Online and Dynamic Algorithms for Geometric Set Cover and Hitting Set

Arindam Khan
Indian Institute of Science, Bengaluru, India

Aditya Lonkar
Indian Institute of Science, Bengaluru, India

Saladi Rahul
Indian Institute of Science, Bengaluru, India

Aditya Subramanian
Indian Institute of Science, Bengaluru, India

Andreas Wiese
Technische Universität München, Germany

Abstract

Set cover and hitting set are fundamental problems in combinatorial optimization which are well-studied in the offline, online, and dynamic settings. We study the geometric versions of these problems and present new online and dynamic algorithms for them. In the online version of set cover (resp. hitting set), m sets (resp. n points) are given and n points (resp. m sets) arrive online, one-by-one. In the dynamic versions, points (resp. sets) can arrive as well as depart. Our goal is to maintain a set cover (resp. hitting set), minimizing the size of the computed solution.

For online set cover for (axis-parallel) squares of arbitrary sizes, we present a tight $O(\log n)$-competitive algorithm. In the same setting for hitting set, we provide a tight $O(\log N)$-competitive algorithm, assuming that all points have integral coordinates in $[0, N)^2$. No online algorithm had been known for either of these settings, not even for unit squares (apart from the known online algorithms for arbitrary set systems).

For both dynamic set cover and hitting set with d-dimensional hyperrectangles, we obtain $(\log m)^{O(d)}$-approximation algorithms with $(\log m)^{O(d)}$ worst-case update time. This partially answers an open question posed by Chan et al. [SODA'22]. Previously, no dynamic algorithms with polylogarithmic update time were known even in the setting of squares (for either of these problems). Our main technical contributions are an *extended quad-tree* approach and a *frequency reduction* technique that reduces geometric set cover instances to instances of general set cover with bounded frequency.

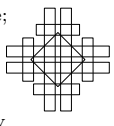
2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Online algorithms

Keywords and phrases Geometric Set Cover, Hitting Set, Rectangles, Squares, Hyperrectangles, Online Algorithms, Dynamic Data Structures

Digital Object Identifier 10.4230/LIPIcs.SoCG.2023.46

Related Version *Full Version*: https://arxiv.org/abs/2303.09524

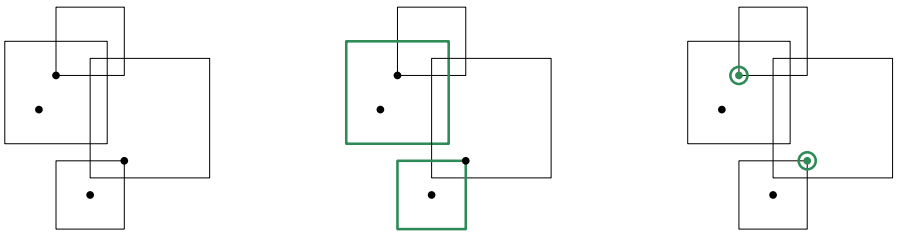
Funding *Arindam Khan*: gratefully acknowledges the generous support due to IUSSTF virtual center on "Polynomials as an Algorithmic Paradigm", Pratiksha Trust Young Investigator Award, Google India Research Award, Google ExploreCS Award, and SERB Core Research Grant (CRG/2022/001176) on "Optimization under Intractability and Uncertainty".
Aditya Subramanian: supported in part by Kotak IISc AI-ML Centre (KIAC) PhD Fellowship.



1 Introduction

Geometric set cover is a fundamental and well-studied problem in computational geometry [19, 16, 31, 26, 29]. Here, we are given a universe P of n points in $\mathbb{R}^d$, and a family $\mathcal{S}$ of m sets, where each set $S \in \mathcal{S}$ is a geometric object (we assume S to be a *closed* set in $\mathbb{R}^d$ and S *covers* all points in $P \cap S$), e.g., a hyperrectangle. Our goal is to select a collection $\mathcal{S}' \subseteq \mathcal{S}$ of these sets that contain (i.e., cover) all elements in P, minimizing the cardinality of $\mathcal{S}'$ (see Figure 1 for an illustration). The *frequency* f of the set system $(P, \mathcal{S})$ is defined as the maximum number of sets that contain an element in P.

In the offline setting of some cases of geometric set cover, better approximation ratios are known than those for the general set cover, e.g., there is a polynomial-time approximation scheme (PTAS) for (axis-parallel) squares [30]. However, much less is understood in the online and in the dynamic variants of geometric set cover. In the online setting, the sets are given offline and the points arrive one-by-one, and for an uncovered point, we have to select a (covering) set in an immediate and irrevocable manner. To the best of our knowledge, even for 2-D unit squares, there is no known online algorithm with an asymptotically improved competitive ratio compared to the $O(\log n \log m)$-competitive algorithm for general online set cover [3, 14]. In the dynamic case, the sets are again given offline and at each time step a point is inserted or deleted. Here, we are interested in algorithms that update the current solution quickly when the input changes. In particular, it is desirable to have algorithms whose update times are polylogarithmic. Unfortunately, hardly any such algorithm is known for geometric set cover. Agarwal et al. [2] initiated the study of dynamic geometric set cover for intervals and 2-D unit squares and presented $(1+\varepsilon)$- and $O(1)$-approximation algorithms with polylogarithmic update times, respectively. To the best of our knowledge, for more general objects, e.g., rectangles, three-dimensional cubes, or hyperrectangles in higher dimensions, no such dynamic algorithms are known. Note that in dynamic geometric set cover, the inserted points are represented by their coordinates, which is more compact than for general (dynamic) set cover (where for each new point p we are given a list of the sets that contain p, and hence, already to read this input we might need $\Omega(f)$ time).



(a) (b) (c)

Figure 1 (a) A set of squares $\mathcal{S}$ and a set of points P, (b) A set cover (in green) $\mathcal{S}' \subseteq \mathcal{S}$ covering P, (c) A hitting set (green points) $P' \subseteq P$ for $\mathcal{S}$.

Related to set cover is the hitting set problem (see Figure 1 for an illustration) where, given a set of points P and a collection of sets $\mathcal{S}$, we seek to select the minimum number of points $P' \subseteq P$ that hit each set $S \in \mathcal{S}$, i.e., such that $P' \cap S \neq \emptyset$. Again, in the offline geometric case, there are better approximation ratios known than for the general case, e.g., a PTAS for squares [30], and an $O(\log \log \mathsf{OPT})$-approximation for rectangles [4]. However, in the online and the dynamic cases, only few results are known that improve on the results for the general case. In the online setting, there is an $O(\log n)$-competitive algorithm for $d = 1$,

i.e., intervals, and an $O(\log n)$-competitive algorithm for unit disks [21]. In the dynamic case, the only known algorithms are for intervals and unit squares (and thus, also for quadrants), yielding approximation ratios of $(1+\varepsilon)$ and $O(1)$, respectively [2].

1.1 Our results

In Section 2 we study online set cover for axis-parallel squares of arbitrary sizes and provide an online $O(\log n)$-competitive algorithm. We also match (asymptotically) the lower bound of $\Omega(\log n)$, and hence, our competitive ratio is tight. In our online model (as in [3]), we assume that the sets (squares) are given initially and the elements (points) arrive online.

Our online algorithm is based on a new offline algorithm that is *monotone*, i.e., it has the property that if we add a new point p to P, the algorithm outputs a superset of the squares that it outputs given only P without p. The algorithm is based on a quad-tree decomposition. It traverses the tree from the root to the leaves, and for each cell C in which points are still uncovered, it considers each edge e of C and selects the "most useful" squares containing e, i.e., the squares with the largest intersection with C. We assume (throughout this paper) that all points and all corners of the squares have integral coordinates in $[0, N)^2$ for a given N, and we obtain a competitive ratio of $O(\log N)$. If we know that all the inserted points come from an initially given set of n candidate points P_0 (as in, e.g., Alon et al. [3]), we improve our competitive ratio to $O(\log n)$. For this case, we use the BBD-tree data structure due to Arya et al. [6] which uses a more intricate decomposition into cells than a standard quad-tree, and adapt our algorithm to it in a non-trivial manner. Due to the monotonicity of our offline algorithm, we immediately obtain an $O(\log n)$-competitive online algorithm.

