Covering Weighted Points Using Unit Squares

Chaeyoon Chung
Department of Computer Science and Engineering, Pohang University of Science and Technology, South Korea

Jaegun Lee
Department of Computer Science and Engineering, Pohang University of Science and Technology, South Korea

Hee-Kap Ahn
Department of Computer Science and Engineering, Graduate School of Artificial Intelligence, Pohang University of Science and Technology, South Korea

Abstract

Given a set of n points in d-dimensional space, each assigned a positive weight, we study the problem of finding k axis-parallel unit hypercubes that maximize the total weight of the points contained in their union. In this paper, we present both exact and $(1-\varepsilon)$-approximation algorithms for the case of $k = 2$. We present an exact algorithm that runs in $O(n^2)$ time in the plane, improving the previous $O(n^2 \log^2 n)$-time result. This algorithm generalizes to higher dimensions and larger k in $O(n^{dk/2})$ time for fixed d and k. We also present a $(1-\varepsilon)$-approximation algorithm that runs in $O(n \log \min\{n, 1/\varepsilon\} + 1/\varepsilon^3)$ time for $k = 2$ in the plane, improving the best known result. Our approximation algorithm also extends to higher dimensions.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Maximum coverage, Unit squares, Approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.21

Funding This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (RS-2023-00219980), and the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.RS-2019-II191906, Artificial Intelligence Graduate School Program (POSTECH)) and (No. 2017-0-00905, Software Star Lab (Optimal Data Structure and Algorithmic Applications in Dynamic Geometric Environment)).

1 Introduction

The maximum coverage problem is a classic and widely studied problem in theoretical computer science and computational geometry. Given a set of weighted elements, a collection of subsets, and an integer k, the goal is to choose up to k subsets that maximize the total weight of the elements contained in the union of the subsets. This problem naturally models many real-world scenarios such as facility location, sensor placement, and resource allocation. It is well-known that the maximum coverage problem is NP-hard and cannot be approximated within a factor better than $(1 - 1/e)$ unless P = NP [5, 6, 13].

In this paper, we study a geometric version of the problem, specifically the variant where the covering objects are axis-parallel unit hypercubes.

▶ **Definition 1** (k-Cover(P)). *Given a set P of n points in d-dimensional space, where each point is assigned a positive weight, find k axis-parallel unit hypercubes such that the total weight of points covered by their union is maximized.*

The problem k-Cover(P) is NP-hard when k is part of the input, even for $d = 2$ [10]. There are results for fixed and small k. For 1-Cover(P), Imai and Asano [7], and Nandy and Bhattacharya [12] gave exact algorithms which run in $O(n \log n)$ time. This is proven to

be optimal in the algebraic decision tree model [1]. For 2-Cover(P), Mahapatra et al. [9] gave an algorithm that runs in $O(n^2 \log^2 n)$ time using $O(n \log n)$ space. They also studied a variant where the solution pair of squares must not share a common input point and gave an $O(n^2)$-time algorithm. Cabello et al. [2] considered a more restricted version of 2-Cover(P) where the solution pair of squares must be disjoint and gave an $O(n \log n)$-time algorithm.

There are also approximation algorithms for this problem. For 1-Cover(P), Tao et al.[14] gave a randomized approximation scheme that computes a $(1-\varepsilon)$-approximate solution with probability at least $1 - 1/n$ and runs in $O(n \log(1/\varepsilon)) + n \log \log n)$ time. This was later improved by Jin et al.[8] with an algorithm with running time $O(n \log(1/\varepsilon))$.

For $k \geq 3$, de Berg et al. [4] gave an algorithm that can be extended to compute a $(1-\varepsilon)$-approximate solution to k-Cover(P) in $O(n(k/\varepsilon)^{O(\sqrt{k})})$ time. Jin et al. [8] gave an $(1-\varepsilon)$-approximation algorithm for k-Cover(P) which runs in $O((n/\varepsilon)\log(1/\varepsilon) + (k/\varepsilon)\log(1/\varepsilon) + k(1/\varepsilon)^{\Delta})$ time, where $\Delta = O(\min(\sqrt{k}, 1/\varepsilon))$.

There are work on the problem in higher dimensions $d \geq 3$. For $k = 1$, the problem reduces to the well-known maximum depth problem that, given a set of weighted hypercubes, finds a point maximizing the total weight of the hypercubes containing the point. Chan [3] gave an $O(n^{d/2})$-time algorithm for this problem when $d \geq 3$. For $k \geq 2$, however, little is known in higher dimensions ($d \geq 3$). Mostafavi and Hamzeh [11] studied the problem of approximating the smallest axis-parallel box that covers a given set of points in $\mathbb{R}^d$ while allowing z outliers, and gave a 2-approximation algorithm with running time $O(nd \log n)$.

