Theoretical Computer Science 815 (2020) 270–288

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Computing coverage kernels under restricted settings

Jérémy Barbay [a,1], Pablo Pérez-Lantero [b,1], Javiel Rojas-Ledesma [a,*,1]

[a] Departamento de Ciencias de la Computación, Universidad de Chile, Chile
[b] Departamento de Matemática y Ciencia de la Computación, Universidad de Santiago, Chile

ARTICLE INFO

Article history:
Received 6 December 2018
Received in revised form 14 December 2019
Accepted 16 January 2020
Available online 20 January 2020

Keywords:
Coverage kernels
Box cover
Orthogonal polygon covering
Computational geometry
High dimensional data

ABSTRACT

Given a set $\mathcal{B}$ of d-dimensional boxes (i.e., axis-aligned hyperrectangles), a *minimum coverage kernel* is a subset of $\mathcal{B}$ of minimum size covering the same region as $\mathcal{B}$. Computing it is NP-hard, but as for many similar NP-hard problems (e.g., Box Cover, and Orthogonal Polygon Covering), the problem becomes solvable in polynomial time under restrictions on $\mathcal{B}$. We show that computing minimum coverage kernels remains NP-hard even when restricting the graph induced by the input to a highly constrained class of graphs. Alternatively, we present two polynomial-time approximation algorithms for this problem: one deterministic with an approximation ratio within $O(\log n)$, and one randomized with an improved approximation ratio within $O(\lg \mathrm{OPT})$ (with high probability).

1. Introduction

Consider a set $\mathcal{B}$ of d-dimensional boxes (i.e., axis-aligned closed hyper-rectangles, d-boxes for short), with $d \geq 2$ a fixed constant. A *coverage kernel* of $\mathcal{B}$ is a subset of $\mathcal{B}$ which covers the same region as $\mathcal{B}$, and a *minimum coverage kernel* is a coverage kernel of minimum size[2] (see Fig. 1 for an illustration). Computing a minimum coverage kernel, namely the Minimum Coverage Kernel problem, is related to two other well known problems in computational geometry: Box Cover, and Orthogonal Polygon Covering. In the Box Cover problem, given a set of boxes $\mathcal{B}$ and an additional set $\mathcal{S}$ of points, the goal is to compute a minimum-size subset of $\mathcal{B}$ containing all the points in $\mathcal{S}$. In the Orthogonal Polygon Covering problem, given an orthogonal polygon $\mathcal{P}$, the goal is to find a minimum-size set $\mathcal{B}$ of boxes covering exactly $\mathcal{P}$. The Orthogonal Polygon Covering problem can be reduced to the Minimum Coverage Kernel problem, which can in turn be reduced to the Box Cover problem. Both reductions run in polynomial time, and since the Orthogonal Polygon Covering problem is NP-hard [8,11], the Box Cover and Minimum Coverage Kernel problems are NP-hard as well.

The Minimum Coverage Kernel problem corresponds to finding and/or removing redundancies in the region covered by a set of boxes, which has applications in various areas [9,17,24]. Consider for instance *blacklists*, a basic mechanism to filter traffic and enforce security policies on a network. A *blacklist* is a set of rules that define which network packets should be blocked or allowed, based on different characteristics of the packets, like source IP, destination IP, and destination port (see Fig. 2 for an example of such rules). Since the filtering of packets must be executed extremely fast, keeping the set of rules

* Corresponding author.
E-mail addresses: jeremy@barbay.cl (J. Barbay), pablo.perez.l@usach.cl (P. Pérez-Lantero).

[1] This work was supported by projects CONICYT Fondecyt/Regular nos 1170366 and 1160543, DICYT 041933PL Vicerrectoría de Investigación, Desarrollo e Innovación USACH (Chile), Programa Regional STICAMSUD 19-STIC-02, CONICYT-PCHA/Doctorado Nacional/2013-63130209 (Chile), and CONICYT Fondecyt/Postdoctorado No. 3190550.

[2] Note that this definition of Kernel is exactly the same as that seen in results in parameterized complexity, even though in this context the result achieved is of non-tractability.

https://doi.org/10.1016/j.tcs.2020.01.021
0304-3975/

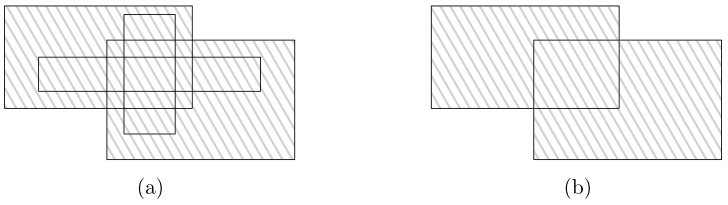

Fig. 1. An example of a minimum coverage kernel. (a) A set $\mathcal{B}$ of four boxes covering the dashed region. (b) A subset of $\mathcal{B}$ of minimum size covering the same region, namely, a minimum coverage kernel of $\mathcal{B}$.

Rule	Source IP	Destination IP	Destination Port
r_1	192.168.*.*	10.16.64.*	*
r_2	192.168.127.*	10.16.*.*	[0, 3999]

Fig. 2. Example of a blacklist. Rule r_1 blocks the packets having at the same time source IP in the range [192.168.0.0 - 192.168.255.255], destination IP in the range [10.16.64.0-10.16.64.255], and destination port in the range [0, 65535]. When a *firewall* filters a packet p trying to ingress/egress the network, p is blocked if it matches any of the two rules, and allowed otherwise.

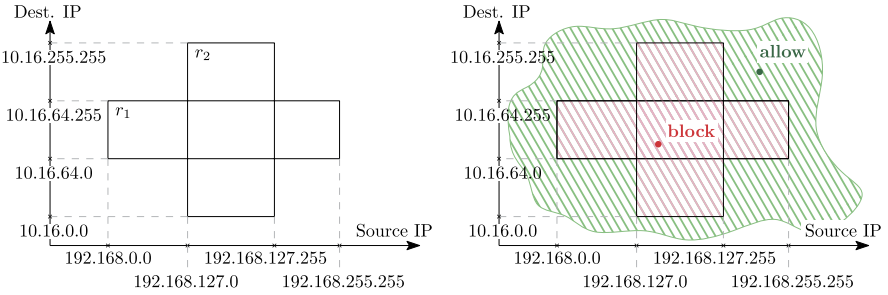

Fig. 3. Representation of a set of blacklisted rules as two-dimensional boxes. A minimum coverage kernel of the set corresponds to a minimum-size subset of the rules enforcing the same security policies.

in a blacklist as small as possible is essential to good network performance. The Minimum Coverage Kernel problem serves this purpose: The rules in a blacklist can be naturally represented as d-boxes, each one is simply a conjunction of ranges over the domains of the d different characteristics considered in the rules (see Fig. 3 for an illustration). Each packet on the network corresponds uniquely to a point in the space of the boxes. While filtering traffic, a packet should be allowed if and only if the respective point is not covered by any of the boxes representing the rules. Under this representation, a minimum coverage kernel corresponds to a minimum-size subset of the rules in the blacklist enforcing the same security policies.

Given the practical importance of the Minimum Coverage Kernel problem, we study this problem beyond its NP-hardness in the worst case: after all, it could be the case that practical instances turn out to be much easier than the worst case; and even if not so, that a kernel of reasonable size can be found in reasonable time. More formally, we attack this problem from two angles:

1. **Under which restrictions is the exact computation of Minimum Coverage Kernel still NP-hard**?
2. **How precisely can one approximate the minimum coverage kernel in polynomial time?**

When the interactions between the boxes in $\mathcal{B}$ are sufficiently simple (e.g., they are pairwise disjoint), its minimum coverage kernel can be computed efficiently. A natural way to capture the complexity of these interactions is through the intersection graph of $\mathcal{B}$: an undirected graph with a vertex for each box, and in which two vertices are adjacent if and only if the respective boxes have a non-empty intersection. When the intersection graph is a tree, for instance, one can show that each box of $\mathcal{B}$ is either completely covered by another box in $\mathcal{B}$, or present in any coverage kernel of the set.[3] Therefore, a simple procedure which tests every box for containment within the others can compute a minimum coverage kernel for this case in polynomial time. For NP-hard problems on graphs, a common approach to understanding when they become easy is to study distinct restricted classes of graphs, in the hope to define some form of "boundary classes" of inputs separating "easy" from "hard" instances [2]. Based on this, we study the hardness of the Minimum Coverage Kernel problem and of

[3] Note that when a box b is covered by the union of two or more boxes intersecting b, at least two of the covering boxes most intersect (because all boxes are closed), so that the intersection graph contains at least one cycle.

related problems (such as Box Cover and Orthogonal Polygon Covering) under different restricted classes of the intersection graph of the input box set.

Our results. We study the Minimum Coverage Kernel and Box Cover problems under three restrictions of the intersection graph, commonly considered for other problems [2]: planarity of the graph, bounded clique-number, and bounded vertex degree. We show that the Minimum Coverage Kernel problem remains NP-hard even when the intersection graph of the boxes has clique-number at most 4, and the maximum vertex degree is at most 10. This result implies, for instance, that computing a minimum-size subset of rules in a blacklist enforcing the same policies when no packet can match more than 4 of the rules at the same time, or when no rule overlaps with more than 10 other rules in the blacklist, is NP-Hard. For the Box Cover problem, we show that it remains NP-hard even under the severely restricted settings where the intersection graph of the boxes is planar, its clique-number is at most 2 (i.e., the graph is triangle-free), the maximum degree is at most 3, and every point is contained in at most two boxes.

We complement these hardness results with two approximation algorithms for the Minimum Coverage Kernel problem running in polynomial time. The first algorithm yields a $\mathcal{O}(\log n)$-approximation, and runs in deterministic time within $\mathcal{O}(\mathtt{OPT} \cdot n^{\frac{d}{2}+1} \log^2 n)$. The second one, is a randomized algorithm computing a $\mathcal{O}(\log \mathtt{OPT})$-approximation with high probability (at least $1 - \frac{1}{n^{\Omega(1)}}$), in expected time within $\mathcal{O}(\mathtt{OPT} \cdot n^{\frac{d+1}{2}} \log^2 n)$, where $\mathtt{OPT}$ is the size of an optimal solution. Our main contribution in this matter is not the existence of polynomial time approximation algorithms (which can be inferred from previous results on Box Cover [6]), but a new data structure which allows to significantly improve the running time of finding those approximations (when compared to the approximation algorithms for Box Cover). This is relevant in applications where a minimum coverage kernel needs to be computed repeatedly, such as for instance, whenever new rules are added to a blacklist.

In the next section, we review the reductions between the three problems we consider, and introduce some basic concepts. We then present the hardness results in Section 3, and describe in Section 4 the two approximation algorithms. We conclude in Section 5 with a discussion on the potential impact of the results, and some perspectives for future work.

2. Preliminaries

To better understand the relation between the Orthogonal Polygon Covering, Box Cover and Minimum Coverage Kernel problems, we briefly review the reductions between them (which are inspired in similar results [8,11,16]). We describe the reductions in two dimensions, as the generalization to higher dimensions is straightforward. Since 2-boxes ares axis-aligned rectangles, we will refer to boxes simply as rectangles in these reductions. Let us start by formally defining the problems that we consider.

Definition 1 *(Box Cover problem).* Given a set $\mathcal{S}$ of n points in $\mathbb{R}^d$ and a set $\mathcal{B}$ of m boxes in $\mathbb{R}^d$, compute the subset C of $\mathcal{B}$ of minimum size covering $\mathcal{S}$.

Definition 2 *(Orthogonal Polygon Covering problem).* Given an orthogonal polygon $\mathcal{P}$ (i.e., a polygon with all its edges either vertical or horizontal), find a set $\mathcal{B}$ of boxes whose union is exactly $\mathcal{P}$, and such that the size of $\mathcal{B}$ is as small possible.

Definition 3 *(Minimum Coverage Kernel problem).* Given a set $\mathcal{B}$ of n boxes in $\mathbb{R}^d$, compute a minimum-size subset C of $\mathcal{B}$ covering the same region as $\mathcal{B}$.

Let us show that the Orthogonal Polygon Covering problem can be reduced to the Minimum Coverage Kernel problem. Let $\mathcal{P}$ be an orthogonal polygon with n horizontal/vertical sides. Consider the grid obtained by drawing infinitely long lines through each side of $\mathcal{P}$ (see Fig. 4.a for an illustration), and let G be the set of $\mathcal{O}(n^2)$ points of this grid lying on the intersection of two lines. Let $\mathcal{B}$ be the set of (at most n^4) different rectangles that: (i) are inside $\mathcal{P}$; and (ii) the vertices in at least one of the diagonals are in G (see Fig. 4.b). Note that by construction $\mathcal{B}$ covers $\mathcal{P}$, and so will any coverage kernel of $\mathcal{B}$ by transitivity. We show that any minimum coverage kernel of $\mathcal{B}$ is an optimal covering of $\mathcal{P}$. For this purpose, let $\mathcal{C}$ be any set of rectangles covering $\mathcal{P}$, and for any rectangle $c \in \mathcal{C}$, let c' be the largest rectangle that contains c, and is contained in $\mathcal{P}$. Note that all the sides of c' must intersect with sides of $\mathcal{P}$, otherwise c' could be enlarged while remaining inside $\mathcal{P}$, contradicting the fact that c' is the largest of such rectangles. Hence, all the vertices of c' must be in G, and thus $c' \in \mathcal{B}$. Therefore, there is a subset $\mathcal{B}' \subseteq \mathcal{B}$ covering $\mathcal{P}$ such that $|\mathcal{B}'| \leq |\mathcal{C}|$. Finally, if we let C be an optimal cover of $\mathcal{P}$, then $|\mathcal{B}'| = |C|$, and thus $\mathcal{B}'$ is also an optimal cover of $\mathcal{P}$.

