Dynamic geometric set cover and hitting set

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

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

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

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

Jie Xue
Department of Computer Science, University of California at Santa Barbara, 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

Lines 500

1 Introduction

Given a pair $(S, \mathcal{R})$ where S is a set of points and $\mathcal{R}$ is a collection of geometric ranges in a Euclidean space, 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 algorithm 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.

Although (static) geometric set cover and hitting set have been extensively studied over the years, their dynamic variants are surprisingly open. For example, even for the most fundamental case, dynamic *interval* set cover and hitting set in one dimension, no nontrivial results were previously known. In this paper, we propose two algorithmic frameworks for the problems, which lead to efficient data structures for dynamic set cover and hitting set for intervals in $\mathbb{R}^1$ and unit squares and quadrants in $\mathbb{R}^2$. We believe that our approaches can be extended to solve dynamic set cover and hitting set in other geometric settings, or more generally, other dynamic problems in computational geometry.

1.1 Related work

The set cover and hitting set problems in general setting are well-known to be NP-complete [17]. A simple greedy algorithm achieves an $O(\log n)$-approximation [11, 18, 20], which is tight under appropriate complexity-theoretic assumptions [12, 19]. In many geometric settings, the problems remain NP-hard or even hard to approximate [5, 21, 22]. However, by exploiting the geometric nature of the problems, efficient algorithms with better approximation factors can be obtained. For example, Mustafa and Ray [23] showed the existence of polynomial-time approximation schemes (PTAS) for halfspace hitting set in $\mathbb{R}^3$ and disk hitting set. There is also a PTAS for unit-square set cover given by Erlebach and van Leeuwen [13]. Agarwal and Pan [3] proposed approximation algorithms with near-linear running time for the set cover and hitting set problems for halfspaces in $\mathbb{R}^3$, disks in $\mathbb{R}^2$, and orthogonal rectangles.

Dynamic problems have received a considerable attention in recent years [4, 6, 7, 9, 10, 16, 24, 25]. In particular, dynamic set cover in general setting has been studied in [1, 8, 15]. All the results were achieved in the partially dynamic setting where ranges are fixed and only points are dynamic. Gupta et al. [15] showed that an $O(\log n)$-approximation can be maintained using $O(f \log n)$ amortized update time and an $O(f^3)$-approximation can be maintained using $O(f^2)$ amortized update time, where f is the maximum number of ranges that a point belongs to. Bhattacharya et al. [8] gave an $O(f^2)$-approximation data structure for dynamic set cover with $O(f \log n)$ amortized update time. Abboud et al. [1] proved that one can maintain a $(1+\varepsilon)f$-approximation using $O(f^2 \log n/\varepsilon^5)$ amortized update time.

In geometric settings, only the dynamic hitting set problem has been considered [14]. Ganjugunte [14] studied two different dynamic settings: **(i)** only the range set $\mathcal{R}$ is dynamic and **(ii)** $\mathcal{R}$ is dynamic and S is semi-dynamic (i.e., insertion-only). Ganjugunte [14] showed that, for pseudo-disks in $\mathbb{R}^2$, dynamic hitting set in setting **(i)** can be solved using $O(\gamma(n) \log^4 n)$ amortized update time with approximation factor $O(\log^2 n)$, and that in setting **(ii)** can be solved using $O(\gamma(n)\sqrt{n} \log^4 n)$ amortized update time with approximation factor $O(\log^6 n/ \log\log n)$, where $\gamma(n)$ is the time for finding a point in X contained in a query pseudo-trapezoid (see [14] for details). Dynamic geometric hitting set in the fully dynamic setting (where both points and ranges can be inserted or deleted) as well as dynamic geometric set cover has not yet been studied before, to the best of our knowledge.

1.2 Our results

Let $(S, \mathcal{R})$ be a dynamic geometric set cover (resp., hitting set) instance. We are interested in proposing efficient data structures to *maintain* an (approximately) optimal solution for $(S, \mathcal{R})$. This may have various definitions, resulting in different variants of the problem. A natural variant is to maintain the number opt, which is the size of an optimal set cover (resp., hitting set) for $(S, \mathcal{R})$, or an approximation of opt. However, in many applications, only maintaining the optimum is not satisfactory and one may hope to maintain a "real" set cover (resp., hitting set) for the dynamic instance. Therefore, in this paper, we formulate the problem as follows. We require the data structure to, after each update, store (implicitly) a solution for the current problem instance satisfying some quality requirement such that certain information about the solution can be queried efficiently. For example, one can ask how large the solution is, whether a specific element in $\mathcal{R}$ (resp., S) is used in the solution, what the entire solution is, etc. We will make this more precise shortly.