1.1 Our results

We present both exact and approximation algorithms for 2-Cover(P). We present an $O(n^2)$-time algorithm for 2-Cover(P) in the plane by reducing the problem to the Depth problem (See Section 1.2). Our approach extends for any fixed $d, k > 2$ with little modification, resulting in an $O(n^{dk/2})$-time algorithm for k-Cover(P) in d-dimensional space. More importantly, we present a $(1-\varepsilon)$-approximation algorithm for 2-Cover(P) in the plane running in $O(n \log \min\{n, 1/\varepsilon\} + 1/\varepsilon^3)$ time. This improves upon the previous best result by Jin et al. [8], which runs in $O((n/\varepsilon)\log(1/\varepsilon) + (1/\varepsilon)^{O(1)})$ time, with big hidden constants. In contrast, our algorithm runs faster, is simple, and easy to implement. Our algorithm extends for any fixed $d \geq 3$ with little modification, running in $O((n/\varepsilon^{d-2})\log(1/\varepsilon) + 1/\varepsilon^{2d-1})$ time.

Hidden Constants in the running time of the algorithm in [8]. For $k = 2$, it takes $O((n/\varepsilon)\log(1/\varepsilon) + (1/\varepsilon)^{O(1)})$ time. It begins by constructing a grid G with cells of size $6/\varepsilon$. For each cell c of G, it first computes a $(1-\varepsilon)$-approximate solution to 1-Cover(P_c), where P_c denotes the set of points in P contained in c. Among all cells in G, it greedily selects two cells whose $(1-\varepsilon)$-approximate solution to 1-Cover(P_c) covers the maximum weight. To approximate 2-Cover within each cell, it constructs an additional finer grid consisting of $O(1/\varepsilon^3)$ horizontal and vertical lines, yielding $O(1/\varepsilon^6)$ grid points. Each grid point is considered as a potential candidate for the upper-left corner of a square. Furthermore, this process is repeated over $O(1/\varepsilon)$ different grids. Thus, the second term is $\Omega(1/\varepsilon^7)$ time.

1.2 Preliminaries

For simplicity, we omit the terms "unit" and "axis-parallel" when they are clear from the context. For a compact set X, we use $\mathsf{int}(X)$ to denote the interior of X.

We denote by P a set of positively weighted points in d-dimensional space, and by $w(P)$ the total weight of points in P. We say that a region R *covers* weight w if the total weight of the points in P contained in R is w. For any region R, we denote by $w(R)$ the total weight of points in P covered by R. Let $\mathsf{opt}(P)$ denote the total weight of points covered by an optimal pair of hypercubes to 2-Cover(P).

▶ **Definition 2** (k-Depth(R))**.** *Given a set R of n positively weighted boxes (hyperrectangles) in d-dimensional space, find k points in $\mathbb{R}^d$ that maximizes the sum of weights of the boxes in R containing at least one of the points. Let Depth(R) denote 1-Depth(R).*

2 Exact algorithm

We show that an instance of 2-Cover in the plane can be reduced to an instance of Depth in 4-dimensional space. This yields an $O(n^2)$-time algorithm for 2-Cover in the plane, improving upon the previously best-known $O(n^2 \log^2 n)$-time algorithm. We then generalize this approach to higher dimensions and larger k, reducing an instance of k-Cover in d-dimensional space to an instance of Depth in dk-dimensional space, which can be solved in $O(n^{dk/2})$ time for $d \geq 3$ [3].

2.1 Two squares in the plane

For a set P of n points in the plane, we give a reduction from 2-Cover(P) to an instance of Depth in 4-dimensional space. For a unit square s in the plane, we represent its location by the coordinates of its top-right corner, denoted by (s_x, s_y) (See Figure 1(a)). By representing a pair of unit squares (s, s') as a point in 4-dimensional space, (s_x, s_y, s'_x, s'_y), there is a one-to-one correspondence between a placement of a pair of squares and a point in 4-dimensional space.

A unit square s covers a point $p = (p_x, p_y)$ in the plane if and only if $p_x \leq s_x \leq p_x + 1$ and $p_y \leq s_y \leq p_y + 1$. Let $H(p)$ denote the set of points in 4-dimensional space that correspond to all placements of two unit squares containing p in their union. Then, $H(p) = H_1(p) \cup H_2(p)$ and $H_i(p) = \{(x_1, y_1, x_2, y_2) \mid p_x \leq x_i \leq p_x + 1, p_y \leq y_i \leq p_y + 1\}$ for $i = 1, 2$. See Figure 1.

