard, we can always use the one we keep, which increases the total weight by only at most εw. Using the estimation $\mathsf{opt}^\sim$ of the optimum, we can actually classify the one-sided intervals in $\mathcal{I}$ into $\widetilde{O}(1/\varepsilon)$ groups, where the intervals in each group have similar weights. In each group, we only keep the interval with the maximum coverage. In this way, we obtain a set of $\widetilde{O}(1/\varepsilon)$ *candidate* one-sided intervals, and we only need to enumerate these candidate intervals, which can be done much more efficiently.

3.11.2 The data structure

Now we are ready to formally present our data structure and analysis. Let $\varepsilon > 0$ be the approximation factor. Our goal is to design a data structure $\mathcal{D}$ that maintains a $(3+\varepsilon)$-approximate set cover solution for the dynamic weighted interval set cover instance $(X, \mathcal{I})$ and supports the size, membership, and reporting queries to the solution.

Let $J_1, \dots, J_r$, $X_1, \dots, X_r$, and $\mathcal{I}_1, \dots, \mathcal{I}_r$ be as defined in Section 3.7. For each $i \in [r]$, we recursively build a sub-structure $\mathcal{D}_i$ on the sub-instance $(X_i, \mathcal{I}_i)$ with approximation factor $\tilde{\varepsilon} = \alpha\varepsilon$ for some parameter $\alpha < 1$. Next, we compute two sets $\mathcal{L}$ and $\mathcal{R}$ of one-sided intervals in $\mathcal{I}$ as follows. Recall that we have the estimation $\mathsf{opt}^\sim$ satisfying $\mathsf{opt} \leq \mathsf{opt}^\sim \leq n^{O(1)}\mathsf{opt}$. Set $\mathsf{opt}^- = (\varepsilon/4) \cdot \mathsf{opt}^\sim/n^c$ for a sufficiently large constant c so that we have

$\mathsf{opt}^- \leq ((\varepsilon - \alpha\varepsilon)/4) \cdot \mathsf{opt}$, assuming $\alpha = \Omega(1)$ (which is the case when we choose α). Define $\delta_0 = 0$ and $\delta_i = \mathsf{opt}^- \cdot (1 + \tilde{\varepsilon}/2)^{i-1}$ for $i \geq 1$. Let m be the smallest number such that $\delta_m \geq (3+\varepsilon)\mathsf{opt}^{\sim}$. Note that $m = \widetilde{O}(\frac{1}{\tilde{\varepsilon}} \log \frac{1}{\tilde{\varepsilon}})$.

For $i \in [m]$, let $L_i \in \mathcal{I}$ be the left one-sided interval with the rightmost right endpoint satisfying $w(L_i) \in [\delta_{i-1}, \delta_i]$. Then we define $\mathcal{L} = \{L_1, \dots, L_m\}$. Similarly, let $R_i \in \mathcal{I}$ be the right one-sided interval with the leftmost left endpoint satisfying $w(R_i) \in [\delta_{i-1}, \delta_i]$, and define $\mathcal{R} = \{R_1, \dots, R_m\}$. The sets $\mathcal{L}$ and $\mathcal{R}$ can be computed in $\widetilde{O}(m)$ time using a (dynamic) 2D range-max/range-min data structure. Indeed, if we map each interval $I = [a,b] \in \mathcal{I}$ into the point $(a, w(I)) \in \mathbb{R}^2$ with weight b, then the interval L_i just corresponds to the maximum-weight point in the range $(-\infty, 0] \times [\delta_{i-1}, \delta_i]$.

Besides $\mathcal{L}$ and $\mathcal{R}$, we need another set $\mathcal{I}_{\text{long}} \subseteq \mathcal{I}$ defined as follows. Recall that $x_1, \dots, x_{r+1}$ are the endpoints of $J_1, \dots, J_r$ sorted from left to right. For an interval $I \in \mathcal{I}$ that contains at least one point in $\{x_1, \dots, x_{r+1}\}$, its *middle piece* refers to the interval $[x_{i^-}, x_{i^+}]$ where x_{i^-} (resp., x_{i^+}) is the leftmost (resp., rightmost) point in $\{x_1, \dots, x_{r+1}\}$ that is contained in I. For every $i^-, i^+ \in [r+1]$ where $i^- < i^+$, we include in $\mathcal{I}_{\text{long}}$ the minimum-weight interval in $\mathcal{I}$ whose middle piece is $[x_{i^-}, x_{i^+}]$. Note that $|\mathcal{I}_{\text{long}}| = O(r^2)$. Also, we can compute $\mathcal{I}_{\text{long}}$ in $\widetilde{O}(r^2)$ time using a (dynamic) 2D range-min data structure. Indeed, if we map each interval $I = [a,b] \in \mathcal{I}$ into the point $(a,b) \in \mathbb{R}^2$ with weight $w(I)$, then the minimum-weight interval in $\mathcal{I}$ whose middle piece is $[x_{i^-}, x_{i^+}]$ just corresponds to the minimum-weight point in the range $[x_{i^- -1}, x_{i^-}] \times [x_{i^+}, x_{i^+ +1}]$.

Update of the sub-structures and reconstruction. Whenever the instance $(X, \mathcal{I})$ changes, we need to update the sub-structures for which the underlying sub-instances change. An insertion/deletion on X or $\mathcal{I}$ can change at most two sub-instances. We also need to re-compute the sets $\mathcal{L}$, $\mathcal{R}$, and $\mathcal{I}_{\text{long}}$.

As before, our data structure will be periodically reconstructed. Specifically, the $(i+1)$-th reconstruction happens after processing n_i/r updates from the i-th reconstruction, where n_i denotes the size of $(X, \mathcal{I})$ at the point of the i-th reconstruction. (The 0-th reconstruction is just the initial construction of $\mathcal{D}$.)

Constructing a solution. For each pair (L, R) where $L \in \mathcal{L}$ and $R \in \mathcal{R}$, we construct a set cover $\mathcal{I}^*(L,R)$ of $(X, \mathcal{I})$ that includes L and R as follows. Suppose the right (resp., left) endpoint of L (resp., R) lies in J_{i^-} (resp., J_{i^+}). If $i^- > i^+$, we simply let $\mathcal{I}^*(L,R) = \{L, R\}$. If $i^- = i^+$, we temporarily insert the intervals L and R with weight 0 to the sub-instance $(X_i, \mathcal{I}_i)$ where $i = i^- = i^+$ and let $\mathcal{I}_i^*$ be the set cover of $(X_i, \mathcal{I}_i \cup \{L, R\})$ maintained by $\mathcal{D}_i$ excluding the weight-0 intervals L and R. We then define $\mathcal{I}^*(L,R) = \{L,R\} \cup \mathcal{I}_i^*$.

Now assume $i^- < i^+$. We temporarily insert the interval L (resp., R) with weight 0 to the sub-instance $(X_{i^-}, \mathcal{I}_{i^-})$ (resp., $(X_{i^+}, \mathcal{I}_{i^+})$) and let $\mathcal{I}_{i^-}^*$ (resp., $\mathcal{I}_{i^+}^*$) be the set cover of $(X_{i^-}, \mathcal{I}_{i^-} \cup \{L\})$ (resp., $(X_{i^+}, \mathcal{I}_{i^+} \cup \{R\})$) maintained by $\mathcal{D}_{i^-}$ (resp., $\mathcal{D}_{i^+}$) excluding the weight-0 interval L (resp., R).

For $i^- < i < i^+$, let $\mathcal{I}_i^*$ be the set cover of $(X_i, \mathcal{I}_i)$ maintained by $\mathcal{D}_i$. We construct $\mathcal{I}^*(L, R)$ using the DP procedure described before. Let $\text{OPT}[0, \dots, i^+], \mathcal{I}_{\text{long}}^*[0, \dots, i^+]$, and $P[0, \dots, i^+]$ be three tables to be computed. Set $\text{OPT}[i] = 0$, $\mathcal{I}_{\text{long}}^*[i] = \emptyset$, and $P[i] = \emptyset$ for all $i < i^-$. For each i from i^- to i^+, we fill out the entries $\text{OPT}[i]$, $\mathcal{I}_{\text{long}}^*[i]$, $P[i]$ as follows. We find the interval $I \in \mathcal{I}_{\text{long}}$ satisfying $J_i \subseteq I$ that minimizes $\text{OPT}[\pi(I)] + w(I)$ where $\pi(I) \in [r]$ is the index such that the left endpoint of the middle piece of I is $x_{\pi(I)+1}$ (or equivalently, the left endpoint of I contains in $J_{\pi(I)}$). If $\text{OPT}[\pi(I)] + w(I) \leq \text{OPT}[i-1] + \mathsf{cost}(\mathcal{I}_i^*)$, then let $\text{OPT}[i] = \text{OPT}[\pi(I)] + w(I)$, $\mathcal{I}_{\text{long}}^*[i] = \mathcal{I}_{\text{long}}^*[\pi(I)] \cup \{I\}$, and $P[i] = P[\pi(I)]$. Otherwise, let $\text{OPT}[i] = \text{OPT}[i-1] + \mathsf{cost}(\mathcal{I}_i^*)$, $\mathcal{I}_{\text{long}}^*[i] = \mathcal{I}_{\text{long}}^*[i-1]$, and $P[i] = P[i-1] \cup \{i\}$. Then we define $\mathcal{I}^*(L, R) = \{L, R\} \sqcup \mathcal{I}_{\text{long}}^* \sqcup (\bigsqcup_{i \in P} \mathcal{I}_i^*)$ where $\mathcal{I}_{\text{long}}^* = \mathcal{I}_{\text{long}}^*[i^+]$ and $P = P[i^+]$. It is clear that the cost of $\mathcal{I}^*(L, R)$ is equal to $w(L) + w(R) + \text{OPT}[i^+]$. Also, as one can easily verify, the DP procedure guarantees the following property of $\mathcal{I}^*(L, R)$.

Fact 3.3. Let $P \subseteq \{i^-, \dots, i^+\}$ and $\mathcal{I}' \subseteq \mathcal{I}$ such that for any $i \in \{i^-, \dots, i^+\} \backslash P$, $J_i \subseteq I$ for some $I \in \mathcal{I}'$. Then $\mathsf{cost}(\mathcal{I}^*(L, R)) \leq w(L) + w(R) + \mathsf{cost}(\mathcal{I}') + \sum_{i \in P} \mathsf{cost}(\mathcal{I}_i^*)$.

We construct $\mathcal{I}^*(L, R)$ for all $L \in \mathcal{L}$ and $R \in \mathcal{R}$. (Clearly, we cannot afford to construct $\mathcal{I}^*(L, R)$ explicitly as the size of $\mathcal{I}^*(L, R)$ can be large. So what we do is to only compute the DP tables, which implicitly represents $\mathcal{I}^*(L, R)$.) Finally, among all $\mathcal{I}^*(L, R)$, we take the one of the smallest cost as the set cover solution $\mathcal{I}_{\text{appx}}$ for $(X, \mathcal{I})$.

Answering queries to the solution. The way to store the approximate solution $\mathcal{I}_{\text{appx}}$ for answering queries is essentially the same as the unweighted case in Section 3.7. We explicitly calculate and store the cost of $\mathcal{I}_{\text{appx}}$, and the membership and reporting queries are handled by recursively querying the sub-structures. By the same analysis as in Section 3.7, we can answer the size, membership, and reporting queries in $O(1)$, $O(\log n)$, and $O(|\mathcal{I}_{\text{appx}}| \log n)$ time, respectively (for reporting queries, we are using the naive analysis here).

Correctness. It is easy to see that $\mathcal{I}_{\text{appx}}$ is a set cover of $(X, \mathcal{I})$. In order to show $w(\mathcal{I}_{\text{appx}}) \leq (3 + \varepsilon) \cdot \mathsf{opt}$, we introduce a new approximation criterion called (c_1, c_2)-*approximation*. We define the (c_1, c_2)-*cost* of a set cover $\mathcal{I}^*$ of $(X, \mathcal{I})$ as the total weight of the one-sided intervals in $\mathcal{I}^*$ times c_1, plus the total weight of the two-sided intervals in $\mathcal{I}^*$ times c_2. We say a set cover of $(X, \mathcal{I})$ is a (c_1, c_2)-*approximate* solution if its (normal) cost is smaller than or equal to the (c_1, c_2)-cost of any set cover of $(X, \mathcal{I})$. We shall show that $\mathcal{I}_{\text{appx}}$ is a $(1 + \frac{\varepsilon}{2}, 3 + \varepsilon)$-approximate solution for $(X, \mathcal{I})$, which implies $w(\mathcal{I}_{\text{appx}}) \leq (3 + \varepsilon) \cdot \mathsf{opt}$.

By induction, we can assume that each sub-structure maintains a $(1+\frac{\tilde{\varepsilon}}{2}, 3+\tilde{\varepsilon})$-approximate solution for $(X_i, \mathcal{I}_i)$. Consider a set cover $\mathcal{I}_{\text{opt}}$ of $(X, \mathcal{I})$ with minimum $(1 + \frac{\varepsilon}{2}, 3 + \varepsilon)$-cost. Note that the $(1 + \frac{\varepsilon}{2}, 3 + \varepsilon)$-cost of $\mathcal{I}_{\text{opt}}$ is at most $(3 + \varepsilon) \cdot \mathsf{opt}$, and hence the (normal) cost is at most $(3 + \varepsilon) \cdot \mathsf{opt}$. Let L and R be the left and right one-sided intervals used in $\mathcal{I}^*$. We have $w(L) \leq (3 + \varepsilon) \cdot \mathsf{opt} \leq \delta_m$, and thus $w(L) \in [\delta_{u-1}, \delta_u]$ for some $u \in [m]$. Similarly, $w(R) \in [\delta_{v-1}, \delta_v]$ for some $v \in [m]$. By construction, we have $L \cap [0, 1] \subseteq L_u \cap [0, 1]$ and $R \cap [0, 1] \subseteq R_v \cap [0, 1]$. Thus, $\mathcal{I}'_{\text{opt}} = (\mathcal{I}_{\text{opt}} \backslash \{L, R\}) \cup \{L_u, R_v\}$ is also a set cover of $(X, \mathcal{I})$. Furthermore, we notice the following.

Fact 3.4. The $(1, 3 + \tilde{\varepsilon})$-cost of $\mathcal{I}'_{\text{opt}}$ is at most the $(1 + \frac{\varepsilon}{2}, 3 + \varepsilon)$-cost of $\mathcal{I}_{\text{opt}}$.

Proof. The $(1, 3+\tilde{\varepsilon})$-cost of $\mathcal{I}'_{\text{opt}}$ is equal to $w(L_u)+w(R_v)+(3+\tilde{\varepsilon})\cdot\mathsf{cost}(\mathcal{I}_{\text{opt}}\backslash\{L, R\})$. Clearly, $(3 + \tilde{\varepsilon}) \cdot \mathsf{cost}(\mathcal{I}_{\text{opt}}\backslash\{L, R\})$ is the $(0, 3 + \tilde{\varepsilon})$-cost of $\mathcal{I}_{\text{opt}}$. We shall show that $w(L_u) + w(R_v)$ is at most the $(1 + \frac{\varepsilon}{2}, (\varepsilon - \alpha\varepsilon)/2)$-cost of $\mathcal{I}_{\text{opt}}$, which implies the claim in the fact. We have

$$w(L_u) \leq (1 + \tilde{\varepsilon}/2) \cdot w(L) + \delta_1 \leq (1 + \tilde{\varepsilon}/2) \cdot w(L) + ((\varepsilon - \alpha\varepsilon)/4) \cdot \mathsf{opt}, \tag{3.22}$$

and similarly $w(R_v) \leq (1+\tilde{\varepsilon}/2)\cdot w(R)+((\varepsilon-\alpha\varepsilon)/4)\cdot\mathsf{opt}$. It follows that $w(L_u)+w(R_v) \leq (1+\tilde{\varepsilon}/2)\cdot(w(L)+w(R))+((\varepsilon-\alpha\varepsilon)/2)\cdot\mathsf{opt}$. Note that $((\varepsilon-\alpha\varepsilon)/2)\cdot\mathsf{opt} \leq ((\varepsilon-\alpha\varepsilon)/2)\cdot\mathsf{cost}(\mathcal{I}_{\text{opt}})$ and $((\varepsilon-\alpha\varepsilon)/2)\cdot\mathsf{cost}(\mathcal{I}_{\text{opt}})$ is the $((\varepsilon-\alpha\varepsilon)/2, (\varepsilon-\alpha\varepsilon)/2)$-cost of $\mathcal{I}_{\text{opt}}$. Also, $(1+\tilde{\varepsilon}/2)\cdot(w(L)+w(R))$ is at most the $(1+\frac{\tilde{\varepsilon}}{2}, 0)$-cost of $\mathcal{I}_{\text{opt}}$. Thus, $w(L_u)+w(R_v)$ is at most the $(1+\frac{\varepsilon}{2}, (\varepsilon-\alpha\varepsilon)/2)$-cost of $\mathcal{I}_{\text{opt}}$. QED.

Now it suffices to show that $\mathsf{cost}(\mathcal{I}^*(L_u, R_v))$ is at most the $(1, 3+\tilde{\varepsilon})$-cost of $\mathcal{I}'_{\text{opt}}$. Suppose the right (resp., left) endpoint of L_u (resp., R_v) lies in J_{i^-} (resp., J_{i^+}). If $i^- > i^+$, then $\mathcal{I}^*(L_u, R_v) = \{L_u, R_v\}$ and hence $\mathsf{cost}(\mathcal{I}^*(L_u, R_v))$ is at most the $(1, 0)$-cost of $\mathcal{I}'_{\text{opt}}$. The remaining cases are $i^- < i^+$ and $i^- = i^+$. Here we only analyze the case $i^- < i^+$, because the other case $i^- = i^+$ is similar and simpler. Recall that when computing $\mathcal{I}^*(L_u, R_v)$, we temporarily inserted the interval L_u (resp., R_v) with weight 0 to the sub-instance $(X_{i^-}, \mathcal{I}_{i^-})$ (resp., $(X_{i^+}, \mathcal{I}_{i^+})$) and let $\mathcal{I}^*_{i^-}$ (resp., $\mathcal{I}^*_{i^+}$) be the set cover of $(X_{i^-}, \mathcal{I}_{i^-} \cup \{L\})$ (resp., $(X_{i^+}, \mathcal{I}_{i^+} \cup \{R\})$) maintained by $\mathcal{D}_{i^-}$ (resp., $\mathcal{D}_{i^+}$) excluding the weight-0 interval L (resp., R). Also, for $i^- < i < i^+$, we let $\mathcal{I}^*_i$ be the set cover of $(X_i, \mathcal{I}_i)$ maintained by $\mathcal{D}_i$. Let $P \subseteq \{i^-, \dots, i^+\}$ consist of all indices i such that $J_i \nsubseteq I$ for all $I \in \mathcal{I}'_{\text{opt}}$ and $\mathcal{I}' \subseteq \mathcal{I}'_{\text{opt}}$ consist of all intervals that contain at least one point in $\{x_1, \dots, x_{r+1}\}$. Now let us define another set cover $\mathcal{I}''_{\text{opt}} = \{L_u, R_v\} \sqcup \mathcal{I}' \sqcup (\bigsqcup_{i \in P} \mathcal{I}^*_i)$. Note that the sets P and $\mathcal{I}'$ satisfy the condition in Fact 3.3.

Thus, by applying Fact 3.3, we have

$$\mathsf{cost}(\mathcal{I}^*(L_u, R_v)) \leq w(L_u) + w(R_v) + \mathsf{cost}(\mathcal{I}') + \sum_{i \in P} \mathsf{cost}(\mathcal{I}_i^*) = \mathsf{cost}(\mathcal{I}''_{\text{opt}}). \tag{3.23}$$

With the above inequality, it suffices to show that $\mathsf{cost}(\mathcal{I}''_{\text{opt}})$ is at most the $(1, 3+\tilde{\varepsilon})$-cost of $\mathcal{I}'_{\text{opt}}$. Equivalently, we show that $\mathsf{cost}(\mathcal{I}') + \sum_{i \in P} \mathsf{cost}(\mathcal{I}_i^*)$ is at most the $(0, 3+\tilde{\varepsilon})$-cost of $\mathcal{I}'_{\text{opt}} \backslash \{L_u, R_v\}$.

Let $\mathcal{I}'_i = \mathcal{I}'_{\text{opt}} \cap \mathcal{I}_i$ for all $i \in P$. Note that $\mathcal{I}'_i$ is a set cover of $(X_i, \mathcal{I}_i)$ for $i \in P \backslash \{i^-, i^+\}$. By assumption, $\mathsf{cost}(\mathcal{I}_i^*)$ is at most the $(1+\tilde{\varepsilon}/2, 3+\tilde{\varepsilon})$-cost of $\mathcal{I}'_i$ for $i \in P \backslash \{i^-, i^+\}$. Also, it is easy to see that if $i^- \in P$ (resp., $i^+ \in P$), then $\mathsf{cost}(\mathcal{I}_{i^-}^*)$ (resp., $\mathsf{cost}(\mathcal{I}_{i^+}^*)$) is at most the $(1+\tilde{\varepsilon}/2, 3+\tilde{\varepsilon})$-cost of $\mathcal{I}'_{i^-}$ (resp., $\mathcal{I}'_{i^+}$), because $\mathcal{I}'_{i^-} \cup L$ (resp., $\mathcal{I}'_{i^+} \cup R$) is a set cover of $(X_{i^-}, \mathcal{I}_{i^-} \cup \{L\})$ (resp., $(X_{i^+}, \mathcal{I}_{i^+} \cup \{R\})$) of the same cost as $\mathcal{I}'_{i^-}$ (resp., $\mathcal{I}'_{i^+}$) when $w(L) = 0$ (resp., $w(R) = 0$). Let cost_i be the $(1+\tilde{\varepsilon}/2, 3+\tilde{\varepsilon})$-cost of $\mathcal{I}'_i$ for $i \in P$. By the above observation, we have $\sum_{i \in P} \mathsf{cost}(\mathcal{I}_i^*) \leq \sum_{i \in P} \mathsf{cost}_i$. Each interval in $\mathcal{I}'$ belongs to (at most) two sub-instances as *one-sided* intervals, so its weight is counted in $\sum_{i \in P} \mathsf{cost}_i$ with a multiplier at most $2 \cdot (1+\tilde{\varepsilon}/2) = 2+\tilde{\varepsilon}$. Each interval in $\mathcal{I}'_{\text{opt}} \backslash (\mathcal{I}' \cup \{L_u, R_v\})$ belongs to one sub-instance, so its weight is counted in $\sum_{i \in P} \mathsf{cost}_i$ with a multiplier at most $3+\tilde{\varepsilon}$. As a result, the weight of each interval in $\mathcal{I}'_{\text{opt}} \backslash \{L_u, R_v\}$ is counted in $\mathsf{cost}(\mathcal{I}') + \sum_{i \in P} \mathsf{cost}_i$ with a multiplier at most $3+\tilde{\varepsilon}$. Because $\mathsf{cost}(\mathcal{I}') + \sum_{i \in P} \mathsf{cost}(\mathcal{I}_i^*) \leq \mathsf{cost}(\mathcal{I}') + \sum_{i \in P} \mathsf{cost}_i$, we know that $\mathsf{cost}(\mathcal{I}') + \sum_{i \in P} \mathsf{cost}(\mathcal{I}_i^*)$ is at most the $(0, 3+\tilde{\varepsilon})$-cost of $\mathcal{I}'_{\text{opt}} \backslash \{L_u, R_v\}$. It follows that $\mathsf{cost}(\mathcal{I}''_{\text{opt}})$ is at most the $(1, 3+\tilde{\varepsilon})$-cost of $\mathcal{I}'_{\text{opt}}$, which in turn implies $\mathcal{I}_{\text{appx}}$ is a $(1+\frac{\varepsilon}{2}, 3+\varepsilon)$-approximate solution of $(X, \mathcal{I})$.