We now describe how to reduce the Minimum Coverage Kernel problem to the Box Cover problem in polynomial time. Let $\mathcal{B}$ be a set of n rectangles, and consider the grid obtained by drawing infinite lines through the sides of each rectangle in $\mathcal{B}$. This grid has within $\mathcal{O}(n^2)$ cells, and each of those cells is either completely inside some rectangle in $\mathcal{B}$, or outside the region covered by $\mathcal{B}$. Create a point-set $\mathcal{D}^*(\mathcal{B})$ as follows: for each cell c of the grid contained in some rectangle in $\mathcal{B}$, add to $\mathcal{D}^*(\mathcal{B})$ the center point of c (see Fig. 4.c for an illustration). By construction, if a rectangle $b \in \mathcal{B}$ covers a point p in $\mathcal{D}^*(\mathcal{B})$, then b covers completely the grid cell whose center point is p. Thus, if a subset $\mathcal{C} \subseteq \mathcal{B}$ covers all the points in $\mathcal{D}^*(\mathcal{B})$, then $\mathcal{C}$ must cover all the grid cells covered by $\mathcal{B}$, and hence the same region as $\mathcal{B}$. Therefore, solving the Box Cover problem on the input $(\mathcal{D}^*(\mathcal{B}), \mathcal{B})$ yields a minimum coverage kernel for $\mathcal{B}$.

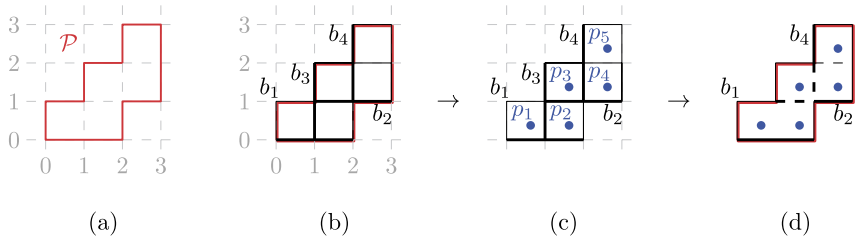

Fig. 4. Reductions between the problems studied. a) An orthogonal polygon $\mathcal{P}$. b) A set $\mathcal{B} = \{b_1 = [0,2] \times [0,1], b_2 = [1,3] \times [1,2], b_3 = [1,2] \times [0,2], b_4 = [2,3] \times [1,3]\}$ of boxes covering exactly $\mathcal{P}$, and such that in any cover of $\mathcal{P}$ with boxes, every box is either in $\mathcal{B}$, or fully covered by a box in $\mathcal{B}$. c) A $\mathcal{D}^*(\mathcal{B}) = \{p_1, p_2, p_3, p_4, p_5\}$ set of points such that any subset of $\mathcal{B}$ covering $\mathcal{D}^*(\mathcal{B})$, covers also $\mathcal{P}$. d) The subset $\{b_1, b_2, b_4\}$ is an optimal solution for the Orthogonal Polygon Covering problem on $\mathcal{P}$, the Minimum Coverage Kernel problem on $\mathcal{B}$, and the Box Cover problem on the input $(\mathcal{D}^*(\mathcal{B}), \mathcal{B})$.

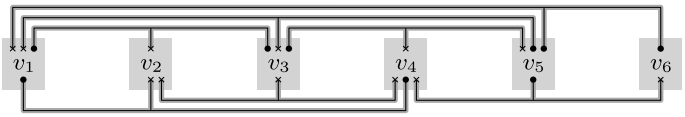

Fig. 5. Planar embedding of the formula $\varphi = (v_1 \vee \overline{v_2} \vee v_3) \wedge (v_3 \vee \overline{v_4} \vee \overline{v_5}) \wedge (\overline{v_1} \vee \overline{v_3} \vee v_5) \wedge (v_1 \vee \overline{v_2} \vee v_4) \wedge (\overline{v_2} \vee \overline{v_3} \vee \overline{v_4}) \wedge (\overline{v_4} \vee v_5 \vee \overline{v_6}) \wedge (\overline{v_1} \vee v_5 \vee v_6)$. The crosses and dots at the end of the clause legs indicate that the connected variable appears in the clause negated or not, respectively.

These reductions between the Orthogonal Polygon Covering, Box Cover, and Minimum Coverage Kernel problems have two main implications. Firstly, polynomial-time hardness results for Minimum Coverage Kernel yield similar results for the Box Cover problem. We use this in Section 3, where we show that Minimum Coverage Kernel remains NP-hard under severely restricted settings, and extend this result to Box Cover. Secondly, polynomial-time approximation algorithms for Box Cover can also be used for Minimum Coverage Kernel through this reduction. Because of the potential size of $\mathcal{D}^*(\mathcal{B})$, however, the computational cost of such solutions might be prohibitive in scenarios where the boxes in $\mathcal{B}$ represent high dimensional data [9,17,24], and coverage kernels need to be computed repeatedly [1]. We introduce in Section 4 two approximation algorithms for the Minimum Coverage Kernel problem for such scenario.

3. Hardness under restricted settings

We prove that the Minimum Coverage Kernel problem remains NP-hard for restricted classes of the intersection graph of the input set of boxes. We consider three main restrictions: when the graph is planar, when the size of its largest clique (namely the clique-number of the graph) is bounded by a constant, and when the maximum degree of the vertices (namely the vertex degree of the graph) is bounded by a constant.

3.1. Hardness of minimum coverage kernel

Given a set $\mathcal{B}$ of n boxes, the k-Coverage Kernel problem is that of deciding whether $\mathcal{B}$ contains a coverage kernel of size k, for $k \leq n$. Proving that k-Coverage Kernel is NP-complete under restricted settings yields the NP-hardness of Minimum Coverage Kernel under the same conditions: k is the size of a minimum coverage kernel of $\mathcal{B}$ if and only if $\mathcal{B}$ contains a coverage kernel of size k and does not contain a coverage kernel of size $k-1$. To prove that k-Coverage Kernel is NP-complete under restricted settings, we reduce instances of the Planar 3-SAT problem (a classical NP-complete problem [22]) to restricted instances of the k-Coverage Kernel problem. In the Planar 3-SAT problem, given a boolean formula in 3-CNF whose incidence graph[4] is planar, the goal is to find whether there is an assignment which satisfies the formula. The (planar) incidence graph of any planar 3-SAT formula φ can be represented in the plane, as illustrated in Fig. 5, where all variables lie on a horizontal line, and all clauses are represented by *non-intersecting* three-legged "combs" [15,18]. Again, we refer to such a representation of φ as the *planar embedding* of φ. Based on this planar embedding, we prove the results in Theorem 1. Although our arguments are for two dimensions, their extension to higher dimensions is simple: construct a set of boxes which in the first two dimensions are defined like the ones in the proofs below, and defined the remaining $d-2$ dimensions of each box as the interval $[0,1]$.

Theorem 1. *Let $\mathcal{B}$ be a set of n 2-boxes and let G be the intersection graph of $\mathcal{B}$. Solving the k-Coverage Kernel problem on $\mathcal{B}$ is NP-Complete even if G has clique-number at most 4, and vertex degree at most 10.*

[4] The *incidence graph* of a 3-SAT formula is a bipartite graph with a vertex for each variable and each clause, and an edge between a variable vertex and a clause vertex for each occurrence of a variable in a clause.

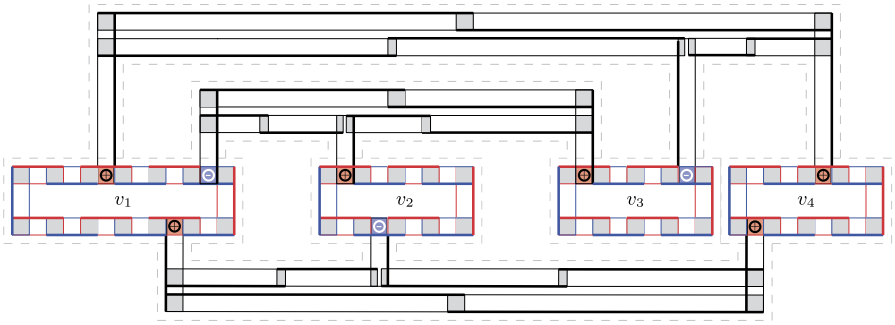

Fig. 6. Variable and clause gadgets for $\varphi = (\overline{v_1} \vee v_2 \vee v_3) \wedge (v_1 \vee \overline{v_2} \vee v_4) \wedge (v_1 \vee \overline{v_3} \vee v_4)$. The bold lines highlight one side of each box in the instance, while the dashed lines delimit the regions of the variable and clause components in the planar embedding of φ. Finding a minimum subset of boxes covering the non-white regions yields an answer for the satisfiability of φ.

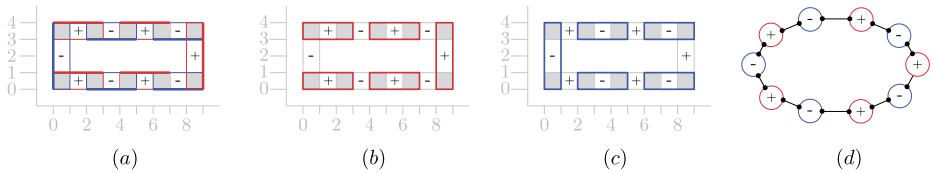

Fig. 7. A variable gadget. *a*) Gadget for a variable appearing in two clauses. The redundant regions of the gadget are shaded in gray, while the connection regions are marked with a dot or a cross. A clause gadget connects with the variable in one connection region. The optimal way to cover all the redundant regions is to choose either all the red boxes (as in *b*) or all the blue boxes (as in *c*). *d*) The intersection graph of the variable gadget. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Proof. Given any set $\mathcal{B}$ of n 2-boxes, and a subset $\mathcal{K} \subseteq \mathcal{B}$, deciding whether $\mathcal{K}$ covers the same region as $\mathcal{B}$ can be done in $\mathcal{O}(n \log n)$-time using Chan's algorithm [7] for computing the area of the region covered by $\mathcal{B}$. Thus, k-Coverage Kernel is in NP. To prove that it is NP-complete, we consider a planar 3-SAT formula φ with n variables and m clauses. We describe how to obtain a set $\mathcal{B}$ with $\mathcal{O}(m+n)$ boxes that contains a coverage kernel of size k if and only if φ is satisfiable, for $k = 32m + 3n$. We use the planar embedding of φ as a starting point, and replace the components corresponding to variables and clauses, respectively, by gadgets composed of several boxes (see Fig. 6 for an illustration). We show that the construction can be obtained in polynomial time, and thus that any polynomial-time algorithm for the k-Coverage Kernel problem yields a polynomial-time algorithm for the Planar 3-SAT problem.

Variable gadgets. Let v be a variable of a planar 3-SAT formula φ, and let c_v be the number of clauses of φ in which v appears. The gadget for v is composed of $4c_v + 2$ boxes: $4c_v$ *horizontal* boxes (of 3×1 units of size), separated into two "rows" with $2c_v$ boxes each, and two *vertical* boxes (of 1×4 units of size) connecting the rows (see Fig. 7 for an illustration). The precise location of each box will be defined in the next two paragraphs. Furthermore, to identify the boxes, we will assign a sign (positive or negative) to each box (positive boxes are colored red in Fig. 7, while negative boxes are colored blue).

Horizontal boxes. The $2c_v$ boxes in the j-th row, for $j \in \{1, 2\}$, are enumerated from left to right, so that the i-th box is defined as $[2i-2, 2i+1] \times [3j-3, 3j-2]$. The gadget occupies a rectangular region of $(4c_v + 1) \times 4$ units. Although the gadget is defined with respect to the origin of coordinates, it is later translated to the region corresponding to v in the planar embedding of φ, which we assume without loss of generality to be large enough to fit the gadget. Every horizontal box is signed *positive* if its numbering is odd, and *negative* otherwise. If b is a horizontal box of the form $[x, x+3] \times [y, y+1]$, we say that the squares $[x, x+1] \times [y, y+1]$ and $[x+2, x+3] \times [y, y+1]$ are the *redundant regions* of b, and that the square $[x+1, x+2] \times [y, y+1]$ is the *connection region* of b (see Fig. 7.*a*). Moreover, we define the sign of the connection region of b to be the same as the sign of b.

Vertical boxes. Each variable gadget has two vertical boxes: the leftmost box, defined as $[0, 1] \times [0, 4]$, signed negative, and the rightmost box defined as $[4c_v, 4c_v + 1] \times [0, 4]$, signed positive. If b is a vertical box of the form $[x, x+1] \times [0, 4]$, the squares $[x, x+1] \times [0, 1]$ and $[x, x+1] \times [3, 4]$ are the redundant regions of b.

Covering the redundant regions. Observe that the boxes of a same sign are disjoint, and that every box b intersects exactly two boxes of the opposed signed, sharing with each a squared region of dimensions 1×1 corresponding to the redundant regions of b. There are $2c_v + 1$ redundant regions in a variable gadget, each one covered by exactly two boxes, one of each possible sign (see Fig. 7.*a*). The optimal way to cover all the redundant regions in a variable gadget with a subset of its boxes is to choose, either all the $2c_v + 1$ positive boxes and no negative box, or all the $2c_v + 1$ negative boxes and no positive box (as in Fig. 7.*b*-*c*). To see why, note that there is a direct one-to-one correspondence between the edges of the intersection

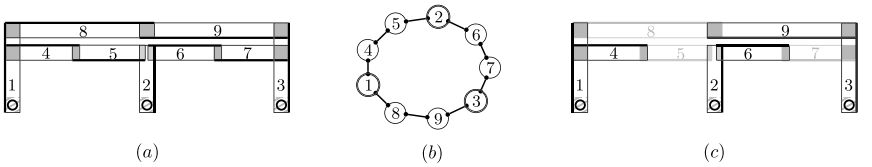

Fig. 8. Clause gadgets. *a*) A gadget with nine clause boxes: three vertical "legs" (1,2, and 3) and six horizontal boxes (4-9). We call the regions where they meet *redundant regions*. The striped regions at the bottom of each leg connect with the variables. *b*) The intersection graph of the gadget. Any minimum cover of the edges (redundant regions) requires 5 vertices (boxes). *c*) Any cover of the redundant regions which includes the three legs has 6 or more boxes.