In Section 3 we present an $O(\log N)$-competitive algorithm for online hitting set for squares of arbitrary sizes, where the points are given initially and the squares arrive online. This matches the best-known $O(\log N)$-competitive algorithm for the much simpler case of intervals [21]. Also, there is a matching lower bound of $\Omega(\log N)$, even for intervals.

In a nutshell, if a new square S is inserted by the adversary, we identify $O(\log N)$ quad-tree cells for which S contains one of its edges. Then, we pick the most useful points in these cells to hit such squares: those are the points closest to the four edges of the cell. We say that this *activates* the cell. In our analysis, we turn this around: we show that for each point $p \in \mathsf{OPT}$ there are only $O(\log N)$ cells that can possibly get activated if a square S is inserted that is hit by p. This yields a competitive ratio of $O(\log N)$.

Then, in Section 4 and 5 we present our dynamic algorithms for set cover and hitting set for hyperrectangles in d dimensions. Note that no dynamic algorithm with polylogarithmic update time and polylogarithmic approximation ratio is known even for set cover for rectangles and it was asked explicitly by Chan et al. [18] whether such an algorithm exists. Thus, we answer this question in the affirmative for the case when only points are inserted and deleted. Note that this is the relevant case when we seek to store our solution explicitly, as discussed above. Even though our considered objects are very general, our algorithms need only polylogarithmic worst-case update time. In contrast, Abboud et al. [1] showed that under Strong Exponential Time Hypothesis any general (dynamic) set cover algorithm with an amortized update time of $O(f^{1-\varepsilon})$ must have an approximation ratio of $\Omega(n^{\alpha})$ for some constant $\alpha > 0$, and f can be as large as $\Theta(m)$.

We first discuss our algorithm for set cover. We start with reducing the case of hyperrectangles in d dimensions to $2d$-dimensional hypercubes with integral corners in $[0, 4m]^{2d}$. Then, a natural approach would be to adapt our algorithm for squares from above to these hypercubes. A canonical generalization would be to build a quad-tree, traverse it from the root to the leaves, and to select for each cell C and for each facet F of C the most

useful hypercube S containing F, i.e., the hypercube S with maximal intersection with C. Unfortunately, this is no longer sufficient, not even in 3-D: it might be that there is a cell C for which it is necessary that we select cubes that contain only an edge of C but not a facet of C (see Figure 2). Here, we introduce a crucial new idea: for each cell C of the (standard) quad-tree and for each dimension $i \in [2d]$, consider the hypercubes which are "edge-covering" C along dimension i. Based on these hypercubes a $(2d-1)$-dimensional recursive secondary structure is built on all the dimensions except the i-th dimension (see Figure 11).

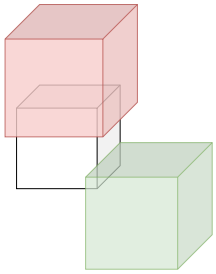


Figure 2 The red cube is the only cube that covers a facet of the (uncolored) cell. The green cube (from OPT) only covers an edge of the cell. Note that there is no corner of a cube from OPT in the cell. Picking the red cube does not cover the the intersection of the green cube with the cell.

We call the resulting tree the *extended quad-tree*. Even though it is much larger than the standard quad-tree, we show that each point is contained in only $(\log m)^{O(d)}$ cells. Furthermore, we use it for our second crucial idea to *reduce the frequency* of the set cover instance: we build an auxiliary instance of general set cover with bounded frequency. It has the same points as the given instance of geometric set cover, but different sets: for each node corresponding to a one-dimensional cell C of the extended quadtree, we consider each of its endpoints p and introduce a set that corresponds to the "most useful" hypercube covering p, i.e., the hypercube covering p with maximal intersection with C. Since each point is contained in only $(\log m)^{O(d)}$ cells, the resulting frequency is bounded by $(\log m)^{O(d)}$. Also, we show that our auxiliary set cover instance admits a solution with at most $\mathsf{OPT} \cdot (\log m)^{O(d)}$ sets. Then we use a dynamic algorithm from [12] for *general* set cover to maintain an approximate solution for our auxiliary instance, which yields a dynamic $(\log m)^{O(d)}$-approximation algorithm.

We further adapt our dynamic set cover algorithm mentioned above to hitting set for d-dimensional hyperrectangles with an approximation ratio of $(\log n)^{O(d)}$. Finally, we extend our algorithms for set cover and hitting set for d-dimensional hyperrectangles even to the weighted case, at the expense of only an extra factor of $(\log W)^{O(1)}$ in the update time and approximation ratio, assuming that all sets/points in the input have weights in $[1, W]$.

Due to space limitations, many proofs are omitted and we refer the readers to the full version [28] for the details. See the following tables for a summary of our results.

Table 1 Online algorithms for geometric set cover and hitting set.

Problem	Objects	Competitive ratio	Lower bound
Set cover	intervals	2	2
	2-D squares	$O(\log n)$	$\Omega(\log n)$
Hitting set	intervals	$O(\log N)$ [21]	$\Omega(\log N)$ [21]
	2-D squares	$O(\log N)$	$\Omega(\log N)$[21]

Table 2 Dynamic algorithms for geometric set cover and hitting set. Update times in [2] are amortized and for the unweighted case. Our results are for worst-case update times.

Problem	Objects	Approximation ratio	Update time
Set cover	2-D unit squares	$O(1)$ [2]	$(\log n)^{O(1)}$
	d-D hyperrectangles	$O(\log^{4d-1} m) \log W$	$O(\log^{2d} m) \log^3(Wn)$
Hitting set	unit squares	$O(1)$ [2]	$(\log n)^{O(1)}$
	d-D hyperrectangles	$O(\log^{4d-1} n) \log W$	$O(\log^{2d-1} n) \log^3(Wm)$

1.2 Other related work

The general set cover is well-studied in both online and dynamic settings. Several variants and generalizations of online set cover have been considered, e.g., online submodular cover [25], online set cover under random-order arrival [22], online set cover with recourse [23], etc.

For dynamic setting, Gupta et al. [23] initiated the study and provided $O(\log n)$-approximation algorithm with $O(f \log n)$-amortized update time, even in the weighted setting. Similar to our model, in their model sets are given offline and only elements can appear or depart. After this, there has been a series of works [1, 9, 11, 10, 12, 23, 24, 7].

Bhattacharya et al. [12] have given deterministic $(1+\varepsilon)f$-approximation in $O(f \log^2(Wn)/\varepsilon^3)$-worst-case update time, and $O\left((f^2/\varepsilon^3) + (f/\varepsilon^2)\log(W)\right)$-amortized update time, where W denotes the ratio of the weights of the highest and lowest weight sets. Assadi and Solomon [7] have given a randomized f-approximation algorithm with $O(f^2)$-amortized update time.

Agarwal et al. [2] studied another dynamic setting for geometric set cover, where both points and sets can arrive or depart, and presented $(1+\varepsilon)$- and $O(1)$-approximation with sublinear update time for intervals and unit squares, respectively. Chan and He [17] extended it to set cover with arbitrary squares. Recently, Chan et al. [18] gave $(1+\varepsilon)$-approximation for the special case of intervals in $O(\log^3 n/\varepsilon^3)$-amortized update time. They also gave $O(1)$-approximation for dynamic set cover for unit squares, arbitrary squares, and weighted intervals in amortized update time of $2^{O(\sqrt{\log n})}$, $n^{1/2+\varepsilon}$, and $2^{O(\sqrt{\log n \log\log n})}$, respectively.

