Dynamic Geometric Set Cover and Hitting Set

Pankaj K. Agarwal
Department of Computer Science, Duke University, Durham, NC, USA
pankaj@cs.duke.edu

Hsien-Chih Chang
Department of Computer Science, Duke University, Durham, NC, USA
hsienchih.chang@duke.edu

Subhash Suri
Department of Computer Science, University of California at Santa Barbara, CA, USA
suri@cs.ucsb.edu

Allen Xiao
Department of Computer Science, Duke University, Durham, NC, USA
axiao@cs.duke.edu

Jie Xue
Department of Computer Science, University of California at Santa Barbara, CA, USA
jiexue@ucsb.edu

Abstract

We investigate dynamic versions of geometric set cover and hitting set where points and ranges may be inserted or deleted, and we want to efficiently maintain an (approximately) optimal solution for the current problem instance. While their static versions have been extensively studied in the past, surprisingly little is known about dynamic geometric set cover and hitting set. For instance, even for the most basic case of one-dimensional interval set cover and hitting set, no nontrivial results were known. The main contribution of our paper are two frameworks that lead to efficient data structures for dynamically maintaining set covers and hitting sets in $\mathbb{R}^1$ and $\mathbb{R}^2$. The first framework uses bootstrapping and gives a $(1+\varepsilon)$-approximate data structure for dynamic interval set cover in $\mathbb{R}^1$ with $O(n^\alpha/\varepsilon)$ amortized update time for any constant $\alpha > 0$; in $\mathbb{R}^2$, this method gives $O(1)$-approximate data structures for unit-square (and quadrant) set cover and hitting set with $O(n^{1/2+\alpha})$ amortized update time. The second framework uses local modification, and leads to a $(1+\varepsilon)$-approximate data structure for dynamic interval hitting set in $\mathbb{R}^1$ with $\widetilde{O}(1/\varepsilon)$ amortized update time; in $\mathbb{R}^2$, it gives $O(1)$-approximate data structures for unit-square (and quadrant) set cover and hitting set in the *partially* dynamic settings with $\widetilde{O}(1)$ amortized update time.

2012 ACM Subject Classification Theory of computation → Computational geometry

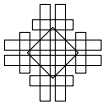
Keywords and phrases Geometric set cover, Geometric hitting set, Dynamic data structures

Digital Object Identifier 10.4230/LIPIcs.SoCG.2020.2

Related Version A full version of this paper is available at `https://arxiv.org/abs/2003.00202`.

Funding *Pankaj K. Agarwal*: NSF grants CCF-15-13816, CCF-15-46392, and IIS-14-08846; ARO grant W911NF-15-1-0408; BSF Grant 2012/229 from the U.S.-Israel Binational Science Foundation.
Hsien-Chih Chang: NSF grants CCF-15-13816, CCF-15-46392, and IIS-14-08846; ARO grant W911NF-15-1-0408; BSF Grant 2012/229 from the U.S.-Israel Binational Science Foundation.
Subhash Suri: NSF grant CCF-18-14172.
Allen Xiao: NSF grants CCF-15-13816, CCF-15-46392, and IIS-14-08846; ARO grant W911NF-15-1-0408; BSF Grant 2012/229 from the U.S.-Israel Binational Science Foundation.
Jie Xue: NSF grant CCF-18-14172.

36th International Symposium on Computational Geometry (SoCG 2020).
Editors: Sergio Cabello and Danny Z. Chen; Article No. 2; pp. 2:1–2:15



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A *range space* $(X, \mathcal{R})$ consists of a set X of objects and a family $\mathcal{R}$ of subsets of X called *ranges*. A subset $H \subseteq X$ is called a *hitting set* of $(X, \mathcal{R})$ if it intersects every (nonempty) range in $\mathcal{R}$, and a subset $\mathcal{C} \subseteq \mathcal{R}$ is called a *set cover* of $(X, \mathcal{R})$ if $\bigcup_{C \in \mathcal{C}} C = X$. In many applications X is a set of points in $\mathbb{R}^d$ and $\mathcal{R}$ is induced by a set of geometric regions (rectangles, balls, simplices, etc.), i.e., each is the subset of points lying inside one of the regions. With a slight abuse of notation, we will use $\mathcal{R}$ to denote the set of ranges as well as the set of regions that define these ranges. Given a geometric range space $(S, \mathcal{R})$, the *geometric set-cover* (resp., *hitting-set*) problem is to find the smallest number of ranges in $\mathcal{R}$ (resp., points in S) that cover all points in S (resp., hit all ranges in $\mathcal{R}$). Geometric set cover and hitting set are classical geometric optimization problems, with numerous applications in databases, sensor networks, VLSI design, etc.

In many applications, the problem instance can change over time and re-computing a new solution after each change is too costly. In these situations, a dynamic data structure that can update the solution after a change more efficiently than constructing the entire new solution from scratch is highly desirable. This motivates the main problem studied in our paper: dynamically maintaining geometric set covers and hitting sets under insertion and deletion of points and ranges. In this paper, we formulate the problem as follows: after each update, our data structure should (implicitly) store an approximate set-cover solution $\mathcal{R}'$ (resp., hitting-set solution S') for the current instance such that the following queries can be supported efficiently.

- *Size query*: report the size of $\mathcal{R}'$ (resp., S').
- *Membership query*: for a given range $R \in \mathcal{R}$ (resp., point $a \in S$), report whether R (resp., a) is contained in $\mathcal{R}'$ (resp., S').
- *Reporting query*: report all elements in $\mathcal{R}'$ (resp., S').

We require the size query to be answered in $O(1)$ time, a membership query to be answered in $O(\log |\mathcal{R}'|)$ time (resp., $O(\log |S'|)$ time), and the reporting query to be answered in $O(|\mathcal{R}'|)$ time (resp., $O(|S'|)$ time); this is the best one can expect in the pointer machine model.

We say that a set-cover (resp., hitting-set) instance is *fully dynamic* if insertions and deletions on both points and ranges are allowed, and *partially dynamic* if only the points (resp., ranges) can be inserted and deleted. In this paper, unless explicitly mentioned otherwise, problems are always considered in the fully dynamic setting.

Related work

The set-cover and hitting-set problems for general range spaces are well-known to be NP-complete [12]. A simple greedy algorithm achieves an $O(\log n)$-approximation [7, 13, 15], which is tight under appropriate complexity-theoretic assumptions [8, 14]. The problems remain NP-hard or even hard to approximate in many geometric settings [5, 16, 17]. However, by exploiting the geometric nature of the problems, efficient algorithms with $o(\log n)$ approximation factors can be obtained. For example, Mustafa and Ray [18] showed the existence of polynomial-time approximation schemes (PTAS) for halfspace hitting set in $\mathbb{R}^3$ and disk hitting set in $\mathbb{R}^2$. There is also a PTAS for unit-square set cover given by Erlebach and van Leeuwen [9]. Agarwal and Pan [3] proposed approximation algorithms with near-linear running time to the set-cover and hitting-set problems for halfspaces in $\mathbb{R}^3$, disks in $\mathbb{R}^2$, and orthogonal rectangles.

Despite extensive work on the static versions of hitting set and set cover, very little is known about these problems in the dynamic setting. There is some recent work on set cover in the partially dynamic setting. Gupta et al. [11] showed that an $O(\log n)$-approximation

can be maintained with $O(f \log n)$ amortized update time and an $O(f^3)$-approximation can be maintained with $O(f^2)$ amortized update time, where f is the maximum number of ranges that a point belongs to. These bounds were subsequently improved by Bhattacharya et al. [6] to $O(f^2)$-approximation factor and $O(f \log n)$ amortized update time, and by Abboud et al. [1] to $(1+\varepsilon)f$-approximation factor and $O(f^2 \log n/\varepsilon^5)$ amortized update time.

In geometric settings, there has been some work on the dynamic hitting-set problem. Agarwal et al. [4] described a dynamic data structure for maintaining an $(1+\varepsilon)$-approximation of the optimal hitting set when the set of points S is $\mathbb{R}^1$ and $\mathcal{R}$ is a set of intervals. Ganjugunte [10] studied the dynamic hitting-set problem for the case where S is a set of points in $\mathbb{R}^2$ and $\mathcal{R}$ is a set of squares or discs, under two different dynamic settings: (a) only the range set $\mathcal{R}$ is dynamic and (b) $\mathcal{R}$ is dynamic and S is semi-dynamic (i.e., insertion-only). We are not aware of any non-trivial results for geometric set cover or hitting set even in 1D, except that the greedy algorithm can be implemented in an output-sensitive manner in some special cases (see below).