In dynamic settings, it is usually more natural and convenient to consider a *multiset* solution for the set cover (resp., hitting set) instance. 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. In the static problems, one can always efficiently remove the duplicates in a multiset solution to obtain a (ordinary) set cover or hitting set with even better quality (i.e., smaller cardinality), hence computing a multiset solution is essentially equivalent to computing an ordinary solution. However, in dynamic settings, the update time has to be sublinear, and in general this is not sufficient for detecting and removing duplicates. Therefore, in this paper, we mainly focus on multiset solutions (though some of our data structures can maintain an ordinary set cover or hitting set). Unless explicitly mentioned otherwise, solutions for set cover and hitting set always refer to multiset solutions hereafter.

Precisely, we require a dynamic set cover (resp., hitting set) data structure to store implicitly, after each update, a set cover $\mathcal{R}'$ (resp., a hitting set S') for the current instance $(S, \mathcal{R})$ such that the following queries are supported.

- **Size query**: reporting the (multiset) size of $\mathcal{R}'$ (resp., S').
- **Membership query**: reporting, for a given range $R \in \mathcal{R}$ (resp., a given point $a \in S$), the number of copies of R (resp., a) contained in $\mathcal{R}'$ (resp., S').
- **Reporting query**: reporting all the 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. This paper mainly focuses on the fully dynamic setting, while some results are achieved in the partially dynamic setting. Thus, unless explicitly mentioned otherwise, problems are always considered in the fully dynamic setting.

The main contribution of this paper are two frameworks for designing 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, quadrant/unit-square set cover and hitting set (see the first three rows of Table 1 for detailed bounds). The second framework is based on local modification, which results in efficient (approximate) dynamic data structures for interval hitting set, quadrant/unit-square set cover and hitting set in the *partially* dynamic setting (see the last three rows of Table 1 for detailed bounds).

Organization. The rest of the paper is organized as follows. Section 2 gives the preliminaries required for the paper and Section 3 presents an overview of our two frameworks. Due to limited space, only the results of our first framework (bootstrapping) are discussed in this paper: Section 4 presents the 1D results and Section 5 presents the 2D results. The results of our second framework (local modification) can be found in the full version [2]. Also, all the omitted proofs and details are presented in the full version [2].

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

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.

2 Preliminaries

In this section, we introduce the basic notions used throughout the paper.

Multi-sets and disjoint union. A *multi-set* is a set in which elements can have multiple copies. The *multiplicity* of an element a in a multi-set A is the number of the copies of a in A. For two multi-sets A and B, we use $A \sqcup B$ to denote the *disjoint union* of A and B, in which the multiplicity of an element a is the sum of the multiplicities of a in A and B.

Basic data structures. A data structure built on a dataset (e.g., point set, range set, set cover or hitting set instances) of size n is *basic* if it can be constructed in $\widetilde{O}(n)$ time and can be dynamized with $\widetilde{O}(1)$ update time (with a bit abuse of terminology, sometimes we also use "basic data structures" to denote the dynamized version of such data structures).

Output-sensitive algorithms. In some set cover and hitting set problems, if the problem instance is properly stored in some data structure, it is possible to compute an (approximate) optimal solution in sub-linear time. An *output-sensitive* algorithm for a set cover or hitting set problem refers to an algorithm that can compute an (approximate) optimal solution in $\widetilde{O}(\mathsf{out})$ time (where out is the size of the output solution), by using some *basic* data structure built on the problem instance.

3 An overview of our two frameworks

The basic idea of our first framework is *bootstrapping*. Namely, we begin from a simple inefficient 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.

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].

4 Warm-up: 1D set cover for intervals

As a warm up for our bootstrapping framework, we first study the 1D problem: dynamic interval set cover. First, we observe that interval set cover admits a simple *exact* output-sensitive algorithm. Indeed, 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, and the algorithm can be easily made output-sensitive if we store the points and intervals in binary search trees.

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

This algorithm will serve an important role in the design of our data structure.

4.1 Bootstrapping

As mentioned before, our data structure is designed using bootstrapping. Specifically, we prove the following bootstrapping theorem, which is the technical heart of our result. The theorem roughly states that given a dynamic interval set cover data structure, one can obtain another dynamic interval set cover data structure with improved update time.

▶ **Theorem 2.** *Let $\alpha \in [0,1]$ be a number. If there exists a $(1+\varepsilon)$-approximate dynamic interval set cover data structure $\mathcal{D}_{\text{old}}$ with $\widetilde{O}(n^\alpha/\varepsilon^{1-\alpha})$ amortized update time and $\widetilde{O}(n_0)$ 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 $\widetilde{O}(n_0)$ construction time for any $\varepsilon > 0$, where $\alpha' = \alpha/(1+\alpha)$. Here n (resp., n_0) denotes the size of the current (resp., initial) problem instance.*

Assuming the existence of $\mathcal{D}_{\text{old}}$ as in the theorem, we are going to design the improved data structure $\mathcal{D}_{\text{new}}$. Let $(S, \mathcal{I})$ be a dynamic interval set cover instance, and $\varepsilon > 0$ be the approximation factor. We denote by n (resp., n_0) the size of the current (resp., initial) $(S, \mathcal{I})$.