Dynamic algorithms are also well-studied for other geometric problems such as maximum independent set of intervals and hyperrectangles [27, 13, 15], and geometric measure [20].

2 Set cover for squares

In this section we present our online and dynamic algorithms for set cover for squares. We are given a set of m squares $\mathcal{S}$ such that each square $S \in \mathcal{S}$ has integral corners in $[0, N)^2$. W.l.o.g. assume that N is a power of 2. We first describe an offline $O(\log N)$-approximate algorithm. Then we construct an online algorithm based on it, such that it has an approximation ratio of $O(\log N)$ as well. For our offline algorithm, we assume that in addition to $\mathcal{S}$ and N, we are given a set of points P that we need to cover, such that $P \subseteq [0, N)^2$, and each point $p \in P$ has integral coordinates.

Quad-tree

We start with the definition of a quad-tree $T = (V, E)$, similarly as in, e.g., [5, 8]. In T each node $v \in V$ corresponds to a square cell $C_v \subseteq [0, N)^2$ whose corners have integral coordinates. The root $r \in V$ of T corresponds to the cell $C_r := [0, N)^2$. Recursively, consider a node $v \in V$, corresponding to a cell C_v and assume that $C_v = [x_1^{(1)}, x_2^{(1)}) \times [x_1^{(2)}, x_2^{(2)})$. If C_v is a unit

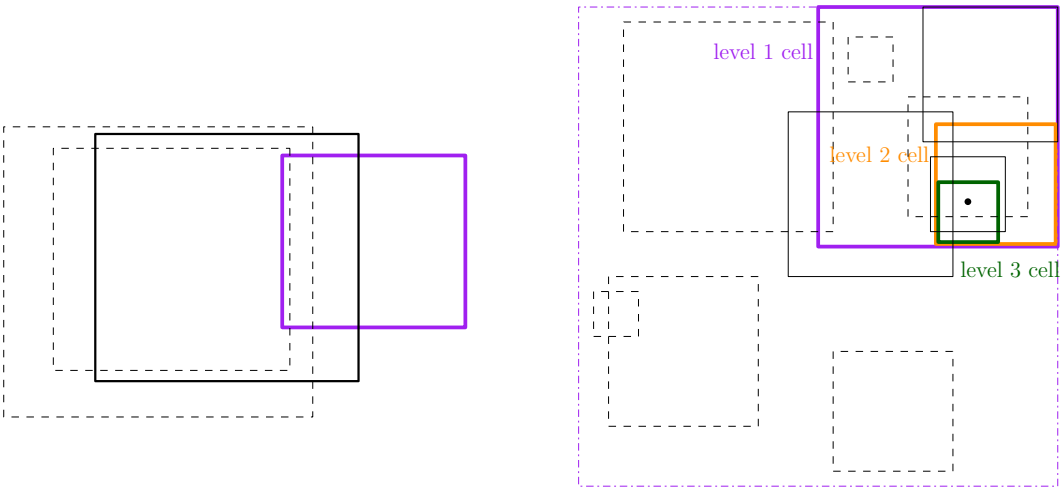


Figure 3 Left figure shows a quad-tree cell in purple. The maximum area-covering square (solid black) is picked, while the other edge-covering squares (dashed) are not. Right figure shows the quad-tree cells (level-wise color-coded) containing an uncovered point. In increasing order of depth of these cells, at most 4 maximum-area covering squares (solid black) are picked together per cell.

square, i.e., $|x_2^{(1)} - x_1^{(1)}| = |x_2^{(2)} - x_1^{(2)}| = 1$, then we define that v is a leaf. Otherwise, we define that v has four children v_1, v_2, v_3, v_4 that correspond to the four cells that we obtain if we *partition* C_v into four equal sized smaller cells, i.e., define $x_{\text{mid}}^{(1)} := (x_2^{(1)} - x_1^{(1)})/2$ and $x_{\text{mid}}^{(2)} := (x_2^{(2)} - x_1^{(2)})/2$ and $C_{v_1} = [x_1^{(1)}, x_{\text{mid}}^{(1)}) \times [x_1^{(2)}, x_{\text{mid}}^{(2)})$, $C_{v_2} = [x_1^{(1)}, x_{\text{mid}}^{(1)}) \times [x_{\text{mid}}^{(2)}, x_2^{(2)})$, $C_{v_3} = [x_{\text{mid}}^{(1)}, x_2^{(1)}) \times [x_1^{(2)}, x_{\text{mid}}^{(2)})$, and $C_{v_4} = [x_{\text{mid}}^{(1)}, x_2^{(1)}) \times [x_{\text{mid}}^{(2)}, x_2^{(2)})$. Note that the depth of this tree is $\log N$, where depth of a node in the tree is its distance from the root of T, and depth of T is the maximum depth of any node in T. By the construction, each leaf node contains at most one point and it will lie on the bottom-left corner of the corresponding cell.

Offline algorithm

In the offline algorithm $\mathcal{A}_{\text{off}}$, we traverse T in a breadth-first-order, i.e., we order the nodes in V by their distances to the root r and consider them in this order (breaking ties arbitrarily but in a fixed manner). Suppose that in one iteration we consider a node $v \in V$, corresponding to a cell C_v. We check whether the squares selected in the ancestors of v cover all points in $P \cap C_v$. If this is the case, we do not select any squares from $\mathcal{S}$ in this iteration (corresponding to v). Observe that hence we also do not select any squares in the iterations corresponding to the descendants of v in T (so we might as well skip the whole subtree rooted at v).

Suppose now that the squares selected in the ancestors of v do *not* cover all points in $P \cap C_v$. We call such a node to be *explored* by our algorithm. Let e be an edge of C_v. We say that a square containing e is *edge-covering for* e. We select a square from $\mathcal{S}$ that is edge-covering for e and that has the largest intersection with C_v among all such squares in $\mathcal{S}$ (we call such a square *maximum area-covering* for C_v for edge e). We break ties in an arbitrary but fixed way. If there is no square in $\mathcal{S}$ that is edge-covering for e then we do not select a square corresponding to e. We do this for each of the four edges of C_v. See Figure 3. If we reach a leaf node, and if there is an uncovered point (note that it must be on the bottom-left corner of the cell), then we select any arbitrary square that covers the point (the existence of such a square is guaranteed as some square in OPT covers it). See Figure 4. This guarantees the feasibility of the solution.

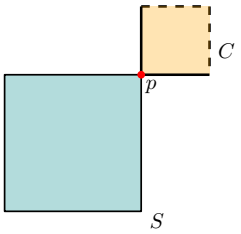


Figure 4 Point p lies in a leaf cell C (which may not even have any edge-covering squares). In this case, we pick an arbitrary square S to cover the point (since one such square always exists).

▶ **Lemma 1.** *$\mathcal{A}_{off}$ outputs a feasible set cover for the points in P.*

Approximation ratio

Let $\mathsf{ALG} \subseteq \mathcal{S}$ denote the selected set of squares and let OPT denote the optimal solution. To prove $O(\log N)$-approximation guarantee, the main idea is the following: consider a node $v \in V$ and suppose that we selected at least one square in the iteration corresponding to v. If C_v contains a corner of a square $S \in \mathsf{OPT}$, then we charge the (at most four) squares selected for v to S. Otherwise, we argue that the squares selected for v cover at least as much of C_v as the squares in OPT, and that they cover all the remaining uncovered points in $P \cap C_v$. Thus we do not select any further squares in the descendants of v. The squares selected for v are charged to the parent of v (which contains a corner of a square $S \in \mathsf{OPT}$). See Figure 5. Since each corner of each square $S \in \mathsf{OPT}$ is contained in $O(\log N)$ cells, we show that each square $S \in \mathsf{OPT}$ receives a total charge of $O(\log N)$. Thus, we obtain the following lemma.