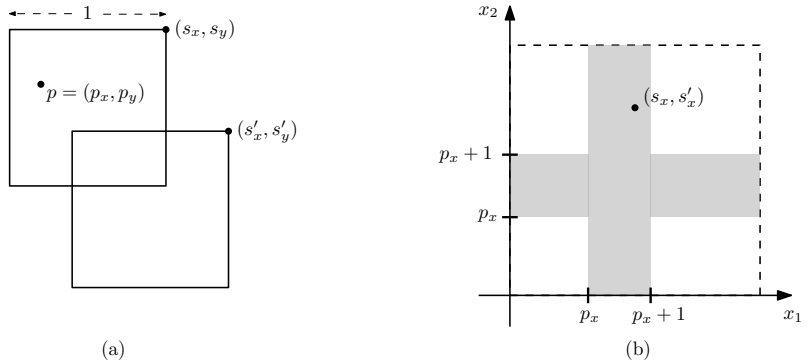


Figure 1 (a) A point $p = (p_x, p_y)$ and a pair of squares (s, s') whose union covers p. (b) A bounding box B is represented by dashed lines. For illustration, we show the projection of $H(p) \cap B$ onto the x_1x_2-plane (gray region). The pair of unit squares (s, s') corresponds to a point (s_x, s_y, s'_x, s'_y) in the 4-dimensional parameter space, and its projection is also shown.

By restricting the parameter space to a sufficiently large bounding box $B = [b_{\min}, b_{\max}]^4$, $H(p)$ can be represented as a union of interior-disjoint boxes in B. (Let $b_{\min}$ be the minimum coordinate value over all points in P and let $b_{\max}$ be the maximum coordinate value over all

points in P plus one.) Clearly, $H(p) \cap B$ can be represented as the union of $O(1)$ mutually interior-disjoint boxes in the 4-dimensional space. Let $B(p)$ denote this set of interior-disjoint boxes such that the weight of each box is set to the weight of p. Let T denote the collection of the boxes in $B(p)$ for all $p \in P$. We now show that an optimal point for $\mathsf{Depth}(T)$ corresponds to an optimal pair of squares for 2-$\mathsf{Cover}(P)$. Here, we slightly abuse the notation so that $\mathsf{opt}(T)$ denotes the total weight of boxes in T containing an optimal point of $\mathsf{Depth}(T)$.

▶ **Lemma 3.** *$opt(T) = opt(P)$.*

Proof. Let (s, s') be an optimal pair of squares for 2-$\mathsf{Cover}(P)$. This pair corresponds to the point $q_{s,s'} = (s_x, s_y, s'_x, s'_y)$ in the 4-dimensional parameter space. Let $P' \subseteq P$ be the set of points covered by the union of s and s', that is, $w(P') = \mathsf{opt}(P)$. By definition, we have $q_{s,s'} \in H(p)$ for every $p \in P'$. Thus, for each $p \in P'$, there is at least one box in $B(p)$ that contains $q_{s,s'}$, implying $\mathsf{opt}(T) \geq \mathsf{opt}(P)$.

Let $q_{r,r'} = (r_x, r_y, r'_x, r'_y)$ be an optimal point for $\mathsf{Depth}(T)$, and let $T' \subseteq T$ be the set of boxes in T that contain $q_{r,r'}$. (By a symbolic perturbation to the boxes, we can avoid double counting while preserving the maximum depth.) Then the total weight of points $p \in P$ such that $q_{r,r'} \in H(p)$ is exactly $\mathsf{opt}(T)$. This means the pair of squares (r, r') together cover the total weight $\mathsf{opt}(T)$ which is at most $\mathsf{opt}(P)$. ◀

As $|T| = O(n)$ and $\mathsf{Depth}(T)$ can be solved in $O(n^{d/2})$ time [3], we have the following.

▶ **Theorem 4.** *Given a set P of n positively weighted points in the plane, we can compute an optimal pair of unit squares for 2-$\mathsf{Cover}(P)$ in $O(n^2)$ time.*

2.2 Extension to larger k and higher dimensions

We can naturally extend this framework to handle k-Cover(P) for general $k \geq 3$. By representing the placement of k squares can also be specified by their top-right corners, it corresponds to a single point in $2k$-dimensional parameter space. There is a one-to-one correspondence between a point in this $2k$-dimensional space and a specific placement of the k squares in the plane. Then $H(p), H_i(p)$, and $B(p)$ can be defined for a point p analogously. The remaining procedure is exactly the same as in the case of $k = 2$.