The construction of $\mathcal{D}_{\text{new}}$. Initially, $|S| + |\mathcal{I}| = n_0$. Essentially, our data structure $\mathcal{D}_{\text{new}}$ consists of two parts[1]. 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 defined as follows. Let f be a function to be determined shortly. We partition the real line $\mathbb{R}$ into $r = \lceil n_0/f(n_0, \varepsilon) \rceil$ connected portions (i.e., intervals) $J_1, \dots, J_r$ such that each portion J_i contains $O(f(n_0, \varepsilon))$ points in S and $O(f(n_0, \varepsilon))$ endpoints of the intervals in $\mathcal{I}$. Define $S_i = S \cap J_i$ and define $\mathcal{I}_i \subseteq \mathcal{I}$ as the sub-collections consisting of the intervals that "partially intersect" J_i, i.e., $\mathcal{I}_i = \{I \in \mathcal{I} : J_i \cap I \neq \emptyset \text{ and } J_i \nsubseteq I\}$. When the instance $(S, \mathcal{I})$ is updated, the partition $J_1, \dots, J_r$ will remain unchanged, but the S_i's and $\mathcal{I}_i$'s will change along with S and $\mathcal{I}$. We view each $(S_i, \mathcal{I}_i)$ as a dynamic interval set cover instance, and let $\mathcal{D}_{\text{old}}^{(i)}$ be

[1] In implementation level, we may need some additional support data structures (which are very simple). For simplicity of exposition, we shall mention them when discussing the implementation details.

the data structure $\mathcal{D}_{\text{old}}$ built on $(S_i, \mathcal{I}_i)$ for the approximation parameter $\tilde{\varepsilon} = \varepsilon/2$. Thus, $\mathcal{D}_{\text{old}}^{(i)}$ maintains a $(1+\tilde{\varepsilon})$-approximate optimal set cover for $(S_i, \mathcal{I}_i)$. The second part of $\mathcal{D}_{\text{new}}$ consists of the data structures $\mathcal{D}_{\text{old}}^{(1)}, \dots, \mathcal{D}_{\text{old}}^{(r)}$.

Update and reconstruction. After an operation on $(S, \mathcal{I})$, we update the basic data structure $\mathcal{A}$. Also, 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 operation on S changes exactly one S_i and an operation 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. Besides the update, we also reconstruct the entire data structure $\mathcal{D}_{\text{new}}$ periodically (as is the case for many dynamic data structures). Specifically, the first reconstruction of $\mathcal{D}_{\text{new}}$ happens after processing $f(n_0, \varepsilon)$ operations. The reconstruction is the same as the initial construction of $\mathcal{D}_{\text{new}}$, except that n_0 is replaced with n_1, the size of $(S, \mathcal{I})$ at the time of reconstruction. Then the second reconstruction happens after processing $f(n_1, \varepsilon)$ operations since the first reconstruction, and so forth.

Maintaining a solution. We now discuss how to maintain a $(1+\varepsilon)$-approximate optimal set cover $\mathcal{I}_{\text{appx}}$ for $(S, \mathcal{I})$. Let opt denote the optimum (i.e., the size of an optimal set cover) of the current $(S, \mathcal{I})$. Set $\delta = \min\{(6+2\varepsilon) \cdot r/\varepsilon, n\}$. If $\mathsf{opt} \leq \delta$, then the output-sensitive algorithm can compute an optimal set cover for $(S, \mathcal{I})$ in $\widetilde{O}(\delta)$ time. Thus, we simulate the output-sensitive 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 $i \in \{1, \dots, r\}$, 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 $P' = \{i : J_i \text{ is uncoverable}\}$. We try to use the intervals in $\mathcal{I}$ to "cover" all coverable portions. That is, for each $i \in P$, we find an interval in $\mathcal{I}$ that contains J_i, and denote by $\mathcal{I}^*$ the collection of these intervals. Then we consider the uncoverable portions. If for some $i \in P'$, the data structure $\mathcal{D}_{\text{old}}^{(i)}$ tells us that the current $(S_i, \mathcal{I}_i)$ does not have a set cover, then we immediately make a no-solution decision, i.e., decide that the current $(S, \mathcal{I})$ has no feasible set cover, and continue to the next operation. Otherwise, for every $i \in P'$, the data structure $\mathcal{D}_{\text{old}}^{(i)}$ maintains a $(1+\tilde{\varepsilon})$-approximate optimal set cover $\mathcal{I}_i^*$ for $(S_i, \mathcal{I}_i)$. We then define $\mathcal{I}_{\text{appx}} = \mathcal{I}^* \sqcup \left(\bigsqcup_{J_i \in \mathcal{P}'} \mathcal{I}_i^*\right)$.