▶ **Lemma 2.** *We have that $|\mathsf{ALG}| = O(\log N) \cdot |\mathsf{OPT}|$.*

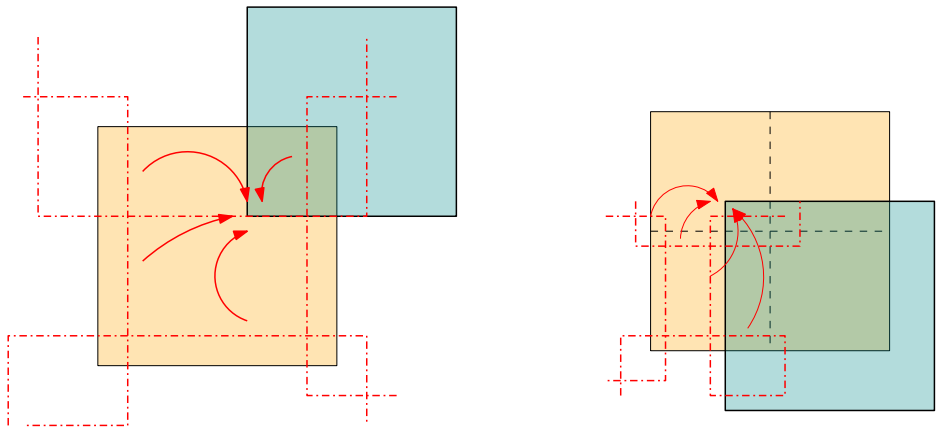


Figure 5 Charging picked (red) edge-covering squares to the corner of a (cyan) square in OPT. In the image on the left, the (yellow) cell contains a corner of the square from OPT, and in the image on the right, the parent of the cell contains such a corner.

2.1 Online set cover for squares

In the following, we first present an $O(\log N)$-competitive online algorithm for online set cover for squares. Then we improve its competitive ratio to $O(\log n)$ in the setting where we are given a set of n points at the beginning, and the adversary can introduce only points from this set.

2.1.1 $O(\log N)$-competitive online algorithm

We want to turn our offline algorithm $\mathcal{A}_{\text{off}}$ into an online algorithm $\mathcal{A}_{\text{on}}$, assuming that in each *round* a new point is introduced by the adversary. The key insight for this is that the algorithm above is *monotone*, i.e., if we add a point to P, then it outputs a superset of the squares from $\mathcal{S}$ that it had output before (when running it on P only). For a given set of points P, let $\mathsf{ALG}(P) \subseteq \mathcal{S}$ denote the set of squares that our (offline) algorithm outputs.

▶ **Lemma 3.** *Consider a set of points P and a point p. Then $\mathsf{ALG}(P) \subseteq \mathsf{ALG}(P \cup \{p\})$.*

Initially, $P = \emptyset$. If a point p is introduced by the adversary, then we compute $\mathsf{ALG}(P)$ (where P denotes the set of previous points, i.e., *without* p) and $\mathsf{ALG}(P \cup \{p\})$ and we add the squares in $\mathsf{ALG}(P \cup \{p\}) \setminus \mathsf{ALG}(P)$ to our solution. Therefore, due to Lemma 2 and Lemma 3 we obtain an $O(\log N)$-competitive online algorithm.

2.1.2 $O(\log n)$-competitive online set cover for squares

We assume now that we are given a set $\tilde{P} \subseteq \mathbb{R}^2$ with $|\tilde{P}| = n$ such that in each round a point from $\tilde{P}$ is inserted to P, i.e., $P \subseteq \tilde{P}$ after each round. We want to get a competitive ratio of $O(\log n)$ in this case. If $N = n^{O(1)}$ then this is immediate. Otherwise, we extend our algorithm such that it uses the balanced box-decomposition tree (or BBD-tree) data structure due to Arya et al. [6], instead of the quad-tree. Before the first round, $P = \emptyset$ and we initialize the BBD-tree which yields a tree $\tilde{T} = (\tilde{V}, \tilde{E})$ with the following properties:

- each node $v \in \tilde{V}$ corresponds to a cell $\tilde{C}_v \subseteq [0, N)^2$ which is described by an outer box $b_O \subseteq [0, N)^2$ and an inner box $b_I \subseteq b_O$; both of them are axis-parallel rectangles and $\tilde{C}_v = b_O \setminus b_I$ (Note that b_I could be the empty set).
- the aspect ratio of b_O, i.e., the ratio between the length of the longest edge to the length of the shortest edge of b_O, is bounded by 3.
- if $b_I \neq \emptyset$, then b_I is *sticky* which intuitively means that in each dimension, the distance of b_I to the boundary of b_O is either 0 or at least the width of b_I. Formally, assume that $b_O = [x_O^{(1)}, x_O^{(2)}] \times [y_O^{(1)}, y_O^{(2)}]$ and $b_I = [x_I^{(1)}, x_I^{(2)}] \times [y_I^{(1)}, y_I^{(2)}]$. Then $x_O^{(1)} = x_I^{(1)}$ or $x_I^{(1)} - x_O^{(1)} \geq x_I^{(2)} - x_I^{(1)}$. Also $x_O^{(2)} = x_I^{(2)}$ or $x_O^{(2)} - x_I^{(2)} \geq x_I^{(2)} - x_I^{(1)}$. Analogous conditions also hold for the y-coordinates.
- each node $v \in \tilde{V}$ is a leaf or it has two children $v_1, v_2 \in \tilde{V}$; in the latter case $\tilde{C}_v = \tilde{C}_{v_1} \dot{\cup} \tilde{C}_{v_2}$.
- the depth of $\tilde{T}$ is $O(\log n)$ and each point $q \in [0, N)^2$ is contained in $O(\log n)$ cells.
- each leaf node $v \in \tilde{V}$ contains at most one point in $\tilde{P}$.

In the construction of the BBD-tree, we make the cells at the same depth disjoint so that a point p may be contained in exactly one cell at a certain depth. Hence, for a cell $\tilde{C}_v = b_O \setminus b_I$ we assume both b_O and b_I to be *closed* set. We now describe an adjustment of our offline algorithm from Section 2, working with $\tilde{T}$ instead of T. Similarly, as before, we traverse $\tilde{T}$ in a breadth-first-order. Suppose that in one iteration we consider a node $v \in \tilde{V}$

corresponding to a cell $\tilde{C}_v$. We check whether the squares selected in the ancestors of v cover all points in $P \cap \tilde{C}_v$. If this is the case, we do not select any squares from $\mathcal{S}$ in this iteration corresponding to v.

Suppose now that the squares selected in the ancestors of v do *not* cover all points in $P \cap \tilde{C}_v$. Similar to Section 2, we want to select $O(1)$ squares for $\tilde{C}_v$ such that if $\tilde{C}_v$ contains no corner of a square $S \in \mathsf{OPT}$, then the squares we selected for $\tilde{C}_v$ should cover all points in $P \cap \tilde{C}_v$. Similarly as before, for each edge e of b_O we select a square from $\mathcal{S}$ that contains e and that has the largest intersection with b_O among all such squares in $\mathcal{S}$. We break ties in an arbitrary but fixed way. However, as $\tilde{C}_v$ may not be a square and can have holes (due to b_I), apart from the edge-covering squares, we need to consider two additional types of squares in OPT with nonempty overlap with $\tilde{C}_v$: (a) crossing $\tilde{C}_v$, i.e., squares that intersect two parallel edges of b_O; (b) has one or two corners inside b_I. See Figure 6.