Our results

The main contribution of this paper are two frameworks for designing fully dynamic geometric set-cover and hitting-set data structures, leading to efficient data structures in $\mathbb{R}^1$ and $\mathbb{R}^2$ (see Table 1). The first framework is based on bootstrapping, which results in efficient (approximate) dynamic data structures for interval set cover and quadrant/unit-square set cover and hitting set (the first three rows of Table 1). The second framework is based on local modification, which results in efficient (approximate) dynamic data structures for interval hitting set and quadrant/unit-square set cover and hitting set in the *partially* dynamic setting (the last three rows of Table 1).

Table 1 Summary of our results for dynamic geometric set cover and hitting set (SC = set cover and HS = hitting set). All update times are amortized. The notation $\widetilde{O}(\cdot)$ hides logarithmic factors; n is the size of the current instance, and $\alpha > 0$ is any small constant. All data structures can be constructed in $\widetilde{O}(n_0)$ time where n_0 is the size of the initial instance.

Framework	Problem	Range	Approx.	Update time	Setting
Bootstrapping	SC	Interval	$1+\varepsilon$	$\widetilde{O}(n^\alpha/\varepsilon)$	Fully dynamic
	SC & HS	Quadrant	$O(1)$	$\widetilde{O}(n^{1/2+\alpha})$	Fully dynamic
	SC & HS	Unit square	$O(1)$	$\widetilde{O}(n^{1/2+\alpha})$	Fully dynamic
Local modification	HS	Interval	$1+\varepsilon$	$\widetilde{O}(1/\varepsilon)$	Fully dynamic
	SC & HS	Quadrant	$O(1)$	$\widetilde{O}(1)$	Part. dynamic
	SC & HS	Unit square	$O(1)$	$\widetilde{O}(1)$	Part. dynamic

For technical reasons, our algorithms maintain a *multiset* solution, as opposed to a regular subset of $\mathcal{R}$ (resp., S). That is, we allow the solution to be a multiset of elements in $\mathcal{R}$ (resp., S) that cover all points in S (resp., hit all ranges in $\mathcal{R}$), and the quality of the solution is also evaluated in terms of the multiset cardinality. Unless explicitly mentioned otherwise, solutions for set cover and hitting set always refer to multiset solutions hereafter.

Overview of the techniques

The basic idea of our *bootstrapping* framework is as follows: We begin from a simple dynamic set-cover or hitting-set data structure (e.g., a data structure that re-computes a solution after each update), and repeatedly use the current data structure to obtain an improved

one. The main challenge here is to design the bootstrapping procedure: how to use a given data structure to construct a new data structure with improved update time. We achieve this by using output-sensitive algorithms and carefully partitioning the problem instances to sub-instances. We say an algorithm is *output-sensitive* if it computes an (approximate) optimal solution in time proportional to the size of the output, using only *basic data structures*. A data structure built on a dataset of size n is *basic* if it can be constructed in $\widetilde{O}(n)$ time and made dynamic with $\widetilde{O}(1)$ update time. One of our technical contributions is in designing an $O(1)$-approximate output-sensitive algorithm for 2D quadrant set cover, which is new to the best of our knowledge.

Our second framework is much simpler, which is based on *local modification*. Namely, we construct a new solution by slightly modifying the previous one after each update, and re-compute a new solution periodically using an output-sensitive algorithm. This framework applies to the problems which are *stable*, i.e., the optimum of a dynamic instance does not change significantly. The discussion of this framework can be found in the full version [2].

Organization

The rest of the paper is organized as follows. Due to limited space, only the results of our first framework (bootstrapping) are discussed in this conference version: Section 2 presents the 1D results and Section 3 presents the 2D results. The results of our second framework (local modification), as well as all the omitted proofs and details, can be found in the full version [2].

2 Warm-up: 1D set cover for intervals

As a warm up for our bootstrapping framework, we first study the 1D problem. Let S be a set of points in $\mathbb{R}^1$ and $\mathcal{I}$ a set of intervals in $\mathbb{R}^1$; set $n = |S| + |\mathcal{I}|$. Our goal is to maintain a small-size set cover of the range space $(S, \mathcal{I})$ as S and $\mathcal{I}$ are updated dynamically; we refer to this instance as the dynamic *interval cover* problem. We note that (static) interval set cover can be solved using the greedy algorithm that repeatedly picks the leftmost uncovered point and covers it using the interval with the rightmost right endpoint. By storing S and $\mathcal{I}$ in a height-balanced tree, the greedy algorithm can be made output sensitive, i.e., it reports an optimal set cover of size opt in $O(\mathsf{opt} \log n)$ time.

▶ **Lemma 1.** *Interval set cover admits an exact output-sensitive algorithm.*

Using the output-sensitive algorithm, now we sketch how to design a fully dynamic data structure to solve the interval-set-cover problem. Set a threshold n^α for some $\alpha \in (0, 1)$. The main bootstrapping step is as follows: Assume that we have a dynamic data structure solving the interval-set-cover problem with $O(n^{\alpha/(1-\alpha)})$ update time (modulo the dependencies on the approximation parameter ε). Using this data structure, we construct a dynamic data structure with $\widetilde{O}(n^\alpha)$ update time as follows: Run the output-sensitive algorithm for $O(n^\alpha)$ steps; if the optimal set cover has size opt at most n^α then the algorithm correctly computes an optimal set cover. Otherwise, we know that opt is at least n^α. We partition the points and the intervals on the real line into roughly εn^α portions in a balanced way. The number of points and endpoints of the intervals per portion is $O(n^{1-\alpha}/\varepsilon)$, and the number of portions is at most $O(\varepsilon n^\alpha)$. On each portion, we build a sub-instance using the points contained in the portion, with all intervals that have an endpoint in the portion. We use the slower data structure to maintain a set cover of this sub-instance. If one of these intervals completely covers the portion (the portion is *coverable*), we solve its sub-instance with a single covering interval. If the portion is not coverable, we rely on the slower data structure

to provide an approximate solution. Because opt is so large, an $(\varepsilon/2)$-approximation on each uncoverable portion combined with the single intervals of each coverable portion still gives an $(1+\varepsilon)$-approximate multiset solution to the entire instance.

The size of each portion is $O(n^{1-\alpha})$ (omitting ε for now), so each update to the dynamic data structure on the portions takes about $O((n^{1-\alpha})^{\alpha/(1-\alpha)}) = O(n^\alpha)$ time, which only happens at most two times per insertion (an interval can partially intersects at most two portions). The data structure periodically reconstructs itself after processing about $n^{1-\alpha}$ operations; which means in amortization each operation costs an extra $O(n^\alpha)$ time. Overall, this gives $O(n^\alpha)$ update time for the new data structure.

We now describe the data structure in detail and analyze its performance.

2.1 Bootstrapping

We begin by stating the bootstrapping theorem, which is the technical heart of our result.

▶ **Theorem 2.** *Let $(S,\mathcal{I})$ be an instance of interval set cover, with $n = |S| + |\mathcal{I}|$. Let $\alpha, \varepsilon \in (0,1)$ be parameters. If there exists a $(1+\varepsilon)$-approximate dynamic set-cover structure $\mathcal{D}_{\text{old}}$ for $(S,\mathcal{I})$ with $\widetilde{O}(n^\alpha/\varepsilon^{1-\alpha})$ amortized update time and $\widetilde{O}(n)$ construction time for any $\varepsilon > 0$, then there exists a $(1+\varepsilon)$-approximate dynamic interval-set-cover data structure $\mathcal{D}_{\text{new}}$ with $\widetilde{O}(n^{\alpha'}/\varepsilon^{1-\alpha'})$ amortized update time and $\widetilde{O}(n)$ construction time for any $\varepsilon > 0$, where $\alpha' = \alpha/(1+\alpha)$. Here n denotes the size of the current problem instance.*

Constructing $\mathcal{D}_{\text{new}}$

The data structure $\mathcal{D}_{\text{new}}$ consists of two parts. The first part is the basic data structure $\mathcal{A}$ required for the output-sensitive algorithm of Lemma 1. The second part is a family of $\mathcal{D}_{\text{old}}$ data structures. Let $f(n,\varepsilon) = \min\{n^{\frac{1}{1+\alpha}}/\varepsilon^{\frac{\alpha}{1+\alpha}}, n/2\}$. $\mathcal{D}_{\text{new}}$ is reconstructed periodically. Let $n_0 = |S| + |\mathcal{I}|$ when the data structure is being constructed. Set $r = \lceil n_0/f(n_0,\varepsilon)\rceil$. We partition the real line $\mathbb{R}$ into r intervals $J_1,\ldots,J_r$ such that each interval J_i contains at most $2f(n_0,\varepsilon)$ points in S plus the endpoints of the intervals in $\mathcal{I}$. For $i \le r$, define $S_i = S \cap J_i$ and $\mathcal{I}_i \subseteq \mathcal{I}$ the subsets of intervals that intersect J_i but do not cover it, i.e., $\mathcal{I}_i = \{I \in \mathcal{I} : J_i \cap I \neq \emptyset \text{ and } J_i \not\subseteq I\}$. When $\mathcal{D}_{\text{new}}$ is updated, the partition $J_1,\ldots,J_r$ remains unchanged, but the S_i's and $\mathcal{I}_i$'s change as S and $\mathcal{I}$ are updated. We view each $(S_i,\mathcal{I}_i)$ as a dynamic interval-set-cover instance, and build the data structure $\mathcal{D}^{(i)}_{\text{old}}$ on $(S_i,\mathcal{I}_i)$ using $\mathcal{D}_{\text{old}}$, with the approximation parameter $\tilde{\varepsilon} = \varepsilon/2$. Thus, $\mathcal{D}^{(i)}_{\text{old}}$ maintains a $(1+\tilde{\varepsilon})$-approximate set cover for $(S_i,\mathcal{I}_i)$. The second part of $\mathcal{D}_{\text{new}}$ consists of the data structures $\mathcal{D}^{(1)}_{\text{old}},\ldots,\mathcal{D}^{(r)}_{\text{old}}$.