Later we will prove that $\mathcal{I}_{\text{appx}}$ is always a $(1+\varepsilon)$-approximate optimal set cover for $(S, \mathcal{I})$. Before this, let us consider 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 thus we are not able to explicitly extract 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 actually only need to compute P, P', and $\mathcal{I}^*$; with these in hand, one can already build a data structure to support the desired queries for $\mathcal{I}_{\text{appx}}$. A detailed discussion is presented in the full version [2].

Correctness. We now prove the correctness of our data structure $\mathcal{D}_{\text{new}}$. We first show the correctness of the no-solution decision.

▶ **Lemma 3.** *$\mathcal{D}_{\text{new}}$ makes a no-solution decision iff the current $(S, \mathcal{I})$ has no set cover.*

Next, we show that the solution $\mathcal{I}_{\text{appx}}$ maintained by $\mathcal{D}_{\text{new}}$ is truly a $(1+\varepsilon)$-approximate optimal set cover for $(S, \mathcal{I})$. If $\mathcal{I}_{\text{appx}}$ is computed by the output-sensitive algorithm, then it is an optimal set cover for $(S, \mathcal{I})$. Otherwise, $\mathsf{opt} > \delta = \min\{(6+2\varepsilon)\cdot r/\varepsilon, n\}$, i.e., either $\mathsf{opt} > (6+2\varepsilon)\cdot r/\varepsilon$ or $\mathsf{opt} > n$. 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 by Lemma 3. So assume $\mathsf{opt} > (6+2\varepsilon)\cdot r/\varepsilon$. In this case, $\mathcal{I}_{\text{appx}} = \mathcal{I}^* \sqcup (\bigsqcup_{i\in P'} \mathcal{I}_i^*)$. For each $i \in P'$, 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 $i \in P'$ where $\tilde{\varepsilon} = \varepsilon/2$. Since $|\mathcal{I}^*| \le r$, we have

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

Let $\mathcal{I}_{\text{opt}}$ be an optimal set cover for $(S, \mathcal{I})$. We observe that for $i \in P'$, $\mathcal{I}_{\text{opt}} \cap \mathcal{I}_i$ 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 $\mathcal{I}\backslash\mathcal{I}_i$). It immediately follows that $\mathsf{opt}_i \le |\mathcal{I}_{\text{opt}} \cap \mathcal{I}_i|$ for all $i \in P'$. Therefore, we have

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

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

The above lemma immediately implies

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

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

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

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

Time analysis. We briefly discuss the update and construction time of $\mathcal{D}_{\text{new}}$; a detailed analysis can be found in the full version [2]. Since $\mathcal{D}_{\text{new}}$ is reconstructed periodically, it suffices to consider the first period (i.e., the period before the first reconstruction). 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 $\mathcal{D}_{\text{old}}^{(1)}, \dots, \mathcal{D}_{\text{old}}^{(r)}$, the time for maintaining the solution, and the time for reconstruction. Since the period consists of $f(n_0, \varepsilon)$ operations, the size of each $(S_i, \mathcal{I}_i)$ is always bounded by $O(f(n_0, \varepsilon))$ during the period. As argued before, we only need to update at most two $\mathcal{D}_{\text{old}}^{(i)}$'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 the solution can be done in $\widetilde{O}(\delta + r)$ time, with a careful implementation. The time for reconstruction is bounded by $\widetilde{O}(n_0 + f(n_0, \varepsilon))$; we amortize it over the $f(n_0, \varepsilon)$ operations in the period and the amortized time cost is then $\widetilde{O}(n_0/f(n_0, \varepsilon))$, i.e., $\widetilde{O}(r)$. In total, the amortized update time of $\mathcal{D}_{\text{new}}$ (during the first period) is $\widetilde{O}(f(n_0,\varepsilon)^\alpha/\varepsilon^{1-\alpha} + \delta + r)$. If we set $f(n,\varepsilon) = \min\{n^{1-\alpha'}/\varepsilon^{\alpha'}, n/2\}$ where α' is as defined in Theorem 2, a careful calculation (see the full version) shows that the amortized update time becomes $\widetilde{O}(n^{\alpha'}/\varepsilon^{1-\alpha'})$.

4.2 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 re-compute 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 5.** *For a given approximation factor $\varepsilon > 0$ and any constant $\alpha > 0$, there exists a $(1+\varepsilon)$-approximate dynamic interval set cover data structure $\mathcal{D}$ with $O(n^{\alpha}/\varepsilon)$ amortized update time and $\widetilde{O}(n_0)$ construction time.*

5 2D set cover and hitting set for quadrants and unit squares

In this section, we present our bootstrapping framework for 2D dynamic set cover and hitting set. Our framework works for quadrants and unit squares.

We first show that dynamic unit-square set cover, dynamic unit-square hitting set, and dynamic quadrant hitting set can all be reduced to dynamic quadrant set cover.