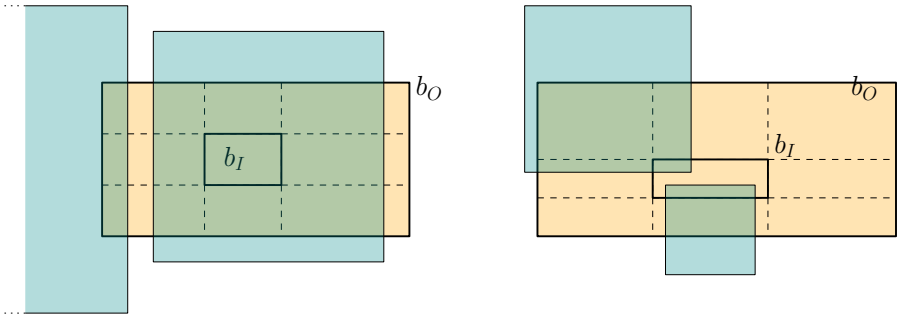


Figure 6 Possible intersections of a (cyan) square from OPT with a cell, such that no corner of the square is in the cell. The left image shows edge-covering, and crossing squares. The right image shows squares with one of two corners inside b_I.

The following *greedy subroutine* $\mathcal{G}$ will be useful in our algorithm to handle such problematic cases. Let R be a box of width w and height h such that $w/h \leq B$, for some constant $B \in \mathbb{N}$; and P_R be a set of points inside R that can be covered by a collection of vertically-crossing (i.e., they intersect both horizontal edges of R) squares $\mathcal{S}'$. Then, the set of squares picked according to $\mathcal{G}$ covers P_R in the following way:

- While there is an uncovered point $p' \in P_R$:
  - Consider the leftmost such uncovered point $p \in P_R$.
  - Select the vertically-crossing square intersecting p (by assumption, such a square exists) with the rightmost edge.

(The above subroutine is for finding vertically-crossing squares. For finding horizontally-crossing squares, we can appropriately rotate the input 90° anti-clockwise, and apply the same subroutine.) Then, we have the following claim about the aforementioned subroutine.

▷ Claim 4. Let R be a box of width w and height h such that $w/h \leq B$, for some constant $B \in \mathbb{N}$; and P_R be a set of points inside R that can be covered by a collection of vertically-crossing (i.e., they intersect both horizontal edges of R) squares $\mathcal{S}'$. Then we can find at most $B + 1$ squares from $\mathcal{S}'$ that can cover all points inside R.

We have an analogous claim for horizontally-crossing squares when $h/w \leq B$.

Now we describe our algorithm. First, we take care of the squares that can cross b_O. So, we apply the greedy subroutine $\mathcal{G}$ on b_O. As b_O has bounded aspect ratio of 3, from Claim 4, we obtain at most $(3 + 1) + (1 + 1) = 6$ squares that can cross C_v vertically or horizontally. If $b_I = \emptyset$, we do not select any more squares. Otherwise, we need to take care of the squares

that can have one or two corners inside b_I. Let $\ell_1, \ell_2, \ell_3, \ell_4$ denote the four lines that contain the four edges of b_I. Observe that $\ell_1, \ell_2, \ell_3, \ell_4$ partition b_O into up to nine rectangular regions, one being identical to b_I. See Figure 7. For each such rectangular region R, if it is sharing a horizontal edge with b_I, we again use $\mathcal{G}$ to select vertically-crossing squares. Otherwise, if R is sharing a vertical edge with b_I, we use the subroutine $\mathcal{G}$ appropriately to select horizontally-crossing squares. This takes care of squares having two corners inside b_I. Otherwise, if the rectangular region R does not share an edge with b_I, then we check if there is a square $S \in \mathcal{S}$ with a corner within b_I that completely contains R. We add S to our solution too. This finally takes care of the case when a square has a single corner inside b_I.

Finally, to complete our algorithm, before its execution, we do the following: for every leaf v for which C_v contains at most one point $p \in \tilde{P}$, we associate a fixed square which covers p. Then, if our algorithm reaches a leaf v while traversing that has an uncovered point p, we pick the associated square with this leaf that covers it. This condition in our algorithm guarantees feasibility.

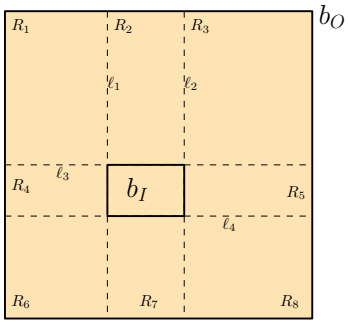


Figure 7 Outer box b_O being partitioned into at most 9 rectangles due to inner box b_I.

Then using the following lemma, we can establish a similar charging scheme as in Section 2.

▶ **Lemma 5.** *Let $\tilde{C}_v$ be a cell such that the squares selected in the ancestors of v do* not *cover all points in $P \cap \tilde{C}_v$. Then*

(a) *we select at most $O(1)$ squares for $\tilde{C}_v$ and*

(b) *if $\tilde{C}_v$ contains no corner of a square $S \in \mathsf{OPT}$, then the squares we selected for $\tilde{C}_v$ cover all points in $P \cap \tilde{C}_v$.*

To pay for our solution, we charge each corner q of a square $S \in \mathsf{OPT}$ at most $O(\log n)$ times. Hence, our approximation ratio is $O(\log n)$. Similarly as in Section 2, we can modify the above offline algorithm to an online algorithm with an approximation ratio of $O(\log n)$.

▶ **Theorem 6.** *There is a deterministic $O(\log n)$-competitive online algorithm for set cover for axis-parallel squares of arbitrary sizes.*

It is a natural question whether algorithms having a competitive factor better than $O(\log n)$ are possible for online set cover for squares. We answer this question in the negative.

▶ **Theorem 7.** *Any deterministic or randomized online algorithm for set cover for unit squares has a competitive ratio of $\Omega(\log n)$, even if all squares contain the origin and all points are contained in the same quadrant.*

3 Online hitting set for squares

Now we present our online algorithm for hitting set for squares. We are given a fixed set of points $P \subseteq [0, N)^2$ with integral coordinates. We maintain a set P' of selected points such that initially $P' := \emptyset$. In each round, we are given a square $S \subseteq [0, N)^2$ whose corners

have integral coordinates. We assume w.l.o.g. that N is a power of 2. Let Q be all (grid) points with integral coordinates in $[0, N)^2$, i.e., $P \subseteq Q$. For each point $q \in Q$ we say that $q = (q_x, q_y)$ *is of level* ℓ if both q_x and q_y are integral multiples of $N/2^\ell$, but not both are integral multiples of $N/2^{(\ell-1)}$. We build the same quad-tree as in Section 2. We say that a cell C_v *is of level* ℓ if its side length equals $N/2^\ell$.

We present our algorithm now. Suppose that in some round a new square S is given. If $S \cap P' \neq \emptyset$ then we do not add any point to P'. Suppose now that $S \cap P' = \emptyset$. Let q be a point of smallest level among all points in $Q \cap S$ (if there are many such points, then we select an arbitrary point in $Q \cap S$ of smallest level). Intuitively, we interpret q as if it were the origin and partition the plane into four *quadrants*. We define $O_{TR} := \{(p_x, p_y) \mid p_x \geq q_x \,,\, p_y \geq q_y\}$, and $S_{TR} := O_{TR} \cap S$, and define similarly O_{TL}, O_{BR}, O_{BL}, and S_{TL}, S_{BR}, S_{BL}. Consider O_{TR} and S_{TR}. For each level $\ell = 0, 1, \ldots, \log N$, we do the following. Consider each cell C of level ℓ in some fixed order such that $C \subseteq O_{TR}$ and S_{TR} is edge-covering for some edge e of C. Then, for each edge identify the point p_b (p_t, p_l, p_r, resp.) in $P \cap C$ that is closest to its bottom (top, left, and right, resp.) edge. We add these (at most 4) points to our solution if at least one of p_b, p_t, p_l, p_r is contained in S_{TR} (see Figure 8). If we add at least one such point p of the cell C to P' in this way, we say that C gets *activated*. Note that we add possibly all of the points p_b, p_t, p_l, p_r to P' even though only one may be contained in S_{TR}. This is to ensure that C gets activated at most once during a run of the online algorithm. If for the current level ℓ we activate at least one cell C of level ℓ, then we stop the loop and do not consider the other levels $\ell + 1, \ldots, \log N$. Otherwise, we continue with level $\ell + 1$. We do a symmetric operation for the pairs (O_{TL}, S_{TL}), (O_{BR}, S_{BR}), and (O_{BL}, S_{BL}).