Maintaining a set cover

We now describe the algorithm for maintaining a set cover $\mathcal{I}_{\text{appx}}$ for $(S,\mathcal{I})$, if there exists one. Set $\delta = \min\{(6+2\varepsilon)\cdot r/\varepsilon, n\}$. We simulate the output-sensitive greedy algorithm for at most δ steps. If the algorithm successfully computes a set cover, we use it as our $\mathcal{I}_{\text{appx}}$. Otherwise, we construct $\mathcal{I}_{\text{appx}}$ as follows. For $i \in \{1,\ldots,r\}$, we say J_i is *coverable* if there exists $I \in \mathcal{I}$ such that $J_i \subseteq I$ and *uncoverable* otherwise. Let $P = \{i : J_i \text{ is coverable}\}$ and $\overline{P} = \{i : J_i \text{ is uncoverable}\}$. For each $i \in P$, we choose an interval in $\mathcal{I}$ that contains J_i, and denote by $\mathcal{I}^*$ the collection of these intervals. If for some $i \in \overline{P}$, the data structure $\mathcal{D}^{(i)}_{\text{old}}$ tells us that the current $(S_i,\mathcal{I}_i)$ does not have a set cover, then we immediately conclude that the current $(S,\mathcal{I})$ has no feasible set cover, record as such. Otherwise, for every $i \in \overline{P}$, $\mathcal{D}^{(i)}_{\text{old}}$ maintains a $(1+\tilde{\varepsilon})$-approximate optimal set cover $\mathcal{I}^*_i$ for $(S_i,\mathcal{I}_i)$. Set $\mathcal{I}_{\text{appx}} = \mathcal{I}^* \sqcup \left(\bigsqcup_{J_i \in \mathcal{P}'} \mathcal{I}^*_i\right)$.

It can be shown that if $(S, \mathcal{I})$ has a feasible set cover, our algorithm maintains one, namely, $\mathcal{I}_{\text{appx}}$. Later we prove that $\mathcal{I}_{\text{appx}}$ is always a $(1+\varepsilon)$-approximate optimal set cover for $(S, \mathcal{I})$. We now describe how to store $\mathcal{I}_{\text{appx}}$ properly to support the size, membership, and reporting queries in the required query times. If $\mathcal{I}_{\text{appx}}$ is computed by the output-sensitive algorithm, then the size of $\mathcal{I}_{\text{appx}}$ is at most δ, and we have all the elements of $\mathcal{I}_{\text{appx}}$ in hand. In this case, it is not difficult to build a data structure on $\mathcal{I}_{\text{appx}}$ to support the desired queries. On the other hand, if $\mathcal{I}_{\text{appx}}$ is defined as the disjoint union of $\mathcal{I}^*$ and $\mathcal{I}_i^*$'s, the size of $\mathcal{I}_{\text{appx}}$ might be very large and we do not explicitly store all elements of $\mathcal{I}_{\text{appx}}$. Fortunately, in this case, each $\mathcal{I}_i^*$ is already maintained in the data structure $\mathcal{D}_{\text{old}}^{(i)}$. Therefore, we only compute P, $\overline{P}$, and $\mathcal{I}^*$; with these in hand, we easily build a data structure to support the desired queries for $\mathcal{I}_{\text{appx}}$. A detailed discussion is presented in the full version [2].

Updating $\mathcal{D}_{\text{new}}$

Let n_0 be the size of the data structure when $\mathcal{D}_{\text{new}}$ was previously constructed. We reconstruct $\mathcal{D}_{\text{new}}$ after $f(n_0, \varepsilon)$ update operations and reset n_0 to the current value of $|S|+|\mathcal{I}|$. If $\mathcal{D}_{\text{new}}$ is not being reconstructed after an update operation, we first update the basic data structure $\mathcal{A}$. Then, we update the data structure $\mathcal{D}_{\text{old}}^{(i)}$ if the instance $(S_i, \mathcal{I}_i)$ changes due to the operation. Note that an update on S changes exactly one S_i and an update on $\mathcal{I}$ changes at most two $\mathcal{I}_i$'s (because an interval can belong to at most two $\mathcal{I}_i$'s). Thus, we in fact only need to update at most two $\mathcal{D}_{\text{old}}^{(i)}$'s.

Correctness

Now we show that the set cover $\mathcal{I}_{\text{appx}}$ maintained by $\mathcal{D}_{\text{new}}$ is a $(1+\varepsilon)$-approximate optimal set cover for $(S, \mathcal{I})$. Let opt be the size of the optimal set cover of $(S, \mathcal{I})$. If $\mathcal{I}_{\text{appx}}$ is computed by the output-sensitive algorithm, then it is an optimal set cover for $(S, \mathcal{I})$. Otherwise, $\mathsf{opt} > \delta = \min\{(6+2\varepsilon)\cdot r/\varepsilon, n\}$. If $\mathsf{opt} > n$, then the current $(S, \mathcal{I})$ has no set cover (i.e., $\mathsf{opt} = \infty$) and thus $\mathcal{D}_{\text{new}}$ makes a no-solution decision. So assume $\mathsf{opt} > (6+2\varepsilon)\cdot r/\varepsilon$. In this case, $\mathcal{I}_{\text{appx}} = \mathcal{I}^* \sqcup (\bigsqcup_{i\in\overline{P}} \mathcal{I}_i^*)$. For each $i \in \overline{P}$, let opt_i be the optimum of the instance $(S_i, \mathcal{I}_i)$. Then we have $|\mathcal{I}_i^*| \le (1+\tilde{\varepsilon})\cdot \mathsf{opt}_i$ for all $i \in \overline{P}$ where $\tilde{\varepsilon} = \varepsilon/2$. Since $|\mathcal{I}^*| \le r$, we have

$$|\mathcal{I}_{\text{appx}}| = |\mathcal{I}^*| + \sum_{i\in\overline{P}} |\mathcal{I}_i^*| \le r + \left(1+\frac{\varepsilon}{2}\right)\sum_{i\in\overline{P}} \mathsf{opt}_i. \tag{1}$$

Let $\mathcal{I}_{\text{opt}}$ be an optimal set cover for $(S, \mathcal{I})$. We observe that for $i \in \overline{P}$, $\mathcal{I}_{\text{opt}} \cap \mathcal{I}_i$ is a set cover for $(S_i, \mathcal{I}_i)$, because J_i is uncoverable (so the points in S_i cannot be covered by any interval in $\mathcal{I}\setminus\mathcal{I}_i$). It immediately follows that $\mathsf{opt}_i \le |\mathcal{I}_{\text{opt}} \cap \mathcal{I}_i|$ for all $i \in \overline{P}$. Therefore, we have

$$\sum_{i\in\overline{P}} \mathsf{opt}_i \le \sum_{i\in\overline{P}} |\mathcal{I}_{\text{opt}} \cap \mathcal{I}_i|. \tag{2}$$

The right-hand side of the above inequality can be larger than $|\mathcal{I}_{\text{opt}}|$ as some intervals in $\mathcal{I}_{\text{opt}}$ can belong to two $\mathcal{I}_i$'s. The following lemma bounds the number of such intervals.

▶ **Lemma 3.** *There are at most $2r$ intervals in $\mathcal{I}_{opt}$ that belong to exactly two $\mathcal{I}_i$'s.*

Proof. Suppose the portions $J_1, \ldots, J_r$ are sorted from left to right. Let s_i be the separation point of J_i and J_{i+1}. Observe that an interval $I \in \mathcal{I}_{\text{opt}}$ belongs to exactly two $\mathcal{I}_i$'s only if I contains one of the separation points $s_1, \ldots, s_{r-1}$. We claim that for each s_i, at most two intervals in $\mathcal{I}_{\text{opt}}$ contain s_i. Assume there are three intervals I^-, I, I^+ that contain s_i.