▶ **Lemma 6.** *Suppose there exists a c-approximate dynamic quadrant set cover data structure with $f(n)$ amortized update time and $\widetilde{O}(n_0)$ construction 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 and $\widetilde{O}(n_0)$ construction time.*

Now it suffices to consider dynamic quadrant set cover. In order to do bootstrapping, we need an output-sensitive algorithm for quadrant set cover, analog to the one in Lemma 1 for intervals. To design such an algorithm is considerably more difficult compared to the 1D case, and we defer it to Section 5.2. Before this, let us first discuss the bootstrapping procedure, assuming the existence of a μ-approximate output-sensitive algorithm for quadrant set cover.

5.1 Bootstrapping

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

▶ **Theorem 7.** *Assume quadrant set cover admits a μ-approximate output-sensitive algorithm for some constant $\mu \geq 1$. Then we have the following result.*

(*) *Let $\alpha \in [0,1]$ be a number. If there exists a $(\mu+\varepsilon)$-approximate dynamic quadrant set cover data structure $\mathcal{D}_{\text{old}}$ with $\widetilde{O}(n^{\alpha}/\varepsilon^{1-\alpha})$ amortized update time and $\widetilde{O}(n_0)$ construction time for any $\varepsilon > 0$, 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 $\widetilde{O}(n_0)$ construction time for any $\varepsilon > 0$, where $\alpha' = 2\alpha/(1+2\alpha)$. Here n (resp., n_0) denotes the size of the current (resp., initial) problem instance.*

Assuming the existence of $\mathcal{D}_{\text{old}}$ as in the theorem, we are going to design the improved data structure $\mathcal{D}_{\text{new}}$. Let $(S, \mathcal{Q})$ be a dynamic quadrant set cover instance. As before, we denote by n (resp., n_0) the size of the current (resp., initial) $(S, \mathcal{Q})$.

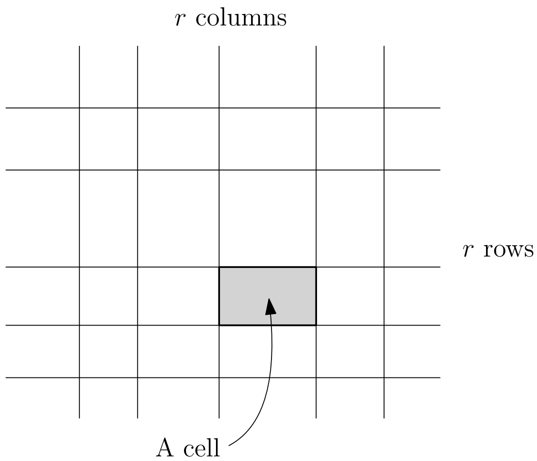

Figure 1 The $r \times r$ grid. Note that the cells may have different sizes.

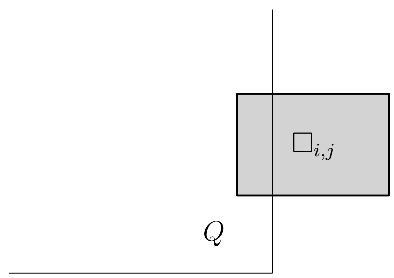

Figure 2 A quadrant Q that left intersects $\square_{i,j}$.

The construction of $\mathcal{D}_{\mathbf{new}}$. Initially, $|S| + |\mathcal{Q}| = n_0$. Essentially, our 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 be a function to be determined shortly. We use an orthogonal grid to partition the plane $\mathbb{R}^2$ into $r \times r$ cells for $r = \lceil n_0/f(n_0, \varepsilon) \rceil$ such that each row (resp., column) of the grid contains $O(f(n_0, \varepsilon))$ points in S and $O(f(n_0, \varepsilon))$ vertices of the quadrants in $\mathcal{Q}$ (see Figure 1 for an illustration). Denote by $\square_{i,j}$ the cell in the i-th row and j-th column. Define $S_{i,j} = S \cap \square_{i,j}$. Also, we need to define a sub-collection $\mathcal{Q}_{i,j} \subseteq \mathcal{Q}$. 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, for technical reasons, here we cannot simply define $\mathcal{Q}_{i,j}$ as the sub-collection of quadrants in $\mathcal{Q}$ that partially intersects $\square_{i,j}$. Instead, we define $\mathcal{Q}_{i,j}$ 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 Figure 2 for an illustration); similarly, we define "right intersects", "top intersects", and "bottom intersects". Among a collection of quadrants, the *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). When the instance $(S, \mathcal{Q})$ is updated, the grid keeps unchanged, but the $S_{i,j}$'s and $\mathcal{Q}_{i,j}$'s change along with S and $\mathcal{Q}$. We view each $(S_{i,j}, \mathcal{Q}_{i,j})$ as a dynamic quadrant set cover instance, and let $\mathcal{D}_{\text{old}}^{(i,j)}$

be the data structure $\mathcal{D}_{\text{old}}$ built on $(S_{i,j}, \mathcal{Q}_{i,j})$ for the 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 $i, j \in \{1, \ldots, r\}$.