▶ **Corollary 5.** *Given a set P of n positively weighted points in the plane, we can compute an optimal set of k unit squares for k-$\mathsf{Cover}(P)$ in $O(n^k)$ time.*

Our approach also extends to higher dimensions with little modification, except that the location of a hypercube, represented by one of its corners, now requires d coordinates. As a result, we obtain a set T of boxes in dk-dimensional space.

▶ **Corollary 6.** *Given a set P of n positively weighted points in d-dimensional space, we can compute an optimal set of k unit hypercubes for k-$\mathsf{Cover}(P)$ in $O(n^{dk/2})$ time.*

3 Approximation algorithm

Given a set P of n positively weighted points in the plane, we present a $(1-\varepsilon)$-approximation algorithm for 2-$\mathsf{Cover}(P)$ that runs in $O(n \log \min\{n, 1/\varepsilon\} + 1/\varepsilon^3)$ time for any fixed $\varepsilon > 0$. Specifically, it returns a pair of squares whose union covers a weight at least $(1-\varepsilon) \cdot \mathsf{opt}(P)$. Compared to the $(1-\varepsilon)$-approximation algorithm by Jin et al. [8], our algorithm runs faster asymptotically. Moreover, our algorithm is simple and easy to implement, and it can be extended to higher dimensions with little modifications.

3.1 Types of solutions

For any pair (s_1, s_2) of squares, either (1) $\mathsf{int}(s_1) \cap \mathsf{int}(s_2) = \emptyset$ or (2) $\mathsf{int}(s_1) \cap \mathsf{int}(s_2) \neq \emptyset$. See Figure 2 for an illustration. We solve two cases of the problem: Type (1) case finds two squares (s_1, s_2) with $\mathsf{int}(s_1) \cap \mathsf{int}(s_2) = \emptyset$ whose union covers the maximum weight. Type (2) case finds two squares (s_1, s_2) with $\mathsf{int}(s_1) \cap \mathsf{int}(s_2) \neq \emptyset$ whose union covers the maximum weight. An optimal solution to the type (1) case can be computed in $O(n \log n)$ time [2]. We show how to find a $(1-\varepsilon)$-approximate solution to the type (1) case for 2-Cover(P) in $O(n \log(1/\varepsilon))$ time.

▶ **Lemma 7.** *Given a set P of n positively weighted points in the plane, a $(1-\varepsilon)$-approximate solution to the type (1) case can be computed in $O(n \log(1/\varepsilon))$ time.*

Proof. We note that Jin et al. [8] presented an algorithm that computes a $(1-\varepsilon)$-approximate square for 1-Cover(P) in $O(n \log(1/\varepsilon))$ time. We adopt their grid-shifting technique for our problem. Let $G_3(a, b)$ denote the square grid with mesh size 3, where the vertical and horizontal grid lines are defined as:

$$G_3(a,b) = \left\{(x,y) \in \mathbb{R}^2 \mid x = a + 3k,\ k \in \mathbb{Z}\right\} \cup \left\{(x,y) \in \mathbb{R}^2 \mid y = b + 3k,\ k \in \mathbb{Z}\right\}.$$

Consider the following nine grids: $G_3(a, b)$ for all $a, b \in \{1, 2, 3\}$. It can be easily shown that for any two unit squares s_1 and s_2, there exists at least one of these grids such that both s_1 and s_2 do not intersect any grid lines Therefore, for every grid $G \in \{G_3(a, b) \mid a, b \in \{1, 2, 3\}\}$, we aim to compute a pair of disjoint unit squares that do not intersect any grid lines of G, and we return the pair that covers the maximum total weight.

For a grid $G \in \{G_3(a, b) \mid a, b \in \{1, 2, 3\}\}$, we perform the following preprocessing. We compute a $(1-\varepsilon)$-approximate square for 1-Cover(P_c) for every nonempty cell c of G where P_c denotes the set of points of P in c. We note that Jin et al. [8] give an algorithm, *MaxCovCell*(c), which computes a $(1-\varepsilon)$-approximate to 1-Cover(P_c) in $O(|P_c| \log(1/\varepsilon) + 1/\varepsilon^2)$ time for a cell c of G if the size of c is constant. For each cell of G which contains larger than $(1/\varepsilon)^2$ points, we run *MaxCovCell*. For the remaining nonempty cells, we run the exact algorithm which runs in $O(|P_c| \log |P_c|)$ time [2]. Let $n_1 \geq \ldots n_j \geq (1/\varepsilon^2) \geq n_{j+1} \geq \ldots \geq n_{j+k}$ be the sorted sequence of the number of points in nonempty cells of G. Then the total running time is $\sum_{i=1}^{j} O(n_i \log(1/\varepsilon) + (1/\varepsilon)^2) + \sum_{i=1}^{k} O(n_{i+j} \log(n_{i+j}))$, which is $O(n \log(1/\varepsilon))$.