Update time. We first observe that, except recursively updating the sub-structures, the (amortized) time cost of all the other work is $\widetilde{O}(r^3 m^2)$. Specifically, computing the sets $\mathcal{L}$ and $\mathcal{R}$ can be done in $\widetilde{O}(m)$ time and computing $\mathcal{I}_{\text{long}}$ takes $\widetilde{O}(r^2)$ time. Using DP to compute each $\mathcal{I}^*(L, R)$ can be done in $O(r^3)$ time, and hence constructing $\mathcal{I}_{\text{appx}}$ takes $O(r^3 m^2)$ time. Storing $\mathcal{I}_{\text{appx}}$ for answering the queries can be done in $\widetilde{O}(r)$ time. The reconstruction of the data structure takes $\widetilde{O}(r)$ amortized time. Next, we consider the recursive updates of the sub-structures. The depth of the recursion is $O(\log_r n)$. If we set $\alpha = 1 - 1/\log_r n$, the approximation factor is $\Theta(\varepsilon)$ in any level of the recursion. When inserting/deleting a point or a interval, we need to update at most two sub-structures whose underlying sub-instances change. Besides, when computing $\mathcal{I}^*(L, R)$, we need to temporarily insert L and R with weight 0 to two sub-instances (and delete them afterwards), which involves a constant number of recursive updates. So the total number of recursive updates is $O(m^2) = \widetilde{O}((\frac{1}{\tilde{\varepsilon}} \log \frac{1}{\tilde{\varepsilon}})^2) = \widetilde{O}((\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})^2)$.

Therefore, if we use $U(n)$ to denote the update time when the instance size is n, we have the recurrence

$$U(n) = \widetilde{O}\left(\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)^2\right) \cdot U(O(n/r)) + \widetilde{O}\left(r^3 \cdot \left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)^2\right), \tag{3.24}$$

which solves to $U(n) = (\log n \cdot \frac{1}{\varepsilon}\log\frac{1}{\varepsilon})^{O(\log_r n)} \cdot r^3$. By setting $r = 2^{\sqrt{\log n \log\log n}+\sqrt{\log n \log(1/\varepsilon)}}$, we have $U(n) = 2^{O\left(\sqrt{\log n \log\log n}+\sqrt{\log n \log(1/\varepsilon)}\right)}$.

Theorem 3.9. There exists a dynamic data structure for $(3+\varepsilon)$-approximate weighted interval set cover with $2^{O\left(\sqrt{\log n \log\log n}+\sqrt{\log n \log(1/\varepsilon)}\right)}$ amortized update time and $\widetilde{O}(n)$ construction time, which can answer size, membership, and reporting queries in $O(1)$, $O(\log n)$, and $O(k \log n)$ time, respectively, where n is the size of the instance and k is the size of the maintained solution.

3.12 WEIGHTED UNIT SQUARES

In this section, we present the first sublinear result for dynamic weighted unit-square set cover, which gets $O(1)$-approximation. It suffices to consider dynamic weighted quadrant set cover, since the reduction from dynamic unit-square set cover to dynamic quadrant set cover [10] still works in the weighted case.

Let $(X, \mathcal{Q})$ be a dynamic weighted quadrant set cover instance where X is the set of points in $\mathbb{R}^2$ and $\mathcal{Q}$ is the set of weighted quadrants, and let $n = |X| + |\mathcal{Q}|$ denote the instance size. We use $w(q)$ to denote the weight of a quadrant q, and $w(\mathcal{Q})$ for the total weight of a set $\mathcal{Q}$ of quadrants. W.l.o.g., assume the points in X lie in the point range $[0,1]^2$. For simplicity, we assume the weights are positive integers bounded by $U = \mathrm{poly}(n)$.

Our idea is based on our unweighted solution for quadrant set cover as explained in Sec. 3.8, which recursively solve for smaller sub-instances and properly combine them to obtain the global solution. In particular, we will again partition the space into $r \times r$ rectangular grid cells. However, our previous solution relies on an output-sensitive algorithm (Lemma 3.11), but such algorithm is not known in the weighted case. Therefore, we will introduce some new ideas.

Data structures. We construct a data structure $\mathcal{D}$ that supports a more powerful type of query:

Given a query rectangle t, compute an $O(1)$-approximate weighted set cover for the points in $X \cap t$, using the quadrants in $\mathcal{Q}$.

For a quadrant $q \in \mathcal{Q}$ intersecting a rectangular range Γ, we say it is *trivial* (resp., *nontrivial*) with respect to Γ if the vertex of q is outside (resp., inside) Γ. Similar to the observation in the unweighted case, our idea is that it suffices to only keep a small subset of the trivial quadrants. In particular, among all trivial quadrants with weights $\in [1, 2^i)$, there are (at most) four maximal quadrants in Γ, which we denote as $\mathcal{M}_{\Gamma,i}$. (In the special case that there exist quadrants with weight in $[1, 2^i)$ that completely contain Γ, $\mathcal{M}_{\Gamma,i}$ will contain any one among them.) We store $\mathcal{M}_{[0,1]^2,i}$ for $i = 1, \ldots, \log U$, and only keep the nontrivial quadrants with respect to $[0,1]^2$ in $\mathcal{Q}$. The intuition is suppose among all trivial quadrants in the optimal solution $\mathcal{Q}_{\text{opt}}$ for Γ, the maximum weight is w, then for an $O(1)$-approximate solution we can just include $\mathcal{M}_{\Gamma,\lceil \log w \rceil}$, and then compute an $O(1)$-approximate solution in the complement region, using only the nontrivial quadrants. The union of the quadrants in $\mathcal{M}_{\Gamma,\lceil \log w \rceil}$ will contain the union of all trivial quadrants in $\mathcal{Q}_{\text{opt}}$.

To build the data structure $\mathcal{D}$, we partition the space into $r \times r$ (nonuniform) grid cells using $r-1$ horizontal/vertical lines, such that each row (resp., column) has size $O(\frac{n}{r})$, where the size of a range Γ is defined as the total number of points in X and vertices of quadrants in $\mathcal{Q}$ inside Γ. Let $\square_{i,j}$ be the cell in the i-th row and j-th column for $(i,j) \in [r]^2$. We define sub-instances for the rows/columns of the partition. In particular, let $R_i = \bigcup_{j=1}^{r} \square_{i,j}$ denote the i-th row and $C_j = \bigcup_{i=1}^{r} \square_{i,j}$ denote the j-th column of the partition. Create a sub-instance $(X_{i,\bullet}, \mathcal{Q}_{i,\bullet})$ for each row i, where $X_{i,\bullet} = X \cap R_i$ contains all points in the row, and $\mathcal{Q}_{i,\bullet}$ contains all quadrants that are nontrivial with respect to the row. Similarly, create a sub-instance $(X_{\bullet,j}, \mathcal{Q}_{\bullet,j})$ for each column j. Recursively construct the data structures $\mathcal{D}_{i,\bullet}$ for each of the r rows, and similarly $\mathcal{D}_{\bullet,j}$ for each of the r columns. Also store the sets of maximal quadrants $\mathcal{M}_{R_i,k}$ and $\mathcal{M}_{C_j,k}$ in the rows and columns, for $k = 1, \ldots, \log U$.

Let $\square_{i,j,k,l} = \bigcup_{i'=i}^{k} \bigcup_{j'=j}^{l} \square_{i',j'}$ denote the grid-aligned rectangular region from row i to k and from column j to l. For each of these $O(r^4)$ grid-aligned rectangles $\square_{i,j,k,l}$, we also maintain an (implicit) $O(1)$-approximate set cover solution $\mathcal{Q}_{\text{appx}}(\square_{i,j,k,l})$ within it, so that its weight $w(\mathcal{Q}_{\text{appx}}(\square_{i,j,k,l}))$ can be retrieved in $O(1)$ time.

Dynamic programming. In the following, we show that given the substructures $\mathcal{D}_{\bullet,j}$ for the columns that support rectangular ranged queries, we can efficiently compute an $O(1)$-approximate solution $\mathcal{Q}_{\text{appx}}(\square_{i,j,k,l})$ for each of the $O(r^4)$ grid-aligned rectangles $\square_{i,j,k,l}$, using dynamic programming.

Consider any grid-aligned rectangle $\square_{i,j,k,l}$. Any set cover solution $\mathcal{Q}_{\text{sol}}(\square_{i,j,k,l})$ can be

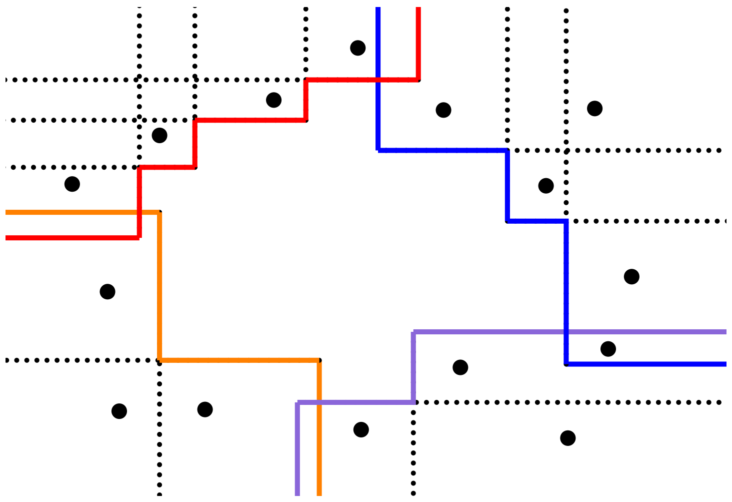


Figure 3.9: Decomposing a quadrant set cover solution into four subsets, based on the four directions of the quadrants. The boundary of the union of each of these subsets forms an orthogonal staircase curve.

decomposed into four subsets $\mathcal{Q}_{\text{sol}}^{\text{NW}}, \mathcal{Q}_{\text{sol}}^{\text{SW}}, \mathcal{Q}_{\text{sol}}^{\text{NE}}, \mathcal{Q}_{\text{sol}}^{\text{SE}}$, based on the four directions of the quadrants (northwest/southwest/northeast/southeast). The boundary of the union of each of these subsets forms an orthogonal staircase curve (as shown in Figure 3.9), which we denote as $U_{\text{sol}}^{\text{NW}}, U_{\text{sol}}^{\text{SW}}, U_{\text{sol}}^{\text{NE}}, U_{\text{sol}}^{\text{SE}}$, respectively.

Let $\ell_0, \ldots, \ell_r$ denote the $r+1$ vertical lines that define the grid (including the boundary), and obtain vertical line ℓ_i^- by slightly shifting ℓ_i to the left. We use $f[i_0][q_\nwarrow][q_\swarrow][q_\nearrow][q_\searrow]$ to denote the weight of an $O(1)$-approximate set cover $\mathcal{Q}_{\text{appx}}$ that covers all points in $\square_{i,j,k,l}$ and to the left of ℓ_{i_0}, such that when we decompose $\mathcal{Q}_{\text{appx}}$ into four orthogonal staircase curves $U_{\text{appx}}^{\text{NW}}, U_{\text{appx}}^{\text{SW}}, U_{\text{appx}}^{\text{NE}}, U_{\text{appx}}^{\text{SE}}$, $\ell_{i_0}^-$ intersects them at quadrants $q_\nwarrow, q_\swarrow, q_\nearrow, q_\searrow$, respectively. (For the special case that ℓ'_{i_0} does not intersect $U_{\text{appx}}^{\text{NW}}$, we let $q_\nwarrow$ to be a special "null" element. Similarly for $q_\swarrow, q_\nearrow, q_\searrow$.)

We decompose each quadrant $q \in \mathcal{Q}$ into two parts: if q has direction west (resp., east), the *long* part is aligned with the rightmost (resp., leftmost) vertical grid line that intersects q, and the *short* part covers the remaining space, which is fully contained in a column (i.e., nontrivial with respect to the column). In this way, each quadrant is duplicated twice, so the approximation factor will multiply by at most 2.

For each vertical grid line ℓ_i, among all long quadrants aligned with ℓ_i, it suffices to keep the lowest (resp., highest) quadrant with direction north (resp., south) with weight $\in [2^k, 2^{k+1})$, for each $k = 1, \ldots, \log U$. We keep $O(r \log U)$ long quadrants in total. The approximation factor will only multiply by a constant.

To compute $f[i_0][q'_\nwarrow][q'_\swarrow][q'_\nearrow][q'_\searrow]$ (corresponding to the set cover solution $\mathcal{Q}'_{\text{appx}}$) where the quadrants $q'_{\nwarrow}, q'_{\swarrow}, q'_{\nearrow}, q'_{\searrow}$ are long (it suffices to only consider the long quadrants after the decomposition, since ℓ_{i_0} is a grid line), we guess that the four orthogonal staircase curves $U'^{\text{NW}}_{\text{appx}}, U'^{\text{SW}}_{\text{appx}}, U'^{\text{NE}}_{\text{appx}}, U'^{\text{SE}}_{\text{appx}}$ corresponding to $\mathcal{Q}'_{\text{appx}}$ intersect $\ell^{-}_{i_0-1}$ at the four long quadrants $q_{\nwarrow}, q_{\swarrow}, q_{\nearrow}, q_{\searrow}$. The set cover solution $\mathcal{Q}_{\text{appx}}$ corresponding to $f[i_0-1][q_{\nwarrow}][q_{\swarrow}][q_{\nearrow}][q_{\searrow}]$ already covers all points to the left of the vertical line ℓ_{i_0-1}, so to obtain $\mathcal{Q}'_{\text{appx}}$, we only need to cover the points between the two grid lines ℓ_{i_0-1} and ℓ_{i_0}. In particular, the region within $\square_{i,j,k,l}$ and between the vertical lines ℓ_{i_0-1} and ℓ_{i_0} that is not covered by the long quadrants $q'_{\nwarrow}, q'_{\swarrow}, q'_{\nearrow}, q'_{\searrow}$ is a rectangle t', and we need to cover t' using the short quadrants in column i_0.

In other words, $f[i_0][q'_{\nwarrow}][q'_{\swarrow}][q'_{\nearrow}][q'_{\searrow}]$ can be computed by the formula

$$f[i_0][q'_{\nwarrow}][q'_{\swarrow}][q'_{\nearrow}][q'_{\searrow}] = \min_{q_{\nwarrow}, q_{\swarrow}, q_{\nearrow}, q_{\searrow}} \Big(f[i_0-1][q_{\nwarrow}][q_{\swarrow}][q_{\nearrow}][q_{\searrow}] + w(\mathcal{Q}_{\text{appx}}(t')) + \sum_{d \in \{\nwarrow, \swarrow, \nearrow, \searrow\}} I[q'_d \neq q_d] \cdot w(q'_d) \Big). \tag{3.25}$$

The weight of an $O(1)$-approximate set cover solution $\mathcal{Q}_{\text{appx}}(t')$ for the rectangle t' can be obtained by querying the column substructure $\mathcal{D}_{\bullet, i_0}$. The final solution is the one with minimum weight among $f[r][q_{\nwarrow}][q_{\swarrow}][q_{\nearrow}][q_{\searrow}]$ for all possible long quadrants $q_{\nwarrow}, q_{\swarrow}, q_{\nearrow}, q_{\searrow}$.

We maintain pointers during the dynamic programming process about how the minimum weight is obtained, so that the actual solution can be easily recovered.

To compute the solution for each grid-aligned rectangle $\square_{i,j,k,l}$ we need to perform $(r \log U)^{O(1)}$ queries, since there are only $O(r \log U)$ choices for each of the long quadrants $q'_{\nwarrow}, q'_{\swarrow}, q'_{\nearrow}, q'_{\searrow}$. There are $O(r^4)$ such grid-aligned rectangles, so the total running time is $(r \log U)^{O(1)} \cdot \mathscr{Q}(O(\frac{n}{r}))$, which is $\widetilde{O}(r^{O(1)})$ by our analysis of the query time $\mathscr{Q}(n)$ later.

Construction time. Let $T(n)$ denote the construction time for the data structure $\mathcal{D}$ when the size of the current instance is n. The construction time satisfies the recurrence

$$T(n) = \sum_{i=1}^{2r} T(n_i) + \widetilde{O}(n) + \widetilde{O}(r^{O(1)}), \tag{3.26}$$

where $n_i = O(\frac{n}{r})$ is the instance size of a row/column. We have $\sum_{i=1}^{2r} n_i \leq 2n$, because each point (resp., quadrant) is copied twice in the row and the column containing that point (resp., the vertex of that quadrant).

Set $r = N^{\delta}$, where N is the global upper bound on the instance size, and $\delta > 0$ is an arbitrarily small constant. The recursion depth is $O(\log_r n) = O(1)$, so the recurrence solves to $T(n) = \widetilde{O}(n \cdot r^{O(1)}) = \widetilde{O}(n \cdot N^{O(\delta)})$.

Query. Given a query rectangle t, we first *guess* that the maximum weight among all trivial quadrants with respect to $[0,1]^2$ used in the optimal solution $\mathcal{Q}_{\text{opt}}(t)$ is within $[2^{k-1}, 2^k)$, and include $\mathcal{M}_{[0,1]^2,k}$ in the approximate solution $\mathcal{Q}_{\text{appx}}(t)$. Let t' denote the complement region of the union of quadrants in $\mathcal{M}_{[0,1]^2,k}$, it suffices to query for the rectangular region $t_0 = t \cap t'$. There are $O(\log U)$ possible choices for $k = 1, \ldots, \log U$, so we need to perform $O(\log U)$ queries and then take the minimum among the results.

The query rectangle t_0 can be decomposed into a grid-aligned rectangle $\square_{i,j,k,l}$ and at most four rectangles $L_{\leftarrow}, L_{\rightarrow}, L_{\uparrow}, L_{\downarrow}$ each contained within a row/column, as shown in Figure 3.6. To compute the approximate set cover $\mathcal{Q}_{\text{appx}}(t_0)$, it suffices to take the union of the approximate solutions within $\square_{i,j,k,l}$ and $L_{\leftarrow}, L_{\rightarrow}, L_{\uparrow}, L_{\downarrow}$, i.e., let

$$\mathcal{Q}_{\text{appx}}(t_0) = \mathcal{Q}_{\text{appx}}(\square_{i,j,k,l}) \cup \mathcal{Q}_{\text{appx}}(L_{\leftarrow}) \cup \mathcal{Q}_{\text{appx}}(L_{\rightarrow}) \cup \mathcal{Q}_{\text{appx}}(L_{\uparrow}) \cup \mathcal{Q}_{\text{appx}}(L_{\downarrow}). \tag{3.27}$$

The approximation factor will only grow by a factor of 5.

$\mathcal{Q}_{\text{appx}}(\square_{i,j,k,l})$ has already been maintained, so we can retrieve its weight in $O(1)$ time. To compute $\mathcal{Q}_{\text{appx}}(L_{\uparrow})$ (and similarly for $L_{\downarrow}, L_{\leftarrow}, L_{\rightarrow}$), we perform a query on the rectangle $L_{\uparrow}$ using the substructure $\mathcal{D}_{i-1,\bullet}$ for row $i-1$, since the rectangle $L_{\uparrow}$ is contained in row $i-1$. In this way, for each guess of k, we need to recursively perform four queries.

Let $\mathscr{Q}(n)$ denote the query time for instance size n. It satisfies the recurrence

$$\mathscr{Q}(n) = O(\log U) \cdot \left(4 \cdot \mathscr{Q}\left(O\left(\frac{n}{r}\right)\right) + O(1)\right). \tag{3.28}$$

The recursion depth is $O(\log_r n) = O(\frac{1}{\delta}) = O(1)$, so we have $\mathscr{Q}(n) = O(\log U)^{O(\frac{1}{\delta})} = \log^{O(1)} n$. The approximation factor grows by a constant factor at each level, so the overall approximation factor is $O(1)^{O(\frac{1}{\delta})} = O(1)$.

Update. When we insert/delete a quadrant q, recursively update the substructures $\mathcal{D}_{i^*,\bullet}$ and $\mathcal{D}_{\bullet,j^*}$ for the i^*-th row and j^*-th column that contain the vertex of q. Update the sets of maximal quadrants $\mathcal{M}_{R_i,k}$ and $\mathcal{M}_{C_j,k}$ in the rows and columns, for $i,j \in [r]$ and $k = 1, \ldots, \log U$. Recompute the $O(1)$-approximate set cover solutions $\mathcal{Q}_{\text{appx}}(\square_{i,j,k,l})$ for each of the $O(r^4)$ grid-aligned rectangles $\square_{i,j,k,l}$, using dynamic programming in $(r \log U)^{O(1)} \cdot \mathscr{Q}(O(\frac{n}{r}))$ time.

When we insert/delete a point p, recursively update the substructures $\mathcal{D}_{i^*,\bullet}$ and $\mathcal{D}_{\bullet,j^*}$ for the i^*-th row and j^*-th column that contain p, and also recompute $\mathcal{Q}_{\text{appx}}(\square_{i,j,k,l})$ for each of the $O(r^4)$ grid-aligned rectangles $\square_{i,j,k,l}$.

We reconstruct the entire data structure after every $\frac{n}{r}$ updates, so that the row and column sizes are always bounded by $O(\frac{n}{r})$. The amortized cost per update is $\frac{T(n)}{n/r} = \widetilde{O}(r^{O(1)})$.

Let $\mathscr{U}(n)$ denote the update time for instance size n. It satisfies the recurrence

$$\mathscr{U}(n) = 2\mathscr{U}\left(O\left(\frac{n}{r}\right)\right) + (r\log U)^{O(1)} \cdot \mathscr{Q}\left(O\left(\frac{n}{r}\right)\right) + \widetilde{O}(r^{O(1)}), \tag{3.29}$$

which solves to $\mathscr{U}(n) = n^{O(\delta)} \cdot N^{O(\delta)} = N^{O(\delta)}$.

Theorem 3.10. There exists a dynamic data structure for $O(1)$-approximate weighted unit-square set cover with $O(n^\delta)$ amortized update time and $\widetilde{O}(n^{1+\delta})$ construction time, for any constant $\delta > 0$ (assuming polynomially bounded integer weights), where n is the size of the instance.