Update and reconstruction. After each operation on $(S, \mathcal{Q})$, we update the data structure $\mathcal{A}$. Also, if some $(S_{i,j}, \mathcal{Q}_{i,j})$ changes, we update the data structure $\mathcal{D}_{\text{old}}^{(i,j)}$. Note that an operation on S changes exactly one $S_{i,j}$, and an operation 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. Besides the update, we also reconstruct the entire data structure $\mathcal{D}_{\text{new}}$ periodically, where the (first) reconstruction happens after processing $f(n_0, \varepsilon)$ operations. This part is totally the same as in our 1D data structure.

Maintaining a solution. We now discuss how to maintain a $(\mu + \varepsilon)$-approximate optimal set cover $\mathcal{Q}_{\text{appx}}$ for $(S, \mathcal{Q})$. Let opt denote the optimum of the current $(S, \mathcal{Q})$. Set $\delta = \min\{(8\mu + 4\varepsilon + 2) \cdot r^2/\varepsilon, n\}$. If $\mathsf{opt} \leq \delta$, then the output-sensitive algorithm can compute a μ-approximate optimal set cover for $(S, \mathcal{Q})$ in $\widetilde{O}(\mu\delta)$ time. Thus, we simulate the output-sensitive algorithm within that amount of time. If the algorithm successfully computes a solution, 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 $P' = \{(i,j) : \square_{i,j} \text{ is uncoverable}\}$. We try to use the quadrants in $\mathcal{I}$ to "cover" all coverable cells. That is, for each $(i,j) \in P$, we find a quadrant in $\mathcal{Q}$ that contains $\square_{i,j}$, and denote by $\mathcal{Q}^*$ the set of all these quadrants. Then we consider the uncoverable cells. If for some $(i,j) \in P'$, the data structure $\mathcal{D}_{\text{old}}^{(i,j)}$ tells us that the instance $(S_{i,j}, \mathcal{Q}_{i,j})$ has no set cover, then we immediately make a no-solution decision, i.e., decide that the current $(S, \mathcal{Q})$ has no feasible set cover, and continue to the next operation. Otherwise, for each $(i,j) \in P'$, the data structure $\mathcal{D}_{\text{old}}^{(i,j)}$ maintains a $(\mu + \tilde{\varepsilon})$-approximate optimal set cover $\mathcal{Q}_{i,j}^*$ for $(S_{i,j}, \mathcal{Q}_{i,j})$. We then define $\mathcal{Q}_{\text{appx}} = \mathcal{Q}^* \sqcup \left(\bigsqcup_{(i,j) \in P'} \mathcal{Q}_{i,j}^*\right)$.

We will see later that $\mathcal{Q}_{\text{appx}}$ is always a $(\mu + \varepsilon)$-approximate optimal set cover for $(S, \mathcal{Q})$. Before this, let us briefly discuss how to store $\mathcal{Q}_{\text{appx}}$ to support the desired queries. If $\mathcal{Q}_{\text{appx}}$ is computed by the output-sensitive algorithm, then we have all the elements of $\mathcal{Q}_{\text{appx}}$ in hand and can easily store them in a data structure to support the queries. Otherwise, $\mathcal{Q}_{\text{appx}}$ is defined as the disjoint union of $\mathcal{Q}^*$ and $\mathcal{Q}_{i,j}^*$'s. In this case, the size and reporting queries can be handled in the same way as that in the 1D problem, by taking advantage of the fact that $\mathcal{Q}_{i,j}^*$ is maintained in $\mathcal{D}_{\text{old}}^{(i,j)}$. However, the situation for the membership query is more complicated, because now a quadrant in $\mathcal{Q}$ may belong to many $\mathcal{Q}_{i,j}^*$'s. This issue can be handled by collecting all special quadrants in $\mathcal{Q}_{\text{appx}}$ and building on them a data structure supporting the membership query. A detailed discussion is presented in the full version [2].

Correctness. We now prove the correctness of our data structure $\mathcal{D}_{\text{new}}$. First, we show that the no-solution decision made by our data structure is correct.