Without loss of generality, assume that I^- (resp., I^+) has the leftmost left endpoint (resp., the rightmost right endpoint) among I^-, I, I^+. Then one can easily see that $I \subseteq I^- \cup I^+$. Therefore, $\mathcal{I}_{\text{opt}} \backslash \{I\}$ is also a set cover for $(S, \mathcal{I})$, contradicting the optimality of $\mathcal{I}_{\text{opt}}$. Thus, at most two intervals in $\mathcal{I}_{\text{opt}}$ contain s_i. It follows that there are at most $2(r-1)$ intervals in $\mathcal{I}_{\text{opt}}$ that contain some separation point, and only these intervals can belong to exactly two $\mathcal{I}_i$'s, which proves the lemma. ◀

The above lemma immediately implies

$$\sum_{i \in \overline{P}} |\mathcal{I}_{\text{opt}} \cap \mathcal{I}_i| \leq |\mathcal{I}_{\text{opt}}| + 2r = \mathsf{opt} + 2r. \tag{3}$$

Combining Inequalities 1, 2, and 3, we deduce that

$$\begin{aligned}
|\mathcal{I}_{\text{appx}}| &\leq r + \left(1 + \frac{\varepsilon}{2}\right) \sum_{i \in \overline{P}} \mathsf{opt}_i \\
&\leq r + \left(1 + \frac{\varepsilon}{2}\right) \sum_{i \in \overline{P}} |\mathcal{I}_{\text{opt}} \cap \mathcal{I}_i| \\
&\leq r + \left(1 + \frac{\varepsilon}{2}\right) \cdot (\mathsf{opt} + 2r) = (3 + \varepsilon) \cdot r + \left(1 + \frac{\varepsilon}{2}\right) \cdot \mathsf{opt} \\
&< \frac{\varepsilon}{2} \cdot \mathsf{opt} + \left(1 + \frac{\varepsilon}{2}\right) \cdot \mathsf{opt} = (1 + \varepsilon) \cdot \mathsf{opt},
\end{aligned}$$

where the last inequality follows from the assumption $\mathsf{opt} > (6 + 2\varepsilon) \cdot r/\varepsilon$.

Time complexity analysis

We briefly discuss the amortized update time of $\mathcal{D}_{\text{new}}$; a detailed analysis can be found in the full version [2]. Recall that $f(n, \varepsilon) = \min\{n^{\frac{1}{1+\alpha}}/\varepsilon^{\frac{\alpha}{1+\alpha}}, n/2\}$ and that $\mathcal{D}_{\text{new}}$ is reconstructed after $f(n_0, \varepsilon)$ update operations, where n_0 is the size of $(S, \mathcal{I})$ when $\mathcal{D}_{\text{new}}$ was last constructed, $|n - n_0| \leq n_0/2$ and the size of each $(S_i, \mathcal{I}_i)$ is $O(f(n_0, \varepsilon))$. The construction of $\mathcal{D}_{\text{new}}$ can be easily done in $\widetilde{O}(n_0)$ time.

The update time of $\mathcal{D}_{\text{new}}$ consists of the time for updating the data structures $\mathcal{A}$ and $\mathcal{D}^{(1)}_{\text{old}}, \ldots, \mathcal{D}^{(r)}_{\text{old}}$, the time for maintaining the solution, and the (amortized) time for reconstruction. As argued before, we only need to update at most two $\mathcal{D}^{(i)}_{\text{old}}$'s after each operation. Thus, updating the $\mathcal{D}_{\text{old}}$ data structures takes $\widetilde{O}(f(n_0, \varepsilon)^\alpha/\varepsilon^{1-\alpha})$ amortized time. Maintaining $\mathcal{I}_{\text{appx}}$ takes $\widetilde{O}(\delta + r) = O(n_0/(f(n_0, \varepsilon) \cdot \varepsilon))$ time, with a careful implementation. The reconstruction time is $\widetilde{O}(n) = \widetilde{O}(n_0 + f(n_0, \varepsilon))$, which we pay for by charging $\widetilde{O}(n_0/f(n_0, \varepsilon)) = \widetilde{O}(r)$ to each update operation since the previous reconstruction. In total, the amortized update time of $\mathcal{D}_{\text{new}}$ is $\widetilde{O}(f(n_0, \varepsilon)^\alpha/\varepsilon^{1-\alpha} + n_0/(f(n_0, \varepsilon) \cdot \varepsilon))$. By substituting the value of $f(n_0, \varepsilon)$ (which balances the two terms in the update time) and using the inequality $|n - n_0| \leq n_0/2$, the amortized update time is $\widetilde{O}(n^{\frac{\alpha}{1+\alpha}}/\varepsilon^{1-\frac{\alpha}{1+\alpha}}) = \widetilde{O}(n^{\alpha'}/\varepsilon^{1-\alpha'})$ (recall that $\alpha' = \alpha/(1+\alpha)$).

2.2 Putting everything together

With the bootstrapping theorem in hand, we are now able to design our dynamic interval-set-cover data structure. The starting point is a "trivial" data structure, which simply uses the output-sensitive algorithm of Lemma 1 to recompute an optimal interval set cover after each update. Clearly, the update time of this data structure is $\widetilde{O}(n)$ and the construction time is $\widetilde{O}(n_0)$. Thus, there exists a $(1+\varepsilon)$-approximate dynamic interval-set-cover data structure with $\widetilde{O}(n^{\alpha_0}/\varepsilon^{1-\alpha_0})$ amortized update time for $\alpha_0 = 1$ and $\widetilde{O}(n_0)$ construction time. Define

$\alpha_i = \alpha_{i-1}/(1+\alpha_{i-1})$ for $i \geq 1$. By applying Theorem 2 i times for a constant $i \geq 1$, we see the existence of a $(1+\varepsilon)$-approximate dynamic interval-set-cover data structure with $\widetilde{O}(n^{\alpha_i}/\varepsilon^{1-\alpha_i})$ amortized update time and $\widetilde{O}(n_0)$ construction time. One can easily verify that $\alpha_i = 1/(i+1)$ for all $i \geq 0$. Therefore, for any constant $\alpha > 0$, we have an index $i \geq 0$ satisfying $\alpha_i < \alpha$ and hence $\widetilde{O}(n^{\alpha_i}/\varepsilon^{1-\alpha_i}) = O(n^{\alpha}/\varepsilon)$. We finally conclude the following.

▶ **Theorem 4.** *Let $(S, \mathcal{I})$ be an instance of interval set cover, with $n = |S| + |\mathcal{I}|$. Let $\alpha, \varepsilon \in (0,1)$ be two constants. There exists a $(1+\varepsilon)$-approximate dynamic interval-set-cover data structure with $O(n^{\alpha}/\varepsilon)$ amortized update time.*

3 2D set cover for quadrants and unit squares

In this section, we present dynamic set-cover data structures for quadrants and unit squares using the bootstrapping framework. Most of the section focuses on dynamic quadrant set cover. At the end of the section, we reduce dynamic unit-square set cover to dynamic quadrant set cover.

Let $(S, \mathcal{Q})$ be a range space where S is a set of points in $\mathbb{R}^2$ and $\mathcal{Q}$ is a set of quadrants in $\mathbb{R}^2$. We wish to maintain a set cover of $(S, \mathcal{Q})$ as both S and $\mathcal{Q}$ are updated dynamically. In order to apply the bootstrapping framework, we need an output-sensitive algorithm for quadrant set cover, analog to the one in Lemma 1 for intervals. Designing such an algorithm is considerably more difficult compared to the 1D case, and we defer it to Section 3.2. We first discuss the bootstrapping procedure, assuming the existence of a μ-approximate output-sensitive algorithm for quadrant set cover.

3.1 Bootstrapping

We prove the following bootstrapping theorem, which is the technical heart of our result.