Remark. It is possible to remove the assumption of polynomially bounded weights with more work. One way is to directly modify the recursive query algorithm, as we now briefly sketch:

First, we solve the approximate decision problem, of deciding whether the optimal value is approximately less than a given value W_0. To this end, it suffices to consider $O(\log n)$ choices for k instead of $O(\log U)$ (namely, $k = \lceil \log \frac{W_0}{cn} \rceil, \ldots, \lceil \log W_0 \rceil$ for a large constant c), since replacing a quadrant with weight less than $\frac{W_0}{cn}$ with another one weight less than $\frac{W_0}{cn}$ causes only additive error $O(\frac{W_0}{cn})$, which is tolerable even when summing over all $O(n)$ quadrants. (We don't explicitly store the maximal quadrants in $\mathcal{M}_{\Gamma,i}$ for all $i = 1, \ldots, \log U$, but can generate them on demand by orthogonal range searching.)

Having solved the approximate decision problem, we can next obtain an $O(n)$-approximation of the optimal value, by binary search on the quadrant weights (since the total weight in the optimal solution is within an $O(n)$ factor of the maximum quadrant weight in the optimal solution); this requires $O(\log n)$ calls to the decision oracle. Knowing an $O(n)$-approximation, we can finally obtain an $O(1)$-approximation, by another binary search; this requires $O(\log \log n)$ additional calls to the decision oracle. Thus, we get the same recurrence as Equation 3.28, but with $\log U$ replaced by $\log^{O(1)} n$. We still obtain $O(n^\delta)$ update time in the end.

CHAPTER 4: GEOMETRIC SET COVER AND DISCRETE K-CENTER OF SMALL SIZE

In this chapter, we study the fine-grained complexity of the discrete k-center problem and related (exact) geometric set cover problems when k or the size of the cover is small.

4.1 OUR RESULTS

We obtain a plethora of new algorithms and conditional lower bounds on size-k geometric set cover for $k = 3$ or 6, as well as the related rectilinear discrete 3-center problem:

- We give the first subquadratic algorithm for *rectilinear discrete 3-center* in 2D, running in $\widetilde{O}(n^{3/2})$ time (Theorem 4.5).
- We prove a lower bound of $\Omega(n^{4/3-\delta})$ for rectilinear discrete 3-center in 4D, for any constant $\delta > 0$, under a standard hypothesis about triangle detection in sparse graphs (Theorem 4.8 and Corollary 4.2).
- Given n points and n *weighted* axis-aligned unit squares in 2D, we give the first subquadratic algorithm for finding a minimum-weight cover of the points by 3 unit squares, running in $\widetilde{O}(n^{8/5})$ time (Theorem 4.4). We also prove a lower bound of $\Omega(n^{3/2-\delta})$ for the same problem in 2D, under the well-known APSP Hypothesis (Theorem 4.6).

  For weighted arbitrary axis-aligned rectangles in 2D, our upper bound is $\widetilde{O}(n^{7/4})$ (Theorem 4.3).
- We investigate the smallest k for which we can obtain a near-quadratic lower bound for weighted size-k set cover, and prove such a result for $k = 6$ assuming the Weighted 4-Clique Hypothesis (Theorem 4.9).

Table 4.1 summarizes our results for size-3 geometric set cover in $\mathbb{R}^2$.

New algorithms. We present the first subquadratic algorithms for rectilinear discrete 3-center in $\mathbb{R}^2$, and more generally, for (unweighted and weighted) geometric size-3 set cover for unit squares, as well as arbitrary rectangles in $\mathbb{R}^2$. More precisely, the time bounds of our algorithms are:

- $\widetilde{O}(n^{3/2})$ for rectilinear discrete 3-center in $\mathbb{R}^2$ and unweighted size-3 set cover for unit squares in $\mathbb{R}^2$;

objects	unweighted	weighted
unit squares	$\widetilde{O}(n^{3/2})$ **(new)**	$\widetilde{O}(n^{8/5})$ **(new)** CLB: $\Omega(n^{3/2-\delta})$ **(new)**
rectangles	$\widetilde{O}(n^{5/3})$ **(new)**	$\widetilde{O}(n^{7/4})$ **(new)** CLB: $\Omega(n^{3/2-\delta})$ **(new)**

Table 4.1: Summary of our results on size-3 geometric set cover in $\mathbb{R}^2$.

- $\widetilde{O}(n^{8/5})$ for weighted size-3 set cover for unit squares in $\mathbb{R}^2$;
- $\widetilde{O}(n^{5/3})$ for unweighted size-3 set cover for rectangles in $\mathbb{R}^2$;
- $\widetilde{O}(n^{7/4})$ for weighted size-3 set cover for rectangles in $\mathbb{R}^2$.

New conditional lower bounds. We also prove the first nontrivial conditional lower bounds on the time complexity of rectilinear discrete 3-center and related size-3 geometric set cover problems. More precisely, our lower bounds are:

- $\Omega(n^{3/2-\delta})$ for weighted size-3 set cover (or hitting set) for unit squares in $\mathbb{R}^2$, assuming the APSP Hypothesis;
- $\Omega(n^{4/3-\delta})$ for unweighted size-3 set cover for boxes in $\mathbb{R}^3$, assuming the Sparse Triangle Hypothesis;
- $\Omega(n^{4/3-\delta})$ for rectilinear discrete 3-center in $\mathbb{R}^4$ and unweighted size-3 set cover (or hitting set) for unit hypercubes in $\mathbb{R}^4$, assuming the Sparse Triangle Hypothesis.

The lower bound in the first bullet is particularly attractive, since it implies that conditionally, our $\widetilde{O}(n^{8/5})$-time algorithm for weighted size-3 set cover for unit squares in $\mathbb{R}^2$ is within a small factor (near $n^{0.1}$) from optimal, and that our $\widetilde{O}(n^{7/4})$-time algorithm for weighted size-3 set cover for rectangles in $\mathbb{R}^2$ is within a factor near $n^{0.25}$ from optimal. The second bullet answers Question 1.2, implying that rectilinear discrete 3-center is strictly harder than rectilinear discrete 2-center and rectilinear (continuous) 3-center, at least when the dimension is 4 or higher. (In contrast, rectilinear (continuous) 4-center is strictly harder than rectilinear discrete 4-center for sufficiently large constant dimensions [50]; see Table 1.2.)

Techniques. Traditionally, in computational geometry, subquadratic algorithms with "intermediate" exponents between 1 and 2 tend to arise from the use of nonorthogonal range searching [12] (Agarwal, Sharir, and Welzl's $\widetilde{O}(n^{4/3})$-time algorithm for Euclidean discrete 2-center in $\mathbb{R}^2$ [17] being one such example). Our subquadratic algorithms for rectilinear discrete 3-center in $\mathbb{R}^2$ and related set-cover problems, which are about "orthogonal" or axis-aligned objects, are different. A natural first step is to use a $g \times g$ grid to divide into cases, for some carefully chosen parameter g. Indeed, a grid-based approach was used in some recent subquadratic algorithms by Chan [67] for size-4 independent set for boxes in any constant dimension, and size-5 independent set for rectangles in $\mathbb{R}^2$ (with running time $\widetilde{O}(n^{3/2})$ and $\widetilde{O}(n^{4/3})$ respectively). However, discrete 3-center or rectangle set cover is much more challenging than independent set (for one thing, the 3 rectangles in the solution may intersect each other). To make the grid approach work, we need new original ideas (notably, a sophisticated argument to assign grid cells to rectangles, which is tailored to the 2D case). Still, the entire algorithm description fits in under 3 pages.

Our conditional lower bounds for rectilinear discrete 3-center and the corresponding set cover problem for unit hypercubes are proved by reduction from unweighted or weighted triangle finding in graphs. It turns out there is a simple reduction in $\mathbb{R}^2$ by exploiting weights. However, lower bounds in the unweighted case (and thus the original rectilinear discrete 3-center problem) are much trickier. We are able to design a clever, simple reduction in $\mathbb{R}^6$ by hand, but reducing the dimension down to 4 is far from obvious and we end up employing a *computer-assisted search*, interestingly. The final construction is still simple, and so is easy to verify by hand.

4.2 PRELIMINARIES

On hypotheses from fine-grained complexity. Let us briefly state the hypotheses that will be used in this section for proving various conditional lower bounds.

- The *APSP Hypothesis* is among the three most popular hypotheses in fine-grained complexity [205] (the other two being the 3SUM Hypothesis and the Strong Exponential Time Hypothesis): it asserts that there is no $O(n^{3-\delta})$-time algorithm for the all-pairs shortest paths problem for an arbitrary weighted graph with n vertices (and $O(\log n)$-bit integer weights). This hypothesis has been used extensively in the algorithms literature (but less often in computational geometry).

- The *Sparse Triangle Hypothesis* asserts that there is no $O(m^{4/3-\delta})$-time algorithm for detecting a triangle (i.e., a 3-cycle) in a sparse unweighted graph with m edges. The current best upper bound for triangle detection, from a 3-decade-old paper by Alon, Yuster, and Zwick [19], is $\widetilde{O}(m^{2\omega/(\omega+1)})$, which is $\widetilde{O}(m^{4/3})$ if $\omega = 2$. (In fact, a stronger version of the hypothesis asserts that there is no $O(m^{2\omega/(\omega+1)-\delta})$-time algorithm.) As supporting evidence, it is known that certain "listing" or "all-edges" variants of the triangle detection problem have an $O(m^{4/3-\delta})$ lower bound, under the 3SUM Hypothesis or the APSP Hypothesis [79, 186, 207]. See [2, 143] for more discussion on the Sparse Triangle Hypothesis, and [67] for a recent application in computational geometry.

- The *Hyperclique Hypothesis* asserts that there is no $O(n^{k-\delta})$-time algorithm for detecting a size-k hyperclique in an ℓ-uniform hypergraph with n vertices, for any fixed $k > \ell \geq 3$. See [160] for discussion on this hypothesis, and [42, 67, 158] for some recent applications in computational geometry, including Künnemann's breakthrough result on conditional lower bounds for Klee's measure problem [158].

4.3 SUBQUADRATIC ALGORITHMS FOR SIZE-3 SET COVER FOR RECTANGLES IN $\mathbb{R}^2$

In this section, we describe the most basic version of our subquadratic algorithm to solve the size-3 geometric set cover problem for weighted rectangles in $\mathbb{R}^2$ (an example of the unweighted problem is shown in Fig. 4.1). The running time is $\widetilde{O}(n^{16/9})$. Refinements of the algorithm will be described in Sec. 4.4, where we will improve the time bound further to $\widetilde{O}(n^{7/4})$, or even better for the unweighted case and unit square case. The rectilinear discrete 3-center problem in $\mathbb{R}^2$ reduces to the unweighted unit square case by standard techniques [54, 123] (see Sec. 4.4).

We begin with a lemma giving a useful geometric data structure:

Lemma 4.1. For a set P of n points and a set R of n weighted rectangles in $\mathbb{R}^2$, we can build a data structure in $\widetilde{O}(n)$ time and space, to support the following kind of queries: given a pair of rectangles $r_1, r_2 \in R$, we can find a minimum-weight rectangle $r_3 \in R$ (if it exists) such that P is covered by $r_1 \cup r_2 \cup r_3$, in $\widetilde{O}(1)$ time.

Proof. By orthogonal range searching [12, 100] on P, we can find the minimum/maximum x- and y-values among the points of P in the complement of $r_1 \cup r_2$ in $\widetilde{O}(1)$ time (since the complement can be expressed as a union of $O(1)$ orthogonal ranges). As a result, we obtain the minimum bounding box b enclosing $P \setminus (r_1 \cup r_2)$. To finish, we find a minimum-weight rectangle in R enclosing b; this is a "rectangle enclosure" query on R and can be solved in $\widetilde{O}(1)$ time, since it also reduces to orthogonal range searching (the rectangle $[x^-, x^+] \times [y^-, y^+]$

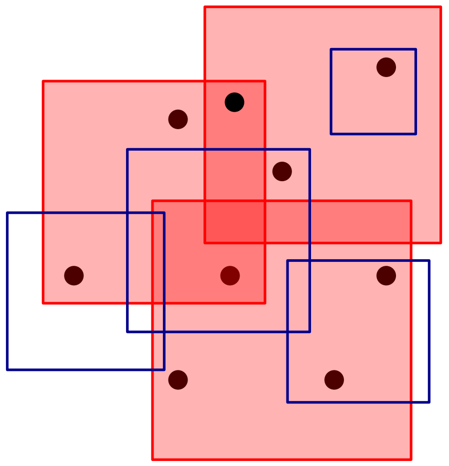


Figure 4.1: An example of size-3 geometric set cover: deciding whether the input points can be covered by three squares. The optimal solution is marked in red.

encloses the rectangle $[\xi^-, \xi^+] \times [\eta^-, \eta^+]$ in $\mathbb{R}^2$ iff the point (x^-, x^+, y^-, y^+) lies in the box $(-\infty, \xi^-] \times [\xi^+, \infty) \times (-\infty, \eta^-] \times [\eta^+, \infty)$ in $\mathbb{R}^4$). QED.

Theorem 4.1. Given a set P of n points and a set R of n weighted rectangles in $\mathbb{R}^2$, we can find 3 rectangles $r_1^*, r_2^*, r_3^* \in R$ of minimum total weight (if they exist), such that P is covered by $r_1^* \cup r_2^* \cup r_3^*$, in $\widetilde{O}(n^{16/9})$ time.

Proof. Let B_0 be the minimum bounding box enclosing P (which touches 4 points). If a rectangle of R has an edge outside of B_0, we can eliminate that edge by extending the rectangle, making it unbounded.

Let g be a parameter to be determined later. Form a $g \times g$ (non-uniform) grid, where each column/row contains $O(n/g)$ rectangle vertices.

Step 1. For each pair of rectangles $r_1, r_2 \in R$ that have vertical edges in a common column or horizontal edges in a common row, we query the data structure in Lemma 4.1 to find a minimum-weight rectangle $r_3 \in R$ (if exists) such that $P \subset r_1 \cup r_2 \cup r_3$, and add the triple $r_1 r_2 r_3$ to a list L. The number of queried pairs $r_1 r_2$ is $O(g \cdot (n/g)^2) = O(n^2/g)$, and so this step takes $\widetilde{O}(n^2/g)$ total time.

Step 2. For each rectangle $r_1 \in R$ and each of its horizontal (resp. vertical) edges e_1, define $\gamma^-(e_1)$ and $\gamma^+(e_1)$ to be the leftmost and rightmost (resp. bottommost and topmost) grid cell that intersects e_1 and contains a point of P not covered by r_1. We can naively find $\gamma^-(e_1)$ and $\gamma^+(e_1)$ by enumerating the $O(g)$ grid cells intersecting e_1 and performing $O(g)$ orthogonal range queries; this takes $\widetilde{O}(gn)$ total time. For each rectangle $r_2 \in R$ that has an edge intersecting $\gamma^-(e_1)$ or $\gamma^+(e_1)$, we query the data

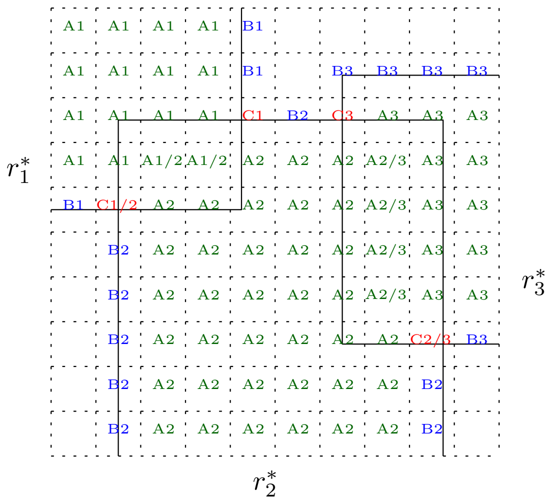

A1 A1 A1 A1 B1
A1 A1 A1 A1 B1 B3 B3 B3 B3
A1 A1 A1 A1 C1 B2 C3 A3 A3 A3
r_1^*
A1 A1 A1/2 A1/2 A2 A2 A2 A2/3 A3 A3
B1 C1/2 A2 A2 A2 A2 A2 A2/3 A3 A3
B2 A2 A2 A2 A2 A2 A2/3 A3 A3
r_3^*
B2 A2 A2 A2 A2 A2 A2/3 A3 A3
B2 A2 A2 A2 A2 A2 A2 C2/3 B3
B2 A2 A2 A2 A2 A2 A2 B2
B2 A2 A2 A2 A2 A2 A2 B2
r_2^*

Figure 4.2: Proof of Theorem 4.1: grid cells in Step 3. The letter in a cell indicates its type (A, B, or C), and the number (or numbers) in a cell indicates the index (or indices) $j \in \{1,2,3\}$ of the rectangle r_j^* that the cell is assigned to.

structure in Lemma 4.1 to find a minimum-weight rectangle $r_3 \in R$ (if exists) such that $P \subset r_1 \cup r_2 \cup r_3$, and add the triple $r_1r_2r_3$ to the list L. The total number of queried pairs r_1r_2 is $O(n \cdot n/g) = O(n^2/g)$, and so this step again takes $\widetilde{O}(n^2/g)$ total time. (This entire Step 2, and the definition of $\gamma^-(\cdot)$ and $\gamma^+(\cdot)$, might appear mysterious at first, but their significance will be revealed later in Step 3.)

Step 3. We guess the column containing each of the vertical edges of r_1^*, r_2^*, r_3^* and the row containing each of the horizontal edges of r_1^*, r_2^*, r_3^*; there are at most 12 edges and so $O(g^{12})$ choices. Actually, 4 of the 12 edges are eliminated after extension, and so the number of choices can be lowered to $O(g^8)$.

After guessing, we know which grid cells are completely inside r_1^*, r_2^*, r_3^* and which grid cells intersect which edges of r_1^*, r_2^*, r_3^*. We may assume that the vertical edges from different rectangles in $\{r_1^*, r_2^*, r_3^*\}$ are in different columns, and the horizontal edges from different rectangles in $\{r_1^*, r_2^*, r_3^*\}$ are in different rows: if not, $r_1^*r_2^*r_3^*$ would have already been found in Step 1. In particular, we know combinatorially what the arrangement of r_1^*, r_2^*, r_3^* looks like, even though we do not know the precise coordinates and identities of r_1^*, r_2^*, r_3^*.

We classify each grid cell γ into the following types (see Figure 4.2):

- Type A: γ is completely contained in some r_j^* ($j \in \{1,2,3\}$). Here, we *assign* γ to each such r_j^*.

- TYPE B: γ is not of type A, and intersects an edge of exactly one rectangle r_j^*. We *assign* γ to this r_j^*. Observe that points in $P \cap \gamma$ can only be covered by r_j^*.
- TYPE C: γ is not of type A, and intersects edges from two different rectangles in $\{r_1^*, r_2^*, r_3^*\}$. W.l.o.g., suppose that γ intersects a horizontal edge e_1^* of r_1^* and a vertical edge e_2^* of r_2^*; note that the intersection point $v^* = e_1^* \cap e_2^*$ lies on the boundary of the union $r_1^* \cup r_2^* \cup r_3^*$ (because γ is not of type A). By examining the arrangement of $\{r_1^*, r_2^*, r_3^*\}$, we know that at least one of the following is true: (i) we can walk horizontally from v^* to an endpoint of e_1^* (or a point at infinity) while staying on the boundary of $r_1^* \cup r_2^* \cup r_3^*$, or (ii) we can walk vertically from v^* to an endpoint of e_2^* (or a point at infinity) while staying on the boundary of $r_1^* \cup r_2^* \cup r_3^*$.

  If (i) is true, we *assign* γ to r_1^*. Observe that if there is a point in $P \cap \gamma$ not covered by r_1^* (and if the guesses are correct), then γ must be equal to $\gamma^-(e_1^*)$ or $\gamma^+(e_1^*)$ (as defined in Step 2), and so $r_1^* r_2^* r_3^*$ would have already been found in Step 2. This is because except for γ, the grid cells encountered while walking from v^* to that endpoint of e_1^* can intersect only r_1^* and so points in those cells can only be covered by r_1^*.

  If (ii) is true, we *assign* γ to r_2^* for a similar reason.

Note that there are at most $O(1)$ grid cells γ of type C; and the grid cells γ of type B form $O(1)$ contiguous blocks. Let ρ_j be the union of all grid cells assigned to r_j^*. Then ρ_j is a rectilinear polygon of $O(1)$ complexity. We compute the minimum/maximum x- and y-values of the points in $P \cap \rho_j$, by orthogonal range searching in $\widetilde{O}(1)$ time. As a result, we obtain the minimum bounding box b_j enclosing $P \cap \rho_j$. We find a minimum-weight rectangle $r_j \in R$ enclosing b_j, by a rectangle enclosure query (reducible to orthogonal range searching, as before). If $P \setminus (r_1 \cup r_2 \cup r_3) = \emptyset$ (testable by orthogonal range searching), we add the triple $r_1 r_2 r_3$ (which should coincide with $r_1^* r_2^* r_3^*$, if it has not been found earlier and if the guesses are correct) to L. The total time over all guesses is $\widetilde{O}(g^8)$.

At the end, we return a minimum-weight triple in L. The overall running time is $\widetilde{O}(g^8 + n^2/g + gn)$. Setting $g = n^{2/9}$ yields the theorem. QED.

4.4 IMPROVED SUBQUADRATIC ALGORITHMS FOR SIZE-3 SET COVER FOR RECTANGLES IN $\mathbb{R}^2$ AND RELATED PROBLEMS

Continuing Section 4.3, we describe refinements of our basic subquadratic algorithm to obtain improved time bounds in different cases.

4.4.1 Improvement for unweighted rectangles

Lemma 4.2. Given a set S of points in a grid $\{1,\ldots,g\}^d$, the number of *maximal points* (i.e., points in S that are not strictly dominated by any other point in S) is $O(g^{d-1})$.

Proof. Along each "diagonal" line of the form $\{(t, c_1+t, \ldots, c_{d-1}+t) : t \in \mathbb{R}\}$, there can be at most one point of S. The grid can be covered by $O(g^{d-1})$ such lines. QED.

Theorem 4.2. The running time in Theorem 4.1 can be improved to $\widetilde{O}(n^{5/3})$ if the rectangles in R are unweighted.

Proof. In the unweighted case, it suffices to keep only the rectangles that are *maximal*, i.e., not strictly contained in other rectangles. (We can test whether a rectangle is maximal in $\widetilde{O}(1)$ time by a rectangle enclosure query, reducible to orthogonal range searching.) We run the algorithm in the proof of Theorem 4.1.

For a p-sided rectangle r_j^* (with $p \in \{1,\ldots,4\}$), the number of choices for the columns/rows containing the vertical/horizontal edges of r_j^* is now $O(g^{p-1})$ instead of $O(g^p)$ by Lemma 4.2, because the p-tuples of such column/row indices correspond to maximal points of a subset of $\{1,\ldots,g\}^p$, by mapping p-sided rectangles to p-dimensional points.

This way, the number of guesses for the 3 rectangles r_1^*, r_2^*, r_3^* can be further lowered from $O(g^8)$ to $O(g^5)$, and the overall running time becomes $\widetilde{O}(g^5 + n^2/g + gn)$. Setting $g = n^{1/3}$ yields the theorem. QED.

4.4.2 Improvement for weighted rectangles

Theorem 4.3. The running time in Theorem 4.1 can be improved to $\widetilde{O}(n^{7/4})$ for weighted rectangles.

Proof. We follow Steps 1–2 of the proof of Theorem 4.1 but will modify Step 3.