For the analysis of the algorithm, we show that for a point $p \in \mathsf{OPT}$, the number of rounds for which the adversary can possibly introduce a square S such that $p \in S$ and $S \cap P' = \emptyset$ is $O(\log N)$. More specifically, we identify a set of cells $\mathcal{C}_p$ such that $|\mathcal{C}_p| = O(\log N)$ and in any such round where $p \in S$, one of the cells in $\mathcal{C}_p$ is activated. The competitive ratio of the algorithm follows from the fact that any cell of the quad-tree is activated at most once.

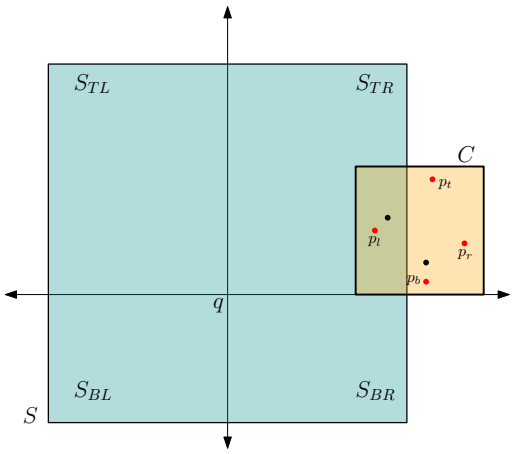


Figure 8 In the cell C (contained in O_{TR}) the red points are chosen by the algorithm.

▶ **Theorem 8.** *There is an $O(\log N)$-competitive deterministic online algorithm for hitting set for axis-parallel squares of arbitrary sizes.*

This is tight, as even for intervals, Even et al. [21] have shown an $\Omega(\log N)$ lower bound.

4 Dynamic set cover for d-dimensional hyperrectangles

In this section, we will design an algorithm to dynamically maintain an approximate set cover for d-dimensional hyperrectangles. The main result we prove in this section is the following.

▶ **Theorem 9.** *After performing a pre-processing step which takes $O(m \log^{2d} m)$ time, there is an algorithm for dynamic set cover for d-dimensional hyperrectangles with an approximation factor of $O(\log^{4d-1} m)$ and an update time of $O(\log^{2d+2} m)$.*

Our goal is to adapt the quad-tree based algorithms designed in the previous sections of the paper. As a first step towards that, we transform the problem such that the points and hyperrectangles in $\mathbb{R}^d$ get transformed to points and *hypercubes* in $\mathbb{R}^{2d}$, and the new problem is to cover the points in $\mathbb{R}^{2d}$ with these hypercubes. As discussed in the introduction, a simple $2d$-dimensional quad-tree on the hypercubes does not suffice for our purpose. We augment the quad-tree in two ways: (a) at each node, we collect the hypercubes which are edge-covering w.r.t. that node and "ignore" that dimension in which they are edge-covering, and (b) recursively construct a $(2d-1)$-dimensional quad-tree on these hypercubes based on the remaining $2d-1$ dimensions. We call this new structure an *extended quad-tree*. This yields the important property that any point in $\mathbb{R}^{2d}$ will belong to only $O(\log^{2d} m)$ cells in the extended quad-tree. Furthermore, at the 1-dimensional cells of the extended quad-tree, for each cell we will identify $O(1)$ "most useful" hypercubes. This ensures that any point belongs to only $O(\log^{2d} m)$ of these most useful hypercubes. As a result, a "bounded frequency" set system can be constructed with the most useful hypercubes. The dynamic algorithm from Bhattacharya et al. [12] (for general set cover) works efficiently on bounded frequency set systems and applying it in our setting leads to an $O(\log^{4d-1} m)$-approximation algorithm.

4.1 Transformation to hypercubes in $\mathbb{R}^{2d}$

Recall that the input is a set of points P and $\mathcal{S}$ is a collection of hyperrectangles in $\mathbb{R}^d$.

By a standard rank-space reduction, we can assume that each corner of each hyperrectangle in $\mathcal{S}$ is contained in $\{0, 1, ..., 2m\}^d$ and that the intersection of any two input hyperrectangles in $\mathcal{S}$ is either d-dimensional or empty. Also, we perturb each input point $p \in P$ slightly so that p is not contained in the face of any hyperrectangle in $\mathcal{S}$, without changing the collection of hyperrectangles that cover p.

The first step of the algorithm is to transform the hyperrectangles in $\mathcal{S}$ to hypercubes in $\mathbb{R}^{2d}$. Consider a hyperrectangle $S \in \mathcal{S}$ with $a = (a_1, \ldots, a_d)$ and $b = (b_1, \ldots, b_d)$ being the "lower-left" and the "upper-right" corners of S, respectively. Let $\Delta = \max_{j=1}^{d}(b_j - a_j)$. Then S is transformed to a hypercube S' in $\mathbb{R}^{2d}$ with side-length Δ and "top-right" corner $(-a_1, -a_2, \ldots, -a_d, b_1, b_2, \ldots, b_d)$. Let $\mathcal{S}'$ be the collection of these m transformed hypercubes. Let P' be the set of n points in $\mathbb{R}^{2d}$ obtained by transforming each point $p = (p_1, \ldots, p_d) \in P$ to $p' = (-p_1, \ldots, -p_d, p_1, \ldots, p_d)$. See Figure 9 for an example.

▶ **Observation 10.** *A point $p = (p_1, \ldots, p_d)$ lies inside S if and only if the point $p' = (-p_1, \ldots, -p_d, p_1, \ldots, p_d)$ lies inside S'.*

After applying the above transformation, we note that the coordinates of each corner of each hypercube in $\mathcal{S}'$ will be contained in $\{-4m, ..., 0\}^d \times \{-2m, ..., 2m\}^d$. We perform a suitable shifting so that all the corners of the hypercubes in $\mathcal{S}'$ will be contained in $\{0, ..., 4m\}^{2d}$. Then, our assumption on the input set of hyperrectangles $\mathcal{S}$ and the input points P implies that for any point $p' \in P'$, it does not lie on a face of any hypercube in $\mathcal{S}'$.

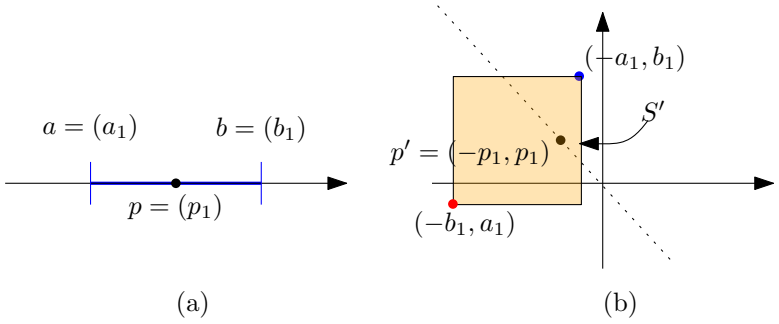


Figure 9 (a) A point p in 1-D lying inside an interval $S = [a_1, b_1]$, and (b) the transformation of p into a point $p' = (-p_1, p_1)$, and the transformation of S into a square S' in 2-D.

4.2 Constructing a bounded frequency set system

We will now present a technique to select a set $\hat{\mathcal{S}} \subseteq \mathcal{S}'$ with the following properties:

1. (*Bounded frequency*) Any point in P' lies inside $O(\log^{2d} m)$ hypercubes in $\hat{\mathcal{S}}$.
2. An α-approximation dynamic set cover algorithm for $(P', \hat{\mathcal{S}})$ implies an $O(\alpha \log^{2d-1} m)$-approximation dynamic set cover algorithm for $(P', \mathcal{S}')$.
3. The time taken to update the solution for the set system $(P', \hat{\mathcal{S}})$ is $O(\log^{2d} m \cdot \log^2 n)$.
4. The time taken to construct the set $\hat{\mathcal{S}}$ is $O(m \log^{2d} m)$.