▶ **Theorem 5.** *Let $(S, \mathcal{Q})$ be an instance of quadrant set cover, with $n = |S| + |\mathcal{Q}|$. Let $\alpha, \varepsilon \in (0,1)$ be parameters and $\mu > 0$. If there exist a $(\mu+\varepsilon)$-approximate output-sensitive algorithm for quadrant set cover and a $(\mu+\varepsilon)$-approximate dynamic set-cover structure $\mathcal{D}_{\text{old}}$ for $(S, \mathcal{Q})$ with $\widetilde{O}(n^{\alpha}/\varepsilon^{1-\alpha})$ amortized update time and $\widetilde{O}(n)$ construction time, then there exists a $(\mu+\varepsilon)$-approximate dynamic quadrant-set-cover data structure $\mathcal{D}_{\text{new}}$ with $\widetilde{O}(n^{\alpha'}/\varepsilon^{1-\alpha'})$ amortized update time and $\widetilde{O}(n)$ construction time, where $\alpha' = 2\alpha/(1+2\alpha)$.*

Constructing $\mathcal{D}_{\text{new}}$

As in the 1D case, the data structure $\mathcal{D}_{\text{new}}$ consists of two parts. The first part is the data structure $\mathcal{A}$ required for the μ-approximate output-sensitive algorithm. The second part is a family of $\mathcal{D}_{\text{old}}$ data structures defined as follows. Let $f(n,\varepsilon) = \min\{n^{\frac{1+\alpha}{1+2\alpha}}/\varepsilon^{\frac{\alpha}{1+2\alpha}}, n/2\}$. $\mathcal{D}_{\text{new}}$ is reconstructed periodically. Let $n_0 = |S| + |\mathcal{I}|$ when the data structure is being constructed. Set $r = \lceil n_0/f(n_0,\varepsilon)\rceil$. We use an orthogonal grid to partition the plane $\mathbb{R}^2$ into $r \times r$ cells such that each row (resp., column) of the grid contains $f(n_0,\varepsilon)$ points in S plus vertices of the quadrants in $\mathcal{Q}$ (see the left of Figure 1 for an illustration). Denote by $\square_{i,j}$ the cell in the i-th row and j-th column. Define $S_{i,j} = S \cap \square_{i,j}$. We also define a sub-collection $\mathcal{Q}_{i,j} \subseteq \mathcal{Q}$, as follows: We include in $\mathcal{Q}_{i,j}$ all the quadrants in $\mathcal{Q}$ whose vertices lie in $\square_{i,j}$. Besides, we also include in $\mathcal{Q}_{i,j}$ the following (at most) four *special* quadrants. We say a quadrant Q *left intersects* $\square_{i,j}$ if Q partially intersects $\square_{i,j}$ and contains the left edge of $\square_{i,j}$ (see the right of Figure 1 for an illustration); similarly, we define "right intersects", "top intersects", and "bottom intersects". Among a collection of quadrants, the

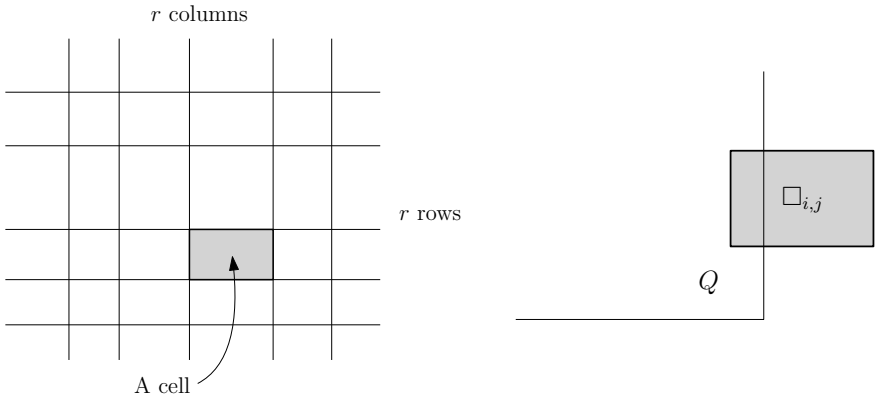


Figure 1 Left: The $r \times r$ grid. Note that the cells may have different sizes. Right: A quadrant Q that left intersects $\square_{i,j}$.

leftmost/rightmost/topmost/bottommost quadrant refers to the quadrant whose vertex is the leftmost/rightmost/topmost/bottommost. We include in $\mathcal{Q}_{i,j}$ the rightmost quadrant in $\mathcal{Q}$ that left intersects $\square_{i,j}$, the leftmost quadrant in $\mathcal{Q}$ that right intersects $\square_{i,j}$, the bottommost quadrant in $\mathcal{Q}$ that top intersects $\square_{i,j}$, and the topmost quadrant in $\mathcal{Q}$ that bottom intersects $\square_{i,j}$ (if these quadrants exist).[1] When the instance $(S, \mathcal{Q})$ is updated, the grid remains unchanged, but the $S_{i,j}$'s and $\mathcal{Q}_{i,j}$'s change as S and $\mathcal{Q}$ are updated. We view each $(S_{i,j}, \mathcal{Q}_{i,j})$ as a dynamic quadrant-set-cover instance, and build the data structure $\mathcal{D}_{\text{old}}^{(i,j)}$ on $(S_{i,j}, \mathcal{Q}_{i,j})$ using $\mathcal{D}_{\text{old}}$, with approximation factor $\tilde{\varepsilon} = \varepsilon/2$. The second part of $\mathcal{D}_{\text{new}}$ consists of the data structures $\mathcal{D}_{\text{old}}^{(i,j)}$ for $i, j \in \{1, \ldots, r\}$.

Maintaining a set cover

We now describe the algorithm for maintaining a set cover $\mathcal{Q}_{\text{appx}}$ for $(S, \mathcal{Q})$, if one exists. Set $\delta = \min\{(8\mu + 4\varepsilon + 2) \cdot r^2/\varepsilon, n\}$. We simulate the output-sensitive algorithm for at most δ steps. If the algorithm successfully computes a set cover, we use it as our $\mathcal{Q}_{\text{appx}}$. Otherwise, we construct $\mathcal{Q}_{\text{appx}}$ as follows. We say the cell $\square_{i,j}$ is *coverable* if there exists $Q \in \mathcal{Q}$ that contains $\square_{i,j}$ and *uncoverable* otherwise. Let $P = \{(i,j) : \square_{i,j} \text{ is coverable}\}$ and $\overline{P} = \{(i,j) : \square_{i,j} \text{ is uncoverable}\}$. For each $(i,j) \in P$, we choose a quadrant in $\mathcal{Q}$ that contains $\square_{i,j}$, and denote by $\mathcal{Q}^*$ the set of these quadrants. If for some $(i,j) \in \overline{P}$, $\mathcal{D}_{\text{old}}^{(i,j)}$ tells us that the instance $(S_{i,j}, \mathcal{Q}_{i,j})$ has no set cover, then we immediately conclude that the current $(S, \mathcal{Q})$ has no feasible set cover, and record as such. Otherwise, for each $(i,j) \in \overline{P}$, $\mathcal{D}_{\text{old}}^{(i,j)}$ maintains a $(\mu + \tilde{\varepsilon})$-approximate optimal set cover $\mathcal{Q}_{i,j}^*$ for $(S_{i,j}, \mathcal{Q}_{i,j})$. Set $\mathcal{Q}_{\text{appx}} = \mathcal{Q}^* \sqcup \left(\bigsqcup_{(i,j)\in\overline{P}} \mathcal{Q}_{i,j}^*\right)$. $\mathcal{Q}_{\text{appx}}$ is stored in roughly the same way as $\mathcal{I}_{\text{appx}}$ is in the 1D case, and we omit the details from here.

[1] Recall that in the 1D case, we define $\mathcal{I}_i$ as the sub-collection of intervals in $\mathcal{I}$ that partially intersect the portion J_i. However, we cannot simply define $\mathcal{Q}_{i,j}$ as the sub-collection of quadrants in $\mathcal{Q}$ that partially intersect $\square_{i,j}$ because a quadrant partially intersects too many cells.

Updating $\mathcal{D}_{\text{new}}$

Let n_0 be the size of the data structure when $\mathcal{D}_{\text{new}}$ was previously constructed. We reconstruct $\mathcal{D}_{\text{new}}$ after $f(n_0, \varepsilon)$ update operations and reset n_0 to the current value of $|S| + |\mathcal{I}|$. If $\mathcal{D}_{\text{new}}$ is not being reconstructed after an update operation, we first update the basic data structure $\mathcal{A}$. Then, we update those data structures $\mathcal{D}_{\text{old}}^{(i,j)}$ for which $(S_{i,j}, \mathcal{Q}_{i,j})$ change. Note that an update on S changes exactly one $S_{i,j}$, and an update on $\mathcal{Q}$ may only change the $\mathcal{Q}_{i,j}$'s in one row and one column (specifically, if the vertex of the inserted/deleted quadrant lies in $\square_{i,j}$, then only $\mathcal{Q}_{i,1}, \ldots, \mathcal{Q}_{i,r}, \mathcal{Q}_{1,j}, \ldots, \mathcal{Q}_{r,j}$ may change). Thus, we in fact only need to update the $\mathcal{D}_{\text{old}}^{(i,j)}$'s in one row and one column.