- Case I: Two of the rectangles in $\{r_1^*, r_2^*, r_3^*\}$ are disjoint.

  W.lo.g., say that these two rectangles are r_2^* and r_3^*, and they are vertically separated, with r_2^* to the left. We guess the identity of r_1^*; there are $O(n)$ choices. We guess a

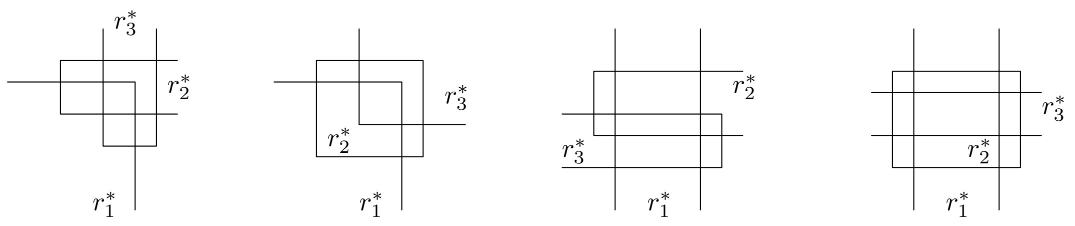


Figure 4.3: Proof of Theorem 4.3: subcases of Case III, when there are no hidden edges.

grid vertical line ℓ^* separating r_1^* and r_2^*; there are $O(g)$ choices. (If a grid separating line does not exist, the answer would have been found in Step 1.) We find the minimum bounding box b_2 enclosing all points of $P \setminus r_1^*$ to the left of ℓ^* by orthogonal range searching, and then find a minimum-weight rectangle $r_2 \in R$ enclosing b_2, by a rectangle enclosure query. We then query the data structure in Lemma 4.1 to find a minimum-weight rectangle $r_3 \in R$ (if exists) such that $P \subset r_1^* \cup r_2 \cup r_3$, and add the triple $r_1^* r_2 r_3$ to the list L. The total number of queries is $O(gn)$, and so this case takes $\widetilde{O}(gn)$ time.

- Case II: r_1^*, r_2^*, r_3^* have at most 7 edges in total, or have 8 edges but at least one of the edges is *hidden*, i.e., does not appear in the boundary of the union $r_1^* \cup r_2^* \cup r_3^*$.

  W.l.o.g., say that this hidden edge is e_3^*, defined by r_3^*. We follow our earlier approach in Step 3, except that we need not guess the column/row containing e_3^*; the number of guesses is thus lowered from $O(g^8)$ to $O(g^7)$. We know the grid cells of types A, B, and C that are assigned to r_1^* and r_2^* (though not necessarily r_3^*), and so we can compute the candidate rectangles r_1 and r_2 as before. We then query the data structure in Lemma 4.1 to find a minimum-weight rectangle $r_3 \in R$ (if exists) such that $P \subset r_1 \cup r_2 \cup r_3$, and add the triple $r_1 r_2 r_3$ to the list L. The total number of queries is $O(g^7)$, and so this case takes $\widetilde{O}(g^7)$ time.

- Case III: r_1^*, r_2^*, r_3^* have a common intersection, and have 8 edges in total, none of which are hidden.

  Some rectangle, say, r_1^*, must have at most 2 edges (i.e., defined either by a vertical and a horizontal ray, or by two parallel lines). By enumerating all scenarios, we see that only 4 main subcases remain, as shown in Figure 4.3 (all other subcases are symmetric). In each of these subcases, one of the rectangles, say, r_3^*, has at most one edge touching $\partial r_1^* \setminus (r_2^* \cup r_3^*)$. Denote this edge by e_3^* (if exists).

  We guess the identity of r_2^*; there are $O(n)$ choices. Once r_2^* is known, we add the vertical and horizontal lines through the edges of r_2^* to the grid. We guess the row/col-

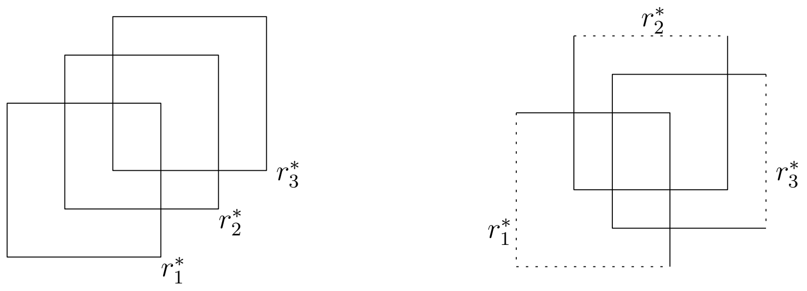


Figure 4.4: Proof of Theorem 4.4: Case II (left) and Case III (right).

umn containing each of r_1^*'s horizontal/vertical edges and the row/column containing r_3^*'s horizontal/vertical edge e_3^* (if exists); there are $O(g^3)$ such choices. We then have enough information to determine the grid cells of types A, B, and C that are assigned to r_1^* (though not necessarily r_3^*). So we can compute the candidate rectangle r_1 as before. We then query the data structure in Lemma 4.1 to find a minimum-weight rectangle $r_3 \in R$ (if exists) such that $P \subset r_1 \cup r_2^* \cup r_3$, and add the triple $r_1 r_2^* r_3$ to the list L. The total number of queries is $O(g^3 n)$, and so this case takes $\widetilde{O}(g^3 n)$ time.

We do not know which case is true beforehand, but can try the algorithms for all cases and return the best triple in L. The overall running time is $\widetilde{O}(g^7 + g^3 n + n^2/g)$. Setting $g = n^{1/4}$ yields the theorem. QED.

4.4.3 Improvement for weighted unit squares

Theorem 4.4. The running time in Theorem 4.1 can be improved to $\widetilde{O}(n^{8/5})$ if the rectangles in R are weighted unit squares.

Proof. We follow the approach in the proof of Theorem 4.1, but will modify Step 3. W.l.o.g., assume that r_1^*, r_2^*, r_3^* have centers ordered from left to right. For unit squares, there are 3 possible cases (up to symmetry):

- Case I: Two of the squares in $\{r_1^*, r_2^*, r_3^*\}$ are disjoint.

  As in the proof of Theorem 4.3, this case can be handled in $\widetilde{O}(gn)$ time.

- Case II: r_1^*, r_2^*, r_3^* have a common intersection, and r_1^* has a lower center than r_2^*, which has a lower center than r_3^*. (See Figure 4.4 (left).)

  We guess the identity of r_2^*; there are $O(n)$ choices. Let $\hat{r}_2^*$ denote the region of all points below and left of the top-right vertex of r_2^*. Notice that $r_1^* \setminus r_2^*$ is contained in $\hat{r}_2^*$, while $r_3^* \setminus r_2^*$ is outside $\hat{r}_2^*$. We find the minimum bounding box b_1 enclosing $P \cap \hat{r}_2^* \setminus r_2^*$, by orthogonal range searching, and find a minimum-weight square $r_1 \in R$ enclosing b_1 by a rectangle enclosure query. We then query the data structure in Lemma 4.1 to find a minimum-weight square $r_3 \in R$ (if exists) such that $P \subset r_1 \cup r_2^* \cup r_3$, and add the triple $r_1 r_2^* r_3$ to the list L. The total number of queries is $O(n)$, and so this case takes $\widetilde{O}(n)$ time.

- Case III: r_1^*, r_2^*, r_3^* have a common intersection, and r_1^* has a lower center than r_3^*, which has a lower center than r_2^*. (See Figure 4.4 (right).)

  After extending the squares to unbounded rectangles, r_1^* has 2 edges, and r_2^* and r_3^* have 3 edges each.

  We guess the column/row containing the vertical/horizontal edge of r_1^*, and the columns containing the two vertical edges of r_2^*, and the rows containing the two horizontal edges of r_3^*; there are $O(g^6)$ choices. We need not guess the bottom horizontal edge of r_2^* and the left vertical edge of r_3^*, as these two edges are hidden. We can assign grid cells of types B and C as before without knowing the hidden edges. For type A, each grid cell completely contained in r_1^* can be assigned to r_1^*; each grid cell that is strictly between the columns containing the two vertical edges of r_2^* and is on or above the row containing the top horizontal edge of r_1^* can be assigned to r_2^*; each grid cell that is strictly between the rows containing the two horizontal edges of r_3^* and is on or to the right of the column containing the right vertical edge of r_1^* can be assigned to r_3^*. The rest of the algorithm is the same.

  Observe that the number of possible pairs of columns containing two vertical edges of r_2^* at unit distance is actually $O(g)$ instead of $O(g^2)$ (to see this, imagine sweeping the grid by two vertical lines at unit distance apart, creating $O(g)$ events). Similarly, the number of possible pairs of rows containing two horizontal edges of r_3^* at unit distance is $O(g)$. With these observations, the number of guesses is further lowered to $O(g^4)$.

The overall running time is $\widetilde{O}(g^4+gn+n^2/g)$. Setting $g = n^{2/5}$ yields the theorem. QED.

4.4.4 Improvement for unweighted unit squares and rectilinear discrete 3-center in $\mathbb{R}^2$

Theorem 4.5. The running time in Theorem 4.1 can be improved to $\widetilde{O}(n^{3/2})$ if the rectangles in R are unweighted unit squares.

Proof. In the unweighted case, it suffices to keep only the rectangles that are maximal (as in the proof of Theorem 4.2). We run the algorithm in the proof of Theorem 4.4.

In Case III, the number of guesses for the column/row containing the vertical/horizontal edge of r_1^* is now $O(g)$ instead of $O(g^2)$ by Lemma 4.2. The overall running time becomes $\widetilde{O}(g^3 + gn + n^2/g)$. Setting $g = \sqrt{n}$ yields the theorem. QED.

Corollary 4.1. Given a set P of n points in $\mathbb{R}^2$, we can solve the rectilinear discrete 3-center problem in $\widetilde{O}(n^{3/2})$ time.

Proof. The decision problem—deciding whether the optimal radius is at most a given value r—amounts to finding 3 squares covering P, among n squares with centers at P and side length $2r$. The decision problem can thus be solved in $\widetilde{O}(n^{3/2})$ time by Theorem 4.2 (after rescaling to make the side length unit). We can then solve the original optimization problem by Frederickson and Johnson's technique [123], with an extra logarithmic factor (or alternatively by Chan's randomized technique [54], without the logarithmic factor). QED.

4.5 CONDITIONAL LOWER BOUNDS FOR SIZE-3 SET COVER FOR BOXES

In this section, we prove conditional lower bounds for size-3 set cover for boxes in certain dimensions (rectilinear discrete 3-center is related to size-3 set cover for unit hypercubes). We begin with the weighted version, which is more straightforward and has a simple proof, and serves as a good warm-up to the more challenging, unweighted version later.

4.5.1 Weighted size-3 set cover for unit squares in $\mathbb{R}^2$

An *orthant* (also called a dominance range) refers to a d-sided box in $\mathbb{R}^d$ which is unbounded along each of the d dimensions. (Note that orthants may be oriented in 2^d ways.) To obtain a lower bound for the unit square or unit hypercube case, it suffices to obtain a lower bound for the orthant case, since we can just replace each orthant with a hypercube with a sufficiently large side length M, and then rescale by a $1/M$ factor.

Theorem 4.6. Given a set P of n points and a set R of n weighted orthants in $\mathbb{R}^2$, finding 3 orthants in R of minimum total weight that cover P requires $\Omega(n^{3/2-\delta})$ time for any constant $\delta > 0$, assuming the APSP Hypothesis.

Proof. The APSP Hypothesis is known to be equivalent [206] to the hypothesis that finding a minimum-weight triangle in a weighted graph with n vertices requires $\Omega(n^{3-\delta})$ time for any constant $\delta > 0$. We will reduce the minimum-weight triangle problem on a graph with n vertices and m edges ($m \in [n, n^2]$) to the weighted size-3 set cover problem for $O(m)$ points and orthants in $\mathbb{R}^2$. Thus, if there is an $O(m^{3/2-\delta})$-time algorithm for the

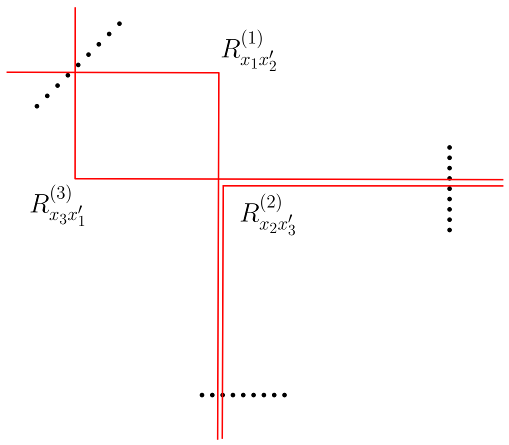


Figure 4.5: Reduction from the minimum-weight triangle problem to weighted size-3 set cover for orthants in $\mathbb{R}^2$.

latter problem, there would be an algorithm for the former problem with running time $O(m^{3/2-\delta}) \le O(n^{3-2\delta})$, refuting the hypothesis.

Let $G = (V, E)$ be the given weighted graph with n vertices and m edges. Without loss of generality, assume that all edge weights are in $[0, 0.1]$, and that $V \subset [0, 0.1]$, i.e., vertices are labelled by numbers that are rescaled to lie in $[0, 0.1]$. Assume that $0 \in V$ and $0.1 \in V$.

The reduction. For each vertex $t \in V$, we create three points $(t, 1+t)$, $(2, t)$, and $(1+t, -1)$ (call them of type 1, 2, and 3, respectively).

Create the following orthants in $\mathbb{R}^2$:

$$\begin{array}{lclclr} \forall x_1x'_2 \in E:\ R^{(1)}_{x_1x'_2} & = & (-\infty, 1+x'_2) & \times & (-\infty, 1+x_1] & \text{(type 1)} \\ \forall x_2x'_3 \in E:\ R^{(2)}_{x_2x'_3} & = & [1+x_2, \infty) & \times & (-\infty, x'_3) & \text{(type 2)} \\ \forall x_3x'_1 \in E:\ R^{(3)}_{x_3x'_1} & = & (x'_1, \infty) & \times & [x_3, \infty) & \text{(type 3)} \end{array} \tag{4.1}$$

The weight of each orthant is set to be the number of points it covers plus the weight of the edge it represents. The total number of points and orthants is $O(n)$ and $O(m)$ respectively. The reduction is illustrated in Figure 4.5.

Correctness. We prove that the minimum-weight triangle in G has weight w (where $w \in [0, 0.3]$) iff the optimal weighted size-3 set cover has weight $3n + w$.

Any feasible solution (if exists) must use an orthant of each type, since the point $(0, 1)$ of type 1 (resp. the point $(2, 0.1)$ of type 2, and the point $(1.1, -1)$ of type 3) can only be covered by an orthant of type 1 (resp. 3 and 2). So, the three orthants in the optimal solution must be of the form $R^{(1)}_{x_1x'_2}$, $R^{(2)}_{x_2x'_3}$ and $R^{(3)}_{x_3x'_1}$ for some $x_1x'_2, x_2x'_3, x_3x'_1 \in E$.

If $x_1 < x'_1$, some point (of type 1) would be uncovered; on the other hand, if $x_1 > x'_1$, some point (of type 1) would be covered twice, and the total weight would then be at least $3n + 1$. Thus, $x_1 = x'_1$. Similarly, $x_2 = x'_2$ and $x_3 = x'_3$. So, $x_1x_2x_3$ forms a triangle in G. We conclude that the minimum-weight solution $R^{(1)}_{x_1x_2}$, $R^{(2)}_{x_2x_3}$ and $R^{(3)}_{x_3x_1}$ correspond to the minimum-weight triangle $x_1x_2x_3$ in G. QED.

4.5.2 Unweighted size-3 set cover for boxes in $\mathbb{R}^3$

Our preceding reduction uses weights to ensure equalities of two variables representing vertices. For the unweighted case, this does not work. We propose a different way to force equalities, by using an extra dimension and extra sides (i.e., using boxes instead of orthants), with some carefully chosen coordinate values.

Theorem 4.7. Given a set P of n points and a set R of n unweighted axis-aligned boxes in $\mathbb{R}^3$, deciding whether there exist 3 boxes in R that cover P requires $\Omega(n^{4/3-\delta})$ time for any constant $\delta > 0$, assuming the Sparse Triangle Hypothesis.

Proof. We will reduce the triangle detection problem on a graph with m edges to the unweighted size-3 set cover problem for $O(m)$ points and boxes in $\mathbb{R}^3$. Thus, if there is an $O(m^{4/3-\delta})$-time algorithm for the latter problem, there would be an algorithm for the former problem with running time $O(m^{4/3-\delta})$, refuting the hypothesis.

Let $G = (V, E)$ be the given unweighted sparse graph with n vertices and m edges ($n \le m$). Without loss of generality, assume that $V \subset [0, 0.1]$, and $0 \in V$ and $0.1 \in V$.

The reduction. For each vertex $t \in V$, create six points

$$\begin{array}{rrrl}
(-1+t, & 0, & 2+t) & \quad\text{(type 1)} \\
(1+t, & 0, & -2+t) & \quad\text{(type 2)} \\
(2+t, & -1+t, & 0) & \quad\text{(type 3)} \\
(-2+t, & 1+t, & 0) & \quad\text{(type 4)} \\
(0, & 2+t, & -1+t) & \quad\text{(type 5)} \\
(0, & -2+t, & 1+t) & \quad\text{(type 6)}
\end{array}$$

Create the following boxes in $\mathbb{R}^3$:

$$\begin{array}{lcccccc}
\forall x_1x_2' \in E: \ R^{(1)}_{x_1x_2'} & = & (-1+x_1, 1+x_1) & \times & [-2+x_2', 2+x_2'] & \times & \mathbb{R} \\
\forall x_2x_3' \in E: \ R^{(2)}_{x_2x_3'} & = & [-2+x_3', 2+x_3'] & \times & \mathbb{R} & \times & (-1+x_2, 1+x_2) \\
\forall x_3x_1' \in E: \ R^{(3)}_{x_3x_1'} & = & \mathbb{R} & \times & (-1+x_3, 1+x_3) & \times & [-2+x_1', 2+x_1']
\end{array} \tag{4.2}$$

(call them of type 1, 2, and 3, respectively).

Correctness. We prove that a size-3 set cover exists iff a triangle exists in G.

Any feasible solution (if exists) must use a box of each type, since the point $(-1, 0, 2)$ of type 1 (resp. the point $(2, -1, 0)$ of type 3, and the point $(0, 2, -1)$ of type 5) can only be covered by a box of type 3 (resp. 2 and 1). So, the three boxes in a feasible solution must be of the form $R^{(1)}_{x_1x_2'}$, $R^{(2)}_{x_2x_3'}$ and $R^{(3)}_{x_3x_1'}$ for some $x_1x_2', x_2x_3', x_3x_1' \in E$.

Consider points of type 1 with the form $(-1+t, 0, 2+t)$. The box $R^{(2)}_{x_2x_3'}$ cannot cover any of them due to the third dimension. The box $R^{(1)}_{x_1x_2'}$ covers all such points corresponding to $t > x_1$, and the box $R^{(3)}_{x_3x_1'}$ covers all such points corresponding to $t \leq x_1'$. So, all points of type 1 are covered iff $x_1 \leq x_1'$. Similarly, all points of type 2 are covered iff $x_1' \leq x_1$. Thus, all points of type 1–2 are covered iff $x_1 = x_1'$. By a symmetric argument, all points of type 3–4 are covered iff $x_3 = x_3'$; and all points of type 5–6 are covered iff $x_2 = x_2'$. We conclude that a feasible solution exists iff a triangle $x_1x_2x_3$ exists in G. QED.

We remark that the boxes above can be made fat, with side lengths between 1 and a constant (by replacing $\mathbb{R}$ with an interval of a sufficiently large constant length).

4.5.3 Unweighted size-3 set cover for unit hypercubes in $\mathbb{R}^4$

Our preceding lower bound for unweighted size-3 set cover for boxes in $\mathbb{R}^3$ immediately implies a lower bound for orthants (and thus unit hypercubes) in $\mathbb{R}^6$, since the point (x, y, z) is covered by the box $[a^-, a^+] \times [b^-, b^+] \times [c^-, c^+]$ in $\mathbb{R}^3$ iff the point (x, x, y, y, z, z) is covered by the orthant $[a^-, \infty) \times (-\infty, a^+] \times [b^-, \infty) \times (-\infty, b^+] \times [c^-, \infty) \times (-\infty, c^+]$ in $\mathbb{R}^6$.

A question remains: can the dimension 6 be lowered? Intuitively, there seems to be some wastage in the above construction: there are several 0's in the coordinates of the points, and several $\mathbb{R}$'s in the definition of the boxes, and these get doubled after the transformation to 6 dimensions. However, it isn't clear how to rearrange coordinates to eliminate this wastage: we would have to give up this nice symmetry of our construction, and there are too many combinations to try. We ended up writing a computer program to exhaustively try all these different combinations, and eventually find a construction that lowers the dimension to 4! Once it is found, correctness is straightforward to check, as one can see in the proof below.

Theorem 4.8. Given a set P of n points and a set R of n unweighted orthants in $\mathbb{R}^4$, deciding whether there exists a size-3 set cover requires $\Omega(n^{4/3-\delta})$ time for any constant $\delta > 0$, assuming the Sparse Triangle Hypothesis.

Proof. We will reduce the triangle detection problem on a graph with m edges to the unweighted size-3 set cover problem for $O(m)$ points and orthants in $\mathbb{R}^4$.

Let $G = (V, E)$ be the given unweighted graph with n vertices and m edges ($n \leq m$). Without loss of generality, assume that $V \subset [0, 0.1]$, and $0 \in V$ and $0.1 \in V$.

The reduction. For each vertex $t \in V$, create six points

$$\begin{array}{rrrrl}
(0+t, & 2+t, & -0.5, & -0.5) & \qquad\text{(type 1)}\\
(2-t, & 0-t, & -0.5, & -0.5) & \qquad\text{(type 2)}\\
(1-t, & 0.5, & 1+t, & 1.5) & \qquad\text{(type 3)}\\
(0.5, & 1+t, & 0.5, & 2-t) & \qquad\text{(type 4)}\\
(-0.5, & -0.5, & 2-t, & 0-t) & \qquad\text{(type 5)}\\
(-0.5, & -0.5, & 0+t, & 1+t) & \qquad\text{(type 6)}
\end{array}$$

Create the following orthants in $\mathbb{R}^4$:

$$\begin{array}{lcl}
\forall x_1x_2' \in E: \ R^{(1)}_{x_1x_2'} & = & [0+x_1, +\infty) \times [0-x_1, +\infty) \times (-\infty, 1+x_2') \times (-\infty, 2-x_2')\\
\forall x_2x_3' \in E: \ R^{(2)}_{x_2x_3'} & = & (-\infty, 1-x_2] \times (-\infty, 1+x_2] \times (0+x_3', +\infty) \times (0-x_3', +\infty)\\
\forall x_3x_1' \in E: \ R^{(3)}_{x_3x_1'} & = & (-\infty, 2-x_1') \times (-\infty, 2+x_1') \times (-\infty, 2-x_3] \times (-\infty, 1+x_3]
\end{array} \tag{4.3}$$

(call them of type 1, 2, and 3, respectively).