To make use of the above results, we consider two cases based on whether there exists an optimal disjoint pair of squares for 2-Cover(P) such that the two squares are contained in different cells of G, or the two squares are contained in different cells of G.

Case 1. We first consider the case where there exists an optimal pair of squares of type (1) for 2-Cover(P) such that the two squares are contained in different cells of G. In this case, there always exist at least two cells of G whose respective $(1-\varepsilon)$-approximate solutions together cover at least the weight of $\varepsilon \cdot \mathsf{opt}(P)$. Therefore, among all cells of G, we select the pair of cells whose $(1-\varepsilon)$-approximate solutions together cover the maximum total weight. The union of these two $(1-\varepsilon)$-approximate solutions then forms a $(1-\varepsilon)$-approximate solution for 2-Cover(P).

Before moving onto the second case, we provide a summary of *MaxCovCell(c)*.

MaxCovCell(c). For a given cell c, they form another (non-uniform) grid G' which consists of $O(1/\varepsilon)$ horizontal lines and $O(1/\varepsilon)$ vertical lines such that the following property holds: For a unit square q, it holds that $w(r(q) \cap c) \geq \varepsilon \cdot w(q \cap c)$ where $r(q)$ denotes the largest axis-parallel rectangle contained in q whose corners all lie on grid points of G'. They construct such grid G' in $O(|P_c| \log(1/\varepsilon))$ time. The details of the method used to construct G' can be found in Lemmas 1 and 2 of [8].

For each cell c' of G', they calculate the sum of weights of points at the interior, edges, or corners of c' in $O(|P_c| \log(1/\varepsilon))$ time. Then they enumerate every vertex of G' and consider the unit square q which has its top-left corner on the vertex. While enumerating those vertices and corresponding unit squares q, they calculate $w(r(q) \cap c)$. Then the one that covers the maximum weight can be found in $O(1/\varepsilon^2)$ time with a standard incremental algorithm, and it is a $(1-\varepsilon)$-approximate solution to 1-Cover(P_c).

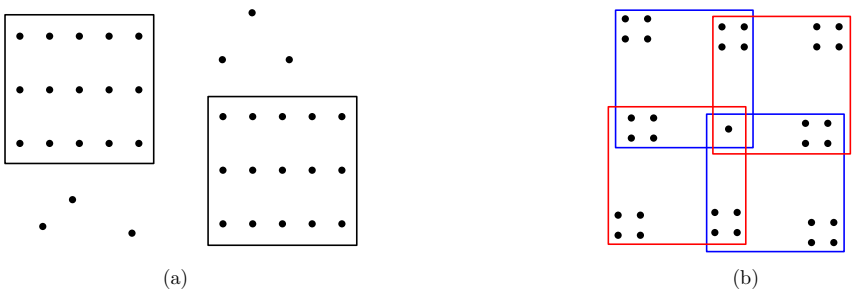
Case 2. Now we consider the case where an optimal pair of squares of type (1) for 2-Cover(P) is contained in the same cell of G. We can observe that once we compute a $(1-\varepsilon)$-approximate pair of squares to 2-Cover(P_c) for every cell c of G, and we can return the pair which covers the maximum total weight as a $(1-\varepsilon)$-approximate solution to 2-Cover(P).

We first note that there exists an $O(n \log n)$-time exact algorithm for 2-Cover(P) in the case where a disjoint optimal pair of squares exists [2]. For any cell c with $|P_c| \leq 1/\varepsilon^2$, we can directly apply this algorithm for 2-Cover(P_c), which takes $O(|P_c| \log |P_c|) = O(|P_c| \log(1/\varepsilon))$ time. Therefore, we now focus on the remaining cells c with $|P_c| > 1/\varepsilon^2$.