Correctness

We show that the set cover $\mathcal{Q}_{\text{appx}}$ maintained by $\mathcal{D}_{\text{new}}$ is a $(\mu + \varepsilon)$-approximate optimal set cover for $(S, \mathcal{Q})$. If $\mathcal{Q}_{\text{appx}}$ is computed by the output-sensitive algorithm, then it is a μ-approximate optimal set cover for $(S, \mathcal{Q})$. Otherwise, $\mathsf{opt} > \delta = \min\{(8\mu + 4\varepsilon + 2) \cdot r^2/\varepsilon, n\}$, i.e., either $\mathsf{opt} > (8\mu + 4\varepsilon + 2) \cdot r^2/\varepsilon$ or $\mathsf{opt} > n$. If $\mathsf{opt} > n$, then $(S, \mathcal{Q})$ has no set cover (i.e., $\mathsf{opt} = \infty$) and $\mathcal{D}_{\text{new}}$ makes a no-solution decision. So assume $(8\mu + 4\varepsilon + 2)r^2/\varepsilon < \mathsf{opt} < n$. In this case, $\mathcal{Q}_{\text{appx}} = \mathcal{Q}^* \sqcup (\bigsqcup_{(i,j) \in \overline{P}} \mathcal{Q}^*_{i,j})$. For each $(i,j) \in \overline{P}$, let $\mathsf{opt}_{i,j}$ be the optimum of $(S_{i,j}, \mathcal{Q}_{i,j})$. Then we have $|\mathcal{Q}^*_{i,j}| \leq (\mu + \tilde{\varepsilon}) \cdot \mathsf{opt}_{i,j}$ for all $(i,j) \in \overline{P}$ where $\tilde{\varepsilon} = \varepsilon/2$. Since $|\mathcal{Q}^*| \leq r^2$, we have

$$\mathcal{Q}_{\text{appx}} = |\mathcal{Q}^*| + \sum_{(i,j) \in \overline{P}} |\mathcal{Q}^*_{i,j}| \leq r^2 + \left(\mu + \frac{\varepsilon}{2}\right) \sum_{(i,j) \in \overline{P}} \mathsf{opt}_{i,j}. \tag{4}$$

Let $\mathcal{Q}'_{i,j} \subseteq \mathcal{Q}_{i,j}$ consist of the non-special quadrants, i.e., those whose vertices are in $\square_{i,j}$.

▶ **Lemma 6.** *We have* $\mathsf{opt}_{i,j} \leq |\mathcal{Q}_{\text{opt}} \cap \mathcal{Q}'_{i,j}| + 4$ *for all* $(i,j) \in \overline{P}$*, and in particular,*

$$\sum_{(i,j) \in \overline{P}} \mathsf{opt}_{i,j} \leq \mathsf{opt} + 4r^2. \tag{5}$$

Using Equations 4 and 5, we deduce that

$$\begin{aligned}
|\mathcal{Q}_{\text{appx}}| &\leq r^2 + \left(\mu + \frac{\varepsilon}{2}\right) \sum_{(i,j) \in \overline{P}} \mathsf{opt}_{i,j} \\
&\leq r^2 + \left(\mu + \frac{\varepsilon}{2}\right) (\mathsf{opt} + 4r^2) \\
&\leq (4\mu + 2\varepsilon + 1) \cdot r^2 + \left(\mu + \frac{\varepsilon}{2}\right) \cdot \mathsf{opt} \\
&< \frac{\varepsilon}{2} \cdot \mathsf{opt} + \left(\mu + \frac{\varepsilon}{2}\right) \cdot \mathsf{opt} = (\mu + \varepsilon) \cdot \mathsf{opt},
\end{aligned}$$

where the last inequality follows from the fact that $\mathsf{opt} > (8\mu + 4\varepsilon + 2) \cdot r^2/\varepsilon$.

Time complexity analysis

We briefly discuss the amortized update time of $\mathcal{D}_{\text{new}}$; a detailed analysis can be found in the full version [2]. Recall that $f(n, \varepsilon) = \min\{n^{1-\alpha'/2}/(\sqrt{\varepsilon})^{\alpha'}, n/2\}$ and that $\mathcal{D}_{\text{new}}$ is reconstructed after $f(n_0, \varepsilon)$ update operations, where n_0 is the size of $(S, \mathcal{Q})$ when $\mathcal{D}_{\text{new}}$ was last constructed, $|n - n_0| \leq n_0/2$. We first observe the following fact.

▶ **Lemma 7.** *At any time between reconstructions, we have* $\sum_{k=1}^{r}(|S_{i,k}| + |\mathcal{Q}_{i,k}|) = O(f(n_0,\varepsilon)+r)$ *for all* $i \in \{1,\ldots,r\}$ *and* $\sum_{k=1}^{r}(|S_{k,j}| + |\mathcal{Q}_{k,j}|) = O(f(n_0,\varepsilon)+r)$ *for all* $j \in \{1,\ldots,r\}$.

The above lemma implies that the sum of the sizes of all $(S_{i,j},\mathcal{Q}_{i,j})$ is $O(n_0+r^2)$ at any time in the first period. Therefore, constructing $\mathcal{D}_{\text{new}}$ can be done in $\widetilde{O}(n_0+r^2)=\widetilde{O}(n_0)$ time.

The update time of $\mathcal{D}_{\text{new}}$ consists of the (amortized) time for reconstruction, the time for updating $\mathcal{A}$ and $\mathcal{D}_{\text{old}}^{(i,j)}$'s, and the time for maintaining the solution. Using almost the same analysis as in the 1D problem, we can show that the reconstruction takes $\widetilde{O}(r+r^2/f(n_0,\varepsilon))$ amortized time and maintaining $\mathcal{Q}_{\text{appx}}$ takes $\widetilde{O}(\delta+r^2)=\widetilde{O}(r^2/\varepsilon)$ time, with a careful implementation. The time for updating the $\mathcal{D}_{\text{old}}^{(i,j)}$ requires a different analysis. Let $m_{i,j}$ denote the current size of $(S_{i,j},\mathcal{Q}_{i,j})$. As argued before, we in fact only need to update the $\mathcal{D}_{\text{old}}^{(i,j)}$ in one row and one column (say the i-th row and j-th column). Hence, updating the $\mathcal{D}_{\text{old}}^{(i,j)}$ takes $\widetilde{O}(\sum_{k=1}^{r} m_{i,k}^{\alpha}/\varepsilon^{1-\alpha}+\sum_{k=1}^{r} m_{k,j}^{\alpha}/\varepsilon^{1-\alpha})$ amortized time. Lemma 7 implies that $\sum_{k=1}^{r} m_{i,k} = O(f(n_0,\varepsilon)+r)$ and $\sum_{k=1}^{r} m_{k,j} = O(f(n_0,\varepsilon)+r)$. Since $\alpha \le 1$, by Hölder's inequality and Lemma 7,

$$\sum_{k=1}^{r} m_{i,k}^{\alpha} \le \left(\frac{\sum_{k=1}^{r} m_{i,k}}{r}\right)^{\alpha} \cdot r = O(r^{1-\alpha}\cdot(f(n_0,\varepsilon)+r)^{\alpha}) = O(r+r^{1-\alpha}f^{\alpha}(n_0/\varepsilon))$$

and similarly $\sum_{k=1}^{r} m_{k,j}^{\alpha} = O(r+r^{1-\alpha}f^{\alpha}(n_0/\varepsilon))$. It follows that updating the $\mathcal{D}_{\text{old}}$ data structures takes $\widetilde{O}((r+r^{1-\alpha}f^{\alpha}(n_0/\varepsilon))/\varepsilon^{1-\alpha})$ amortized time. In total, the amortized update time of $\mathcal{D}_{\text{new}}$ (during the first period) is $\widetilde{O}((r+r^{1-\alpha}f^{\alpha}(n_0/\varepsilon))/\varepsilon^{1-\alpha}+r^2/\varepsilon)$. By substituting the value of $f(n_0,\varepsilon) = n^{\frac{1+\alpha}{1+2\alpha}}/\varepsilon^{\frac{\alpha}{1+2\alpha}}$ (which balances the two main terms in the update time) and using the fact that $|n-n_0| \le n_0/2$, we obtain that the amortized update time is $\widetilde{O}(n^{\frac{2\alpha}{1+2\alpha}}/\varepsilon^{1-\frac{2\alpha}{1+2\alpha}}) = \widetilde{O}(n^{\alpha'}/\varepsilon^{1-\alpha'})$.

3.2 An output-sensitive cover algorithm

The key to Theorem 5 is an output-sensitive algorithm for the quadrant-set-cover problem. In this section, we develop such an algorithm, which computes an $O(1)$-approximation of the set cover in $\widetilde{O}(\mathsf{opt})$ time, using basic data structures.

For simplicity, let us assume that $(S,\mathcal{Q})$ has a set cover; how the no-solution case is handled is discussed in the full version [2]. There are four types of quadrants in $\mathcal{Q}$, southeast, southwest, northeast, northwest; we denote by $\mathcal{Q}^{\text{SE}},\mathcal{Q}^{\text{SW}},\mathcal{Q}^{\text{NE}},\mathcal{Q}^{\text{NW}} \subseteq \mathcal{Q}$ the sub-collections of these types of quadrants, respectively. Let U^{SE} denote the union of the quadrants in $\mathcal{Q}^{\text{SE}}$, and define $U^{\text{SW}},U^{\text{NE}},U^{\text{NW}}$ similarly. Since $(S,\mathcal{Q})$ has a set cover, we have $S = (S\cap U^{\text{SE}})\cup(S\cap U^{\text{SW}})\cup(S\cap U^{\text{NE}})\cup(S\cap U^{\text{NW}})$. Therefore, if we can compute $O(1)$-approximate optimal set covers for each of $(S\cap U^{\text{SE}},\mathcal{Q})$, $(S\cap U^{\text{SW}},\mathcal{Q})$, $(S\cap U^{\text{NE}},\mathcal{Q})$, and $(S\cap U^{\text{NW}},\mathcal{Q})$, then the union of these four set covers is an $O(1)$-approximate optimal set cover for $(S,\mathcal{Q})$.