Correctness. We prove that a size-3 set cover exists iff a triangle exists in G.

Any feasible solution (if exists) must use an orthant of each type, as one can easily check (like before). So, the three orthants in a feasible solution must be of the form $R^{(1)}_{x_1x_2'}$, $R^{(2)}_{x_2x_3'}$ and $R^{(3)}_{x_3x_1'}$ for some $x_1x_2', x_2x_3', x_3x_1' \in E$.

Consider points of type 1 with the form $(0+t, 2+t, -0.5, -0.5)$. The orthant $R^{(2)}_{x_2x_3'}$ cannot cover any of them due to the third dimension. The orthant $R^{(1)}_{x_1x_2'}$ covers all such points corresponding to $t \geq x_1$, and the orthant $R^{(3)}_{x_3x_1'}$ covers all such points corresponding to $t < x_1'$. So, all points of type 1 are covered iff $x_1 \leq x_1'$. By similar arguments, it can be checked that all points of type 2 are covered iff $x_1 \geq x_1'$; all points of type 3 are covered iff $x_2 \leq x_2'$; all points of type 4 are covered iff $x_2 \geq x_2'$; all points of type 5 are covered iff $x_3 \leq x_3'$; all points of type 6 are covered iff $x_3 \geq x_3'$. We conclude that a feasible solution exists iff a triangle $x_1x_2x_3$ exists in G. QED.

In the computer search, we basically tried different choices of points with coordinate values of the form $c \pm t$ or c for some constant c, and orthants defined by intervals of the form $[c \pm x_j, +\infty)$ or $(-\infty, c \pm x_j]$ (closed or open) for some variable x_j (or x_j'). The constraints are not exactly easy to write down, but are self-evident as we simulate the correctness proof above. Naively, the number of cases is in the order of 10^{14}, but can be drastically reduced to about 10^7 with some optimization and careful pruning of the search space. The C++ code is not long (under 150 lines) and, after incorporating pruning, runs in under a second.

It is now straightforward to modify the above lower bound proof for unweighted orthants (or unit hypercubes) in $\mathbb{R}^4$ to the rectilinear discrete 3-center problem in $\mathbb{R}^4$; see Sec. 4.6.1. In Sec. 4.6.2, we also prove a higher conditional lower bound for weighted size-6 set cover for rectangles in $\mathbb{R}^2$.

4.6 OTHER CONDITIONAL LOWER BOUNDS FOR SMALL-SIZE SET COVER FOR BOXES AND RELATED PROBLEMS

Continuing Section 4.5, we prove a few more conditional lower bounds for set cover related problems.

4.6.1 Rectilinear discrete 3-center in $\mathbb{R}^4$

It is easy to modify our conditional lower bound proof for size-3 set cover for orthants in $\mathbb{R}^4$ (Theorem 4.8) to obtain a lower bound for the rectilinear discrete 3-center problem in $\mathbb{R}^4$.

First recall that a lower bound for orthants in $\mathbb{R}^4$ automatically implies a lower bound for unit hypercubes in $\mathbb{R}^4$. In our construction, we can replace each orthant with a hypercube of side length 10 (i.e., L_∞-radius 5), keeping the corner vertex the same, and then rescale.

For rectilinear discrete 3-center in $\mathbb{R}^4$, the new point set consists of the constructed points and the centers of the constructed hypercubes of side length 10 (before rescaling), together with three auxiliary points:

$$(9.5, 9.5, -8.5, -7.5),\ (-8.5, -8.5, 9.5, 9.5),\ (-7.5, -7.5, -7.5, -8.5).$$

The purpose of the auxiliary points is to force the 3 centers in the solution to be from the centers of the constructed hypercubes.

Corollary 4.2. For any constant $\delta > 0$, there is no $O(n^{4/3-\delta})$-time algorithm for rectilinear discrete 3-center in $\mathbb{R}^4$, assuming the Sparse Triangle Hypothesis.

4.6.2 Weighted size-6 set cover for rectangles in $\mathbb{R}^2$

In Section 4.5, we have obtained a superlinear conditional lower bound for weighted size-3 set cover for rectangles in $\mathbb{R}^2$. Another direction is to investigate the smallest k for which we can obtain a near-quadratic lower bound for weighted size-k set cover. We prove such a result for $k = 6$.

For a fixed $\ell \geq 3$, the *Weighted ℓ-Clique Hypothesis* [205] asserts that there is no $O(n^{\ell-\delta})$ time algorithm for finding an ℓ-clique of minimum total edge weight in an n-vertex graph with $O(\log n)$-bit integer weights.

Theorem 4.9. Given a set P of n points and a set R of n weighted rectangles in $\mathbb{R}^2$, any algorithm for computing the minimum weight size-6 set cover requires $\Omega(n^{2-\delta})$ time for any constant $\delta > 0$, assuming the Weighted 4-Clique Hypothesis.

Proof. We will reduce the minimum-weight 4-clique problem on a graph with n vertices and m edges ($m \in [n, n^2]$) to the weighted size-6 set cover problem for $O(m)$ points and rectangles in $\mathbb{R}^2$. Thus, if there is an $O(m^{2-\delta})$-time algorithm for the latter problem, there would be an algorithm for the former problem with running time $O(m^{2-\delta}) \leq O(n^{4-2\delta})$, refuting the hypothesis.

Let $G = (V, E)$ be the given weighted graph with n vertices and m edges. Without loss of generality, assume that all edge weights are in $[0, 0.1]$, and that $V \subset [0, 0.1]$. Assume that $0 \in V$ and $0.1 \in V$.

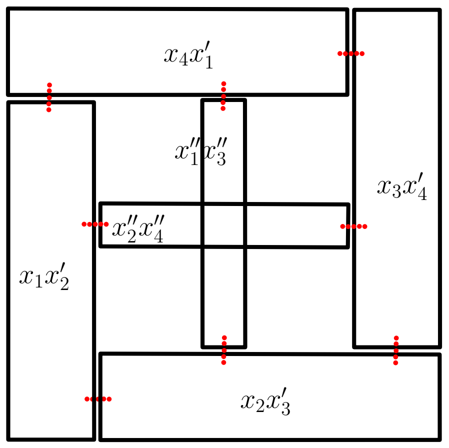


Figure 4.6: Reduction from the minimum-weight 4-clique problem to weighted size-6 set cover for rectangles in $\mathbb{R}^2$.

The reduction. For each vertex $t \in V$, create 8 points

$$\begin{array}{rrl}
(0, & 2-t) & \text{(type 1)} \\
(t, & 0) & \text{(type 2)} \\
(2, & t) & \text{(type 3)} \\
(2-t, & 2) & \text{(type 4)} \\
(1, & t) & \text{(type 5)} \\
(1, & 2-t) & \text{(type 6)} \\
(t, & 1) & \text{(type 7)} \\
(2-t, & 1) & \text{(type 8)}
\end{array}$$

Create the following rectangles in $\mathbb{R}^2$:

$$\begin{array}{rclcll}
\forall x_1x'_2 \in E: \; R^{(1)}_{x_1x'_2} & = & [0, x'_2) & \times & [0, 2-x_1] & \text{(type 1)} \\
\forall x_2x'_3 \in E: \; R^{(2)}_{x_2x'_3} & = & [x_2, 2] & \times & [0, x'_3) & \text{(type 2)} \\
\forall x_3x'_4 \in E: \; R^{(3)}_{x_3x'_4} & = & (2-x'_4, 2] & \times & [x_3, 2] & \text{(type 3)} \\
\forall x_4x'_1 \in E: \; R^{(4)}_{x_4x'_1} & = & [0, 2-x_4] & \times & (2-x'_1, 2] & \text{(type 4)} \\
\forall x''_1x''_3 \in E: \; R^{(5)}_{x''_1x''_3} & = & [1, 1.1] & \times & [x''_3, 2-x''_1] & \text{(type 5)} \\
\forall x''_2x''_4 \in E: \; R^{(6)}_{x''_2x''_4} & = & [x''_2, 2-x''_4] & \times & [1, 1.1] & \text{(type 6)}
\end{array} \qquad (4.4)$$

The weight of each rectangle is set to be the number of points it covers plus the weight of the edge it represents. The total number of points and rectangles is $O(n)$ and $O(m)$ respectively. The reduction is illustrated in Figure 4.6.

Correctness. We prove that the minimum-weight 4-clique in G has weight w (where $w \in [0, 0.6]$) iff the optimal weighted size-6 set cover has weight $8n + w$.

Any feasible solution (if exists) must use a rectangle of each type, since the point $(0, 1.9)$ of type 1 (resp. the point $(0.1, 0)$ of type 2, the point $(2, 0.1)$ of type 3, the point $(1.9, 2)$ of type 4, the point $(1, 0.1)$ of type 5, and the point $(0.1, 1)$ of type 7) can only be covered by a rectangle of type 1 (resp. 2, 3, 4, 5 and 6). So the six rectangles in a feasible solution must be of the form $R^{(1)}_{x_1x'_2}$, $R^{(2)}_{x_2x'_3}$, $R^{(3)}_{x_3x'_4}$, $R^{(4)}_{x_4x'_1}$, $R^{(5)}_{x''_1x''_3}$ and $R^{(6)}_{x''_2x''_4}$ for some $x_1x'_2$, $x_2x'_3$, $x_3x'_4$, $x_4x'_1$, $x''_1x''_3$, $x''_2x''_4 \in E$.

If $x_1 > x'_1$, some point (of type 1) would be uncovered; on the other hand, if $x_1 < x'_1$, some point (of type 1) would be covered twice, and the total weight would then be at least $8n + 1$. Thus, $x_1 = x'_1$. Similarly, $x'_1 = x''_1$, $x_2 = x'_2 = x''_2$, $x_3 = x'_3 = x''_3$, and $x_4 = x'_4 = x''_4$. So, $x_1x_2x_3x_4$ forms a 4-clique in G. We conclude that the minimum-weight solution $R^{(1)}_{x_1x_2}$, $R^{(2)}_{x_2x_3}$, $R^{(3)}_{x_3x_4}$, $R^{(4)}_{x_4x_1}$, $R^{(5)}_{x_1x_3}$ and $R^{(6)}_{x_2x_4}$ correspond to the minimum-weight 4-clique $x_1x_2x_3x_4$ in G. QED.

4.7 CONCLUSIONS

In this chapter, we have obtained a plethora of nontrivial new results on a fundamental class of problems in computational geometry related to discrete k-center and size-k geometric set cover for small values of k. (See Tables 1.1–4.1.) In particular, we have one result where the upper bounds and conditional lower bounds are close: for weighted size-3 set cover for rectangles in $\mathbb{R}^2$, we have given the first subquadratic $\widetilde{O}(n^{7/4})$-time algorithm, and an $\Omega(n^{3/2-\delta})$ lower bound under the APSP Hypothesis.

For all of our results, we have managed to find simple proofs (each with 1–3 pages). We view the simplicity and accessibility of our proofs as an asset—they would make good examples illustrating fine-grained complexity techniques in computational geometry. Generally speaking, there has been considerable development on fine-grained complexity in the broader algorithms community over the last decade [205], but to a lesser extent in computational geometry. A broader goal of this chapter is to encourage more work at the intersection of these two areas. We should emphasize that while our conditional lower bound proofs may appear simple in hindsight, they are not necessarily easy to come up with; for example, see one of our proofs that require computer-assisted search (Theorem 4.8).

Additional remarks. For "combinatorial" algorithms (i.e., algorithms that avoid fast matrix multiplication or algebraic techniques), some of our conditional lower bounds can be improved. For example, our near-$n^{4/3}$ lower bounds on unweighted size-3 set cover for boxes in $\mathbb{R}^3$ and rectilinear discrete 3-center in $\mathbb{R}^4$ (Theorems 4.7–4.8 and Corollary 4.2) can be increased to near-$n^{3/2}$, under the Combinatorial BMM Hypothesis [205] (which asserts that Boolean matrix multiplication of two $n \times n$ matrices requires $\Omega(n^{3-\delta})$ time for combinatorial algorithms). These follow from the same reductions, because the Combinatorial BMM Hypothesis is equivalent [206] to the hypothesis that triangle detection in a graph with n vertices requires $\Omega(n^{3-\delta})$ time for combinatorial algorithms. Note that although the notion of "combinatorial" algorithms is vague, all our algorithms in Sec. 4.3 and Sec. 4.4 are combinatorial. (However, the best combinatorial algorithm we are aware of for Euclidean discrete 2-center in higher dimensions are slower than the best noncombinatorial algorithm, and has time bound near $n^{3-1/O(d)}$ using range searching techniques.)

It is interesting to compare our conditional lower bound proofs with the known hardness proofs by Marx [163] and Chitnis and Saurabh [85] on fixed-parameter intractibility with respect to the parameter k. These were also obtained by reduction from k'-clique for some function relating k and k'. Thus, in principle, by setting $k' = 3$, they should yield superlinear lower bounds for discrete k-center (and related unweighted set cover problems) for some specific constant k, but this value of k is probably large (much larger than 3). On the other hand, Cabello et al.'s proofs of fixed-parameter intractibility with respect to d [50] also produced reductions from k'-clique to Euclidean 2-center and rectilinear 4-center, but the continuous problems are different from the discrete problems. (Still, Cabello et al.'s proofs appeared to use some similar tricks, though ours are simpler.)

One of our initial reasons for studying the fine-grained complexity of small-size geometric set cover is to prove hardness of approximation for fast approximation algorithms for geometric set cover. For example, Theorem 4.7 (with its subsequent remark) implies that no $\widetilde{O}(n)$-time approximation algorithm for set cover for fat boxes in $\mathbb{R}^3$ with side lengths $\Theta(1)$ can achieve approximation factor strictly smaller than $4/3$ (otherwise, it would be able to decide whether the optimal size is 3 or at least 4). For such fat boxes, near-linear-time approximation algorithms with some (large) constant approximation factor follow from known techniques [14, 71] (since fat boxes of similar sizes in $\mathbb{R}^3$ have linear union complexity); see Chapter 3.

CHAPTER 5: ENCLOSING POINTS WITH GEOMETRIC OBJECTS

In this chapter, we develop approximation algorithms for the problem of enclosing all points with geometric objects (Enclosing-All-Points), which is related to geometric set cover. We solve this problem by adapting techniques for approximating geometric set cover.

A generalized colored enclosing problem. To solve the Enclosing-All-Points problem (Problem 1.5), we design a polynomial-time approximation algorithm that in fact works for a generalized version of it, where each input object has a color. The goal is to ensure that each point is enclosed by a subset of the chosen objects with the same color. The colored version of the Enclosing-All-Points problem is formally defined as follows:

Problem 5.1 (Enclosing-All-Points-Colored)**.** Given a set X of points and a set S of colored geometric objects (possibly overlapping) in $\mathbb{R}^2$, find the smallest subset $S^* \subseteq S$ to enclose all points in X. More precisely, for each point $p \in X$, there exists a subset $S^*_{c_p} \subseteq S^*$ of objects with the same color c_p that encloses p, i.e., any curve connecting p with infinity $(+\infty, 0)$ must intersect at least one object in S^* with color c_p.

The previous (uncolored) Enclosing-All-Points problem (Problem 1.5) can be viewed as a special case of the above Enclosing-All-Points-Colored problem (Problem 5.1), since in the colored version, we can let all input objects have the same color.

Hardness. It is easy to see that geometric set cover is also a special case of the above colored enclosing problem, by assigning a unique color for each object in Problem 5.1. In this way, if in the colored enclosing problem a point $p \in X$ is enclosed by objects with the same color c_p, then in the set cover problem, p must be covered by the unique object with color c_p. Therefore, Problem 5.1 is NP-hard for a wide range of geometric objects, e.g., unit disks or unit squares, because of the known hardness results for geometric set cover [120].

5.1 OUR RESULTS

For enclosing all points with colored geometric objects (Problem 5.1), we obtain the following results:

- A polynomial-time $O(\alpha(n) \log n)$-approximation algorithm for enclosing all points in $\mathbb{R}^2$ using colored arbitrary line segments (Theorem 5.1).

- A polynomial-time $O(\log n)$-approximation algorithm for enclosing all points in $\mathbb{R}^2$ using colored disks (Theorem 5.2).

- More generally, our results hold for curves that pairwise intersect only at most $s = O(1)$ times, achieving $O(\frac{\lambda_{s+2}(n)}{n} \log n)$-approximation (Theorem 5.3).

We remark that in particular, these results hold for the original (uncolored) Enclosing-All-Points problem as a special case. To the best of our knowledge, no polynomial-time approximation algorithms were known for these problems before our work.

5.2 PRELIMINARIES

Union complexity of geometric objects. For a type of geometric objects, its *union complexity* is defined as the combinatorial complexity of the boundary of the union of n such objects. It is known that disks have linear union complexity [146].

For our applications, it suffices to look at the combinatorial complexity of the outer face in the arrangement of the objects, which is upper-bounded by the union complexity of the objects. Therefore for disks, this complexity is still $O(n)$. Pollack et al. [185] proved that for n arbitrary line segments in $\mathbb{R}^2$, the complexity of the outer face in the arrangement is bounded by $O(n\alpha(n))$ (by Davenport-Schinzel sequences), where $\alpha(n)$ denote the inverse Ackermann function.

More generally, for curves that pairwise intersect only at most $s = O(1)$ times, a similar bound holds: the combinatorial complexity of the outer face in the arrangement is $O(\lambda_{s+2}(n))$, where the function $\lambda_s(n) = n\alpha(n)^{O(\alpha(n)^{s-3})}$ is the maximum length of an (n, s) Davenport-Schinzel sequence [16, 132]. For any constant s, $\lambda_s(n)$ is almost linear in n.

Point-in-polygon check. Given a point q and a polygon P, the point-in-polygon problem asks whether q lies inside, outside, or on the boundary of P. A standard method for solving this problem is to use the ray-casting algorithm [195]: cast a ray $\vec{r}$ starting from q and going in any fixed direction, if $\vec{r}$ intersects P an even number of times, then q is outside P; otherwise if $\vec{r}$ intersects P an odd number of times, then q is inside P. (For our applications, we consider the points on the boundary of P as inside P.)

Another method for point-in-polygon check is by computing the *winding number* of q with respect to P, denote as $wind(q, P)$. The idea is to modify the ray-casting algorithm as follows: first orient the edges of P in counter-clockwise direction. Initialize a counter c_q with value 0. Cast a ray $\vec{r}_q$ starting from q and going in any fixed direction, for each edge $\vec{e}$ of P intersecting $\vec{r}_q$, if $\vec{e}$ crosses $\vec{r}_q$ in counter-clockwise direction, then increase c_q by 1;

otherwise if $\vec{e}$ crosses $\vec{r}_q$ in clockwise direction, decrease c_q by 1. In the end, q is inside P iff the counter c_q is nonzero. In particular, when P is a simple polygon, q is inside P iff $c_q = 1$. Intuitively, if q is outside P, then the weighted crossings between the ray $\vec{r}_q$ and the polygon P will cancel each other. An example for this process is shown in Fig. 5.1.

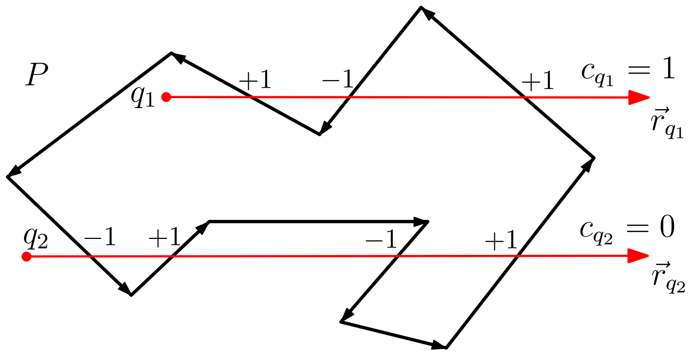


Figure 5.1: The winding number of points with respect to a (simple) polygon P. The point q_1 has winding number $c_{q_1} = 1$, and is inside P; the point q_2 has winding number $c_{q_2} = 0$, and is outside P.

5.3 APPROXIMATION ALGORITHMS

Here we present approximation algorithms for enclosing points with colored geometric objects. Our approach works for disks, arbitrary line segments, and more generally curves that pairwise intersect $O(1)$ times. In the following, we explain our approach in terms of line segments as an example.

A common technique for designing approximation algorithms for set cover related problems is to use LP rounding. The idea is to first design and solve an LP relaxation (i.e., getting a fractional solution), and then round the LP solution to get an integral solution. Our algorithm follows this line of approach, which requires nontrivial geometric insights for designing and rounding the LP, as we will show.

Forming the LP relaxation. We first formalize an LP relaxation of the colored enclosing problem. Intuitively, the idea is as follows: a point q is enclosed by a set of line segments S_c of color c, iff the outer boundary of these line segments enclose q. So it suffices to specify the outer boundary $\mathcal{B}_c$ of the union of line segments with color c that we select in the solution. $\mathcal{B}_c$ is a set of disjoint simple polygons (as shown in Fig. 5.2), which can also be viewed as a set of cycles on a graph G_c: the vertices of G_c are intersection points between two line segments in S_c, and the edges of G_c are subsegments between a pair of vertices in G_c that lie on the same segment in S_c. To get a fractional solution, our goal is to select a set of colored fractional cycles on G_c to enclose all points. We can form the constraints for each color class c separately.

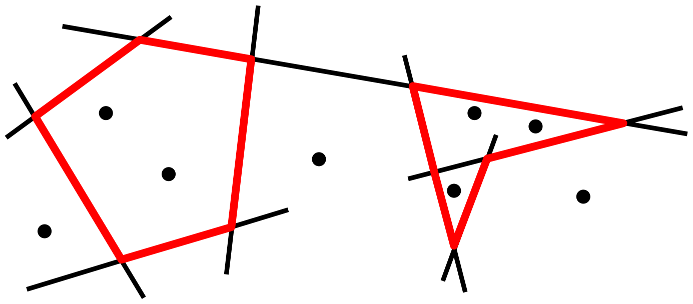


Figure 5.2: The outer boundary $\mathcal{B}_c$ of the union of line segments with color c is a set of disjoint simple polygons (shown in red).

We now formally describe our LP. We start with specifying the variables in the LP. For each color class c, let S_c denote the set of the input line segments with color c. The arrangement of S_c can be viewed as a planar graph Arr_c, which has $O(|S_c|^2)$ vertices and edges. Each vertex in Arr_c is an intersection point of two line segments in S_c. We then create a new graph G_c, using the same set of vertices as Arr_c. For each line segment $\ell \in S_c$, let $V(\ell)$ denote the set of all vertices in G_c that lie on ℓ. For each pair of points $u, v \in V(\ell)$, create a variable $x_{uv}^{(\ell)}$ where $0 \le x_{uv}^{(\ell)} \le 1$, indicating the directed subsegment uv of ℓ (with direction from u to v) is selected $x_{uv}^{(\ell)}$ times in the fractional solution. We call $x_{uv}^{(\ell)}$ the weight of the subsegment uv. Also create a directed edge $e_{uv}^{(\ell)}$ from u to v in G_c.