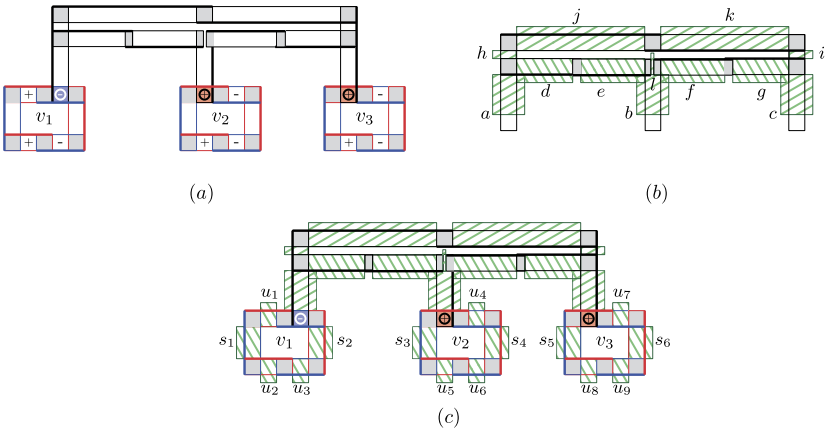

Fig. 9. An instance for $(\overline{v_1} \vee v_2 \vee v_3)$. *a*) The clause gadget connects with a variable gadget in one of its positive (colored red) or negative (colored blue) connection regions depending on the sign of the variable in the clause. *b*) Neutral boxes (dashed, and colored green) are added to cover the regions of depth 1 in the clause gadget. *c*) The regions of depth 1 of the variable gadgets (including the unused connection regions) are also covered with neutral (green) boxes.

graph of the boxes in a variable gadget graph and the redundant regions of the gadget. Covering the redundant regions in a variable gadget with boxes from the gadget, is equivalent to covering the edges of the intersection graph with a subset of its vertices. Since the intersection graph of the boxes is a cycle (ring) graph of even length (see Fig. 7.*d*), the optimal way to cover the edges is to choose one half of the vertices in an alternating way while traveling the cycle, or equivalently, to choose all the boxes of a same sign. Later, we will establish a correspondence between the sign of the boxes chosen to cover the redundant regions of a variable gadget, and the value of the variable in an assignment.

Clause gadgets. Let C be a clause with variables u, v, and w, appearing in this order from left to right in the embedding of φ. Assume, without loss of generality, that the three-legged comb ψ for C in the embedding is above the variables. We create a gadget for C composed of 9 boxes which form an structure similar in shape to ψ. These boxes are located and enumerated as in Fig. 8.*a*, and we will refer to them as *clause boxes*. The vertical boxes numbered 1, 2 and 3 (which we name *leg boxes*) correspond to the legs of ψ, and connect with the gadgets of u, v, and w, respectively. The remaining six horizontal boxes connect the three leg boxes between them. The leg boxes have one unit of width and their height is given by the height of the respective legs in ψ. Similarly, the horizontal boxes have one unit of height and their width is given by the separation between the legs of ψ (see Fig. 6 for an example of how these boxes are extended or stretched as needed). Note that: (i) every clause box intersects exactly two others (again, we call *redundant regions* the regions where they meet); (ii) any minimum cover of the redundant regions (edges in Fig. 8.*b*) has five clause boxes, one of which must be a leg; and (iii) any cover of the redundant regions which includes the three legs must have at least six boxes (e.g., see Fig. 8.*c*).

Connecting the gadgets. Let v be a variable in a formula φ, let $\mathcal{C}_v$ be the set of clauses in which v occurs, and let $c_v = |\mathcal{C}_v|$. For a clause $C \in C_v$ we denote by $\texttt{Lit}(C, v)$ the literal corresponding to v in C. The literal $\texttt{Lit}(C, v)$ is called *positive* when $\texttt{Lit}(C, v) = v$, and *negative* when $\texttt{Lit}(C, v) = \overline{v}$. We connect the leg boxes of the gadgets for the clauses in $\mathcal{C}_v$ with the gadget for v, from left to right, in the same order as the respective legs appear in the embedding of φ. This connection is done as follows. Let G_C be the gadget for a clause $C \in \mathcal{C}_v$, and assume that the three-legged comb for C in the embedding of φ connects with v from above (the case when C connects with v from below is symmetric). We connect the corresponding leg of G_C with the variable gadget for v in the connection region of one of the boxes in the upper row. We choose a positive box if $\texttt{Lit}(C, v)$ is positive, and a negative box if $\texttt{Lit}(C, v)$ negative (see Fig. 9.*a*). Note that a variable gadget has enough connection regions for all the c_v clauses in which it appears, because each row of the gadget has c_v boxes of each sign.

Completing the instance. Each box in a variable or clause gadget, as described, has a region that no other box covers (namely, of *depth* 1). Thus, the coverage kernel of the instances described up to here is trivial: all the boxes. To avoid this, we cover

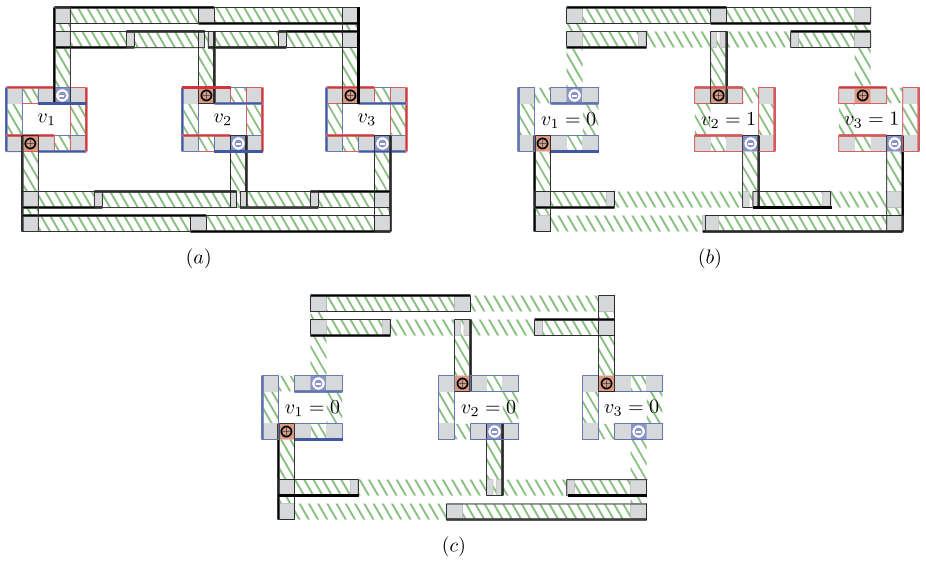

Fig. 10. Variable assignments and coverage kernels. *a*) An instance $\mathcal{B}(\varphi)$ for $\varphi = (\overline{v_1} \vee v_2 \vee v_3) \wedge (v_1 \vee \overline{v_2} \vee \overline{v_3})$ (neutral boxes were omitted, but the area they cover is highlighted with green falling lines). The gadget for first clause connects with the variable gadgets from above, while the gadget for the second clause connects with them from below. *b*) A coverage kernel $\mathcal{K}_1$ corresponding to the assignment $v_1 = 0, v_2 = 1, v_3 = 1$ that does not satisfy φ. *c*) A coverage kernel $\mathcal{K}_2$ corresponding to the assignment $v_1 = 0, v_2 = 0, v_3 = 0$ that satisfies φ, of size $|\mathcal{K}_2| < |\mathcal{K}_1|$.

all these regions of depth 1 with additional boxes (as illustrated in Fig. 9.*c* as boxes with dashed interior). To identify these boxes, we assign them a sign: *neutral*.[5] For every clause gadget we add 12 such neutral boxes: 6 covering the regions of depth 1 in the leg boxes (see boxes a, b, c, h, i, l in Fig. 9.b), and 6 covering the regions of depth 1 of the horizontal clause boxes connecting the legs (see boxes d, e, f, g, j, k in Fig. 9.b). Similarly, for each gadget for a variable v which occurs in c_v clauses, we add $3c_v + 2$ neutral boxes: one for each of the $(4c_v - c_v) = 3c_v$ unused connection regions (see boxes $u_1, \ldots, u_9$ in Fig. 9.c), and one for each of the two vertical boxes of the gadget (see boxes $s_1, \ldots, s_6$ in Fig. 9.c). The neutral boxes are forced to be in any coverage kernel by making them large enough so that each one overflows enough to cover an exclusive region of the plane that no other box covers.

Let φ be a formula with n variables and m clauses, and let $\mathcal{B}(\varphi)$ denote the instance of k-Coverage Kernel that can be generated for φ according to the procedure described above. Then, $\mathcal{B}(\varphi)$ has a total of $41m+4n$ boxes: (i) each clause gadget has 9 clause boxes representing the three-legged comb, and 12 neutral boxes, for a total of $21m$ boxes over all the clauses; (ii) a gadget for a variable v has $2c_v+1$ positive boxes, $2c_v+1$ negative boxes, and $3c_v+2$ neutral boxes, thus adding a total of $7c_v + 4$ boxes per variable gadget; and (iii) over all variables, we add a total of $\sum_{i=1}^{n} (7c_{v_i} + 4) = 7(3m) + 4n = 21m + 4n$ boxes.[6]

Intuition: from minimum kernels to boolean values. Let φ be a formula, and let $\mathcal{K}$ be a minimum coverage kernel of the instance $\mathcal{B}(\varphi)$ generated for φ. Consider a gadget G_v for a variable v that occurs in c_v clauses of φ. $\mathcal{K}$ must contain all the $3c_v + 2$ neutral boxes of G_v (because they cover an exclusive region of the plane), and at least $2c_v + 1$ of its positive/negative boxes (because this is the smallest number required to cover the redundant regions G_v). Thus, the size of $\mathcal{K} \cap G_v$ is at least $5c_v + 3$. If the equality holds (i.e., $|\mathcal{K} \cap G_v| = 5c_v + 3$), then the non-neutral boxes in $G_v \cap \mathcal{K}$ must be either all positive, or all negative, because that is the only possible way to cover the redundant regions of G_v with $2c_v + 1$ boxes. In this case, we say that $\mathcal{K}$ is consistent with an assignment for v, and we say that $v = 1$ when all the positive boxes of G_v belong to $\mathcal{K}$, and that $v = 0$ when all the negative boxes of G_v belong to $\mathcal{K}$ (see Fig. 10 for an example).

In the same way that choosing a value for v may affect the output of a clause C in which v occurs, choosing a sign of the boxes to cover the redundant regions in G_v may affect the number of boxes required to cover the gadget for C. For instance, consider that $G_v \cap \mathcal{K}$ contains only the neutral and negative boxes of G_v (i.e., $v = 0$), and that $\texttt{Lit}(C, v)$ is positive (i.e., $\texttt{Lit}(C, v) = v$, and thus evaluates to 0). Since $\texttt{Lit}(C, v)$ is positive, the gadget for C must connect with G_v in one of its positive connection regions (see e.g., the gadgets for v_1 and the second clause in Fig. 10). Moreover, since $G_v \cap \mathcal{K}$ does not contain positive boxes, the leg connecting C with G_v must cover the respective connection region of G_v, and thus must necessarily belong to $\mathcal{K}$. When $\mathcal{K}$ is consistent with an assignment for the three variables on C, and the three respective literals evaluate all to 0 (i.e., C is not satisfied by the assignment), then the three legs of the clause gadget must belong to $\mathcal{K}$, covering the respective connection regions of the variable gadgets. Thus, to cover the redundant regions of the clause

[5] For simplicity, these neutral boxes were omitted in Fig. 6 and Fig. 10.

[6] Note that $\sum_{i=1}^{n} c_{v_i} = 3m$ since exactly 3 variables occur in each clause.

gadget, at least six of the black boxes will be required (see the gadget for the second clause in Fig. 10.*b*). However, if at least one leg can be discarded (when at least one of the literals evaluates to 1), then the clause gadget can be covered with five of its clause boxes (see the clause gadgets in Fig. 10.*c*). Therefore, it suffices to show that there is an assignment satisfying φ if and only if there is a coverage kernel $\mathcal{K}$ in $\mathcal{B}(\varphi)$ that is consistent with an assignment for all the variables of φ, covers the redundant regions of every clause gadget with only five of its clause boxes, and covers every variable gadget G_v with $5c_v + 3$ boxes.

Reduction. We prove the theorem in two steps. First, we show that $\mathcal{B}(\varphi)$ has a coverage kernel of size $32m + 3n$ if and only if φ is satisfiable. Therefore, answering k-Coverage Kernel over this instance with $k = 32m + 3n$ yields an answer for the Planar 3-SAT problem on φ. Finally, we will show that the instance described matches all the restrictions in Theorem 1, under minor variations.

($\Leftarrow$). Let φ be a 3-CNF formula over a set V of n variables $V = \{v_1, \ldots, v_n\}$, and a set C of m clauses $C = \{C_1, \ldots, C_m\}$ Let $\alpha : V \to \{0, 1\}$ be an assignment which satisfies φ. We create a coverage kernel $\mathcal{K}$ of $\mathcal{B}(\varphi)$ as follows:

- For each variable gadget for $v_i \in V$ such that $\alpha(v_i) = 0$, we add to $\mathcal{K}$ all its non-positive boxes, thus covering the entire gadget minus its positive connection regions. If there is any positive connection region uncovered, it must connect with clause C in which $\texttt{Lit}(C, v)$ is positive (and thus, $\texttt{Lit}(C, v) = v$, which evaluates to 0). Symmetrically, for each variable gadget for $v_i \in V$ such that $\alpha(v_i) = 1$, we add to $\mathcal{K}$ all the non-negative boxes of the gadget, covering the entire gadget minus its negative connection regions. If there is any negative connection region uncovered, it must connect with clause C such that $\texttt{Lit}(C, v)$ is negative (and thus, $\texttt{Lit}(C, v) = \overline{v}$, which again evaluates to 0). These connection regions will be covered later with the respective legs of the corresponding clause gadgets. Over all variables, we add to $\mathcal{K}$ a total of $\sum_{i=1}^{n} \big((7-2)c_{v_i} + (4-1)\big) = 5(3m) + 3n = 15m + 3n$ boxes from the variable gadgets. By construction, $\mathcal{K}$ is up to here consistent with α.
- For each clause $C_i \in \{C_1, \ldots, C_n\}$, we add to $\mathcal{K}$ all its neutral boxes and the leg boxes that connect with connection regions of the variable gadgets left uncovered in the previous step. Since φ is satisfiable by α, at least one of the literals in C_i evaluates to 1. Thus, at least one of the legs of C_i is not added to $\mathcal{K}$ because $\mathcal{K}$ is consistent with α and the connection region corresponding to that literal is already covered by a box from the variable gadget present in $\mathcal{K}$. As at least one of the leg boxes of the clause gadget for C_i can be discarded, its redundant regions can be covered with five of its clause boxes (including the legs already added). So, we add to $\mathcal{K}$ clause boxes from the gadget for C_i as needed (up to a total of five), until all the redundant regions are covered, and the 12 neutral boxes of the gadget. Over all clauses, we add a total of $(12 + 5)m = 17m$ boxes.