With this observation, it now suffices to show how to compute an $O(1)$-approximate optimal set cover for $(S\cap U^{\text{SE}},\mathcal{Q})$ in $\widetilde{O}(\mathsf{opt}^{\text{SE}})$ time, where opt^{SE} is the optimum of $(S\cap U^{\text{SE}},\mathcal{Q})$. The main challenge is to guarantee the running time and approximation ratio simultaneously. We begin by introducing some notation. Let γ denote the boundary of U^{SE}, which is an orthogonal staircase curve from bottom-left to top-right. If $\gamma\cap U^{\text{SW}}\neq\emptyset$, then $\gamma\cap U^{\text{SW}}$ is a connected portion of γ that contains the bottom-left end of γ. Define σ as the "endpoint" of $\gamma\cap U^{\text{SW}}$, i.e., the point on $\gamma\cap U^{\text{SW}}$ that is closest the top-right end of γ. See Figure 2 for an illustration. If $\gamma\cap U^{\text{SW}}=\emptyset$, we define σ as the bottom-left end of γ (which

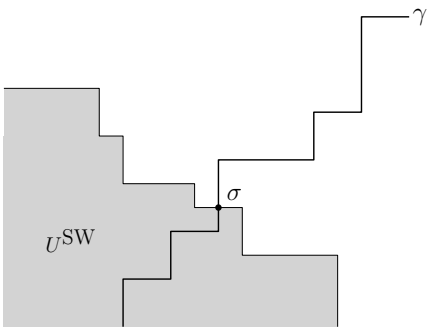


Figure 2 Illustrating the curve γ and the point σ.

is a point whose y-coordinate equals to $-\infty$). For a number $\tilde{y} \in \mathbb{R}$, we define $\phi(\tilde{y})$ as the *leftmost* point in $S \cap U^{\mathrm{SE}}$ whose y-coordinate is greater than $\tilde{y}$; we say $\phi(\tilde{y})$ does not exist if no point in $S \cap U^{\mathrm{SE}}$ has y-coordinate greater than $\tilde{y}$. For a point $a \in \mathbb{R}^2$ and a collection $\mathcal{P}$ of quadrants, we define $\Phi_{\rightarrow}(a, \mathcal{P})$ and $\Phi_{\uparrow}(a, \mathcal{P})$ as the rightmost and topmost quadrants in $\mathcal{P}$ that contains a, respectively. For a quadrant Q, we denote by $x(Q)$ and $y(Q)$ the x- and y-coordinates of the vertex of Q, respectively.

To get some intuition, let us consider a very simple case, where $\mathcal{Q}$ only consists of southeast quadrants. In this case, one can compute an optimal set cover for $(S \cap U^{\mathrm{SE}}, \mathcal{Q})$ using a greedy algorithm similar to the 1D interval-set-cover algorithm: repeatedly pick the leftmost uncovered point in $S \cap U^{\mathrm{SE}}$ and cover it using the topmost (southeast) quadrant in $\mathcal{Q}$. Using the notations defined above, we can describe this algorithm as follows. Set $\mathcal{Q}_{\mathrm{ans}} \leftarrow \emptyset$ and $\tilde{y} \leftarrow -\infty$ initially, and repeatedly do $a \leftarrow \phi(\tilde{y})$, $Q \leftarrow \Phi_{\uparrow}(\sigma, \mathcal{Q}^{\mathrm{SE}})$, $\mathcal{Q}_{\mathrm{ans}} \leftarrow \mathcal{Q}_{\mathrm{ans}} \cup \{Q\}$, $\tilde{y} \leftarrow y(Q)$ until $\phi(\tilde{y})$ does not exist. Eventually, $\mathcal{Q}_{\mathrm{ans}}$ is the set cover we want.

Now we try to extend this algorithm to the general case. However, the situation here becomes much more complicated, since we may have three other types of quadrants in $\mathcal{Q}$, which have to be carefully dealt with in order to guarantee the correctness. But the intuition remains the same: we still construct the solution in a greedy manner. The following procedure describes our algorithm, see also Figure 3.

1. $\mathcal{Q}_{\mathrm{ans}} \leftarrow \emptyset$. $\tilde{y} \leftarrow -\infty$. If $\phi(\tilde{y})$ does not exist, then go to Step 6.
2. $\mathcal{Q}_{\mathrm{ans}} \leftarrow \{\Phi_{\rightarrow}(\sigma, \mathcal{Q}^{\mathrm{SW}}), \Phi_{\uparrow}(\sigma, \mathcal{Q}^{\mathrm{SE}})\}$. $\tilde{y} \leftarrow y(\Phi_{\uparrow}(\sigma, \mathcal{Q}^{\mathrm{SE}}))$. If $\phi(\tilde{y})$ exists, then $a \leftarrow \phi(\tilde{y})$, else go to Step 6.
3. If $a \in U^{\mathrm{NE}}$, then $\mathcal{Q}_{\mathrm{ans}} \leftarrow \mathcal{Q}_{\mathrm{ans}} \cup \{\Phi_{\uparrow}(a, \mathcal{Q}^{\mathrm{NE}}), \Phi_{\uparrow}(a, \mathcal{Q}^{\mathrm{SE}})\}$ and go to Step 6.
4. If $a \in U^{\mathrm{NW}}$, then $\mathcal{Q}_{\mathrm{ans}} \leftarrow \mathcal{Q}_{\mathrm{ans}} \cup \{\Phi_{\rightarrow}(a, \mathcal{Q}^{\mathrm{NW}}), \Phi_{\uparrow}(a, \mathcal{Q}^{\mathrm{SE}})\}$ and $Q \leftarrow \Phi_{\uparrow}(v, \mathcal{Q}^{\mathrm{SE}})$ where v is the vertex of $\Phi_{\rightarrow}(a, \mathcal{Q}^{\mathrm{NW}})$, otherwise $Q \leftarrow \Phi_{\uparrow}(a, \mathcal{Q}^{\mathrm{SE}})$.
5. $\mathcal{Q}_{\mathrm{ans}} \leftarrow \mathcal{Q}_{\mathrm{ans}} \cup \{Q\}$. $\tilde{y} \leftarrow y(Q)$. If $\phi(\tilde{y})$ exists, then $a \leftarrow \phi(\tilde{y})$ and go to Step 3.
6. Output $\mathcal{Q}_{\mathrm{ans}}$.

The following lemma proves the correctness of our algorithm.

▶ **Lemma 8.** *$\mathcal{Q}_{\mathrm{ans}}$ covers all points in $S \cap U^{\mathrm{SE}}$, and $|\mathcal{Q}_{\mathrm{ans}}| = O(\mathsf{opt}^{\mathrm{SE}})$.*

The remaining task is to show how to perform our algorithm in $\widetilde{O}(\mathsf{opt}^{\mathrm{SE}})$ time using basic data structures. It is clear that our algorithm terminates in $O(\mathsf{opt}^{\mathrm{SE}})$ steps, since we include at least one quadrant to $\mathcal{Q}_{\mathrm{ans}}$ in each iteration of the loop Step 3–5 and eventually $|\mathcal{Q}_{\mathrm{ans}}| = O(\mathsf{opt}^{\mathrm{SE}})$ by Lemma 8. Thus, it suffices to show that each step can be done in $\widetilde{O}(1)$ time. In every step of our algorithm, all work can be done in constant time except the tasks of computing the point σ, testing whether $a \in U^{\mathrm{NE}}$ and $a \in U^{\mathrm{NW}}$ for a given point