We emphasize that we need to create two variables (and respectively, two directed edges in G_c) for each undirected subsegment uv, because to ensure all points are enclosed according to the winding number constraints to be defined later, we need to know whether uv appears on the boundary of the solution in clockwise or counter-clockwise order.

We next present the formulas for the constraints of our LP. There are two types of constraints that need to be handled:

1. *Flow constraints.* To ensure that the solution to the LP is a set of colored fractional cycles, we design the flow constraints as follows:

For each color c, and for each vertex u of G_c, we create the constraint

$$\sum_{v,\ell:\ e_{uv}^{(\ell)} \in E(G_c)} x_{uv}^{(\ell)} = \sum_{v,\ell:\ e_{vu}^{(\ell)} \in E(G_c)} x_{vu}^{(\ell)}. \tag{5.1}$$

The solution to the LP with respect to each color c is a circulation on G_c. Using standard techniques, this circulation can be decompose into $O(\sum_c |E(G_c)|) = O(\sum_c |S_c|^2) = O(n^2)$ fractional cycles in polynomial time: repeatedly select the smallest variable $x_{uv}^{(\ell)}$ that is strictly positive, find a path P_{vu} from v to u on G_c using only edges with strictly positive values (such path always exists, and the value of each edge on the path is at least $x_{uv}^{(\ell)}$), and subtract the fractionally weighted cycle $\mathcal{C}$ formed by combining $e_{uv}^{(\ell)}$ and P_{vu} from the solution. The fractional cycle $\mathcal{C}$ has weight $w_{\mathcal{C}} = x_{uv}^{(\ell)}$. Each round sets the value of at least one variable $x_{uv}^{(\ell)}$ to 0, so the process will terminate in $O(|E(G_c)|)$ steps. Each cycle on G_c corresponds to a monochromatic polygon with color c, so the LP solution can be decomposed into $O(n^2)$ fractionally weighted monochromatic polygons. We remark that these polygons are not necessarily simple.

2. *Winding number constraints.* To ensure that each point in X is enclosed by the outer boundary of the union of the chosen subsegments, the idea is to constrain their winding number. After orienting each (unknown) simple polygon on the outer boundary of the optimal solution S^* in counter-clockwise order, the winding number of each point $q \in X$ is exactly 1. So it suffices to ensure a similar inequality when we form the constraints for the fractional solution to the LP.

For each point $q \in X$, let $\vec{r}_q$ be an arbitrary ray starting from q. Inspired by the winding-number algorithm, we add the following constraint to ensure the winding number of q with respect to the LP solution that we selected is at least 1:

$$\sum_{c,\ell,u,v:\ e_{uv}^{(\ell)} \text{ crosses } \vec{r}_q \text{ in counter-clockwise order}} x_{uv}^{(\ell)} - \sum_{c,\ell,u,v:\ e_{uv}^{(\ell)} \text{ crosses } \vec{r}_q \text{ in clockwise order}} x_{uv}^{(\ell)} \geq 1,\ \forall q \in X. \tag{5.2}$$

The (weighted) winding number of q equals to the total weight of the edges that cross the ray $\vec{r}_q$ in counter-clockwise direction, minus the total weight of the edges that cross the ray $\vec{r}_q$ in clockwise direction.

We remark that we require the winding number of each point to be at least 1 instead of exactly 1, since in the LP solution, each polygon may not be simple, which means the winding numbers can be greater than 1.

The winding number constraints together with the flow constraints ensure that each point is enclosed by the fractional cycles in the LP solution. Namely, for each point $p \in X$, we have

$$\sum_{\text{cycle } \mathcal{C}} w_{\mathcal{C}} \cdot wind(q, \mathcal{C}) \geq 1. \tag{5.3}$$

The objective function is to minimize the total weight of the (fractionally) selected subsegments:

$$\mathsf{opt} = \min \sum_{c,\ell,u,v:\ e_{uv}^{(\ell)} \in E(G_c)} x_{uv}^{(\ell)}. \tag{5.4}$$

Rounding the LP. After solving the LP to compute the optimal fractional solution in polynomial time, we represent the LP solution as a set of fractionally weighted monochromatic cycles on the disjoint union of graphs G_c, then rounding the fractional solution to obtain an integral solution. This is a standard technique that is commonly used for approximating set cover solutions [208].

Here we provide a simple self-contained description, which is by using randomized rounding [189]. Namely, for each cycle $\mathcal{C}$ of weight $w_\mathcal{C}$, we randomly and independently select it with probability $\min\{10 w_\mathcal{C} \log n, 1\}$. Here selecting a cycle means selecting all subsegments contained as edges on that cycle, and if a subsegment $e_{uv}^{(\ell)}$ of an input line segment ℓ is selected, then the whole segment ℓ is included in our integral solution. Note that a line segment ℓ may be included because of multiple subsegments of it are selected during the rounding process; although it suffices to include ℓ just once, in the analysis below, we treat as if ℓ would be included multiple times. We show that this will not affect the approximation factor by much, because the combinatorial complexity of the outer boundary of the objects we consider is close to linear.

Our goal is to ensure all points $q \in X$ have winding number ≥ 1 w.h.p. after the rounding process, which implies that they are enclosed by the integral solution, but a technical issue arises for cycles that are not simple: they may wind around a point multiple times, resulting in a large positive winding number for the point. In this case, even giving a small weight to the cycle can still ensure the (weighted) winding number of that point is at least 1 in the fractional solution to the LP, but that means the cycle will only be selected with a small probability during the rounding process (selecting the cycle with a larger probability would be too costly).

To handle the above issue, the idea is to unwind the non-simple cycles first before rounding, i.e., decompose them into a set of simple cycles, and then randomly select each simple cycle independently. This ensures that each cycle $\mathcal{C}$ in the fractional solution contributes at most once to the winding number of any point q.

To unwind the non-simple cycles, we repeatedly choose a self-intersection point v of the cycle $\mathcal{C}$. Suppose v is the intersection point of a pair of crossing edges u_1v_1 and u_2v_2 of $\mathcal{C}$. We create a new vertex at v, subdivide the edge u_1v_1 into two edges u_1v and vv_1, and similarly subdivide u_2v_2 into two edges u_2v and vv_2. In this way, we decompose the cycle $\mathcal{C}$ into two cycles $\mathcal{C}_1$ and $\mathcal{C}_2$ by splitting at v. The total length of the cycles increases by 2, since the two crossing edges are split into four edges. Recursively subdivide the cycles $\mathcal{C}_1$ and $\mathcal{C}_2$ if they are not simple. An example for the unwinding process is shown in Fig. 5.3.

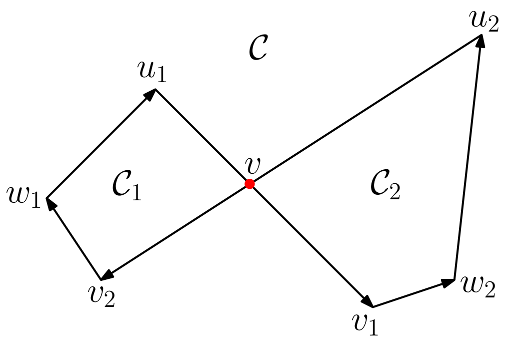


Figure 5.3: During the unwinding process, the non-simple cycle $\mathcal{C} = u_1v_1w_2u_2v_2w_1$ with length 6 is decomposed into two simple cycles $\mathcal{C}_1 = vv_2w_1u_1$ and $\mathcal{C}_2 = vv_1w_2u_2$ each with length 4.

To upper bound the approximation ratio, we show that it only loses a constant approximation factor at the step of decomposing non-simple cycles into simple cycles. Let $f(n)$ denote the maximum total number of edges that a non-simple cycle of length n can decompose into. $f(n)$ satisfies the recurrence

$$f(n) \leq \max_{3 \leq n_1 \leq n-1} \left(f(n_1) + f(n - n_1 + 2)\right), \tag{5.5}$$

since a simple cycle formed by line segments would have length at least 3. The edge case is $f(n) = n$ for $n \leq 3$. Solving the recurrence, we get $f(n) \leq 3n - 6 = O(n)$.

After decomposing the cycles into simple cycles, it still holds that for each point $q \in X$,

$$\sum_{\text{cycle } \mathcal{C}} w_{\mathcal{C}} \cdot wind(q, \mathcal{C}) \tag{5.6}$$

$$= \sum_{\text{simple cycle } \mathcal{C}} w_{\mathcal{C}} \cdot wind(q, \mathcal{C}) \tag{5.7}$$

$$= \sum_{\text{simple counter-clockwise cycle } \mathcal{C}\colon\ \mathcal{C} \text{ encloses } q} w_{\mathcal{C}} - \sum_{\text{simple clockwise cycle } \mathcal{C}\colon\ \mathcal{C} \text{ encloses } q} w_{\mathcal{C}} \tag{5.8}$$

$$\geq 1. \tag{5.9}$$

We can ignore the simple cycles that appear in clockwise order, because they only contribute negatively to the winding number constraints. If we only consider the counter-clockwise cycles, it still holds that

$$\sum_{\text{simple counter-clockwise cycle } \mathcal{C}\text{: } \mathcal{C} \text{ encloses } q} w_{\mathcal{C}} \geq 1. \tag{5.10}$$

Now we analyze the probability of failure. After the rounding process, for any $q \in X$,

$$\begin{aligned}
&\Pr[q \text{ is not enclosed}] && (5.11)\\
&= \prod_{\text{simple counter-clockwise cycle } \mathcal{C}\text{: } \mathcal{C} \text{ encloses } q} \Pr[\mathcal{C} \text{ is not selected}] && (5.12)\\
&= \prod_{\text{simple counter-clockwise cycle } \mathcal{C}\text{: } \mathcal{C} \text{ encloses } q} (1 - \min\{10 w_{\mathcal{C}} \log n, 1\}) && (5.13)\\
&\leq \lim_{k\to\infty} \left(1 - \frac{10\log n}{k}\right)^k && (5.14)\\
&\leq e^{-10\log n} \leq \frac{1}{n^{10}}. && (5.15)
\end{aligned}$$

By union bound, the probability of existing any point $q \in X$ that is not enclosed is at most

$$\Pr[\exists q \in X,\ q \text{ is not enclosed}] \leq n \cdot \frac{1}{n^{10}} \leq \frac{1}{n^9}. \tag{5.16}$$

Therefore, the rounding process succeeds w.h.p. We remark that the approach can be derandomized, by the standard method of conditional probabilities [188].

Analyzing the approximation ratio. It is easy to see that the optimal solution S^* corresponds to a feasible solution to the LP: if for each color class c, set the variables corresponding to the directed subsegments on the outer boundary (ordered counter-clockwise) of the line segments S^*_c with color c in the optimal solution to be 1, and let all other variables being 0, we will obtain a feasible solution to the LP. From the known combinatorial complexity of the outer face in the arrangement of line segments, the outer boundary of S^*_c can be decomposed into $O(|S^*_c| \cdot \alpha(|S^*_c|))$ subsegments. Therefore the optimal LP solution satisfies

$$\mathsf{opt} \leq O\left(\sum_c |S^*_c| \cdot \alpha(|S^*_c|)\right) \leq O\left(|S^*| \cdot \alpha(|S^*|)\right). \tag{5.17}$$

As we have shown before, after the unwinding step, the total weight of edges among all cycles only increases by a constant factor, which is still $O(\mathsf{opt})$. So after performing randomized rounding on the unwound fractional solution, the expected size of the obtained integral solution is at most

$$
\begin{aligned}
&\sum_{\text{simple counter-clockwise cycle } \mathcal{C}} 10 w_{\mathcal{C}} \log n \cdot O\left(|\mathcal{C}|\right) && (5.18)\\
&= 10 \log n \cdot \sum_{\text{simple counter-clockwise cycle } \mathcal{C}} \ \sum_{\ell,u,v\colon\ e_{uv}^{(\ell)} \in \mathcal{C}} w_{\mathcal{C}} && (5.19)\\
&= O(\log n) \cdot \sum_{\ell,u,v} \ \sum_{\text{simple counter-clockwise cycle } \mathcal{C}\colon\ \mathcal{C} \ni e_{uv}^{(\ell)}} w_{\mathcal{C}} && (5.20)\\
&\leq O(\log n) \cdot \sum_{\ell,u,v} \ \sum_{\text{cycle } \mathcal{C}\colon\ \mathcal{C} \ni e_{uv}^{(\ell)}} w_{\mathcal{C}} && (5.21)\\
&\leq O(\log n) \cdot \sum_{\ell,u,v} x_{uv}^{(\ell)} && (5.22)\\
&\leq O(\mathsf{opt} \log n) && (5.23)\\
&\leq O\left(|S^*| \cdot \alpha(|S^*|) \log n\right). && (5.24)
\end{aligned}
$$

In summary, the problem of enclosing points by geometric objects can be viewed as selecting the smallest subset of objects to cover all curves connecting a point q with infinity (i.e., a set cover problem). The difficulty is there are infinitely many such curves, so directly applying the known set cover results does not give us a good approximation. Using the winding number idea, we transform the problem into covering each of a finite number of rays with a finite number of possible boundary curves of the input objects, so that we can apply the set cover approximation techniques.

Using the above algorithm, we obtain the following approximation result for enclosing all points in $\mathbb{R}^2$ using (colored) line segments:

Theorem 5.1. *Given a set X of points and a set S of colored arbitrary line segments in $\mathbb{R}^2$, where $|X| + |S| = n$, there exists a polynomial-time $O(\alpha(n) \log n)$-approximation algorithm for enclosing all points in X using the smallest subset of objects in S.*

Disks and more general types of curves. The technique we described above can be modified to work for more general types of curves with minor changes. For curves that pairwise intersect only $s = O(1)$ times, one technical issue is during unwinding, a simple cycle may only have length 2. If we decompose a cycle $\mathcal{C}$ into a cycle $\mathcal{C}_1$ with length 2 and another cycle $\mathcal{C}_2$, then the length of $\mathcal{C}_2$ is the same as $\mathcal{C}$, therefore not getting a good recurrence for the maximum total size $f(n)$ of the simple cycles that we decompose into.

To fix this issue, we first try to unwind at a self-intersection point v of $\mathcal{C}$, such that the two subcycles have length strictly less than $|\mathcal{C}|$. If such point exists, then our previous analysis for the recurrence of $f(n)$ still works. Otherwise if there are no such points, then it means only pairs of consecutive edges on the cycle $\mathcal{C}$ can intersect. In this case, there are only at most $s \cdot |\mathcal{C}|$ self-intersections on $\mathcal{C}$, so we can decompose $\mathcal{C}$ into $O(s|\mathcal{C}|)$ subcycles, increasing the total number of edges by only a factor of $O(s) = O(1)$.

Since it is known that the outer boundary for disks has complexity $O(n)$, we get the following result for disks:

Theorem 5.2. Given a set X of points and a set S of colored disks in $\mathbb{R}^2$, where $|X|+|S| = n$, there exists a polynomial-time $O(\log n)$-approximation algorithm for enclosing all points in X using the smallest subset of objects in S.

More generally, for curves that pairwise intersect at most s times, it is known that the combinatorial complexity of the outer face in the arrangement is bounded by $O(\lambda_{s+2}(n))$ [16, 132] as mentioned earlier in Sec. 5.2. So our approximation algorithm yields the following result:

Theorem 5.3. Given a set X of points and a set S of colored curves that pairwise intersect at most $s = O(1)$ times in $\mathbb{R}^2$, where $|X| + |S| = n$, there exists a polynomial-time $O(\frac{\lambda_{s+2}(n)}{n} \log n)$-approximation algorithm for enclosing all points in X using the smallest subset of objects in S.

CHAPTER 6: MONOTONE CONVEX CHAIN COVER

In this chapter, we inspect the FPT status of the monotone convex chain cover problem (Problem 1.6), which asks for finding the minimum number of x-monotone convex chains $\kappa(P)$ that can together cover a given point set P. One open question is whether this problem or the related convex chain cover problem is FPT [112, Open Problem 11.16]. Here we show that this problem is not likely to have a polynomial kernel, taking a step towards resolving the parameterized complexity of the problem.

6.1 OUR RESULTS

We obtain two hardness results on the monotone convex chain cover problem:

- We show that computing the monotone convex cover number $\kappa(P)$ for a set of points P in $\mathbb{R}^2$ is NP-hard (Theorem 6.1).
- Furthermore, we show the problem of deciding whether $\kappa(P) \leq k$ does not have a polynomial kernel, unless NP $\subseteq$ coNP/poly (Theorem 6.2).

6.2 NP-HARDNESS

We first show that the monotone convex chain cover problem is NP-hard, which serves as a building block for proving nonexistence of polynomial kernels later.

It is known that computing the convex cover number $\kappa_c(P)$ for a given point set P is NP-hard, as shown by Arkin et al. [22] (which turns out to be inspired by the hardness proof for Angular Metric TSP by Aggarwal et al. [18]). Since convex chain cover and monotone convex chain cover are closely related, by slightly modifying their proof, we are able to prove a similar hardness result for monotone convex chain cover.

Theorem 6.1. Given a set of points P in the plane and an integer k, it is NP-hard to decide whether the monotone convex cover number $\kappa(P)$ of P is at most k.

We first sketch the argument of Arkin et al.'s NP-hardness proof, then highlight the modifications we made for proving NP-hardness for monotone convex chain cover.

Proof. Arkin et al.'s NP-hardness proof for convex cover number is via a reduction from the 1-in-3 SAT problem, i.e., deciding whether there exists a satisfying assignment of the given 3-SAT formula, such that exactly one literal in each clause is true. The 1-in-3 SAT problem

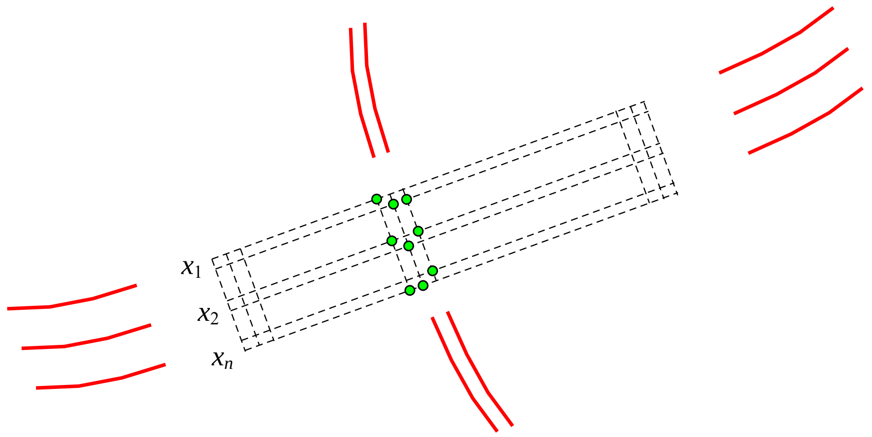


Figure 6.1: The point set P_I we create for a 1-in-3 SAT instance I for reducing to the monotone convex chain cover problem. Each red curve illustrates a gadget that we create, which is an x-monotone convex chain with sufficiently large size. Two red curves are paired together to form a "staple" gadget. The pivot points for the clause $(x_1 \vee \overline{x_2} \vee x_n)$ are shown in green.

is known to be NP-hard [191]. In their reduction, for each variable x_i they created a pair of rows: if the upper row is covered by a single chain, then the variable x_i is assigned to be "true", otherwise "false". For each clause c_j in the formula, they created three columns, where each column corresponds to a variable in the clause (so each clause is represented using a 2×3 grid). Then they added "pivot" points (the green points illustrated in Fig. 6.1) to encode the variable-clause incidence information: if x_i (resp., $\overline{x_i}$) appears in c_j, then create a point at the intersection of x_i's "true" (resp., "false") row and the column representing x_i in c_j, and for the other two columns in c_j, add a point at their intersection with x_i's "false" (resp., "true") row. Finally, they added "staple" gadgets to the rows and columns each containing sufficiently many points which are in convex position, one staple for each variable and two staples for each clause. Each horizontal staple can cover a row (and only that), and each vertical staple can cover a column (and only that). The aim is to use those gadgets as convex chains to cover all the "pivot" points. They proved that the 1-in-3 SAT instance I is satisfiable iff $\kappa_c(P_I) \leq n + 2m$, where n is the number of variables and m is the number of clauses.

Our modifications for monotone convex chain cover are as follows. Here one can only use x-monotone convex chains to cover the points, therefore one needs to replace each "staple" gadget with a pair of x-monotone convex chains (the red curves illustrated in Fig. 6.1), each is slightly perturbed from a straight line segment and contains $\Omega(n^4)$ points, which is sufficiently large to ensure each pair of such monotone convex chains will be selected as a whole. As before, each "staple" can be linked as a single x-monotone convex chain that is able to cover all the pivot points in one row or one column, but cannot cover any other pivot point. We carefully set the positions and the slopes of the monotone convex chains to ensure each pair of convex chains that is created will be contained in a single x-monotone convex chain in the optimal solution. One need to also slightly rotate the whole point set P_I counterclockwise by a sufficiently small angle, in order to ensure x-monotonicity of the (nearly vertical) chains that we want to appear in the optimal solution. An example of our modified reduction construction is shown in Fig. 6.1. The rest of the correctness proof follows from Arkin et al.'s work. QED.

Remark. Intuitively, the monotone convex chain cover problem is hard to solve, since it is known that covering points by lines is NP-hard [103, 170] and even APX-hard [45, 157], and monotone convex chains are more complicated geometric objects than lines. However, we are not aware of a direct reduction between these two problems.

6.3 NO POLYNOMIAL KERNEL

Next, we show that the monotone convex chain cover problem does not have a polynomial sized kernel, unless NP $\subseteq$ coNP/poly, which is believed to be unlikely, as it will imply that the polynomial hierarchy collapses [51, 214]. For more discussions on this assumption, see the book on complexity theory by Arora and Barak [26]. For basic definitions related to FPT and kernels, see Sec. 1.6.

Composing problem instances. The key observation is that given t point sets $P_1, \ldots, P_t$, we can create a new point set $\hat{P}$, such that its monotone convex cover number $\kappa(\hat{P}) \leq k$ iff $\kappa(P_i) \leq k$ for all $1 \leq i \leq t$.

The detailed construction is as follows. Note that convexity is preserved under affine transformations, in particular, translation, rotation and scaling. The idea is to apply an affine transformation on each point set P_i to get a new point set $\hat{P}_i$, and let $\hat{P}$ be the disjoint union of all $\hat{P}_i$'s. We first scale each point set P_i to ensure that $\hat{P}_i$ is contained in an axis-aligned bounding box which is thin. In particular, after applying appropriate horizontal and vertical scaling, we can w.l.o.g. assume the width of the bounding box is 1, and the absolute value of the slope between each pair of points in $\hat{P}_i$ is bounded by ε, where $\varepsilon = \varepsilon(t)$ is a sufficiently small value depending only on t. As a consequence, the height of the bounding box is bounded by 2ε. Next, we rotate the bounding box of $\hat{P}_i$ by $\frac{i}{t}$ radians counterclockwise, and finally translate the bounding box to let its lower left corner lie on coordinate $(\sum_{j=0}^{2i-3} \cos \frac{j}{2t}, \sum_{j=0}^{2i-3} \sin \frac{j}{2t})$, specifically $(0, 0)$ for $\hat{P}_1$. See Fig. 6.2 for an example.