To compute a $(1-\varepsilon)$-approximate solution of type (1) for 2-Cover(P_c), we further utilize the information obtained from the procedure *MaxCovCell(c)*. Recall that we calculated $w(r(q) \cap c)$ for the squares q which has its top-left corner on the vertex of G'. Among the rectangle $r(q)$'s, we find a disjoint pair whose union covers the maximum total weight. This can be done using the observation that there always exists a horizontal or vertical line that separates the two disjoint squares. Without loss of generality, we can sweep a horizontal line across the plane and, at each position, track the maximum value of $w(r(q) \cap c)$ achievable by placing q on one side of the sweep line. We repeat this process by sweeping the line in both directions. By combining the results from both sweeps, we can identify two disjoint rectangles among the $r(q)$'s whose union covers the maximum total weight. We note that this procedure can be implemented as part of the $MaxCovCell(c)$. Then a pair of unit squares, each containing one of the rectangles, is a $(1-\varepsilon)$-approximate solution to 2-Cover(P_c).

Among all the cells c of G, the pair of squares for 2-Cover(P_c) that covers the maximum total weight is a $(1-\varepsilon)$-approximate solution to 2-Cover(P).

For each grid $G \in \{G_3(a,b) \mid a,b \in \{1,2,3\}\}$, we compute a $(1-\varepsilon)$-approximate solution for both cases in $O(n \log(1/\varepsilon))$ time and take the one that covers the larger total weight. Then we repeat this process for every grid $G \in \{G_3(a,b) \mid a,b \in \{1,2,3\}\}$, and we return the pair of squares that covers the maximum total weight. ◀



(a) (b)

Figure 2 Solutions for 2-Cover with unit-weighted points. (a) A solution for the type (1) case, which is optimal for 2-Cover. No optimal to the type (2) case is optimal to 2-Cover. (b) Two optimal solutions (red and blue) to the type (2) case, which are optimal for 2-Cover. No optimal to the type (1) case is optimal to 2-Cover.

If an optimal pair of squares for the type (1) case is an optimal pair of squares for 2-Cover(P), we can compute a $(1-\varepsilon)$-approximate solution for 2-Cover(P) in $O(n\log(1/\varepsilon))$ time by Lemma 7. From now on, we assume that every optimal pair of squares for 2-Cover(P) intersect each other in their interior, and we focus on computing a $(1-\varepsilon)$-approximate solution to the type (2) case. Once we obtain a $(1-\varepsilon)$-approximate solution to the type (2) case, we compare it with the approximate solution to the type (1) case and return the one with the larger total weight.

3.2 Search space of the type (2) case

We begin with reducing the search space for 2-Cover(P). We use s^* to denote an optimal square for 1-Cover(P).

▶ **Lemma 8.** *There is an optimal pair of squares for* 2-Cover(P) *such that both squares intersect* int(s^*).

Proof. Suppose that, for an optimal pair (s_1, s_2) for 2-Cover(P), s_2 does not intersect int(s^*). See Figure 3(a). Since s^* is an optimal square for 1-Cover(P), $w(s_1) \leq w(s^*)$. Thus, $w(s_1 \cup s_2) \leq w(s_1) + w(s_2) \leq w(s^*) + w(s_2)$, implying that (s^*, s_2) is an optimal pair of squares for 2-Cover(P). Observe that int(s^*) $\cap$ int(s_2) $= \emptyset$, which contradicts our assumption that every optimal pair of squares for 2-Cover(P) intersect each other in their interiors. ◀

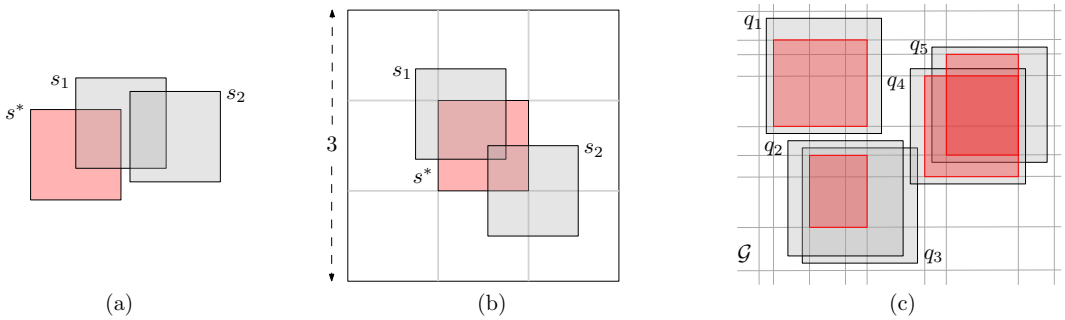


Figure 3 (a) An optimal square s^* for 1-Cover(P) and an optimal pair (s_1, s_2) for 2-Cover(P). (s^*, s_2) is also an optimal pair for 2-Cover(P). (b) 3×3 square centered at s^*. (c) A set $\mathsf{Q} = \{q_1, q_2, \ldots, q_5\}$ of squares and a grid $\mathcal{G}$ formed by L_v and L_h (gray lines). The rectangles in $\mathsf{R} = \{r(q) \mid q \in \mathsf{Q}\}$ are drawn in red. Note that $r(q_2) = r(q_3)$.