By construction, $\mathcal{K}$ is consistent with α because in the second step no new boxes were added to $\mathcal{K}$ from the variable gadgets, and is a coverage kernel of $\mathcal{B}(\varphi)$ because it covers completely every variable and clause gadget. Moreover, the size of $\mathcal{K}$ is $(15m + 3n) + 17m = 32m + 3n$.

($\Rightarrow$). Let φ be 3-CNF formula over a set V of n variables $V = \{v_1, \ldots, v_n\}$ with a set C of m clauses $C = \{C_1, \ldots, C_m\}$. Let $\mathcal{B}(\varphi)$ be a set of boxes created as described above for φ, and let $\mathcal{K}$ be a coverage kernel of $\mathcal{B}$ whose size is $32m + 3n$, and that is consistent with an assignment $\alpha : V \to \{0, 1\}$ for φ.

Any coverage kernel of $\mathcal{B}$, including $\mathcal{K}$, must contain all the neutral boxes in $\mathcal{B}$ because each one covers an exclusive region. Over all the variable gadgets the number of neutral boxes is $\sum_{i=1}^{n} (3c_{v_i} + 2) = 3(3m) + 2n = 9m + 2n$, and over all the clause gadgets the number of neutral boxes is $12m$. Thus, $\mathcal{K}$ must contain all the $21m + 2n$ neutral boxes in $\mathcal{B}$. Furthermore, to cover the redundant regions of any clause gadget at least five of its clause boxes are required, and to cover the redundant regions of any variable gadget at least half of its positive/negative boxes are required. Hence, $\mathcal{K}$ must have at least $5m$ black boxes from the clause gadgets, and at least $\sum_{i=1}^{n} (2c_{v_i} + 1) = 2(3m) + n = 6m + n$ positive/negative boxes from the variable gadgets. Therefore, $|\mathcal{K}| \geq (21m + 2n) + (11m + n) = 32m + n$. Since $\mathcal{K}$ has precisely this size, it must cover the redundant regions of every clause gadget in $\mathcal{B}$ with exactly 5 of its clause boxes, and hence $\mathcal{B} \setminus \mathcal{K}$ must contain at least one leg box of every clause gadget. Therefore, for any clause $C_i \in C$, α makes the literal corresponding to the leg of C_i in $\mathcal{B} \setminus \mathcal{K}$ evaluate to 1, and thus α satisfies φ.

Meeting the restrictions. Now we prove that the instance of k-Coverage Kernel generated for the reduction meets the restrictions of the theorem. First, we show the bounded clique-number and vertex degree properties for the intersection graph of a clause gadget and its three respective variable gadgets. We do this based on Fig. 11, where we illustrate the intersection graph for a clause $\varphi = (\overline{v_1} \vee v_2 \vee v_3)$. Since the sign of a literal in a clause does not change the maximum vertex degree or the clique-number of the graph,[7] our arguments about the vertex degree and clique-number of the gadget for φ apply to any clause gadget. By definition the boxes composing the gadgets are closed, thus if two boxes contain at least one point in common (in their interior or boundary), their respective vertices in the intersection graph are adjacent. Note that in Fig. 11

[7] The sign of a literal $l(v)$ in a clause c changes only the sign of the connection region of the gadget for v that connects with a leg box from the gadget for c. No matter the sign of v in c, the leg connecting c and v will intersect only three boxes from the variable gadget: the box b containing the connection region, and the two neighbors of b sharing its redundant regions.

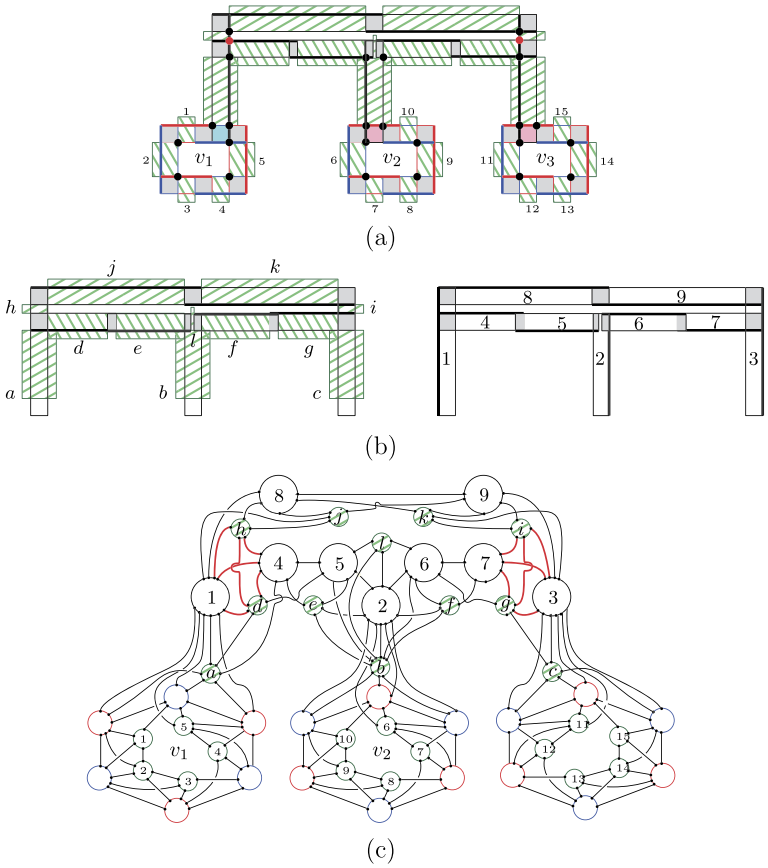

Fig. 11. The intersection graph of the instance $\mathcal{B}(\varphi)$ corresponding to the clause $\varphi = (\overline{v_1} \vee v_2 \vee v_3)$. The positive boxes (and the respective vertices) are colored red, while negative boxes (and vertices) are colored blue. Clause boxes are colored black, while neutral boxes are dashed and colored green. *a*) The boxes of $\mathcal{B}(\varphi)$ and a numbering for the neutral boxes of the variable gadgets. The dots highlight the points covered by 4 boxes at the same time, the maximum possible. *b*) Numbering of the boxes of the clause gadget. *c*) The intersection graph of the instance: the vertices corresponding to the legs have degree 10, which is the highest; the red fat edges highlight two 4-cliques. The vertices in the cliques correspond to the boxes containing the red dots in *a*, respectively.

the vertices with highest degree are the ones corresponding to legs of the clause gadget (boxes 1, 2, and 3). To bound the clique-number, note in Fig. 11.*a* that there is no point covered at the same time by five boxes: the dots in the figure highlight the points covered by the maximum number of boxes, which is 4. The intersection graph of a set of rectangles contains a 5-clique if and only if there are five boxes whose intersection is non-empty (i.e., share a common point) because a set of boxes satisfies the Helly property [13]. Thus, there are no 5-cliques in $\mathcal{B}(\varphi)$. Note that there are indeed 4-cliques in the graph corresponding to the points covered by 4 boxes. For instance, the right lower corner of the green box denoted *h* is covered also by boxes $1, 4$ and *d*, and their respective vertices form a clique. To extend these arguments to instances with more than one clause note that, since the clause gadgets are located according to the planar embedding of the formula, any two different clause gadgets are pairwise independent.[8] Thus, our bounds on the clique-number and vertex degree of the intersection graph of any particular clause gadget extend also to the intersection graph of general instance with multiple clauses. □

3.2. Extension to box cover

Since the Minimum Coverage Kernel problem is a special case of the Box Cover problem, the result of Theorem 1 also applies to the Box Cover problem. However, in Theorem 2 we show that this problem remains hard under even more restricted settings.

[8] Two clause gadgets are pairwise independent if the boxes composing them are pairwise independent, as well as the boxes where they connect with their respective variables gadgets.

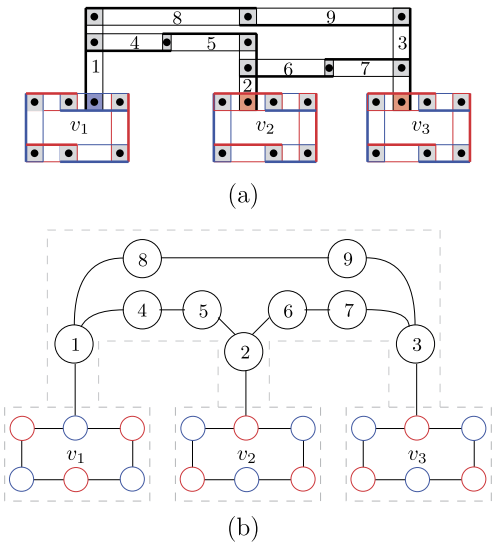

Fig. 12. An instance of Box Cover and its intersection graph. *a*) The instance corresponding to $\varphi = (\overline{v_1} \vee v_2 \vee v_3)$. *b*) The intersection graph of the instance: the vertices corresponding to the legs have degree 3, which is the highest; there are no 3-cliques in the graph; and the graph is planar and can be drawn within the planar embedding of φ (highlighted with dashed lines).

Theorem 2. *Let P and $\mathcal{B}$ be a set of m points and a set of n boxes in the plane, respectively, and let G be the intersection graph of $\mathcal{B}$. Solving the Box Cover problem on P and $\mathcal{B}$ is* NP*-hard even if every point in P is covered by at most two boxes of $\mathcal{B}$, and G is planar, has clique-number at most 2, and vertex degree at most 3.*

Proof. We use the same reduction from Planar 3-SAT, but with three main variations in the gadgets. Firstly, we do not need to use neutral boxes in the variable or clause gadgets, so we drop them out. Secondly, we create a set of points P as follows: for each redundant and connection region of a variable or clause gadgets we add to P the point at the center of the region. And finally, we increase the width of the connection regions of the variable gadgets so that any leg box connecting a variable intersects only one box from the gadget; and separate vertically the boxes 5 and 6 of the clause gadgets so that they do not intersect. See Fig. 12 for an illustration of these changes for the clause $\varphi = (\overline{v_1} \vee v_2 \vee v_3)$. *b*). Note in Fig. 12.*a* that the points covered by the maximum number of boxes are precisely those in the connection and redundant regions of the variable and clause gadgets (like the ones added to P), and that each one is covered only by two boxes. In Fig. 12.b, we illustrate the intersection graph for $\mathcal{B}(\varphi)$. We argued in the proof of Theorem 1 that the sign of the literals in a clause do not change the maximum vertex degree or the clique-number of the graph. Similarly, the sign of the literals in φ do not affect the planarity of the intersection graph of $\mathcal{B}(\varphi)$: compare the gadgets for v_1 and v_2 in Fig. 12.*b* and note that changing the sign only alternates the color of the vertices corresponding to boxes of the variable gadget, but does not change the structure of the graph. Thus, the properties that follow are also true for a clause with any variation of the literal signs. Note that as there is no point covered by three boxes at the same time there are no 3-cliques in the graph. Also note that again the vertices of maximum degree are the ones corresponding to the leg boxes and their neighbors in the respective variable gadgets, whose vertex degree is three. Finally, we check that the intersection graph can be drawn within the planar embedding of φ so that no two edges cross, and hence the graph is planar. Again, due to the pairwise independence of the clause gadgets, these properties extend to the intersection graph of any general instance with more than one clauses. □

In the next section, we complement these hardness results with two approximation algorithms for the Minimum Coverage Kernel problem.

4. Efficient approximation of minimum coverage kernels

In Section 2, we described how to reduce the Minimum Coverage Kernel problem over a set $\mathcal{B}$ with n boxes to an instance of Box Cover, for dimension $d = 2$. For this, we showed how to find a set $\mathcal{D}^*(\mathcal{B})$ with $\mathcal{O}(n^2)$ points such that, if a subset $\mathcal{B}' \subseteq \mathcal{B}$ covers $\mathcal{D}^*(\mathcal{B})$, then $\mathcal{B}'$ is a coverage kernel of $\mathcal{B}$. Thus, for $d = 2$ the Minimum Coverage Kernel problem can be approximated in polynomial time by generating $\mathcal{D}^*(\mathcal{B})$, and using an approximation algorithm for the Box Cover problem. The approximation factors achieved are asymptotically the same as the ones known for Box Cover (i.e., $\log n$ deterministic [19], and $\log \log \mathtt{OPT}$ with high probability [6]) since $|\mathcal{D}^*(\mathcal{B})|$ is polynomial in n. In Section 4.1, we show that this approach can be generalized to any dimension $d > 2$, but that in the worst-case it requires a running time within $\Omega(n^d)$ because of the size of $\mathcal{D}^*(\mathcal{B})$. To improve this running time down to $\mathcal{O}(\mathtt{OPT} \cdot n^{\frac{d+1}{2}} \log^2 n)$ we introduce, in Section 4.2,

a new data structure that represents implicitly the (generalization of the) set $|\mathcal{D}^*(\mathcal{B})|$, and supports a few basic operations on it. These operations allow to adapt currently known approximation algorithms for Box Cover to approximate the Minimum Coverage Kernel problem. We show how to perform such adaptations in Section 4.3, where we introduce two approximation algorithms for the Minimum Coverage Kernel problem.