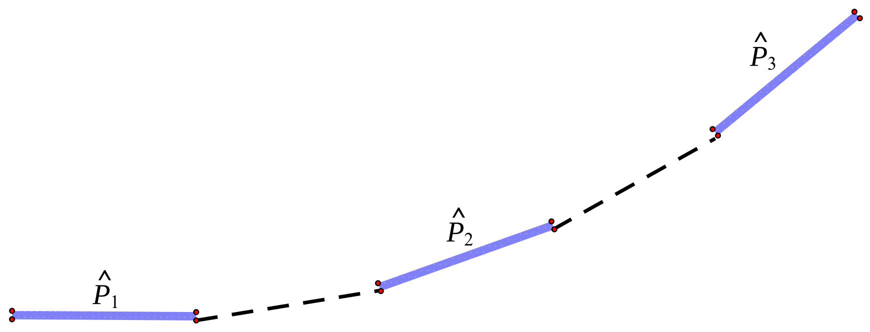


Figure 6.2: The point set $\hat{P}$ created from composing P_1, P_2 and P_3. Each transformed point set $\hat{P}_i$ is contained in a bounding box shown in blue.

The above construction ensures that an x-monotone convex chain in P_i is still an x-monotone convex chain in $\hat{P}$ after applying the corresponding affine transformation. Moreover, for any pair $i < j$, we can concatenate any x-monotone convex chain in $\hat{P}_i$ with any other x-monotone convex chain in $\hat{P}_j$, to get a single x-monotone convex chain in $\hat{P}$ after the transformation. Therefore $\kappa(\hat{P}_i \cup \hat{P}_j) \leq \max\{\kappa(\hat{P}_i), \kappa(\hat{P}_j)\}$ by greedily concatenating the chains in $\hat{P}_i$ with the chains in $\hat{P}_j$. On the other hand, clearly $\kappa(\hat{P}_i) \leq \kappa(\hat{P}_i \cup \hat{P}_j)$ as $\hat{P}_i \subseteq \hat{P}_i \cup \hat{P}_j$, and similarly $\kappa(\hat{P}_j) \leq \kappa(\hat{P}_i \cup \hat{P}_j)$. Thus, $\kappa(\hat{P}_i \cup \hat{P}_j) = \max\{\kappa(\hat{P}_i), \kappa(\hat{P}_j)\}$.

As a result, $\hat{P}$ can be covered by at most k x-monotone convex chains if and only if each part P_i can be covered by at most k x-monotone convex chains.

AND-composition. The above seemingly simple construction implies we can compose multiple instances of monotone convex chain cover into a single instance, where the answer for the combined instance equals to the Boolean AND of the answers for those instances. The next step is to use the framework of Bodlaender et al. [40, 41] to prove that the monotone convex chain cover problem does not have a polynomial kernel, unless NP $\subseteq$ coNP/poly. To begin with, we introduce the concept of AND-composition.

Definition 6.1 (AND-composition [40]). Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. An AND-composition algorithm for L is an algorithm that takes as input a sequence $((x_1, k), \ldots, (x_t, k))$ where $(x_i, k) \in \Sigma^* \times \mathbb{N}^+ \ \forall\ 1 \leq i \leq t$, runs in time polynomial in $\sum_{i=1}^{t} |x_i| + k$, and outputs $(y, k') \in \Sigma^* \times \mathbb{N}^+$ such that:
(a) $(y, k') \in L$ iff $(x_i, k) \in L \ \forall\ 1 \leq i \leq t$, and
(b) $k' = \text{poly}(k)$.

We claim that our above construction gives an AND-composition algorithm for the parameterized language of the monotone convex chain cover problem. One can encode a point set P containing n points using poly(n) bits, because one only need the information about the rank of the x-coordinates of the points, and the convexity of each triple of points. Namely, the input can be encoded using $O(n^3)$ bits. Using our construction, in time polynomial in the total size of the problem instances, t problem instances each with parameter k can be combined into a single instance with parameter $k' = k$. It is easy to verify that the requirements within the definition of AND-composition are met.

Bodlaender et al. [40] showed that if an AND-compositional parameterized language L has a polynomial kernel, and if its unparameterized version $\tilde{L}$ is NP-complete, then NP $\subseteq$ coNP/poly. Combining our AND-composition construction in this section with the NP-hardness result in Sec. 6.2 (Theorem 6.1), we obtain the following theorem:

Theorem 6.2. The monotone convex chain cover problem parameterized by k does not admit a polynomial kernel, unless NP $\subseteq$ coNP/poly.

We remark that natural problems which are known to admit AND-compositions are rare (apart from the ones about computing certain graph parameters, e.g., the treewidth, where simply taking the disjoint union of the graphs yields an AND-composition) [97]. Our work explores the geometry of monotone convex chain cover, in order to compose the problem instances.

CHAPTER 7: CONCLUSIONS AND OPEN PROBLEMS

We conclude this thesis with a number of open problems from each chapter.

7.1 STATIC GEOMETRIC SET COVER

For unweighted geometric set cover, it would be interesting to see whether our approach can be extended for more types of geometric ranges. For weighted geometric set cover, a remaining open problem is to find efficient deterministic algorithms. The MWU part in our algorithm is deterministic, but the ε-net construction is not. Chan et al. [69] noted that the quasi-uniform sampling technique can be derandomized via the method of conditional probabilities, but the running time is high.

Open Problem 7.1. Is there a deterministic algorithm for weighted geometric set cover for halfspaces in 3D that runs in near-linear time?

Another direction is to see whether our ideas will be useful for solving the more general geometric set multicover problem, where each given point p is required to be covered $d(p)$ times. There are $O(\log \mathsf{opt})$-approximation for set systems of bounded VC-dimension, and $O(1)$-approximation for 3D halfspaces [81]. The result by Chekuri and Quanrud [83] for solving the underlying LP relaxations requires efficient range searching data structures. Later, Chekuri, Har-Peled and Quanrud [82] gave an $O(nm)$ time approximation algorithm for solving the LP for set multicover for disks in $\mathbb{R}^2$.

7.2 DYNAMIC GEOMETRIC SET COVER

In Chapter 3, we have described improved dynamic approximation data structures for various versions of the geometric set cover problem, and in particular, achieving very low (polylogarithmic or $n^{o(1)}$) update time for 1D intervals and 2D unit squares, in both the unweighted and weighted settings. Besides obtaining further improvements of our update time bounds, there are a number of interesting directions to explore for future work:

- We have given sublinear results for unweighted 2D halfplanes with regards to reporting queries, but could similar results be obtained for unweighted 3D halfspaces and 2D disks, that can handle reporting queries in $\widetilde{O}(\mathsf{opt})$ time? Our current method in Sec. 3.5 can only handle size queries.

- Are there data structures with sublinear update time for the dynamic hitting set problem for ranges such as 2D arbitrary squares? We can use duality to reduce hitting set to set cover in the unit square case, but not in the arbitrary square case (not even for "nearly unit" squares with side lengths in $[1, 1+\varepsilon]$).

- Are there data structures with sublinear update time for 2D arbitrary rectangles with polylogarithmic approximation factor? (Demanding constant approximation factor would be unreasonable, because of the lack of known efficient static $O(1)$-approximation algorithms, but there are static $O(\log n)$-approximation algorithms with near-linear running time for 2D rectangles [14].)

  In a subsequent work in SoCG'23, Khan et al. [147] partly answered this question, by providing an $(\log m)^{O(d)}$-approximation algorithm with $(\log m)^{O(d)}$ worst-case update time, but only for the special case where only the points can be inserted or deleted.

- In view of the recent developments in fine-grained complexity and reductions [205], could one prove $n^{\Omega(1)}$ conditional lower bounds on the update time for dynamic approximate geometric set cover, e.g., for arbitrary squares or other ranges, based on the conjectured hardness of standard problems such as 3SUM, all-pairs shortest paths, or orthogonal vectors?

- More generally, can we get more dynamic approximation results for other NP-hard geometric optimization problems?

7.3 GEOMETRIC SET COVER AND DISCRETE K-CENTER OF SMALL SIZE

In Chapter 4, we have provided a number of new algorithms and conditional hardness results for rectilinear discrete k-center and the geometric set cover problem of small size k. As many versions of the problems studied here still do not have matching upper and lower bounds, our work raises many interesting open questions. For example:

- Is it possible to make our subquadratic algorithm for rectilinear discrete 3-center in $\mathbb{R}^2$ work in dimension 3 or higher?

- Is it possible to make our conditional lower bound proof for rectilinear discrete 3-center in $\mathbb{R}^4$ work in dimension 2 or 3?

We should remark that some of these questions could be quite difficult. In fine-grained complexity, there are many examples of basic problems that still do not have tight conditional lower bounds (to mention one well-known geometric example, Künnemann's recent

FOCS'22 paper [158] has finally obtained a near-optimal conditional lower bound for Klee's measure problem in $\mathbb{R}^3$, which was later extended in SoCG'23 [129], but tight lower bounds in dimension 4 and higher are still not known for non-combinatorial algorithms). Still, we hope that our work would inspire more progress in both upper and lower bounds for this rich class of problems.

7.4 ENCLOSING POINTS WITH GEOMETRIC OBJECTS

Although many problems related to the enclosing problem have been proved to be NP-hard, the hardness status of the particular (uncolored) version ENCLOSING-ALL-POINTS that we study here is still unknown. To complement our approximation results, it would be nice to prove that this problem is NP-hard (if that's true).

Open Problem 7.2. Is the ENCLOSING-ALL-POINTS problem NP-hard, e.g., when the objects are disks or arbitrary line segments in $\mathbb{R}^2$?

For future work on this topic, it would be interesting to see whether our ideas can be extended to other geometric objects, and whether the approximation factor of our algorithms can be further improved, e.g., to near-constant. More generally, can our ideas be applied to solve other similar geometric optimization problems?

7.5 MONOTONE CONVEX CHAIN COVER

For the last part of the thesis about the FPT status of monotone convex chain cover, we conclude with a number of open problems to explore for future work:

- Our results exclude the existence of a polynomial kernel for monotone convex chain cover, unless NP $\subseteq$ coNP/poly. But whether this problem is FPT remains open: it is still possible that the problem has a super-polynomial sized kernel. Currently, the best known algorithm for this problem runs in $n^{O(k)}$ time.

- Since monotone convex chain cover and convex chain cover seem to be closely related, it would be interesting to see whether we can prove similar kernelization hardness results for convex chain cover as well. Our current technique does not obviously generalize, because in our construction for AND-composition, x-monotonicity is crucial for combining multiple monotone convex chains into a single one.

  Another direction is to directly relate the hardness between monotone convex/concave chain cover and convex chain cover, if that's possible.

- Since a large portion of this thesis is focused on designing approximation algorithms, the same direction can also be studied on (monotone) convex chain cover. The current best polynomial-time approximation algorithms for convex chain cover and monotone convex chain cover only achieve $O(\log n)$-approximation, via a simple greedy approach. It is not known whether polynomial-time constant factor approximation algorithms exist for these two problems, or whether these problems are APX-hard.

REFERENCES

[1] Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. Dynamic set cover: improved algorithms and lower bounds. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 114–125, 2019. doi:10.1145/3313276.3316376.

[2] Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir. Hardness of approximation in P via short cycle removal: Cycle detection, distance oracles, and beyond. In *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1487–1500, 2022. doi:10.1145/3519935.3520066.

[3] Eyal Ackerman, Oswin Aichholzer, and Balázs Keszegh. Improved upper bounds on the reflexivity of point sets. *Comput. Geom.*, 42(3):241–249, 2009. doi:10.1016/j.comgeo.2008.05.004.

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

[5] Anna Adamaszek and Andreas Wiese. Approximation schemes for maximum weight independent set of rectangles. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 400–409. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.50.

[6] Peyman Afshani and Timothy M. Chan. On approximate range counting and depth. *Discrete & Computational Geometry*, 42(1):3–21, 2009. doi:10.1007/s00454-009-9177-z.

[7] Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 180–186. SIAM, 2009. doi:10.1137/1.9781611973068.21.

[8] Peyman Afshani, Chris Hamilton, and Norbert Zeh. A general approach for cache-oblivious range reporting and approximate range counting. *Computational Geometry*, 43(8):700–712, 2010.

[9] Pankaj K. Agarwal, Rinat Ben Avraham, and Micha Sharir. The 2-center problem in three dimensions. *Comput. Geom.*, 46(6):734–746, 2013. Preliminary version in SoCG'10. doi:10.1016/j.comgeo.2012.11.005.

[10] 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. Preliminary version in SoCG'20 (https://doi.org/10.4230/LIPIcs.SoCG.2020.2). doi:10.1145/3551639.

[11] Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing*, 29(3):912–953, 1999. doi:10.1137/S0097539795295936.

[12] 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. URL: http://jeffe.cs.illinois.edu/pubs/survey.html.

[13] Pankaj K. Agarwal, Esther Ezra, and Micha Sharir. Near-linear approximation algorithms for geometric hitting sets. *Algorithmica*, 63(1-2):1–25, 2012.

[14] Pankaj K. Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. *Discrete & Computational Geometry*, 63(2):460–482, 2020. Preliminary version in SoCG'14. doi:10.1007/s00454-019-00099-6.

[15] Pankaj K. Agarwal and Cecilia Magdalena Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002. doi:10.1007/s00453-001-0110-y.

[16] Pankaj K. Agarwal and Micha Sharir. Davenport-Schinzel sequences and their geometric applications. In *Handbook of Computational Geometry*, pages 1–47. North Holland / Elsevier, 2000. doi:10.1016/b978-044482537-7/50002-4.

[17] Pankaj K Agarwal, Micha Sharir, and Emo Welzl. The discrete 2-center problem. *Discrete & Computational Geometry*, 20(3):287–305, 1998. Preliminary version in SoCG'97.

[18] Alok Aggarwal, Don Coppersmith, Sanjeev Khanna, Rajeev Motwani, and Baruch Schieber. The angular-metric traveling salesman problem. *SIAM Journal on Computing*, 29(3):697–711, 2000.

[19] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

[20] Helmut Alt, Sergio Cabello, Panos Giannopoulos, and Christian Knauer. Minimum cell connection and separation in line segment arrangements. *CoRR*, abs/1104.4618, 2011. arXiv:1104.4618.

[21] Helmut Alt, Sergio Cabello, Panos Giannopoulos, and Christian Knauer. On some connection problems in straight-line segment arrangements. *Proceedings of the 27th European Workshop on Computational Geometry (EuroCG)*, pages 27–30, 2011.

[22] Esther M. Arkin, Sándor P. Fekete, Ferran Hurtado, Joseph S. B. Mitchell, Marc Noy, Vera Sacristán, and Saurabh Sethia. On the reflexivity of point sets. *Discrete & Computational Geometry*, pages 139–156, 2003. Preliminary version in WADS'01. Extended version at arXiv:cs/0210003. URL: https://arxiv.org/pdf/cs/0210003.pdf.

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

[24] Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. *SIAM Journal on Computing*, 38(3):899–921, 2008. doi:10.1137/060669474.

[25] Sanjeev Arora. Polynomial time approximation schemes for euclidean TSP and other geometric problems. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 2–11. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548458.

[26] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

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

[28] Sepehr Assadi and Shay Solomon. Fully dynamic set cover via hypergraph maximal matching: An optimal approximation through a local approach. In *Proceedings of the 29th Annual European Symposium on Algorithms (ESA)*, volume 204 of *LIPIcs*, pages 8:1–8:18, 2021. doi:10.4230/LIPIcs.ESA.2021.8.

[29] Sayan Bandyapadhyay, Neeraj Kumar, Subhash Suri, and Kasturi R. Varadarajan. Improved approximation bounds for the minimum constraint removal problem. *Comput. Geom.*, 90:101650, 2020. doi:10.1016/j.comgeo.2020.101650.

[30] Michael Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 80–86, 1983. doi:10.1145/800061.808735.

[31] Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

[32] Sergey Bereg and David G. Kirkpatrick. Approximating barrier resilience in wireless sensor networks. In *Algorithmic Aspects of Wireless Sensor Networks, 5th International Workshop (ALGOSENSORS)*, volume 5804 of *Lecture Notes in Computer Science*, pages 29–40. Springer, 2009. doi:10.1007/978-3-642-05434-1_5.

[33] Sergei Bespamyatnikh and David G. Kirkpatrick. Rectilinear 2-center problems. In *Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG)*, 1999. URL: http://www.cccg.ca/proceedings/1999/fp55.pdf.

[34] Sergei Bespamyatnikh and Michael Segal. Rectilinear static and dynamic discrete 2-center problems. In *Proceedings of the 6th Workshop on Algorithms and Data Structures (WADS)*, pages 276–287, 1999. doi:10.1007/3-540-48447-7_28.

[35] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Design of dynamic algorithms via primal-dual method. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 9134 of *Lecture Notes in Computer Science*, pages 206–218. Springer, 2015. doi:10.1007/978-3-662-47672-7_17.

[36] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. A new deterministic algorithm for dynamic set cover. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 406–423. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00033.

[37] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Xiaowei Wu. Dynamic set cover: Improved amortized and worst-case update time. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2537–2549, 2021. doi:10.1137/1.9781611976465.150.

[38] Sujoy Bhore, Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Dynamic geometric independent set. *CoRR*, abs/2007.08643, 2020. arXiv:2007.08643.

[39] Sujoy Bhore, Guangping Li, and Martin Nöllenburg. An algorithmic study of fully dynamic independent sets for map labeling. *ACM J. Exp. Algorithmics*, 27:1.8:1–1.8:36, 2022. Preliminary version in ESA'20. doi:10.1145/3514240.

[40] Hans L Bodlaender, Rodney G Downey, Michael R Fellows, and Danny Hermelin. On problems without polynomial kernels. In *Proceedings of the 35th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 563–574, 2008.

[41] Hans L Bodlaender, Bart MP Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014.

[42] Karl Bringmann, Sándor Kisfaludi-Bak, Marvin Künnemann, André Nusser, and Zahra Parsaeian. Towards sub-quadratic diameter computation in geometric intersection graphs. In *Proceedings of the 38th Symposium on Computational Geometry (SoCG)*, pages 21:1–21:16, 2022. doi:10.4230/LIPIcs.SoCG.2022.21.

[43] Karl Bringmann, Sándor Kisfaludi-Bak, Michal Pilipczuk, and Erik Jan van Leeuwen. On geometric set cover for orthants. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA)*, pages 26:1–26:18, 2019. doi:10.4230/LIPIcs.ESA.2019.26.

[44] Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 617–626. IEEE Computer Society, 2002. doi:10.1109/SFCS.2002.1181985.

[45] Björn Brodén, Mikael Hammar, and Bengt J. Nilsson. Guarding lines and 2-link polygons is APX-hard. In *Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG)*, pages 45–48, 2001. URL: http://www.cccg.ca/proceedings/2001/mikael-2351.ps.gz.

[46] Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[47] Norbert Bus, Shashwat Garg, Nabil H. Mustafa, and Saurabh Ray. Limits of local search: Quality and efficiency. *Discrete & Computational Geometry*, 57(3):607–624, 2017. doi:10.1007/s00454-016-9819-x.

[48] Norbert Bus, Nabil H. Mustafa, and Saurabh Ray. Practical and efficient algorithms for the geometric hitting set problem. *Discrete Applied Mathematics*, 240:25–32, 2018.

[49] Sergio Cabello and Panos Giannopoulos. The complexity of separating points in the plane. *Algorithmica*, 74(2):643–663, 2016. doi:10.1007/s00453-014-9965-6.

[50] Sergio Cabello, Panos Giannopoulos, Christian Knauer, Dániel Marx, and Günter Rote. Geometric clustering: Fixed-parameter tractability and lower bounds with respect to the dimension. *ACM Trans. Algorithms*, 7(4):43:1–43:27, 2011. Preliminary version in SODA'08. doi:10.1145/2000807.2000811.

[51] Jin-yi Cai, Venkatesan T. Chakaravarthy, Lane A. Hemaspaandra, and Mitsunori Ogihara. Competing provers yield improved Karp-Lipton collapse results. *Inf. Comput.*, 198(1):1–23, 2005. doi:10.1016/j.ic.2005.01.002.

[52] 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.

[53] David Yu Cheng Chan and David G. Kirkpatrick. Approximating barrier resilience for arrangements of non-identical disk sensors. In *Algorithms for Sensor Systems, 8th International Symposium on Algorithms for Sensor Systems, Wireless Ad Hoc Networks and Autonomous Mobile Entities (ALGOSENSORS)*, volume 7718 of *Lecture Notes in Computer Science*, pages 42–53. Springer, 2012. doi:10.1007/978-3-642-36092-3_6.

[54] Timothy M. Chan. Geometric applications of a randomized optimization technique. *Discrete & Computational Geometry*, 22(4):547–567, 1999. doi:10.1007/PL00009478.

[55] Timothy M. Chan. More planar two-center algorithms. *Computational Geometry*, 13(3):189–198, 1999.

[56] Timothy M. Chan. Random sampling, halfspace range reporting, and construction of ($\leq k$)-levels in three dimensions. *SIAM Journal on Computing*, 30(2):561–575, 2000. doi:10.1137/S0097539798349188.

[57] Timothy M. Chan. Random sampling, halfspace range reporting, and construction of ($\leq k$)-levels in three dimensions. *SIAM Journal on Computing*, 30(2):561–575, 2000. doi:10.1137/S0097539798349188.

[58] Timothy M. Chan. A fully dynamic algorithm for planar width. *Discrete & Computational Geometry*, 30(1):17–24, 2003. doi:10.1007/s00454-003-2923-8.

[59] Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *Journal of Algorithms*, 46(2):178–189, 2003. doi:10.1016/S0196-6774(02)00294-8.

[60] Timothy M. Chan. Semi-online maintenance of geometric optima and measures. *SIAM Journal on Computing*, 32(3):700–716, 2003. doi:10.1137/S0097539702404389.

[61] Timothy M. Chan. Comparison-based time-space lower bounds for selection. *ACM Trans. Algorithms*, 6(2):26:1–26:16, 2010. Preliminary version in SODA'09. doi:10.1145/1721837.1721842.

[62] Timothy M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. *Journal of the ACM*, 57(3):16:1–16:15, 2010. doi:10.1145/1706591.1706596.

[63] Timothy M. Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012. Preliminary version in SoCG'10. doi:10.1007/s00454-012-9410-z.

[64] Timothy M. Chan. Three problems about dynamic convex hulls. *International Journal of Computational Geometry & Applications*, 22(4):341–364, 2012. doi:10.1142/S0218195912600096.

[65] Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. *Discrete & Computational Geometry*, 64(4):1235–1252, 2020. doi:10.1007/s00454-020-00229-5.

[66] Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. *Discrete & Computational Geometry*, 64(4):1235–1252, 2020. Preliminary version in SoCG'19. doi:10.1007/s00454-020-00229-5.

[67] Timothy M. Chan. Finding triangles and other small subgraphs in geometric intersection graphs. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1777–1805. SIAM, 2023. doi:10.1137/1.9781611977554.ch68.

[68] Timothy M. Chan and Elyot Grant. Exact algorithms and APX-hardness results for geometric packing and covering problems. *Comput. Geom.*, 47(2):112–124, 2014. doi:10.1016/j.comgeo.2012.04.001.