▶ **Lemma 8.** *$\mathcal{D}_{\text{new}}$ makes a no-solution decision iff the current $(S, \mathcal{Q})$ has no set cover.*

Next, we show that the solution $\mathcal{Q}_{\text{appx}}$ maintained by $\mathcal{D}_{\text{new}}$ is truly 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 by Lemma 8. So assume $\mathsf{opt} > (8\mu + 4\varepsilon + 2)r^2/\varepsilon$. In this case, $\mathcal{Q}_{\text{appx}} = \mathcal{Q}^* \sqcup (\bigsqcup_{(i,j) \in P'} \mathcal{Q}_{i,j}^*)$. For each $(i,j) \in P'$, let

$\mathsf{opt}_{i,j}$ be the optimum of $(S_{i,j}, \mathcal{Q}_{i,j})$. Then we have $|\mathcal{Q}^*_{i,j}| \le (\mu + \tilde{\varepsilon}) \cdot \mathsf{opt}_{i,j}$ for all $(i,j) \in P'$ where $\tilde{\varepsilon} = \varepsilon/2$. Since $|\mathcal{Q}^*| \le r^2$, we have

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

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

$$\mathsf{opt} + 4r^2 = |\mathcal{Q}_{\text{opt}}| + 4r^2 \ge \sum_{(i,j)\in P'} \mathsf{opt}_{i,j}. \tag{5}$$

Using Equations 4 and 5, we deduce that

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

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

Time analysis. We briefly discuss the update and construction time of $\mathcal{D}_{\text{new}}$; a detailed analysis can be found in the full version [2]. It suffices to consider the first period (i.e., the period before the first reconstruction). We first observe the following fact.

▶ **Lemma 10.** *At any time in the first period, we have $\sum_{k=1}^{r}(|S_{i,k}| + |\mathcal{Q}_{i,k}|) = O(f(n_0, \varepsilon) + r)$ for all $i \in \{1, \dots, r\}$ and $\sum_{k=1}^{r}(|S_{k,j}| + |\mathcal{Q}_{k,j}|) = O(f(n_0, \varepsilon) + r)$ for all $j \in \{1, \dots, r\}$.*

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 easily done in $\widetilde{O}(n_0 + r^2)$ time. The update time of $\mathcal{D}_{\text{new}}$ consists of the time for reconstruction, the time for updating $\mathcal{A}$ and $\mathcal{D}^{(i,j)}_{\text{old}}$'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 the solution can be done in $\widetilde{O}(\delta + r^2)$ time, with a careful implementation. The time for updating the $\mathcal{D}_{\text{old}}$ data structures 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 $\mathcal{D}_{\text{old}}$ data structures in one row and one column (say the i-th row and j-th column). Hence, updating the $\mathcal{D}_{\text{old}}$ data structures 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 10 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 10,

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

and similarly $\sum_{k=1}^{r} m_{k,j}^{\alpha} = O(r^{1-\alpha} \cdot (f(n_0, \varepsilon) + r)^{\alpha})$. It follows that updating the $\mathcal{D}_{\text{old}}$ data structures takes $\widetilde{O}(r^{1-\alpha} \cdot (f(n_0, \varepsilon) + r)^{\alpha}/\varepsilon^{1-\alpha})$ amortized time. In total, the amortized update time of $\mathcal{D}_{\text{new}}$ (during the first period) is $\widetilde{O}(r^{1-\alpha} \cdot (f(n_0, \varepsilon) + r)^{\alpha} + \delta + r^2)$. If we set $f(n, \varepsilon) = \min\{n^{1-\alpha'/2}/(\sqrt{\varepsilon})^{\alpha'}, n/2\}$ where α' is as defined in Theorem 7, a careful calculation (see the full version [2]) shows that the amortized update time becomes $\widetilde{O}(n^{\alpha'}/\varepsilon^{1-\alpha'})$.

5.2 An output-sensitive quadrant set cover algorithm

We propose an $O(1)$-approximate output-sensitive algorithm for quadrant set cover, which is needed for applying Theorem 7. Let $(S, \mathcal{Q})$ be a quadrant set cover instance of size n, and opt be its optimum. Our goal is to compute an $O(1)$-approximate optimal set cover for $(S, \mathcal{Q})$ in $\widetilde{O}(\mathsf{opt})$ time, using some basic data structure built on $(S, \mathcal{Q})$.

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

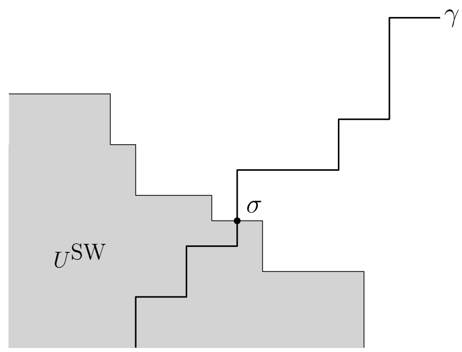

Figure 3 Illustrating the curve γ and the point σ.