Lemma 8 immediately implies the existence of an optimal pair of squares for 2-Cover(P) that are contained in the 3×3 square centered at s^*. See Figure 3(b). We have a similar statement for a $(1-\varepsilon)$-approximate square $\tilde{s}$ for 1-Cover(P).

▶ **Lemma 9.** *There exists a* $(1-\varepsilon)$*-approximate pair of squares* (s_1, s_2) *for* 2-Cover(P) *such that* int(s_1) $\cap$ int($\tilde{s}$) $\neq \emptyset$ *or* int(s_2) $\cap$ int($\tilde{s}$) $\neq \emptyset$.

Proof. Suppose that neither s_1 nor s_2 intersects int($\tilde{s}$). Without loss of generality, assume that $w(s_1) \geq w(s_2)$. Then, $w(s_1) \geq (1-\varepsilon)\mathsf{opt}(P)/2$. Since $\tilde{s}$ is a $(1-\varepsilon)$-approximate square for 1-Cover(P), $2w(\tilde{s}) \geq (1-\varepsilon)\mathsf{opt}(P)$. Thus, $w(s_1 \cup \tilde{s}) = w(s_1) + w(\tilde{s}) \geq (1-\varepsilon)\mathsf{opt}(P)$, implying that $(\tilde{s}, s_1)$ is a $(1-\varepsilon)$-approximate pair of squares for 2-Cover(P). ◀

Recall that we compute a $(1-\varepsilon)$-approximate solution to the type (2) case. By Lemma 9, we have the following.

▶ **Lemma 10.** *There exists a $(1-\varepsilon)$-approximate pair (s_1, s_2) of squares for 2-Cover(P) such that both s_1, s_2 are contained in the 5×5 square centered at $\tilde{s}$.*

By Lemmas 8, 9, and 10, the search space for 2-Cover(P) is a 5×5 square centered at s^* or $\tilde{s}$. We compute either s^* or $\tilde{s}$ depending on the comparison between n and $1/\varepsilon$, identify the corresponding search space, and remove the points of P lying outside this search space. Since s^* can be computed in $O(n \log n)$ time [2] and $\tilde{s}$ can be computed in $O(n \log(1/\varepsilon))$ time by Lemma 7, this step can be completed in $O(n \log \min\{1/\varepsilon, n\})$ time.

From now on, we assume that every point of P lies in the 5×5 square. Clearly, there is a unit square that contains a subset of P whose total weight is at least $w(P)/25$.

▶ **Lemma 11.** *opt$(P) \geq w(P)/25$.*

3.3 Reduction to 2-Depth

Consider a dual mapping between a weighted point and a weighted square. For a point $p = (a, b)$ with weight w_p, the dual $\bar{p}$ of p is a unit square centered at (a, b) and with weight w_p. For a square s with weight w_s centered at (a, b), the dual $\bar{s}$ of s is the point (a, b) with weight w_s. Observe that s contains p if and only if $\bar{p}$ contains $\bar{s}$.

Let Q denote the set of weighted squares $\bar{p}$ for each $p \in P$. Throughout this section, for a square $q \in \mathsf{Q}$, we denote by $w(q)$ the weight q. For an optimal pair (p_1, p_2) of points for 2-Depth(Q), $(\bar{p}_1, \bar{p}_2)$ form an optimal pair of squares for 2-Cover(P). Thus, we transform P into Q, and solve 2-Depth(Q). We then return the pair of squares that is dual to the solution to 2-Depth(Q) as a solution for 2-Cover(P). Since we aims to compute a $(1-\varepsilon)$-approximate solution for 2-Cover(P), it suffices to compute a $(1-\varepsilon)$-approximate solution to 2-Depth(Q). To do this, we construct a grid $\mathcal{G}$ and snap round each rectangle in Q to $\mathcal{G}$.

Grid. We construct a grid $\mathcal{G}$ with $O(1/\varepsilon)$ horizontal and vertical lines with respect to Q using the following lemma, which is obtained by combining Lemmas 1 and 2 of [8].