4.2.1 Extended quad-tree for 2-dimensional squares

Given a set of squares $\mathcal{S}'$, we construct a 2-dimensional quad-tree $\mathbb{T}$ (as defined in Section 2) such that its root cell contains all the squares in $\mathcal{S}'$. Consider a node $v \in \mathbb{T}$ and a square $S \in \mathcal{S}'$. Let C and $par(C)$ be the cell corresponding to node v and the parent node of v, respectively. Let $\mathsf{proj}_i(C)$, $\mathsf{proj}_i(par(C))$ and $\mathsf{proj}_i(S)$ be the projection of C, $par(C)$ and S, respectively, on to the i-th dimension. Then S is *i-long* at v if and only if $\mathsf{proj}_i(C) \subseteq \mathsf{proj}_i(S)$ but $\mathsf{proj}_i(par(C)) \not\subseteq \mathsf{proj}_i(S)$. See Figure 10(a). For all $u \in \mathbb{T}$, let $\mathcal{S}(u, i) \subseteq \mathcal{S}'$ be the squares which are i-long at node u. Intuitively, these are squares that cover the edge of C in the i-th dimension but do not cover any edge of $par(C)$ in the i-th dimension. Now, at each node of $\mathbb{T}$ we will construct two *secondary structures* as follows: the first structure is a 1-dimensional quad-tree built on the projection of the squares in $\mathcal{S}(u, 1)$ on to the second dimension, and the second structure is a 1-dimensional quad-tree built on the projection of the squares in $\mathcal{S}(u, 2)$ on to the first dimension.

In each secondary structure, an interval I (corresponding to a square $S \in \mathcal{S}'$) is *assigned* to a node u if and only if u is the node with the smallest depth (the root is at depth zero) where I intersects either the left endpoint or the right endpoint of the cell C_u. See Figure 10(b). By this definition, any interval will be assigned to at most two nodes in the secondary structure.

Now we will use $\mathbb{T}$ to construct the geometric collection $\hat{\mathcal{S}}$. Let V_{sec} be the set of nodes in all the secondary structures of $\mathbb{T}$. For any node $u \in V_{\text{sec}}$, among its assigned intervals which intersect the left (resp., right) endpoint of the cell C_u, identify the *maximal* interval I_ℓ (resp., I_r), i.e., the interval which has maximum overlap with C_u. See Figure 10(c). We then do the following set of operations over all the nodes in V_{sec}: For a node $u \in V_{\text{sec}}$, denote by S' and S'' the corresponding squares for the assigned intervals I_ℓ and I_r, respectively. Further, let w be the node in $\mathbb{T}$, on which the secondary structure of u was constructed. Then, we include in $\hat{\mathcal{S}}$ the rectangles $S_1 \cap C_w$ and $S_2 \cap C_w$.

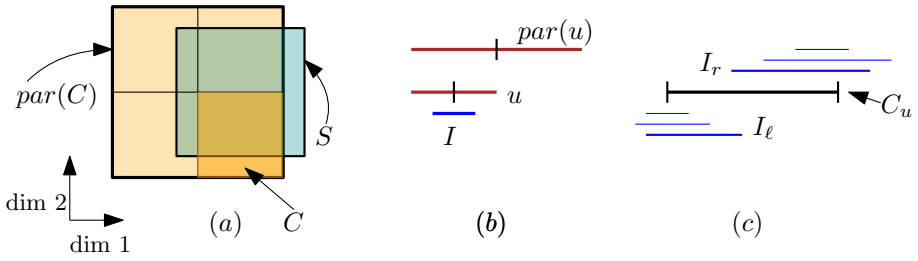


Figure 10 (a) A square S which is 1-long at node v (corrs. cell C is highlighted in darker orange), (b) I is assigned to the two children of v, and (c) the maximal intervals I_ℓ and I_r at C_v.

4.2.2 Extended quad-tree for $2d$-dimensional hypercubes

For $2d$-dimensions, we need a generalization of the quad-tree defined in Section 2. For $d' > 2$, a d'-dimensional quad-tree is defined analogously to the the quad-tree defined in Section 2, where instead of four, each internal node will now have $2^{d'}$ children. Assume by induction that we have defined how to construct the extended quad-tree for all dimensions less than or equal to $2d-1$. (The base case is the extended quad-tree built for 2-dimensional squares.) We define now how to construct the structure for $2d$-dimensional hypercubes. First construct the regular $2d$-dimensional quad-tree $\mathbb{T}$ for the set of hypercubes $\mathcal{S}'$. Consider any node $v \in \mathbb{T}$. Generalizing the previous definition, for any $1 \leq i \leq 2d$, a hypercube $S \in \mathcal{S}'$ is defined to be *i-long* at node v if and only if $\mathsf{proj}_i(C) \subseteq \mathsf{proj}_i(S)$, but $\mathsf{proj}_i(par(C)) \not\subseteq \mathsf{proj}_i(S)$. For all $v \in \mathbb{T}$, let $\mathcal{S}(v,i) \subseteq \mathcal{S}'$ be the hypercubes which are i-long at node v. Now, at each node of $\mathbb{T}$ we will construct $2d$ secondary structures as follows: for all $1 \leq i \leq 2d$, the i-th secondary structure is a $(2d-1)$-dimensional extended quad-tree built on $\mathcal{S}(v,i)$ and all its $2d$ dimensions except the i-th dimension. Specifically, any hypercube $S \in \mathcal{S}(v,i)$ of the form $\ell_1 \times \cdots \times \ell_i \times \cdots \times \ell_{2d}$ is projected to a $(2d-1)$-dimensional hypercube $\ell_1 \times \cdots \times \ell_{i-1} \times \ell_{i+1} \times \cdots \times \ell_{2d}$. Let $\hat{\mathcal{S}}_v$ be the collection of the $(2d-1)$-dimensional hyperrectangles that are inductively picked for the secondary structure constructed at $v \in \mathbb{T}$ using the routine. Define the function g which maps a $(2d-1)$-dimensional hyperrectangle picked as part of the collection $\hat{\mathcal{S}}_v$ (for a $v \in \mathbb{T}$) to its corresponding $2d$-dimensional hypercube $S \in \mathcal{S}'$. We now define the collection of sets $\hat{\mathcal{S}}$ consisting of $2d$-dimensional hyperrectangles: $\hat{\mathcal{S}} \leftarrow \bigcup_{v \in \mathbb{T}} \left(\bigcup_{S' \in \hat{\mathcal{S}}_v} (g(S') \cap C_v) \right)$. Then we prove the following three key properties of $\hat{\mathcal{S}}$.

▶ **Lemma 11.** *(Feasibility) Any point $p \in P'$ is covered by at least one set in $\hat{\mathcal{S}}$.*

▶ **Lemma 12.** *(Bounded frequency) Any point in P' lies inside $O(\log^{2d} m)$ sets in $\hat{\mathcal{S}}$.*

▶ **Lemma 13.** *If there is an α-approximation dynamic set cover algorithm for $(P', \hat{\mathcal{S}})$ then there is an $O(\alpha \log^{2d-1} m)$-approximation dynamic set cover algorithm for $(P', \mathcal{S}')$.*

4.3 The final algorithm

We run the $O(f)$-approximate algorithm by Bhattacharya et al. [12] for the dynamic set cover problem as a black box on the instance $(P', \hat{\mathcal{S}})$. If ALG is the reported solution, we also report ALG as our solution for the instance $(P', \mathcal{S}')$. One can prove that this yields Theorem 9.