4.1. Coverage discretization of a set of boxes

To approximate a minimum coverage kernel of a set $\mathcal{B}$ using an approximation algorithm for the Box Cover problem as a "black box" one must generate a set P of points such that, for any subset $\mathcal{B}' \subseteq B$, $\mathcal{B}'$ covers P if and only if $\mathcal{B}'$ is a coverage kernel of $\mathcal{B}$. We name a set with such properties as a *coverage discretization of* $\mathcal{B}$.

Definition 4 *(Coverage Discretization).* Let $\mathcal{B}$ be a set of d-dimensional boxes, and let $\mathcal{D}(\mathcal{B})$ be a finite set of points such that any point in $\mathcal{D}(\mathcal{B})$ is covered by at least one box from B. $\mathcal{D}(\mathcal{B})$ is said to be a *coverage discretization* of $\mathcal{B}$ if for any subset $\mathcal{B}' \subseteq \mathcal{B}$, $\mathcal{B}'$ covers every point in $\mathcal{D}(\mathcal{B})$ if and only if $\mathcal{B}'$ covers the exact same region as $\mathcal{B}$ (i.e., $\mathcal{B}'$ is a coverage kernel of $\mathcal{B}$).

Note that in Section 2, in the reduction from the Minimum Coverage Kernel problem to Box Cover, we already described how to compute a coverage discretization of a set $\mathcal{B}$ of 2-dimensional boxes of size $|\mathcal{D}(\mathcal{B})| \in \mathcal{O}(|\mathcal{B}|^2)$. This construction can be generalized to any higher dimension as follows: Let $\mathcal{B}$ be a set of n d-boxes, and consider the grid obtained by "drawing" infinite hyperplanes through the $(d$-1)-faces of each box in $\mathcal{B}$. This grid has within $\mathcal{O}(n^d)$ cells, and each of those cells is either completely inside some box in $\mathcal{B}$, or outside the region covered by $\mathcal{B}$. Create a point-set P as follows: for each cell c of the grid contained in at some box in $\mathcal{B}$, add to P the center point of c. By construction, if a box $b \in \mathcal{B}$ covers a point p in P, then b covers completely the grid cell for which p is the center point. Thus, if a subset $\mathcal{B}' \subseteq \mathcal{B}$ covers all the points in P, then $\mathcal{B}'$ must cover all the grid cells covered by $\mathcal{B}$, and hence the same region as $\mathcal{B}$. Thus, P is a coverage discretization of $\mathcal{B}$. We name the unique set P that can be obtained for $\mathcal{B}$ according to this procedure as the *full coverage discretization* of $\mathcal{B}$, and use the notation $\mathcal{D}^*(\mathcal{B})$ for such a set.

While a set of boxes $\mathcal{B}$ might have multiple different coverage discretizations, by definition the full coverage discretization of $\mathcal{B}$ is unique. The size of a coverage discretization $\mathcal{D}(\mathcal{B})$ of $\mathcal{B}$ might be considerably larger than that of $\mathcal{D}^*(\mathcal{B})$. However, there are instances for which any coverage discretization must contain $\Omega(n^d)$ points, as we show next.

Lemma 1. *Let B be a set of n boxes, and let $\mathcal{D}(\mathcal{B})$ and $\mathcal{D}^*(\mathcal{B})$ be a coverage and a full coverage discretization of $\mathcal{B}$, respectively. In the worst case, $|\mathcal{D}(\mathcal{B})| \in \Theta(|\mathcal{D}^*(\mathcal{B})|) = \Theta(n^d)$.*

Proof. Consider, for instance, a set $\mathcal{B}$ obtained as follows: for all $i \in \{1, ..., d\}$, and for all $j \in \{0, ..., n\}$ add to $\mathcal{B}$ the box which extends from j to $j+1$ in the i-th dimension, and from 0 to n in the remaining $d-1$ dimensions. Note that $\mathcal{B}$ contains nd boxes, and their union covers a d-dimension hypercube Γ of side-length n and volume n^d. $\mathcal{B}$ induces a partition of Γ into n^d cells of the form $[x_1, x_1+1] \times \cdots \times [x_d, x_d+1]$, for all $(x_1, \ldots, x_d) \in \{1, ..., n\}^d$. These cells are precisely the ones obtained during the construction of a full coverage discretization for this particular $\mathcal{B}$. A cell c of the form $[x_1, x_1+1] \times \cdots \times [x_d, x_d+1]$ is covered only by d boxes from $\mathcal{B}$, the ones defined by the interval $[x_i, x_i+1]$ in the i-th dimension and by $[0, n]$ in the remaining $d-1$, for all $i \in \{1, ..., d\}$. Let $\mathcal{B}_c \subseteq \mathcal{B}$ denote the subset of boxes that cover a cell c. Note that for any two different cells c_1, c_2 the sets $\mathcal{B}_{c_1}$ and $\mathcal{B}_{c_2}$ contain at least one box that does not belong to the other, that is, $|B_{c_1} \cap \mathcal{B}_{c_2}| < d$, thus for any cell c the set $\mathcal{B} \setminus \mathcal{B}_c$ covers all the cells induced by $\mathcal{B}$ in Γ except c.

Now, consider a coverage discretization $\mathcal{D}(\mathcal{B})$ of $\mathcal{B}$. We will show that $\mathcal{D}(\mathcal{B})$ contain a point intersecting each of the n^d cells induced by $\mathcal{B}$ in Γ, and hence that the result of the lemma follows (because a point can intersect at most a constant number of such cells). For this, simply suppose that there is a cell $c = [x_1, x_1+1] \times \cdots \times [x_d, x_d+1]$ such that no point in $\mathcal{D}(\mathcal{B})$ is within c. Let $\mathcal{B}_c$ be the set of d boxes that cover c. Then, the subset $\mathcal{B}' = \mathcal{B} \setminus \mathcal{B}_c$ covers all the cells except c, and thus must cover all the points in $\mathcal{D}(\mathcal{B})$. However, $\mathcal{B}'$ does not cover the same region as $\mathcal{B}$, contradicting the fact that $\mathcal{D}(\mathcal{B})$ is a coverage discretization of $\mathcal{B}$. □

This result implies that, if one uses an approximation algorithm for Box Cover as a "black box" to obtain approximations for the Minimum Coverage Kernel problem, then the running time of such approach is within $\Omega(n^d)$ in the worst case. However, one should be able to exploit the regularities that appear in a full coverage discretization in order to improve this running time. We present in the next section a data structure that can represent a full coverage discretization implicitly, while supporting a few basic operations that allow to obtain approximation algorithms that depend only on this implicit implementation.

4.2. An efficient weight index for a set of boxes

Let $\mathcal{B}$ be a set of n boxes in $\mathbb{R}^d$, and let $\mathcal{D}^*(\mathcal{B})$ be a full coverage discretization of $\mathcal{B}$. A *weight index* for $\mathcal{D}^*(\mathcal{B})$ is a data structure which can perform the following operations:

- *Initialization:* Assign an initial unitary weight to every point in $\mathcal{D}^*(\mathcal{B})$;
- *Query:* Given a box $b \in \mathcal{B}$, find the total weight of the points in b.
- *Update:* Given a box $b \in \mathcal{B}$, multiply the weights of all the points within b by a given value $\alpha \geq 0$;

Note that this data structure is "static" with respect to $\mathcal{B}$: none of its operations involve adding or removing boxes from $\mathcal{B}$. The only operation that modifies the state of the data-structure is the update operation, which can only affect the weights of the points of $\mathcal{D}^*(\mathcal{B})$. Although the operations supported by a weight index are very basic, they are enough for our purpose of obtaining approximation algorithms for the Minimum Coverage Kernel problem.

We devote this section to describe an efficient implementation of a weight index, and then show, in Section 4.3, how to use this data structure together with two existing approximation algorithms for the Box Cover problem [6,19] to obtain approximation algorithms for the Minimum Coverage Kernel problem. We present a weight index for $\mathcal{D}^*(\mathcal{B})$ which can be initialized in time within $\mathcal{O}(n^{\frac{d+1}{2}})$, and with query and update time within $\mathcal{O}(n^{\frac{d-1}{2}} \log n)$. We assume that the weights are small enough so that arithmetic operations over the weights can be performed in constant time. We consider first the simpler case of intervals.

A weight index for a set of intervals. Let $\mathcal{D}^*(I)$ be a full coverage discretization of a set I of n intervals. In this case, $\mathcal{D}^*(I)$ is a set of values from $\mathbb{R}$. Consider the trivial solution which explicitly saves the weights of each point in $\mathcal{D}^*(I)$ in an array, and pre-computes and stores the answers to each of the $|I|$ possible queries. This weight index can be initialized in $\mathcal{O}(n \log n)$-time, has linear update time, and constant query time. We show that by sacrificing query time up to $\mathcal{O}(\log n)$-time, one can improve updates to $\mathcal{O}(\log n)$-time. The main idea is to maintain the weights of each point of $\mathcal{D}^*(I)$ indirectly using a tree. The data structure is basically a segment tree [5] with some extra *ad-hoc* values that are maintained at each node.

Consider a balanced binary tree whose leaves are in one-to-one correspondence with the points in $\mathcal{D}^*(I)$ (from left to right in a non-decreasing order). Let p_l denote the point corresponding to a leaf node l of the tree. In order to represent the weights of the points in $\mathcal{D}^*(I)$ so that updates can be supported faster, we store a value $\mu(v)$ at each node v of the tree subject to the following invariant: for each leaf l, the weight of the point p_l equals the product of the values $\mu(v)$ of all the ancestors v of l (including l itself). The μ values allow to increase the weights of many points with only a few changes. For instance, if we want to double the weights of all the points we simply multiply by 2 the value $\mu(r)$ of the root r of the tree. Besides the μ values, to allow efficient query time, we also store at each node v three values $\texttt{min}(v), \texttt{max}(v), \omega(v)$:

- the value $\texttt{min}(v)$ is the minimum weight of a leaf of the tree rooted at v;
- the value $\texttt{max}(v)$ is the maximum weight of a leaf of the tree rooted at v; and
- the value $\omega(v)$ is the sum of the weights of the leaves of the tree rooted at v, divided by the product of the μ values of the ancestors of v.

Note that the value of $\omega(r)$ of the root r of the tree is equal to the sum of the weights of all the points in $\mathcal{D}^*(I)$), namely the *total weight of* $\mathcal{D}^*(I)$ (namely, the *total weight* of $\mathcal{D}^*(I)$).

Initially, all the μ values are set to one. Besides, for every leaf l of the tree $\omega(l)$ is set to one, while $\texttt{min}(l)$ and $\texttt{max}(l)$ are set to p_l. The $\texttt{min}$, $\texttt{max}$ and ω values of every internal node v with children l, r, are initialized in a bottom-up fashion as follows: $\texttt{min}(v) = \min\{\texttt{min}(l), \texttt{min}(r)\}$; $\texttt{max}(v) = \max\{\texttt{max}(l), \texttt{max}(r)\}$; $\omega(v) = \mu(v) \cdot (\omega(l) + \omega(r))$. It is simple to verify that after this initialization, the tree meets all the invariants mentioned above. We show in Theorem 3 that this tree can be used as a weight index for $\mathcal{D}^*(I)$.

Theorem 3. *Let I be a set of n intervals in $\mathbb{R}$. There exists a weight index for $\mathcal{D}^*(I)$ which can be initialized in time within $\mathcal{O}(n \log n)$, and supports queries and updates in time within $\mathcal{O}(\log n)$.*

Proof. We show that the balanced binary tree described above fulfils the conditions of the theorem. To initialize it, we first compute explicitly $\mathcal{D}^*(I)$. Since intervals have linear union complexity, the size of $\mathcal{D}^*(I)$ is within $\mathcal{O}(n)$. Thus, it can be computed in linear time after sorting, or equivalently in time within $\mathcal{O}(n \log n)$ in the worst case. Then, we store these points in a balanced binary tree, which can be done in linear time from the sorted list of values in $\mathcal{D}^*(I)$ in a bottom up fashion. Due to the tree being built bottom-up, the initialization of the $\mu, \omega, \texttt{min}$, and $\texttt{max}$ values, respectively, can be done in constant time per node. Thus, the total initialization time is within $\mathcal{O}(n \log n)$.

To analyze the query time, let $\textsf{totalWeight}(a, b, t)$ denote the procedure which finds the total weight of the points corresponding to leaves of the tree rooted at t that are in the interval $[a, b]$. This procedure can be implemented as follows:

1. if $[a, b]$ and $[\texttt{min}(t), \texttt{max}(t)]$ are disjoint, return 0;
2. if $[a, b]$ completely contains $[\texttt{min}(t), \texttt{max}(t)]$ return $\omega(t)$;
3. if both conditions fail (leaves must meet either 1. or 2.), let l, r be the left and right children of t, respectively;
4. if $a > \texttt{max}(l)$ return $\mu(t) \cdot \textsf{totalWeight}(a, b, r)$;
5. if $b < \texttt{min}(r)$ return $\mu(t) \cdot \textsf{totalWeight}(a, b, l)$;
6. otherwise return $\mu(t)(\textsf{totalWeight}(a, \infty, l) + \textsf{totalWeight}(-\infty, b, r))$.

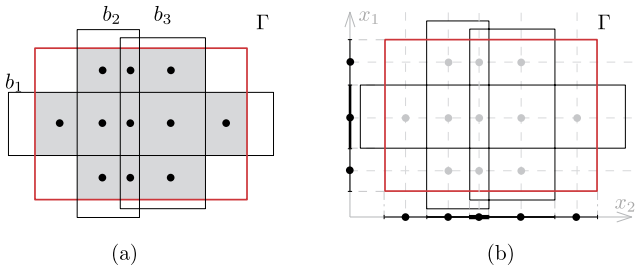

Fig. 13. An illustration in the plane of a set $\mathcal{B}$ with three boxes equivalent to slabs when restricted to the box Γ. a) The set of boxes $\mathcal{B}$ and its full coverage discretization $\mathcal{D}^*(\mathcal{B})$ of $\mathcal{B}$ within Γ. Any subset of $\mathcal{B}$ that covers $\mathcal{D}^*(\mathcal{B})$ also covers the (grayed) region covered by the slabs within Γ. b) The set of points in $\mathcal{D}^*(\mathcal{B})$ can be inferred by the discretization of intervals resulting from the projecting the slabs onto the axes.