With this observation, it now suffices to show how to compute an $O(1)$-approximate optimal set cover for $(S \cap U^{\mathrm{SE}}, \mathcal{Q})$ in $\widetilde{O}(\mathsf{opt}^{\mathrm{SE}})$ time, where $\mathsf{opt}^{\mathrm{SE}}$ is the optimum of $(S \cap U^{\mathrm{SE}}, \mathcal{Q})$. 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 $\gamma \cap U^{\mathrm{SW}} \neq \emptyset$, then $\gamma \cap U^{\mathrm{SW}}$ is a connected portion of γ that contains the bottom-left end of γ. Define σ as the "endpoint" of $\gamma \cap U^{\mathrm{SW}}$, i.e., the point on $\gamma \cap U^{\mathrm{SW}}$ that is closest the top-right end of γ. See Figure 3 for an illustration. If $\gamma \cap U^{\mathrm{SW}} = \emptyset$, we define σ as the bottom-left end of γ (which 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 $\varPhi_{\rightarrow}(a, \mathcal{P})$ and $\varPhi_{\uparrow}(a, \mathcal{P})$ 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 $a \leftarrow \phi(\tilde{y})$, $Q \leftarrow \varPhi_{\uparrow}(\sigma, \mathcal{Q}^{\mathrm{SE}})$, $\mathcal{Q}_{\mathrm{ans}} \leftarrow \mathcal{Q}_{\mathrm{ans}} \cup \{Q\}$, $\tilde{y} \leftarrow y(Q)$ 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.

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

The following lemma proves the correctness of our algorithm.

▶ **Lemma 11.** *$\mathcal{Q}_{\text{ans}}$ covers all points in $S \cap U^{\text{SE}}$, and $|\mathcal{Q}_{\text{ans}}| = O(\mathsf{opt}^{\text{SE}})$.*

The remaining task is to show how to perform our algorithm in $\widetilde{O}(\mathsf{opt}^{\text{SE}})$ time using basic data structures. It is clear that our algorithm terminates in $O(\mathsf{opt}^{\text{SE}})$ steps, since we include at least one quadrant to $\mathcal{Q}_{\text{ans}}$ in each iteration of the loop Step 3–5 and eventually $|\mathcal{Q}_{\text{ans}}| = O(\mathsf{opt}^{\text{SE}})$ by Lemma 11. 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 $a \in U^{\text{NE}}$ and $a \in U^{\text{NW}}$ for a given point a, computing the quadrants $\Phi_{\rightarrow}(a, \mathcal{Q}^{\text{SW}}), \Phi_{\rightarrow}(a, \mathcal{Q}^{\text{NW}}), \Phi_{\uparrow}(a, \mathcal{Q}^{\text{SE}}), \Phi_{\uparrow}(a, \mathcal{Q}^{\text{NE}})$ 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 $\mathcal{Q}^{\text{SE}}$. The details are presented in the full version [2].

Using the above algorithm, we can compute $O(1)$-approximate optimal set covers for $(S \cap U^{\text{SE}}, \mathcal{Q})$, $(S \cap U^{\text{SW}}, \mathcal{Q})$, $(S \cap U^{\text{NE}}, \mathcal{Q})$, and $(S \cap U^{\text{NW}}, \mathcal{Q})$. 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 12.** *Quadrant set cover admits an $O(1)$-approximate output-sensitive algorithm.*

5.3 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 12 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 7 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 an index $i \geq 0$ 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 13.** *For any constant $\alpha > 0$, there exists an $O(1)$-approximate dynamic quadrant set cover data structure with $O(n^{1/2+\alpha})$ amortized update time and $\widetilde{O}(n_0)$ construction time.*

By the reduction of Lemma 6, we have the following corollary.

▶ **Corollary 14.** *For any constant $\alpha > 0$, there exist $O(1)$-approximate dynamic unit-square set cover, dynamic unit-square hitting set, and dynamic quadrant hitting set data structures with $O(n^{1/2+\alpha})$ amortized update time and $\widetilde{O}(n_0)$ construction 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 arXiv:20XX.XXXXX*, 2020.
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. Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $o(\log n)$ update time. *SIAM Journal on Computing*, 44(1):88–113, 2015.
5. Piotr Berman and Bhaskar DasGupta. Complexities of efficient solutions of rectilinear polygon cover problems. *Algorithmica*, 17(4):331–356, 1997.
6. Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the twenty-seventh annual ACM-SIAM Symposium on Discrete Algorithms*, pages 692–711. Society for Industrial and Applied Mathematics, 2016.
7. Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in $o(1)$ amortized update time. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 86–98. Springer, 2017.
8. 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.
9. Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM Journal on Computing*, 47(3):859–887, 2018.
10. Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the forty-eighth annual ACM Symposium on Theory of Computing*, pages 398–411. ACM, 2016.
11. Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.
12. 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.
13. 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.
14. Shashidhara K Ganjugunte. *Geometric hitting sets and their variants*. PhD thesis, Duke University, 2011.
15. 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.
16. Manoj Gupta and Richard Peng. Fully dynamic (1+e)-approximate matchings. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 548–557. IEEE, 2013.
17. Juris Hartmanis. Computers and intractability: a guide to the theory of np-completeness. *Siam Review*, 24(1):90, 1982.

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

19 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.

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

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

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

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

24 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Transactions on Algorithms (TALG)*, 12(1):7, 2016.

25 Shay Solomon. Fully dynamic maximal matching in constant update time. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 325–334. IEEE, 2016.