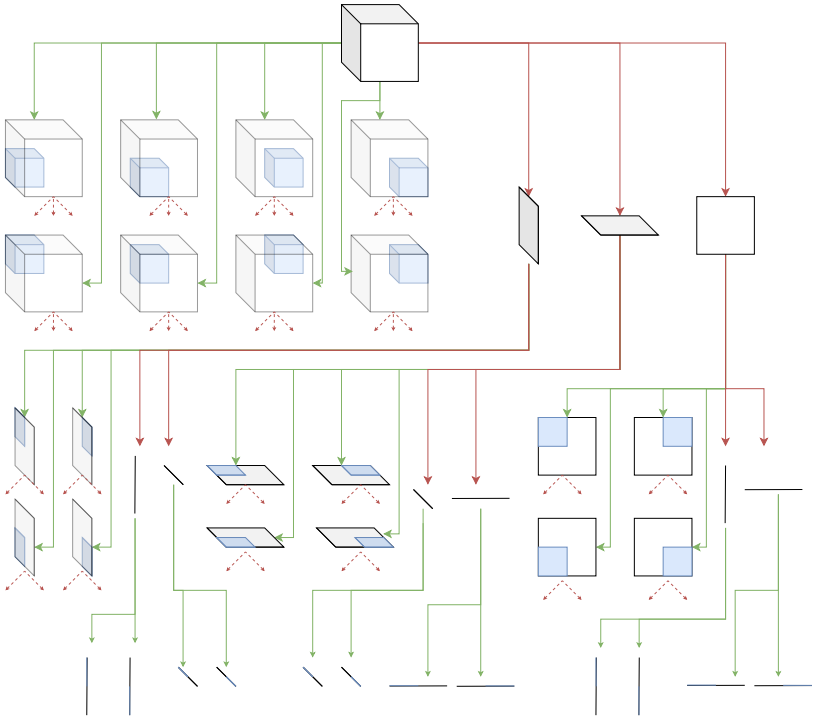


Figure 11 Extended quad-tree with a $2 \times 2 \times 2$ cube as the root.

4.4 Weighted setting

We present an extension of our algorithm to the setting where each hyperrectangle $S \in \mathcal{S}$ has a weight $w_S \in [1, W]$. First, we *round* the weight of each set S to the smallest power of two greater than or equal to w_S, leading to $O(\log W)$ weight classes. Next, for each weight class, we build an extended quad-tree as in the previous section. Finally, let $\hat{\mathcal{S}}$ be the collection of (maximal) hypercubes obtained from all the $O(\log W)$ extended quad-trees. We run the dynamic set cover algorithm of Bhattacharya et al. [12] on $(P', \hat{\mathcal{S}})$.

▶ **Theorem 14.** *There is an algorithm for weighted dynamic set cover for d-dimensional hyperrectangles with an approximation factor of $O(\log^{4d-1} m \cdot \log W)$ and an update time of $O(\log^{2d} m \cdot \log^3(Wm))$.*

5 Dynamic hitting set for d-dimensional hyperrectangles

In this section we claim a dynamic algorithm for hitting set for d-dimensional hyperrectangles. We obtain this by reducing the problem to an instance of dynamic set cover for $2d$-dimensional hypercubes and use the algorithm designed in the previous section to solve the instance.

▶ **Theorem 15.** *After performing a pre-processing step which takes $O(n \log^{2d} n)$ time, there is an algorithm for hitting set for d-dimensional hyperrectangles with an approximation factor of $O(\log^{4d-1} n)$ and an update time of $O(\log^{2d+2} n)$. In the weighted setting, the approximation factor is $O(\log^{4d-1} n \cdot \log W)$ and the update time is $O(\log^{2d} n \log^3(Wn))$.*

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 (STOC)*, pages 114–125, 2019.

2 Pankaj K. Agarwal, Hsien-Chih Chang, Subhash Suri, Allen Xiao, and Jie Xue. Dynamic geometric set cover and hitting set. In *36th International Symposium on Computational Geometry, SoCG 2020*, volume 164 of *LIPIcs*, pages 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

3 Noga Alon, Baruch Awerbuch, and Yossi Azar. The online set cover problem. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing (STOC)*, pages 100–105, 2003.

4 Boris Aronov, Esther Ezra, and Micha Sharir. Small-size \eps-nets for axis-parallel rectangles and boxes. *SIAM Journal on Computing*, 39(7):3248–3282, 2010.

5 Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.

6 Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

7 Sepehr Assadi and Shay Solomon. Fully dynamic set cover via hypergraph maximal matching: An optimal approximation through a local approach. In *29th Annual European Symposium on Algorithms, ESA 2021*, page 8. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2021.

8 Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 1997.

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 Giuseppe F. Italiano. Dynamic algorithms via the primal-dual method. *Information and Computation*, 261:219–239, 2018.

11 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. A new deterministic algorithm for dynamic set cover. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 406–423. IEEE, 2019.

12 Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Xiaowei Wu. Dynamic set cover: Improved amortized and worst-case update time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2537–2549. SIAM, 2021.

13 Sujoy Bhore, Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Dynamic geometric independent set. In *Japan conference on Discrete and Computational Geometry, Graphs, and Games*, 2021.

14 Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Found. Trends Theor. Comput. Sci.*, 3(2-3):93–263, 2009.

15 Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Worst-case efficient dynamic geometric independent set. In *29th Annual European Symposium on Algorithms (ESA 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

16 Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 1576–1585. SIAM, 2012.

17 Timothy M. Chan and Qizheng He. More dynamic data structures for geometric set cover with sublinear update time. In *37th International Symposium on Computational Geometry (SoCG 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

18 Timothy M. Chan, Qizheng He, Subhash Suri, and Jie Xue. Dynamic geometric set cover, revisited. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3496–3528. SIAM, 2022.

19 Kenneth L. Clarkson and Kasturi Varadarajan. Improved approximation algorithms for geometric set cover. In *Proceedings of the twenty-first annual symposium on Computational geometry (SoCG)*, pages 135–141, 2005.

20 Justin Dallant and John Iacono. Conditional lower bounds for dynamic geometric measure problems. *arXiv preprint arXiv:2112.10095*, 2021.

21 Guy Even and Shakhar Smorodinsky. Hitting sets online and vertex ranking. In *Algorithms - ESA 2011 - 19th Annual European Symposium*, volume 6942 of *Lecture Notes in Computer Science*, pages 347–357. Springer, 2011.

22 Anupam Gupta, Gregory Kehne, and Roie Levin. Random order online set cover is as easy as offline. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1253–1264. IEEE, 2022.

23 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 (STOC)*, pages 537–550, 2017.

24 Anupam Gupta and Roie Levin. Fully-dynamic submodular cover with bounded recourse. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1147–1157. IEEE, 2020.

25 Anupam Gupta and Roie Levin. The online submodular cover problem. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1525–1537. SIAM, 2020.

26 Sariel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *Journal of Computational Geometry*, 3(1):65–85, 2012.

27 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In *36th International Symposium on Computational Geometry, SoCG 2020*, volume 164 of *LIPIcs*, pages 51:1–51:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

28 Arindam Khan, Aditya Lonkar, Saladi Rahul, Aditya Subramanian, and Andreas Wiese. Online and dynamic algorithms for geometric set cover and hitting set. *CoRR*, abs/2303.09524, 2023. `arXiv:2303.09524`.

29 Nabil H. Mustafa, Rajiv Raman, and Saurabh Ray. Settling the apx-hardness status for geometric set cover. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 541–550. IEEE, 2014.

30 Nabil H. Mustafa and Saurabh Ray. PTAS for geometric hitting set problems via local search. In *Proceedings of the 25th ACM Symposium on Computational Geometry (SoCG)*, pages 17–22. ACM, 2009.

31 Kasturi Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of the forty-second ACM symposium on Theory of computing (STOC)*, pages 641–648, 2010.