Due to the invariants to which the `min` and `max` values are subjected, every leaf v of t corresponding to a point in $[a, b]$ has an ancestor (including v itself) which is visited during the call to totalWeight and which meets the condition in step 2. Because of this, and because of the invariants to which the ω and μ values are subjected, the procedure totalWeight is correct. Note that the number of nodes visited is at most 4 times the height h of the tree: when both children need to be visited, one of the endpoints of the interval to query is replaced by $\pm\infty$, which ensures that in subsequent calls at least one of the children is completely covered by the query interval. Since $h \in \mathcal{O}(\log n)$, and the operations at each node consume constant time, the running time of totalWeight is within $\mathcal{O}(\log n)$.

Similarly, to analyze the update time, let updateWeights(a, b, t, α) denote the procedure which multiplies by the value α the weights of the points in the interval $[a, b]$ stored in leaves descending from t. This can be implemented as follows:

1. if $[a, b]$ is disjoint to $[\texttt{min}(t), \texttt{max}(t)]$, finish;
2. if $[a, b]$ completely contains $[\texttt{min}(t), \texttt{max}(t)]$ set $\mu(t) = \alpha \cdot \mu(t)$, set $\omega(t) = \alpha \cdot \omega(t)$, and finish;
3. if both conditions fail, let l, r be the left and right child of t, respectively;
4. if $a > \texttt{max}(l)$, call updateWeights(a, b, r, α);
5. else if $b < \texttt{min}(r)$, call updateWeights(a, b, l, α);
6. otherwise, call updateWeights(a, ∞, l, α), and updateWeights$(-\infty, b, r, \alpha)$;
7. finally, after the recursive calls set $\omega(t) = \mu(t) \cdot (\omega(l) + \omega(r))$, and finish.

Note that, for every point p_v in $[a, b]$ corresponding to a leaf v descending from t, the μ value of exactly one of the ancestors of v changes (by a factor of α): at least one changes because of the invariants to which the `min` and `max` values are subjected (as analyzed for totalWeight); and no more than one can change because once μ is assigned for the first time to some ancestor u of v, the procedure finishes leaving the descendants of v untouched. The analysis of the running time is analogous to that of totalWeight, the conditions used to branch and travel down the tree are the same, and the work done at each node of the tree consumes constant time. Thus, the running time of updates is within $\mathcal{O}(\log n)$. □

The weight index for a set of intervals described in Theorem 3 plays an important role in obtaining an index for a higher dimensional set of boxes. In a step towards such a data structure, we first describe how to use one-dimensional indexes to obtain indexes for another special case of sets of boxes, this time in higher dimension.

A weight index for a set of slabs. A box b is said to be a *slab* when restricted to another box Γ if b covers completely Γ in all but one dimension (see Fig. 13.a for an illustration). Let $\mathcal{B}$ be a set of n d-dimensional boxes that are slabs when restricted to another box d-dimensional box Γ. Let $\mathcal{B}|_\Gamma$ denote the set $\{b \cap \Gamma \mid b \in \mathcal{B}\}$ of the boxes of $\mathcal{B}$ restricted to Γ. We describe a weight index for $\mathcal{D}^*(\mathcal{B}|_\Gamma)$ with initialization time within $\mathcal{O}(n \log n)$, and with update and query time within $\mathcal{O}(\log n)$.

For all $i \in \{1, \ldots, d\}$, let $\mathcal{B}_i \subseteq \mathcal{B}$ be the subset of slabs that are orthogonal to the i-th dimension, and let I_i be the set of intervals resulting from projecting Γ and each box in $\mathcal{B}|_\Gamma$ to the i-th dimension (see Fig. 13.b for an illustration). The key to obtain an efficient weight index for a set of slabs is the fact that weight indexes for $\mathcal{D}^*(I_1), \ldots \mathcal{D}^*(I_d)$ can be combined without much extra computational effort into a weight index for $\mathcal{D}^*(\mathcal{B}|_\Gamma)$. Let p be a point of $\mathcal{D}^*(\mathcal{B}|_\Gamma)$ and let $x_i(p)$ denote the i-th coordinate of p. Observe that for all $i \in \{1, ..., d\}$, $x_i(p) \in \mathcal{D}^*(I_i)$ (see Fig. 13.b for an illustration). This allows the representation of the weight of each point $p \in \mathcal{D}^*(\mathcal{B})$ by means of the weight $\omega_i(p)$ of $x_i(p)$ in $\mathcal{D}^*(I_i)$, for all $i \in \{1, ..., d\}$. We do this by maintaining the following *weight invariant*: the weight of a point $p \in \mathcal{D}^*(\mathcal{B}|_\Gamma)$ is equal to $\prod_{i=1}^{d} \omega_i(p)$.

Lemma 2. *Let $\mathcal{B}$ be a set of n d-dimensional boxes that are equivalent to slabs when restricted to another d-dimensional box Γ. There exists a weight index for $\mathcal{D}^*(\mathcal{B})$ which can be initialized in time within $\mathcal{O}(n \log n)$, supporting queries and updates in time within $\mathcal{O}(\log n)$, respectively.*

Proof. The initialization step consists in creating and initializing a weight index for $\mathcal{D}^*(I_i)$ as in Theorem 3, for all $i \in \{1, ..., d\}$. Since the weights of all the points in the one-dimensional indexes are initialized to one, the weight of every point

in $\mathcal{D}^*(\mathcal{B}|_\Gamma)$ is also initialized to one, according to the weight invariant. This initialization can be done in total time within $\mathcal{O}(n \log n)$.

Let $b \in \mathcal{B}|_\Gamma$ be a box which covers Γ in every dimension except for the i-th one, for some $i \in \{1, ..., d\}$ (i.e., $b \in B_i$), and let P_i be the subset of $\mathcal{D}^*(I_i)$ contained within the projection of b into the i-th dimension. The set of points of $\mathcal{D}^*(\mathcal{B}|_\Gamma)$ that are inside b is precisely the set $\{(a_1, \ldots, a_d) \mid a_1 \in \mathcal{D}^*(I_1), \ldots, a_{i-1} \in \mathcal{D}^*(I_{i-1}), a_i \in P_i, a_{i+1} \in \mathcal{D}^*(I_{i+1}), \ldots, a_d \in \mathcal{D}^*(I_d)\}$ (see Fig. 13.b for an illustration).

Therefore, because of the weight invariant and the fact that the components $a_1, \ldots, a_d$ are pairwise independent, the sum of the weights of the points within b is given by the sum of $\omega_i(p)$ for all $p \in P_i$ multiplied by the sum of the weights of the points in $\mathcal{D}^*(I_j)$, for all $j \in \{1, ..., d\}$, $j \neq i$. Formally, such a sum equals

$$\left(\sum_{p \in P_i} \omega_i(p)\right) \cdot \prod_{j \in \{1,...,d\} \setminus \{i\}} \left(\sum_{p \in \mathcal{D}^*(I_j)} \omega_j(p)\right).$$

To query the sum of the weights of the points of $\mathcal{D}^*(\mathcal{B}|_\Gamma)$ within a box $b \in B_i$, we query the weight index of $\mathcal{D}^*(I_i)$ to find the total weight of the points in the projection of b into the i-th dimension (in time within $\mathcal{O}(\log n)$), then query the remaining $d-1$ indexes to find the total weight stored in the index (i.e., the ω value of the root of the trees), and return the product of those d values. Clearly, the running time is within $\mathcal{O}(\log n)$.

The update is similar: to multiply by a value α the weights of all the points of $\mathcal{D}^*(\mathcal{B}|_\Gamma)$ within a box $b \in B_i$, we simply update the weight index of $\mathcal{D}^*(I_i)$ multiplying by α all the weights of the points within the projection of b into the i-th dimension, and leave the other $d-1$ weight indexes untouched. The running time of this operation is clearly within $\mathcal{O}(\log n)$ since it only calls the update of one weight index for intervals. The invariant remains valid after the update: if a point p of $\mathcal{D}^*(\mathcal{B}|_\Gamma)$ is inside b then $\omega_i(p)$ will increase by a factor of α after the update, and $\omega_{j \neq i}(p)$ will remain untouched, effectively increasing the weight of p by a factor of α. □

Lemma 2 shows that there are weight indexes for a set of slabs $\mathcal{B}$ within another box Γ that significantly improve the approach of explicitly constructing $\mathcal{D}^*(\mathcal{B}|_\Gamma)$, an improvement that grows exponentially with the dimension d. We take advantage of this to describe a similar improvement for the general case.

A Weight Index for The General Case. We now show how to maintain a weight index of a general set $\mathcal{B}$ of d-dimensional boxes. The main idea is to partition the space into disjoint cells (i.e., hyper-rectangular axis-aligned regions) such that, within each cell c, any box $b \in \mathcal{B}$ either completely contains c or is equivalent to a *slab*. Then, we use weight indexes for slabs (as described in Lemma 2) to index the weights within each of the cells. This approach was first introduced by Overmars and Yap [23] in order to compute the volume of the region covered by a set boxes, and similar variants were used since then to compute other measures [4,7,25]. The following lemma summarizes the key properties of the partition we use:

Lemma 3 *(Lemma 4.2 of Overmars and Yap [23]). Let $\mathcal{B}$ be a set of n boxes in d-dimensional space. There is a binary space partition tree for storing any subset of $\mathcal{B}$ such that*

- *It can be computed in time within $\mathcal{O}(n^{d/2})$, and it has $\mathcal{O}(n^{\frac{d}{2}})$ nodes;*
- *Each box is stored in $\mathcal{O}(n^{\frac{d-1}{2}})$ leafs;*
- *The boxes stored in a leaf node are slabs within the corresponding cell;*
- *Each leaf stores $\mathcal{O}(\sqrt{n})$ boxes.*

Consider the tree of Lemma 3. Analogously to the case of intervals, we augment this tree with information to efficiently support the operations of a weight index. At every node v we store two values $\mu(v), \omega(v)$: the first one allows to multiply all the weights of the points of $\mathcal{D}^*(\mathcal{B})$ that are maintained in leaves descending from v (allowing to support updates efficiently); while $\omega(v)$ stores the total weight of these points (allowing to support queries efficiently). To ensure that the set of nodes that intersect a box b is exactly the set of nodes visited during a query or update operation, we store at each node the boundaries of the cell corresponding to that node. Furthermore, at every leaf node l we implicitly represent the points of $\mathcal{D}^*(\mathcal{B})$ that are inside the cell corresponding to the node l using a weight index for slabs.

To initialize this data structure, all the μ values are set to one. Then, the weight indices within each leaf cell are initialized. Finally, the ω values of every node v with children l, r, are initialized in a bottom-up fashion by setting $\omega(v) = \mu(v) \cdot (\omega(l) + \omega(r))$. We show in Theorem 4 how to implement the weight index operations over this tree and we analyze its running times.

Theorem 4. *Let $\mathcal{B}$ be a set of n d-dimensional boxes. There is a weight index for $\mathcal{D}^*(\mathcal{B})$ which can be initialized in time within $\mathcal{O}(n^{\frac{d+1}{2}})$, supporting queries and updates in time within $\mathcal{O}(n^{\frac{d-1}{2}} \log n)$.*

Proof. The initialization of the index, when implemented as described before, runs in constant time for each internal node of the tree, and in time within $\mathcal{O}(\sqrt{n}\log n)$ for each leaf (due to Lemma 2, and to the last item of Lemma 3). Since the tree has $\mathcal{O}(n^{\frac{d}{2}})$ nodes, the total running time of the initialization is within $\mathcal{O}(n^{\frac{d+1}{2}}\log n)$.

Since the implementations of the query and update operations are analogous to those of weight index for intervals (see the proof of Theorem 3), we omit the details of their correctness. While performing a query/update operation at most $\mathcal{O}(n^{\frac{d-1}{2}})$ leaves are visited (due to the third item of Lemma 3), and since the height of the tree is within $\mathcal{O}(\log n)$, at most $\mathcal{O}(n^{\frac{d-1}{2}}\log n)$ internal nodes are visited in total. Hence, the cost of a query/update operation within each leaf is within $\mathcal{O}(\log n)$ (by Lemma 2), and is constant in each internal node. Thus, the total running of a query/update operation is within $\mathcal{O}(n^{\frac{d-1}{2}}\log n)$. □

4.3. Approximation algorithms for minimum coverage kernel

Approximating the Minimum Coverage Kernel of a set $\mathcal{B}$ of boxes via approximation algorithms for the Box Cover problem requires that $\mathcal{D}^*(\mathcal{B})$ is explicitly constructed. However, the weight index described in the proof of Theorem 4 can be used to significantly improve the running time of these algorithms. We describe below two examples.

The first algorithm we consider is the greedy $\mathcal{O}(\log n)$-approximation algorithm by Lovász [19]. Such a greedy strategy applies naturally to the Minimum Coverage Kernel problem: iteratively pick the box which covers the most yet uncovered points of $\mathcal{D}^*(\mathcal{B})$, until there are no points of $\mathcal{D}^*(\mathcal{B})$ left to cover (we describe this approach in details in Appendix A). To avoid the explicit construction of $\mathcal{D}^*(\mathcal{B})$, three operations must be simulated: (*i.*) count the uncovered points within a given a box $b \in \mathcal{B}$; (*ii.*) delete the points that are covered by a box $b \in \mathcal{B}$; and (*iii.*) decide whether a subset $\mathcal{B}'$ of $\mathcal{B}$ covers all the points of $\mathcal{D}^*(\mathcal{B})$.