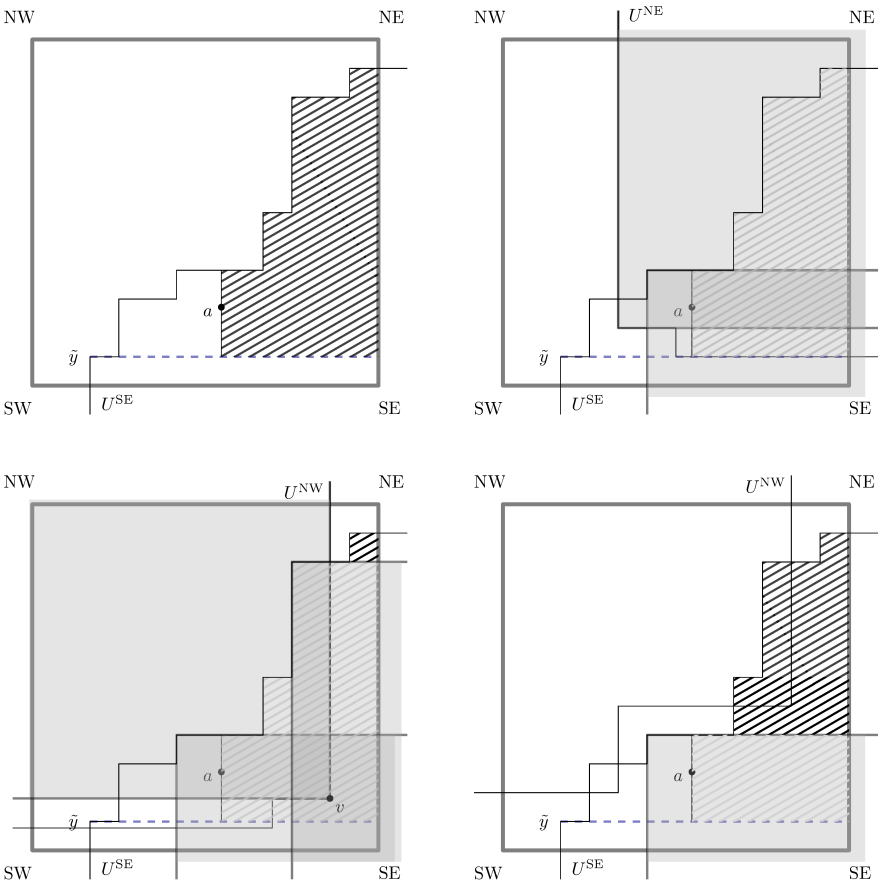


Figure 3 Yet uncovered points lie in the hatch-shaded region (top left). In Step 3, the entire region can be covered by two quadrants if $a \in U^{\mathrm{NE}}$ (top right). In Step 4, the hatch-shaded region can be reduced using one or three quadrants, depending on whether $a \in U^{\mathrm{NW}}$ (bottom). After Step 4, any quadrant intersecting the remaining hatch-shaded region will not cover a.

a, computing the quadrants $\Phi_{\rightarrow}(a, \mathcal{Q}^{\mathrm{SW}}), \Phi_{\rightarrow}(a, \mathcal{Q}^{\mathrm{NW}}), \Phi_{\uparrow}(a, \mathcal{Q}^{\mathrm{SE}}), \Phi_{\uparrow}(a, \mathcal{Q}^{\mathrm{NE}})$ for a given point a, and computing $\phi(\tilde{y})$ for a given number $\tilde{y}$. All these tasks except the computation of $\phi(\cdot)$ can be easily done in $\widetilde{O}(1)$ time by storing the quadrants in binary search trees. To compute $\phi(\cdot)$ in $\widetilde{O}(1)$ time is more difficult, and we achieve this by properly using range trees built on both S and $\mathcal{Q}^{\mathrm{SE}}$. The details are presented in the full version [2].

Using the above algorithm, we can compute $O(1)$-approximate optimal set covers for $(S \cap U^{\mathrm{SE}}, \mathcal{Q})$, $(S \cap U^{\mathrm{SW}}, \mathcal{Q})$, $(S \cap U^{\mathrm{NE}}, \mathcal{Q})$, and $(S \cap U^{\mathrm{NW}}, \mathcal{Q})$. As argued before, the union of these four set covers, denoted by $\mathcal{Q}^*$, is an $O(1)$-approximate optimal set covers for $(S, \mathcal{Q})$.

▶ **Theorem 9.** *Quadrant set cover admits an $O(1)$-approximate output-sensitive algorithm.*

3.3 Putting everything together

With the bootstrapping theorem in hand, we are now able to design our dynamic quadrant-set-cover data structure. Again, the starting point is a "trivial" data structure which uses the output-sensitive algorithm of Theorem 9 to re-compute an optimal quadrant set cover after

each update. Clearly, the update time of this data structure is $\widetilde{O}(n)$ and the construction time is $\widetilde{O}(n_0)$. Let $\mu = O(1)$ be the approximation ratio of the output-sensitive algorithm. The trivial data structure implies the existence of a $(\mu+\varepsilon)$-approximate dynamic quadrant-set-cover data structure with $\widetilde{O}(n^{\alpha_0}/\varepsilon^{1-\alpha_0})$ amortized update time for $\alpha_0 = 1$ and $\widetilde{O}(n_0)$ construction time. Define $\alpha_i = 2\alpha_{i-1}/(1+2\alpha_{i-1})$ for $i \geq 1$. By applying Theorem 5 i times for a constant $i \geq 1$, we see the existence of a $(\mu+\varepsilon)$-approximate dynamic quadrant-set-cover data structure with $\widetilde{O}(n^{\alpha_i}/\varepsilon^{1-\alpha_i})$ amortized update time and $\widetilde{O}(n_0)$ construction time. One can easily verify that $\alpha_i = 2^i/(2^{i+1}-1)$ for all $i \geq 0$. Therefore, for any constant $\alpha > 0$, we have a constant $i \geq 0$ satisfying $\alpha_i < 1/2 + \alpha$ and hence $\widetilde{O}(n^{\alpha_i}/\varepsilon^{1-\alpha_i}) = O(n^{1/2+\alpha}/\varepsilon)$. Setting ε to be any constant, we finally conclude the following.

▶ **Theorem 10.** *Let $(S, \mathcal{Q})$ be an instance of quadrant set cover, with $n = |S| + |\mathcal{Q}|$. Let $\alpha > 0$ be an arbitrarily small constant. There exists an $O(1)$-approximate dynamic quadrant-set-cover data structure with $O(n^{1/2+\alpha})$ amortized update time.*

As mentioned earlier, set cover for unit squares can be reduced to instances of quadrant set cover. In particular, we prove the following lemma in the full version [2]:

▶ **Lemma 11.** *Suppose there exists a c-approximate dynamic quadrant-set-cover data structure with $f(n)$ amortized update time, where f is an increasing function. Then there exist $O(c)$-approximate dynamic unit-square-set-cover, dynamic unit-square-hitting-set, and dynamic quadrant-hitting-set data structures with $\widetilde{O}(f(n))$ amortized update time.*

Finally, it can be verified that the hitting-set problem for quadrants (resp., unit squares) is the same as the set-cover problem for quadrants (resp., unit squares). We thus conclude the following.

▶ **Theorem 12.** *Let $(S, \mathcal{R})$ be an instance of unit-square set cover, unit-square hitting set, or quadrant hitting set, with $n = |S| + |\mathcal{R}|$. Let $\alpha > 0$ be an arbitrarily small constant. There exists an $O(1)$-approximate dynamic data structure for $(S, \mathcal{R})$ with $O(n^{1/2+\alpha})$ amortized update time.*

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 SIGACT Symposium on Theory of Computing*, pages 114–125. ACM, 2019.
2. Pankaj K. Agarwal, Hsien-Chih Chang, Subhash Suri, Allen Xiao, and Jie Xue. Dynamic geometric set cover and hitting set. *arXiv preprint*, 2020. `arXiv:2003.00202`.
3. Pankaj K. Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. In *Proceedings of the thirtieth annual symposium on Computational geometry*, page 271. ACM, 2014.
4. Pankaj K. Agarwal, Junyi Xie, Jun Yang, and Hai Yu. Monitoring continuous band-join queries over dynamic data. In *16th International Symposium on Algorithms and Computation (ISAAC)*, pages 349–359. Springer, 2005.
5. Piotr Berman and Bhaskar DasGupta. Complexities of efficient solutions of rectilinear polygon cover problems. *Algorithmica*, 17(4):331–356, 1997.
6. Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. Design of dynamic algorithms via primal-dual method. In *International Colloquium on Automata, Languages, and Programming*, pages 206–218. Springer, 2015.
7. Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

8 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM Symposium on Theory of Computing*, pages 624–633. ACM, 2014.

9 Thomas Erlebach and Erik Jan Van Leeuwen. Ptas for weighted set cover on unit squares. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 166–177. Springer, 2010.

10 Shashidhara K. Ganjugunte. *Geometric hitting sets and their variants*. PhD thesis, Duke University, 2011.

11 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 537–550. ACM, 2017.

12 Juris Hartmanis. Computers and intractability: a guide to the theory of np-completeness. *Siam Review*, 24(1):90, 1982.

13 David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.

14 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

15 László Lovász. On the ratio of optimal integral and fractional covers. *Discrete mathematics*, 13(4):383–390, 1975.

16 Nimrod Megiddo and Kenneth J. Supowit. On the complexity of some common geometric location problems. *SIAM journal on computing*, 13(1):182–196, 1984.

17 Nimrod Megiddo and Arie Tamir. On the complexity of locating linear facilities in the plane. *Operations research letters*, 1(5):194–197, 1982.

18 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.