[69] Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1576–1585. SIAM, 2012. doi:10.1137/1.9781611973099.125.

[70] Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012. doi:10.1007/s00454-012-9417-5.

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

[72] Timothy M. Chan and Qizheng He. More dynamic data structures for geometric set cover with sublinear update time. *Journal of Computational Geometry (JoCG)*, 2022. Preliminary version in SoCG'21 (https://doi.org/10.4230/LIPIcs.SoCG.2021.25). URL: https://jocg.org/index.php/jocg/article/view/4021/3145.

[73] 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. SIAM, 2022. doi:10.1137/1.9781611977073.139.

[74] Timothy M. Chan, Qizheng He, and Yuancheng Yu. On the fine-grained complexity of small-size geometric set cover and discrete k-center for small k. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 261 of *LIPIcs*, pages 34:1–34:19, 2023. doi:10.4230/LIPIcs.ICALP.2023.34.

[75] Timothy M. Chan and Nan Hu. Geometric red-blue set cover for unit squares and related problems. *Comput. Geom.*, 48(5):380–385, 2015. doi:10.1016/j.comgeo.2014.12.005.

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

[77] Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM Journal on Computing*, 40(2):333–349, 2011. doi:10.1137/090751670.

[78] Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discrete & Computational Geometry*, 56(4):866–881, 2016. Preliminary version in SoCG'15. doi:10.1007/s00454-016-9784-4.

[79] Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Hardness for triangle problems under even more believable hypotheses: Reductions from Real APSP, Real 3SUM, and OV. In *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1501–1514, 2022. doi:10.1145/3519935.3520032.

[80] Bernard Chazelle and Jirı Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *Journal of Algorithms*, 21(3):579–597, 1996.

[81] Chandra Chekuri, Kenneth L. Clarkson, and Sariel Har-Peled. On the set multicover problem in geometric settings. *ACM Trans. Algorithms*, 9(1):9:1–9:17, 2012. Preliminary version in SoCG'09. doi:10.1145/2390176.2390185.

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

[83] Chandra Chekuri and Kent Quanrud. Near-linear time approximation schemes for some implicit fractional packing problems. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 801–820. SIAM, 2017. doi:10.1137/1.9781611974782.51.

[84] Chandra Chekuri and Kent Quanrud. Randomized MWU for positive LPs. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 358–377. SIAM, 2018. doi:10.1137/1.9781611975031.25.

[85] Rajesh Chitnis and Nitin Saurabh. Tight lower bounds for approximate & exact k-center in $\mathbb{R}^d$. In *Proceedings of the 38th Symposium on Computational Geometry (SoCG)*, pages 28:1–28:15, 2022. doi:10.4230/LIPIcs.SoCG.2022.28.

[86] Jongmin Choi and Hee-Kap Ahn. Efficient planar two-center algorithms. *Comput. Geom.*, 97:101768, 2021. doi:10.1016/j.comgeo.2021.101768.

[87] Kenneth L. Clarkson. New applications of random sampling in computational geometry. *Discrete & Computational Geometry*, 2:195–222, 1987. doi:10.1007/BF02187879.

[88] Kenneth L. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17(4):830–847, 1988. doi:10.1137/0217052.

[89] Kenneth L. Clarkson. Algorithms for polytope covering and approximation. In *Proceedings of the 3rd Workshop on Algorithms and Data Structures (WADS)*, pages 246–252, 1993.

[90] Kenneth L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM*, 42(2):488–499, 1995.

[91] Kenneth L. Clarkson and Peter W. Shor. Application of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989. doi:10.1007/BF02187740.

[92] Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4(5):387–421, 1989.

[93] Kenneth L. Clarkson and Kasturi Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007.

[94] 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:10.4230/LIPIcs.ICALP.2023.45.

[95] Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM Journal on Computing*, 11(3):467–471, 1982. doi:10.1137/0211037.

[96] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. The MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

[97] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

[98] Artur Czumaj, Funda Ergün, Lance Fortnow, Avner Magen, Ilan Newman, Ronitt Rubinfeld, and Christian Sohler. Approximating the weight of the Euclidean minimum spanning tree in sublinear time. *SIAM Journal on Computing*, 35(1):91–109, 2005. doi:10.1137/S0097539703435297.

[99] Ketan Dalal. Counting the onion. *Random Structures & Algorithms*, 24(2):155–165, 2004.

[100] 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/title/227584184.

[101] Robert P Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51:161–166, 1950.

[102] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 624–633. ACM, 2014. doi:10.1145/2591796.2591884.

[103] Adrian Dumitrescu and Minghui Jiang. On the approximability of covering points by lines and related problems. *Comput. Geom.*, 48(9):703–717, 2015. doi:10.1016/j.comgeo.2015.06.006.

[104] Martin E. Dyer. On a multidimensional search technique and its application to the Euclidean one-centre problem. *SIAM Journal on Computing*, 15(3):725–738, 1986.

[105] 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.

[106] Eduard Eiben, Jonathan Gemmell, Iyad A. Kanj, and Andrew Youngdahl. Improved results for minimum constraint removal. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 6477–6484. AAAI Press, 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16615.

[107] Eduard Eiben and Iyad Kanj. A colored path problem and its applications. *ACM Trans. Algorithms*, 16(4):47:1–47:48, 2020. doi:10.1145/3396573.

[108] Eduard Eiben and Iyad A. Kanj. How to navigate through obstacles? In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 48:1–48:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.48.

[109] Eduard Eiben and Daniel Lokshtanov. Removing connected obstacles in the plane is FPT. In *Proceedings of the 36th Symposium on Computational Geometry (SoCG)*, volume 164 of *LIPIcs*, pages 39:1–39:14, 2020. doi:10.4230/LIPIcs.SoCG.2020.39.

[110] Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004. doi:10.1016/j.tcs.2004.05.009.

[111] David Eppstein. Faster construction of planar two-centers. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 131–138, 1997. URL: https://dl.acm.org/doi/10.5555/314161.314198.

[112] David Eppstein. *Forbidden configurations in discrete geometry*. Cambridge University Press, 2018.

[113] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997. Preliminary version in FOCS'92. doi:10.1145/265910.265914.

[114] Paul Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio mathematica*, 2:463–470, 1935.

[115] Paul Erdős and George Szekeres. On some extremum problems in elementary geometry. In *Annales Univ. Sci. Budapest*, pages 3–4, 1960.

[116] Lawrence H. Erickson and Steven M. LaValle. A simple, but NP-hard, motion planning problem. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*. AAAI Press, 2013. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6280.

[117] Thomas Erlebach and Erik Jan van Leeuwen. PTAS for weighted set cover on unit squares. In *Proceedings of 13th International Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX)*, pages 166–177, 2010. doi:10.1007/978-3-642-15369-3_13.

[118] Thomas Fevens, Henk Meijer, and David Rappaport. Minimum convex partition of a constrained point set. *Discrete Applied Mathematics*, 109(1-2):95–107, 2001.

[119] Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

[120] Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, 1981. doi:10.1016/0020-0190(81)90111-3.

[121] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798, 1985. Preliminary version in STOC'83. doi:10.1137/0214055.

[122] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987.

[123] Greg N. Frederickson and Donald B. Johnson. The complexity of selection and ranking in $X + Y$ and matrices with sorted columns. *J. Comput. Syst. Sci.*, 24(2):197–208, 1982. doi:10.1016/0022-0000(82)90048-4.

[124] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

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

[126] Panos Giannopoulos, Christian Knauer, and Sue Whitesides. Parameterized complexity of geometric problems. *The Computer Journal*, 51(3):372–384, 2008.

[127] Matt Gibson, Gaurav Kanade, Rainer Penninger, Kasturi R. Varadarajan, and Ivo Vigan. On isolating points using unit disks. *J. Comput. Geom.*, 7(1):540–557, 2016. doi:10.20382/jocg.v7i1a22.

[128] Matt Gibson, Gaurav Kanade, and Kasturi R. Varadarajan. On isolating points using disks. In *Proceedings of the 19th Annual European Symposium on Algorithms (ESA)*, volume 6942, pages 61–69. Springer, 2011. doi:10.1007/978-3-642-23719-5_6.

[129] Egor Gorbachev and Marvin Künnemann. Combinatorial designs meet hypercliques: Higher lower bounds for Klee's measure problem and related problems in dimensions $d \geq 4$. In *Proceedings of the 39th Symposium on Computational Geometry (SoCG)*, volume 258 of *LIPIcs*, pages 36:1–36:14, 2023. doi:10.4230/LIPIcs.SoCG.2023.36.

[130] Magdalene Grantson and Christos Levcopoulos. A fixed parameter algorithm for the minimum number convex partition problem. In *Discrete and Computational Geometry, Japanese Conference (JCDCG), Revised Selected Papers*, volume 3742 of *Lecture Notes in Computer Science*, pages 83–94, 2004. doi:10.1007/11589440_9.

[131] Magdalene Grantson and Christos Levcopoulos. Covering a set of points with a minimum number of lines. In *Algorithms and Complexity, 6th Italian Conference (CIAC)*, volume 3998 of *Lecture Notes in Computer Science*, pages 6–17, 2006. doi:10.1007/11758471_4.

[132] Leonidas J. Guibas, Micha Sharir, and Shmuel Sifrony. On the general motion-planning problem with two degrees of freedom. *Discret. Comput. Geom.*, 4:491–521, 1989. Preliminary version in SoCG'88. doi:10.1007/BF02187744.

[133] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC)*, pages 537–550, 2017. doi:10.1145/3055399.3055493.

[134] Sariel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *J. Comput. Geom.*, 3(1):65–85, 2012. doi:10.20382/jocg.v3i1a4.

[135] Sariel Har-Peled and Bernard Lidicky. Peeling the grid. *SIAM Journal on Discrete Mathematics*, 27(2):650–655, 2013.

[136] Qizheng He. On the FPT status of monotone convex chain cover. In *Proceedings of the 35th Canadian Conference on Computational Geometry (CCCG)*, pages 307–312, 2023.

[137] Pinar Heggernes, Dieter Kratsch, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Fixed-parameter algorithms for cochromatic number and disjoint rectangle stabbing via iterative localization. *Information and Computation*, 231:109–116, 2013.

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

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

[140] Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Fully dynamic connectivity in $O(\log n(\log\log n)^2)$ amortized expected time. *TheoretiCS*, 2, 2023. Preliminary version in SODA'17. doi:10.46298/theoretics.23.6.

[141] R. Z. Hwang, R. C. Chang, and Richard C. T. Lee. The searching over separators strategy to solve some NP-hard problems in subexponential time. *Algorithmica*, 9(4):398–423, 1993. doi:10.1007/BF01228511.

[142] R. Z. Hwang, Richard C. T. Lee, and R. C. Chang. The slab dividing approach to solve the Euclidean p-center problem. *Algorithmica*, 9(1):1–22, 1993. doi:10.1007/BF01185335.

[143] Ce Jin and Yinzhan Xu. Removing additive structure in 3SUM-based reductions. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 405–418. ACM, 2023. doi:10.1145/3564246.3585157.

[144] Matthew J. Katz, Klara Kedem, and Michael Segal. Discrete rectilinear 2-center problems. *Comput. Geom.*, 15(4):203–214, 2000. doi:10.1016/S0925-7721(99)00052-8.

[145] Matthew J. Katz and Frank Nielsen. On piercing sets of objects. In *Proceedings of the 12th Symposium on Computational Geometry (SoCG)*, pages 113–121, 1996. doi:10.1145/237218.237253.

[146] Klara Kedem, Ron Livne, János Pach, and Micha Sharir. On the union of jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete & Computational Geometry*, 1:59–70, 1986. doi:10.1007/BF02187683.

[147] Arindam Khan, Aditya Lonkar, Saladi Rahul, Aditya Subramanian, and Andreas Wiese. Online and dynamic algorithms for geometric set cover and hitting set. In *Proceedings of the 39th Symposium on Computational Geometry (SoCG)*, volume 258 of *LIPIcs*, pages 46:1–46:17, 2023. doi:10.4230/LIPIcs.SoCG.2023.46.

[148] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-ε. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. Preliminary version in CCC'03. doi:10.1016/j.jcss.2007.06.019.

[149] Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 81–91. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814580.

[150] Christian Knauer and Andreas Spillner. Approximation algorithms for the minimum convex partition problem. In *10th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 4059 of *Lecture Notes in Computer Science*, pages 232–241, 2006. doi:10.1007/11785293_23.

[151] Matias Korman, Maarten Löffler, Rodrigo I. Silveira, and Darren Strash. On the complexity of barrier resilience for fat regions. In *Algorithms for Sensor Systems - 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS)*, volume 8243 of *Lecture Notes in Computer Science*, pages 201–216, 2013. doi:10.1007/978-3-642-45346-5_15.

[152] Matias Korman, Maarten Löffler, Rodrigo I. Silveira, and Darren Strash. On the complexity of barrier resilience for fat regions and bounded ply. *Comput. Geom.*, 72:34–51, 2018. doi:10.1016/j.comgeo.2018.02.006.

[153] Christos Koufogiannakis and Neal E. Young. A nearly linear-time PTAS for explicit fractional packing and covering linear programs. *Algorithmica*, 70(4):648–674, 2014. doi:10.1007/s00453-013-9771-6.

[154] Neeraj Kumar, Daniel Lokshtanov, Saket Saurabh, and Subhash Suri. A constant factor approximation for navigating through connected obstacles in the plane. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 822–839. SIAM, 2021. doi:10.1137/1.9781611976465.52.

[155] Neeraj Kumar, Daniel Lokshtanov, Saket Saurabh, Subhash Suri, and Jie Xue. Point separation and obstacle removal by finding and hitting odd cycles. In *Proceedings of the 38th Symposium on Computational Geometry (SoCG)*, volume 224 of *LIPIcs*, pages 52:1–52:14, 2022. doi:10.4230/LIPIcs.SoCG.2022.52.

[156] Santosh Kumar, Ten-Hwang Lai, and Anish Arora. Barrier coverage with wireless sensors. *Wirel. Networks*, 13(6):817–834, 2007. doi:10.1007/s11276-006-9856-0.

[157] V. S. Anil Kumar, Sunil Arya, and H. Ramesh. Hardness of set cover with intersection 1. In *Proceedings of the 27th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1853 of *Lecture Notes in Computer Science*, pages 624–635. Springer, 2000. doi:10.1007/3-540-45022-X_53.

[158] Marvin Künnemann. A tight (non-combinatorial) conditional lower bound for Klee's measure problem in 3D. In *Proceedings of the 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 555–566. IEEE, 2022. doi:10.1109/FOCS54457.2022.00059.

[159] Stefan Langerman and Pat Morin. Covering things with things. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, pages 662–674, 2002.

[160] Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1236–1252, 2018. doi:10.1137/1.9781611975031.80.

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

[162] Chih-Hung Liu, Evanthia Papadopoulou, and D. T. Lee. An output-sensitive approach for the L_1/L_∞ k-nearest-neighbor Voronoi diagram. In *Proceedings of the 19th Annual European Symposium on Algorithms (ESA)*, volume 6942 of *Lecture Notes in Computer Science*, pages 70–81. Springer, 2011. doi:10.1007/978-3-642-23719-5_7.

[163] Dániel Marx. Efficient approximation schemes for geometric problems? In *Proceedings of the 13th Annual European Symposium of Algorithms (ESA)*, pages 448–459, 2005. doi:10.1007/11561071_41.

[164] Dániel Marx and Michal Pilipczuk. Optimal parameterized algorithms for planar facility location problems using Voronoi diagrams. In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA)*, pages 865–877, 2015. doi:10.1007/978-3-662-48350-3_72.

[165] Jiří Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8(3):315–334, 1992.

[166] Jiří Matoušek. Reporting points in halfspaces. *Computational Geometry*, 2:169–186, 1992. doi:10.1016/0925-7721(92)90006-E.

[167] Jiří Matoušek. Range searching with efficient hierarchical cutting. *Discrete & Computational Geometry*, 10:157–182, 1993. doi:10.1007/BF02573972.

[168] Jirí Matoušek, Raimund Seidel, and Emo Welzl. How to net a lot with little: Small ε-nets for disks and halfspaces. In *Proceedings of the 6th Symposium on Computational Geometry (SoCG)*, pages 16–22. ACM, 1990. doi:10.1145/98524.98530.

[169] Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

[170] Nimrod Megiddo and Arie Tamir. On the complexity of locating linear facilities in the plane. *Oper. Res. Lett.*, 1(5):194–197, 1982. doi:10.1016/0167-6377(82)90039-6.

[171] 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. IEEE, 2021. doi:10.1109/FOCS52979.2021.00042.

[172] Joseph SB Mitchell, Günter Rote, Gopalakrishnan Sundaram, and Gerhard J Woeginger. Counting convex polygons in planar point sets. *Inf. Process. Lett.*, 56(1):45–49, 1995.

[173] Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Johns Hopkins series in the mathematical sciences. Johns Hopkins University Press, 2001. URL: https://www.press.jhu.edu/books/title/1675/graphs-surfaces.

[174] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[175] Ketan Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms*. Prentice Hall, 1994.

[176] Nabil H. Mustafa. Computing optimal epsilon-nets is as easy as finding an unhit set. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 87:1–87:12, 2019. doi:10.4230/LIPIcs.ICALP.2019.87.

[177] Nabil H. Mustafa, Rajiv Raman, and Saurabh Ray. Quasi-polynomial time approximation scheme for weighted geometric set cover on pseudodisks and halfspaces. *SIAM Journal on Computing*, 44(6):1650–1669, 2015. doi:10.1137/14099317X.

[178] Nabil H. Mustafa and Saurabh Ray. PTAS for geometric hitting set problems via local search. In *Proceedings of the 25th Symposium on Computational Geometry (SoCG)*, pages 17–22. ACM, 2009. doi:10.1145/1542362.1542367.

[179] Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010. doi:10.1007/s00454-010-9285-9.

[180] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 950–961. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.92.

[181] Doron Nussbaum. Rectilinear p-piercing problems. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 316–323, 1997. doi:10.1145/258726.258828.

[182] J Pach. Discrete and computational geometry, 19. *Special issue dedicated to Paul Erdös*, 1998.

[183] János Pach and Gábor Tardos. Tight lower bounds for the size of epsilon-nets. In *Proceedings of the 27th Symposium on Computational Geometry (SoCG)*, pages 458–463, 2011. doi:10.1145/1998196.1998271.

[184] Rainer Penninger and Ivo Vigan. Point set isolation using unit disks is NP-complete. *CoRR*, abs/1303.2779, 2013. arXiv:1303.2779.

[185] Ricky Pollack, Micha Sharir, and Shmuel Sifrony. Separating two simple polygons by a sequence of translations. *Discrete & Computational Geometry*, 3:123–136, 1988. doi:10.1007/BF02187902.

[186] Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 603–610, 2010. doi:10.1145/1806689.1806772.

[187] Evangelia Pyrga and Saurabh Ray. New existence proofs for ϵ-nets. In *Proceedings of the 24th Symposium on Computational Geometry (SoCG)*, pages 199–207, 2008. doi:10.1145/1377676.1377708.

[188] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *J. Comput. Syst. Sci.*, 37(2):130–143, 1988. Preliminary version in FOCS'86. doi:10.1016/0022-0000(88)90003-7.

[189] Prabhakar Raghavan and Clark D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Comb.*, 7(4):365–374, 1987. doi:10.1007/BF02579324.

[190] Edgar A. Ramos. On range reporting, ray shooting and k-level construction. In *Proceedings of the 15th Symposium on Computational Geometry (SoCG)*, pages 390–399. ACM, 1999. doi:10.1145/304893.304993.

[191] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 216–226. ACM, 1978. doi:10.1145/800133.804350.

[192] Micha Sharir. On k-sets in arrangement of curves and surfaces. *Discrete & Computational Geometry*, 6:593–613, 1991. doi:10.1007/BF02574706.

[193] Micha Sharir. A near-linear algorithm for the planar 2-center problem. *Discrete & Computational Geometry*, 18(2):125–134, 1997. Preliminary version in SoCG'96.

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

[195] M. Shimrat. Algorithm 112: Position of point relative to polygon. *Commun. ACM*, 5(8):434, 1962. doi:10.1145/368637.368653.

[196] Andreas Spillner. Optimal convex partitions of point sets with few inner points. In *Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG)*, pages 39–42, 2005.

[197] Andrew Suk. On the Erdős-Szekeres convex polygon problem. *Journal of the American Mathematical Society*, 30(4):1047–1053, 2017.

[198] Carsten Thomassen. Embeddings of graphs with no short noncontractible cycles. *J. Comb. Theory, Ser. B*, 48(2):155–177, 1990. doi:10.1016/0095-8956(90)90115-G.

[199] Constantine Toregas, Ralph Swain, Charles S. ReVelle, and Lawrence Bergman. The location of emergency service facilities. *Operations Research*, 19(6):1363–1373, 1971. doi:10.1287/opre.19.6.1363.

[200] Kuan-Chieh Robert Tseng. *Resilience of wireless sensor networks*. PhD thesis, University of British Columbia, 2011.

[201] Kuan-Chieh Robert Tseng and David G. Kirkpatrick. On barrier resilience of sensor networks. In *Algorithms for Sensor Systems - 7th International Symposium on Algorithms for Sensor Systems, Wireless Ad Hoc Networks and Autonomous Mobile Entities (ALGOSENSORS)*, volume 7111 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2011. doi:10.1007/978-3-642-28209-6_11.

[202] Masatsugu Urabe. On a partition of point sets into convex polygons. In *Proceedings of the 9th Canadian Conference on Computational Geometry (CCCG)*, 1997.

[203] Kasturi R. Varadarajan. Epsilon nets and union complexity. In *Proceedings of the 25th Symposium on Computational Geometry (SoCG)*, pages 11–16, 2009. doi:10.1145/1542362.1542366.

[204] Kasturi R. Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 641–648. ACM, 2010. doi:10.1145/1806689.1806777.

[205] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472. World Scientific, 2018. URL: https://people.csail.mit.edu/virgi/eccentri.pdf.

[206] Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. Preliminary version in FOCS'10. doi:10.1145/3186893.

[207] Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and APSP. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 786–797, 2020. doi:10.1109/FOCS46700.2020.00078.

[208] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001. URL: http://www.springer.com/computer/theoretical+computer+science/book/978-3-540-65367-7.

[209] Klaus W. Wagner. Monotonic coverings of finite sets. *J. Inf. Process. Cybern.*, 20(12):633–639, 1984.

[210] Haitao Wang. On the planar two-center problem and circular hulls. *Discrete & Computational Geometry*, 68(4):1175–1226, 2022. Preliminary version in SoCG'20. doi:10.1007/s00454-021-00358-5.

[211] Jianxin Wang, Wenjun Li, and Jianer Chen. A parameterized algorithm for the hyperplane-cover problem. *Theor. Comput. Sci.*, 411(44-46):4005–4009, 2010. doi:10.1016/j.tcs.2010.08.012.

[212] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*, pages 359–370. Springer, 1991.

[213] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.

[214] Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26:287–300, 1983. doi:10.1016/0304-3975(83)90020-8.

[215] Neal E. Young. Sequential and parallel algorithms for mixed packing and covering. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 538–546, 2001. doi:10.1109/SFCS.2001.959930.