For the first two operations, we use the weight index described in the proof of Theorem 4: to delete the points within a given box $b \in \mathcal{B}$, we simply multiply the weights of all the points of $\mathcal{D}^*(\mathcal{B})$ within b by $\alpha = 0$; and finding the number of uncovered points within a box b is equivalent to finding the total weight of the points of $\mathcal{D}^*(\mathcal{B})$ within b. For the last operation we use the following straightforward observation:

Observation 1. Let $\mathcal{B}$ be a set of d-dimensional boxes, and let $\mathcal{B}'$ be a subset of $\mathcal{B}$. The volume of the region covered by $\mathcal{B}'$ equals that of $\mathcal{B}$ if and only if $\mathcal{B}'$ and $\mathcal{B}$ cover the exact same region.

Let $\texttt{OPT}$ denote the size of a minimum coverage kernel of $\mathcal{B}$, and let N denote the size of $\mathcal{D}^*(\mathcal{B})$ ($N \in \mathcal{O}(n^d)$). The greedy algorithm described by Lovász [19], when run over the sets $\mathcal{B}$ and $\mathcal{D}^*(\mathcal{B})$, works in $\mathcal{O}(\texttt{OPT}\log N)$ steps; and at each stage a box is added to the solution. The size of the output is within $\mathcal{O}(\texttt{OPT}\log N) \subseteq \mathcal{O}(\texttt{OPT}\log n)$, and its total running time is within $\Omega(n^{d+1})$ in the worst-case, because of the size of $\mathcal{D}^*(\mathcal{B})$. This algorithm can be modified to run in $\mathcal{O}(\texttt{OPT}\cdot n^{\frac{d+1}{2}}\log^2 n)$-time, while achieving the same approximation ratio:

Theorem 5. *Let $\mathcal{B}$ be a set of n boxes in $\mathbb{R}^d$ with a minimum coverage kernel of size $\texttt{OPT}$. Then, a coverage kernel of $\mathcal{B}$ of size within $\mathcal{O}(\texttt{OPT}\log n)$ can be computed in time within $\mathcal{O}(\texttt{OPT}\cdot n^{\frac{d+1}{2}}\log^2 n)$.*

Proof. We initialize a weight index as described in Theorem 4, which can be done in time within $\mathcal{O}(n^{\frac{d+1}{2}})$, and compute the volume of the region covered by $\mathcal{B}$, which can be done in time within $\mathcal{O}(n^{d/2})$ [7]. Let C be initialized to an empty set. At each step of the algorithm, for every box $b \in \mathcal{B} \setminus C$ we compute the total weight of the points inside b (which can be done in time within $\mathcal{O}(n^{\frac{d-1}{2}}\log n)$ using the weight index). We add to C the box with the highest total weight, and update the weights of all the points within this box to zero (by multiplying their weights by $\alpha = 0$) in time within $\mathcal{O}(n^{\frac{d-1}{2}}\log n)$. If the volume of the region covered by C (which can be computed in $\mathcal{O}(n^{d/2})$-time [7]) is the same as that of $\mathcal{B}$, then we stop and return C as the approximate solution. The total running time of each step is within $\mathcal{O}(n^{\frac{d+1}{2}}\log n)$. This, along with the fact that the number of stages is within $\mathcal{O}(\texttt{OPT}\log n)$, yields the result of the theorem. □

Now, we show how to improve the running time of Brönnimann and Goodrich's $\mathcal{O}(\log \texttt{OPT})$-approximation algorithm [6] via a weight index. Brönnimann and Goodrich's algorithm can be though as the composition of two nested loops. The inner loop assumes that an approximation k for the size of an optimal solution is known, and either computes an approximation of size k, or determines that k is a bad approximation and ends without a solution. The outer loop guesses a value for k, tests it with the inner loop, and return the solution if k is a good approximation, or doubles the value of k if not. Below, we review their main idea (to make this article self-contained, we provide a detailed description of their algorithm in Appendix A).

Let $w : \mathcal{D}(\mathcal{B}) \to \mathbb{R}$ be a weight function for the points of $\mathcal{D}^*(\mathcal{B})$, and for a subset $\mathcal{P} \subseteq \mathcal{D}^*(\mathcal{B})$ let $w(P)$ denote the sum of all the weights of the points in $\mathcal{P}$. A point p is said to be ε-heavy, for a value $\varepsilon \in (0, 1]$, if $w(p) \geq \varepsilon \cdot w(\mathcal{D}^*(\mathcal{B}))$, and ε-light otherwise. A subset $\mathcal{B}' \subseteq \mathcal{B}$ is said to be an ε-net with respect to w if for every ε-heavy point $p \in \mathcal{D}^*(\mathcal{B})$ there is a box in $\mathcal{B}'$ which contains p.

Let $\mathtt{OPT}$ denote the size of a minimum coverage kernel of $\mathcal{B}$, and let k be an integer such that $k/2 \leq \mathtt{OPT} < k$. The inner loop of Brönnimann and Goodrich's algorithm initializes the weight of each point in $\mathcal{D}^*(\mathcal{B})$ to 1, and repeats the following *weight-doubling step* until every point is $\frac{1}{2k}$-heavy: find a $\frac{1}{2k}$-light point p and double the weights of all the points inside every box $b \in \mathcal{B}$ that contain p. Brönnimann and Goodrich [6] showed that for a given k, if more than $4k\log(n/k)$ weight-doubling steps are performed, then $\mathtt{OPT} > 2k$. If this happens, the inner loop determines that k is a bad approximation for $\mathtt{OPT}$, and ends without a solution. If this is not the case, the weight doubling steps end when all the points are $\frac{1}{2k}$-heavy. In this case, a Monte-Carlo algorithm is used to compute (and return as the approximate solution) a set C of boxes which is an $\frac{1}{2k}$-net with respect to the final weights. Since each point in $\mathcal{D}^*(\mathcal{B})$ is $\frac{1}{2k}$-heavy, C covers all the points of $\mathcal{D}^*(\mathcal{B})$. Hence, if a $\frac{1}{2k}$-net of size $\mathcal{O}(k \cdot g(k))$ can be computed efficiently, for some function $g(k)$, this algorithm computes a solution of size $\mathcal{O}(k \cdot g(k))$. The outer loop of the algorithm simply guesses values for k using doubling search, testing with the inner loop whether the guess is correct. For more details on the complete algorithm, please refer to Appendix A.

We simulate the operations over the weights of $\mathcal{D}^*(\mathcal{B})$ using again a weight index, but this time with a minor variation to that described in Theorem 4: in every node of the space partition tree, besides the ω, μ values, we also store the minimum weight of the points within the cell corresponding to the node. During the initialization and update operations of the weight index, this value can be maintained as follows: for a node v with children l, r, the minimum weight $\mathtt{min}_\omega(v)$ of a point in the cell of v can be computed as $\mathtt{min}_\omega(v) = \mu(v) \cdot \min\{\mathtt{min}_\omega(l), \mathtt{min}_\omega(r)\}$. This value allows to efficiently detect whether there are $\frac{1}{2k}$-light points, and to find one if such one exists by tracing down, in the partition tree, the path from which that value comes.

To compute a $\frac{1}{2k}$-net, we choose a sample of $\mathcal{B}$ by performing at least $(16dk \log 16dk)$ random independent draws from $\mathcal{B}$. We then check whether it is effectively a $\frac{1}{2k}$-net, and if not, we repeat the process, up to a maximum of $\mathcal{O}(\log n)$ times. Haussler and Welzl [12] showed that such a sample is a $\frac{1}{2k}$-net with probability at least $1/2$. Thus, the expected number of samples needed to obtain a $\frac{1}{2k}$-net is constant, and since we repeat the process up to $\mathcal{O}(\log n)$ times, the probability of effectively finding one is at least $1 - \frac{1}{n^{\Omega(1)}}$. We analyze the running time of this approach in the following theorem.

Theorem 6. *Let $\mathcal{B}$ be a set of n boxes in $\mathbb{R}^d$ with a minimum coverage kernel of size $\mathtt{OPT}$. There is an algorithm which computes a coverage kernel of $\mathcal{B}$ of size within $\mathcal{O}(\mathtt{OPT} \cdot \log \mathtt{OPT})$, with high probability within $1 - \frac{1}{n^{\Omega(1)}}$, running in expected time within $\mathcal{O}(\mathtt{OPT} \cdot n^{\frac{d+1}{2}} \log^2 n)$.*

Proof. We first show how to adapt inner loop of Brönnimann and Goodrich's algorithm to use a weight index. Within each step of the loop we initialize a weight index for $\mathcal{D}^*(\mathcal{B})$ in time within $\mathcal{O}(n^{\frac{d+1}{2}})$. Finding whether there is a $\frac{1}{2k}$-light point in $\mathcal{D}^*(\mathcal{B})$ can be done in constant time: the root of the partition tree stores both $w(\mathcal{D}^*(\mathcal{B}))$ and the minimum weight of any point in the ω and $\mathtt{min}_\omega$ values, respectively. For every light point, the weight-doubling steps consume time within $\mathcal{O}\left(n \cdot \left(n^{\frac{d-1}{2}} \log n\right)\right) \subseteq \mathcal{O}(n^{\frac{d+1}{2}} \log n)$ (by Theorem 4). Since at each step at most $4k\log(n/k)$ weight-doubling steps are performed, the total running time of each stage is within $\mathcal{O}(kn^{\frac{d+1}{2}} \log n \log \frac{n}{k}) \subseteq \mathcal{O}(kn^{\frac{d+1}{2}} \log^2 n)$.

The outer loop of the algorithm does not require any adaptation. It performs several steps guessing the value of k, until $k/2 < \mathtt{OPT} < k$. The inner loop is used to test the validity of the guess, and after each bad guess, the value of k is doubled. Because of this, k increases geometrically, and since the running time of each step is a polynomial function, the sum of the running times of all the steps is asymptotically dominated by that of the last step, for which we have that $k \leq \mathtt{OPT} \leq 2k$. Thus, the theorem follows. □

Compared to the algorithm described in Theorem 5, this last approach yields a better approximation factor on instances with small Coverage Kernels ($\mathcal{O}(\log n)$ vs. $\mathcal{O}(\log \mathtt{OPT})$), but the improvement comes at a cost not only in the running time (worst case vs. expected), but in the probability of finding such a good approximation: the algorithm of Theorem 6 is a Monte-Carlo algorithm which, with a small probability, does not compute an approximation at all.

5. Discussion

We studied the computational complexity of the Minimum Coverage Kernel problem, intermediate between the Box Cover and the Orthogonal Polygon Covering problems. We proved that Minimum Coverage Kernel remains NP-hard under various restrictions over the intersection graph of the input set of boxes, and showed that the Box Cover problem remains NP-hard under even more extreme settings. We also showed that while the Box Cover and Minimum Coverage Kernel can be approximated within at least the same factors, the running time required to obtain some of those approximations can be significantly improved for the Minimum Coverage Kernel problem.

Fixed-Parameter Tractability. Another approach to understand what makes a problem hard is Parameterized Complexity [10], where the hardness of a problem is analyzed with respect to multiple parameters of the input, with the hope of finding measures gradually separating "easy" instances form the "hard" ones. The hardness results described in Section 3 show that

for the Minimum Coverage Kernel and Box Cover problems, the vertex degree and clique-number of the underlaying graph are not good candidates of such kind of measures, as opposed to what happens for other related problems [2].

Tighter bounds. We showed that the Minimum Coverage Kernel problem remains NP-hard even if the intersection graph of the boxes has clique-number bounded by a constant $c \geq 4$, or has vertex degree bounded by a constant $\delta \geq 10$. However, it is open whether these bounds are tight: for the special cases where the intersection graph has clique-number bounded by $c \geq 3$, or vertex degree $\delta \geq 7$, it remains open whether the problem is NP-hard or not. For the Box Cover problem we showed it is NP-hard even if the intersection graph of the boxes has clique-number bounded by a constant $c \geq 2$, or has bound vertex degree $\delta \geq 4$. In this case, the bound for the clique-number is trivially tight: if the intersection graph has clique-number $c = 1$, then no two boxes intersect, and the solution to the problem is trivial. However, it is open whether the vertex degree bound is tight.

Improvements in small dimensions. In two and three dimensions, the Box Cover problem can be approximated up to a factor within $\mathcal{O}(\log\log \mathtt{OPT})$ [3]. Whether the running time of this algorithm can be also improved for the case of Minimum Coverage Kernel is open. The approach described in Section 4, based on weight indices, seems to be relevant only when the dimension of the boxes is high (while still constant), as in various applications [9,17,24] of the Minimum Coverage Kernel problem.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We thank the anonymous referees for pointing out an error in the proof of Theorems 5 and 6 in an early draft, as well as for carefully reading the manuscript, and providing many insightful comments and suggestions.

Appendix A. Approximation algorithms for Box Cover

As the Minimum Coverage Kernel problem is a special case of the Box Cover problem, all the approximation algorithms for the later one can be used to obtain approximated solutions for the Minimum Coverage Kernel problem. We review below two such algorithms.

The first algorithm we consider is the greedy $\mathcal{O}(\log n)$-approximation algorithm by Lovász [19] and Johnson [14]. This algorithm was described originally for the Set Cover problem, but it can be naturally described in terms of the Box Cover problem: iteratively pick the box which covers the most yet uncovered points, until there are no points left to cover.

Algorithm 1 `Greedy-Set-Cover`$(\mathcal{T}, \mathcal{R})$.

Require: A set system $(\mathcal{T}, \mathcal{R})$
Ensure: A subset $C \subseteq \mathcal{R}$ covering $\mathcal{T}$
1: $U \leftarrow \mathcal{T}$, $C \leftarrow \emptyset$
2: **while** $U \neq \emptyset$ **do**
3: select an element $r \in \mathcal{R}$ that maximizes $|U \cap r|$
4: $U \leftarrow U \setminus (U \cap r)$
5: $C \leftarrow C \cup \{r\}$
6: **return** C

Lovász [19] and Johnson [14] showed that this algorithm yields a cover of at most $\mathtt{OPT} \cdot \ln n$ boxes, where n is the number of points, and its running time is polynomial in the size of the input. Note that, although the approximation factor depends on the number of points, this algorithm yields a $\mathcal{O}(\log n)$-approximation for the Minimum Coverage Kernel problem. This, because the number of points generated in the reduction to Box Cover is within $\mathcal{O}(n^d)$, n being this time the number of boxes of the Minimum Coverage Kernel instance.

For the general Set Cover problem, the greedy algorithm is the best that one can hope for, unless P=NP. However, one would expect that in the case of the Box Cover problem, better approximation factors can be obtained by exploiting the underlying geometry. Brönnimann and Goodrich [6] showed that indeed this is possible, and described a $\mathcal{O}(\log \mathtt{OPT})$-approximation algorithm running in polynomial time. Their algorithm employs and adapts a wide range of techniques, including the usage of ε-nets [12]:

Definition 5 *(ε-net).* Let $\Sigma = (\mathcal{T}, \mathcal{R})$ be a set system, and let $\varepsilon \in (0, 1]$ be a parameter. An *ε-net* for Σ is a subset $N \subseteq \mathcal{R}$ such that, for every $t \in \mathcal{T}$ that is contained in at least $\varepsilon|\mathcal{R}|$ elements of $\mathcal{R}$, there is one element in N that contains t.[9]

The concept of an ε-net can be naturally generalized to consider weights. Given a set system $\Sigma = (\mathcal{T}, \mathcal{R})$, and a weight function $\omega : \mathcal{R} \to \mathbb{R}_{\geq 0}$, let $\omega(S)$ denote the total weight of the elements in a subset $S \subset \mathcal{R}$; and for any $x \in \mathcal{T}$ let $\mathcal{R}_x = \{R \in \mathcal{R} \mid R \text{ contains } x\}$.

Definition 6 *(ε-heavy, ε-light).* Let $\Sigma = (\mathcal{T}, \mathcal{R})$ be a set system, let $\omega : \mathcal{R} \to \mathbb{R}_{\geq 0}$ be a weight function and let $\varepsilon \in (0, 1]$ be a parameter. An element $t \in \mathcal{T}$ is said to be *ε-heavy* if $\omega(\mathcal{R}_t) \geq \varepsilon\omega(\mathcal{R})$, and *ε-light* otherwise.

Definition 7 *(Weighted ε-net).* Let $\Sigma = (\mathcal{T}, \mathcal{R})$ be a set system, let $\omega : \mathcal{R} \to \mathbb{R}_{\geq 0}$ be a weight function and let $\varepsilon \in (0, 1]$ be a parameter. A *weighted ε-net* for Σ and ω is a subset $N \subseteq \mathcal{R}$ such that $N \cap \mathcal{R}_t \neq \emptyset$ for every ε-heavy element $t \in \mathcal{T}$.

Typically, ε-net algorithms are designed for the unweighted case [6]. However, simple methods allow to reduce the weighted case to an unweighted one. For instance, the following was described by Brönnimann and Goodrich [6]: first, let $m = |\mathcal{R}|$, and scale the weights such that $w(\mathcal{R}) = m$; then, take $\lfloor w(R) + 1 \rfloor$ copies of each range $R \in \mathcal{R}$. Note that the new set of ranges $\mathcal{R}'$ thus obtained contains all the elements of $\mathcal{R}$, and has cardinality at most $2m$. An ε-net for the set $\mathcal{R}'$ is also an ε-net for the original set $\mathcal{R}$ with weights ω [6,20].

Algorithm 2 `BG-Set-Cover`$(\mathcal{T}, \mathcal{R})$.

Require: A set system $(\mathcal{T}, \mathcal{R})$
Ensure: A subset $C \subseteq \mathcal{R}$ covering $\mathcal{T}$
$k \leftarrow 1$, $C \leftarrow \emptyset$
do
  $k \leftarrow 2k$ ▷ (guess a new bound for OPT)
  $\mu_k = \lceil 4k \log(m/k) \rceil$ ▷ (set the maximum number of steps to test the guess)
  for $R \in \mathcal{R}$ **do** $\omega(R) \leftarrow 1$
  while $\mu_k > 0$ **do**
    if there is an $\frac{1}{2k}$-light element $t \in \mathcal{T}$ **then**
      $\mu_k \leftarrow \mu_k - 1$
      for $R \in \mathcal{R}_t$ **do** $w(R) \leftarrow 2\omega(R)$ ▷ (weight-doubling step)
    else ▷ (all elements of $\mathcal{T}$ are $\frac{1}{2k}$-heavy)
      find a $\frac{1}{2k}$**-net** N **with respect to** ω ▷ (and thus any $\frac{1}{2k}$-net covers $\mathcal{T}$)
      $C \leftarrow N$, $\mu_k \leftarrow 0$ ▷ (store the solution, and end both cycles)
while $C = \emptyset$
return C

Brönnimann and Goodrich's algorithm. Let $\Sigma = (\mathcal{T}, \mathcal{R})$ be a set system, and let OPT denote the size of an optimal Set Cover for Σ. Let k be an integer value such that $k/2 < \mathtt{OPT} < k$. Initialize the weight of each $R \in \mathcal{R}$ to 1, and repeat the following *weight-doubling step* until every element $t \in \mathcal{T}$ is $\frac{1}{2k}$-heavy: find an $\frac{1}{2k}$-light point p and double the weights of all the elements of $\mathcal{R}$ containing p. When this process stops, return a $\frac{1}{2k}$-net C with respect to the final weights as the approximated solution.

Since each point in $\mathcal{T}$ is $\frac{1}{2k}$-heavy, C covers all the points of $\mathcal{T}$. Hence, if a $\frac{1}{2k}$-net of size $\mathcal{O}(kg(k))$ can be computed efficiently, this algorithm computes a solution of size $\mathcal{O}(kg(k))$ (i.e., a $g(\mathcal{O}(\mathsf{OPT}))$-approximation). The following lemma is the key to the performance of the algorithm:

Lemma 4 *(Lemma 3.4 of Brönnimann and Goodrich [6]). If Σ has a Set Cover of size at most k, then the algorithm performs at most $\mu_k = 4k \log(m/k)$ weight-doubling steps. The final weight of $\mathcal{R}$ is at most m^4/k^3.*

By Lemma 4, if the algorithm does not terminate within μ_k steps, then one can conclude that Σ does not have a Set Cover of size at most k (i.e., $\mathtt{OPT} > 2k$). This allows to guess the correct k via doubling search in $\mathcal{O}(\log \mathsf{OPT})$ stages (see Algorithm 2 for a detailed description). Since each step can be trivially implemented in time polynomial in $n + m$, and the weights can be represented with $\mathcal{O}(\log n)$ bits, the total running time of the algorithm is polynomial in $n + m$.

Note that Brönnimann and Goodrich's algorithm [6] is a general method which is not bound to geometric instances of the Set Cover problem, but to the existence of algorithms computing good ε-nets. And it is precisely in this last part that geometry becomes relevant. Haussler and Welzl [12] proved that for a set system of VC-dimension[10] δ, an ε-net of size

[9] The definition introduced by Haussler and Welzl [12] slightly differs from the one in Definition 5. However, it is simple to check that they are equivalent by considering their original formulation over the dual set system. The adaptation of Definition 5 is more natural for the Set Cover problem and its special cases, while the original definition is more natural for the dual Hitting Set problem.

[10] The *VC-dimension* of a set system $\Sigma = (\mathcal{T}, \mathcal{R})$ is the size of the largest subset $A \subseteq \mathcal{T}$ such that $|\{A \cap R \mid R \in \mathcal{R}\}| = 2^{|A|}$.

$\mathcal{O}((\delta/\varepsilon)\log(\delta/\varepsilon))$ can be computed in polynomial time. The VC-dimension of a set system $(\mathcal{P}, \mathcal{B})$, where $\mathcal{P}$ and $\mathcal{B}$ are sets of d-dimensional points and boxes, respectively, is constant in any constant dimension d. Thus, the result of Haussler and Welzl [12] automatically yields a polynomial-time $\mathcal{O}(\log \mathsf{OPT})$-approximation algorithm. But the bounds on the size of ε-nets have been improved for several other geometric set systems. For instance, for set systems of points and halfspaces in $\mathbb{R}^2$ and $\mathbb{R}^3$, ε-nets of size $\mathcal{O}(1/\varepsilon)$ can be computed in polynomial time [21]; and for the set systems of the Box Cover problem in $d = 2, 3$ dimensions, Aronov et al. [3] gave a polynomial-time randomized algorithm to construct an ε-net of size within $\mathcal{O}(1/\varepsilon \log\log(1/\varepsilon))$.

References

[1] P.K. Agarwal, J. Pan, Near-linear algorithms for geometric hitting sets and set covers, in: Proceedings of the 30th Annual Symposium on Computational Geometry (SoCG), ACM, New York, NY, USA, ISBN 978-1-4503-2594-3, 2014, pp. 271:271–271:279.

[2] V.E. Alekseev, R. Boliac, D.V. Korobitsyn, V.V. Lozin, Np-hard graph problems and boundary classes of graphs, Theor. Comput. Sci. 389 (1–2) (2007) 219–236, https://doi.org/10.1016/j.tcs.2007.09.013.

[3] B. Aronov, E. Ezra, M. Sharir, Small-size ε-nets for axis-parallel rectangles and boxes, SIAM J. Comput. 39 (7) (2010) 3248–3282, https://doi.org/10.1137/090762968.

[4] J. Barbay, P. Pérez-Lantero, J. Rojas-Ledesma, Depth distribution in high dimensions, in: Y. Cao, J. Chen (Eds.), Proceedings of the 23rd International Conference on Computing and Combinatorics (COCOON), Hong Kong, China, August 3-5, 2017, in: Lecture Notes in Computer Science, vol. 10392, ISBN 978-3-319-62388-7, Springer, 2017, pp. 38–49.

[5] J.L. Bentley, Algorithms for Klee's rectangle problems, Technical report, Computer Science Department, Carnegie Mellon University, 1977.

[6] H. Brönnimann, M.T. Goodrich, Almost optimal set covers in finite vc-dimension, Discrete Comput. Geom. 14 (4) (1995) 463–479, https://doi.org/10.1007/BF02570718.

[7] T.M. Chan, Klee's measure problem made easy, in: 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Berkeley, CA, USA, 26–29 October, 2013, IEEE Computer Society, ISBN 978-0-7695-5135-7, 2013, pp. 410–419.

[8] J.C. Culberson, R.A. Reckhow, Covering polygons is hard, J. Algorithms 17 (1) (1994) 2–44, https://doi.org/10.1006/jagm.1994.1025.

[9] J. Daly, A.X. Liu, E. Torng, A difference resolution approach to compressing access control lists, IEEE/ACM Trans. Netw. 24 (1) (2016) 610–623, https://doi.org/10.1109/TNET.2015.2397393.

[10] R.G. Downey, M.R. Fellows, Parameterized Complexity, Monographs in Computer Science, Springer, ISBN 978-1-4612-6798-0, 1999.

[11] R.J. Fowler, M. Paterson, S.L. Tanimoto, Optimal packing and covering in the plane are NP-complete, Inf. Process. Lett. 12 (3) (1981) 133–137.

[12] D. Haussler, E. Welzl, epsilon-nets and simplex range queries, Discrete Comput. Geom. 2 (1987) 127–151, https://doi.org/10.1007/BF02187876.

[13] H. Imai, T. Asano, Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane, J. Algorithms 4 (4) (1983) 310–323, https://doi.org/10.1016/0196-6774(83)90012-3.

[14] D.S. Johnson, Approximation algorithms for combinatorial problems, J. Comput. Syst. Sci. 9 (3) (1974) 256–278, https://doi.org/10.1016/S0022-0000(74)80044-9.

[15] D.E. Knuth, A. Raghunathan, The problem of compatible representatives, SIAM J. Discrete Math. 5 (3) (1992) 422–427, https://doi.org/10.1137/0405033.

[16] V.S.A. Kumar, H. Ramesh, Covering rectilinear polygons with axis-parallel rectangles, SIAM J. Comput. 32 (6) (2003) 1509–1541, https://doi.org/10.1137/S0097539799358835.

[17] L.V.S. Lakshmanan, R.T. Ng, C.X. Wang, X. Zhou, T. Johnson, The generalized MDL approach for summarization, in: Proceedings of 28th International Conference on Very Large Data Bases (VLDB), August 20-23, 2002, Hong Kong, China, Morgan Kaufmann, 2002, pp. 766–777, http://www.vldb.org/conf/2002/S22P01.pdf, 2002.

[18] D. Lichtenstein, Planar formulae and their uses, SIAM J. Comput. 11 (2) (1982) 329–343, https://doi.org/10.1137/0211025.

[19] L. Lovász, On the ratio of optimal integral and fractional covers, Discrete Math. 13 (4) (1975) 383–390, https://doi.org/10.1016/0012-365X(75)90058-8.

[20] J. Matoušek, Cutting hyperplane arrangements, Discrete Comput. Geom. 6 (1991) 385–406, https://doi.org/10.1007/BF02574697.

[21] J. Matoušek, Reporting points in halfspaces, Comput. Geom. 2 (1992) 169–186, https://doi.org/10.1016/0925-7721(92)90006-E.

[22] W. Mulzer, G. Rote, Minimum-weight triangulation is NP-hard, J. ACM 55 (2) (2008).

[23] M.H. Overmars, C. Yap, New upper bounds in Klee's measure problem, SIAM J. Comput. 20 (6) (1991) 1034–1045, https://doi.org/10.1137/0220065.

[24] K.Q. Pu, A.O. Mendelzon, Concise descriptions of subsets of structured sets, ACM Trans. Database Syst. 30 (1) (2005) 211–248, https://doi.org/10.1145/1061318.1061324.

[25] H. Yildiz, J. Hershberger, S. Suri, A discrete and dynamic version of klee's measure problem, in: Proceedings of the 23rd Annual Canadian Conference on Computational Geometry (CCCG), Toronto, Ontario, Canada, August 10–12, 2011, 2011, pp. 1–6, http://www.cccg.ca/proceedings/2011/papers/paper28.pdf.