▶ **Lemma 12** ([8])**.** *Given n positively weighted points with distinct x-coordinates in the plane whose weight sum is W, and a value w_d with $0 < w_d \leq W$, we can find $O(W/w_d)$ vertical lines such that the sum of the weights of points lying in the interior of the slab bounded by any two consecutive lines is at most w_d in time $O(n \log(W/w_d))$.*

Let V_{Q} denote the set of corners of the squares in Q. The weight of $v \in V_{\mathsf{Q}}$ is set to the weight of its corresponding square. Let L_v be the set of vertical lines obtained by applying Lemma 12 with $w_d = \varepsilon \cdot w(P)/50$. Let L_h be the set of horizontal lines defined in exactly the same way. Let $\mathcal{G}$ denote the grid formed by L_v and L_h. By Lemma 12, $\mathcal{G}$ can be constructed in $O(n \log(\min\{n, 1/\varepsilon\}))$ time. In particular, when $n < 1/\varepsilon$, we can simply draw horizontal and vertical lines through every point in V_{Q}.

Snap rounding. For each square $q \in \mathsf{Q}$, let $r(q)$ be the largest axis-parallel rectangle contained in q whose corners all lie on grid points of $\mathcal{G}$. Let $\mathsf{R} = \{r(q) \mid q \in \mathsf{Q}\}$. The weight of $r(q)$ is set to the weight of q. If two or more squares in Q have the same rectangle snap-rounded in $\mathcal{G}$, R contains exactly one snap-rounded rectangle for those squares and the weight of the rectangle is set to the sum of the weights of those squares. See Figure 3(c). For a rectangle $r \in \mathsf{R}$, we denote by $w(r)$ the weight of r. Let opt(R) and opt(Q) denote the total weights obtained by 2-Depth(R) and 2-Depth(Q), respectively.

▶ **Lemma 13.** *opt$(R) \geq (1-\varepsilon) \cdot$ opt(Q).*

Proof. For a point $p \in \mathbb{R}^2$, let $\mathsf{Q}(p) = \{q \in \mathsf{Q} \mid p \in q\}$. Note that there may exist squares $q \in \mathsf{Q}(p)$ with $p \notin r(q)$. Let $\mathsf{Q}^c(p) = \{q \in \mathsf{Q}(p) \mid p \notin r(q)\}$.

Let (p_1, p_2) be an optimal pair of points for 2-$\mathsf{Depth}(\mathsf{Q})$. Assume that p_1 lies in the interior of a vertical slab Γ_v bounded by two consecutive lines in L_v. Also, assume that p_1 lies in the interior of a horizontal slab Γ_h bounded by two consecutive horizontal lines in L_h. Every square in $\mathsf{Q}^c(p_1)$ must have at least two corners lying in $\mathsf{int}(\Gamma_v)$ or at least two corners in $\mathsf{int}(\Gamma_h)$. Since the total weight of the corners in V_{Q} contained in $\mathsf{int}(\Gamma_v)$ or $\mathsf{int}(\Gamma_h)$ is at most $\varepsilon \cdot w(P)/25$, $w(\mathsf{Q}^c(p_1)) \leq \varepsilon \cdot w(P)/50$. Since $\mathsf{opt}(P) = \mathsf{opt}(\mathsf{Q}) \geq w(P)/25$ by Lemma 11, $w(\mathsf{Q}^c(p_1)) \leq \varepsilon \cdot \mathsf{opt}(\mathsf{Q})/2$. The same argument applies to p_2, implying $w(\mathsf{Q}^c(p_2)) \leq \varepsilon \cdot \mathsf{opt}(\mathsf{Q})/2$.

By defining $\mathsf{R}(p)$ analogously to $\mathsf{Q}(p)$, we have $w(\mathsf{R}(p_1) \cup \mathsf{R}(p_2)) \geq w(\mathsf{Q}(p_1) \cup \mathsf{Q}(p_2)) - w(\mathsf{Q}^c(p_1)) - w(\mathsf{Q}^c(p_2))$. Since $w(\mathsf{Q}^c(p_1)) + w(\mathsf{Q}^c(p_2)) \leq \varepsilon \cdot \mathsf{opt}(\mathsf{Q})$ and $w(\mathsf{Q}(p_1) \cup \mathsf{Q}(p_2)) = \mathsf{opt}(\mathsf{Q})$, we have $\mathsf{opt}(\mathsf{R}) \geq (1-\varepsilon) \cdot \mathsf{opt}(\mathsf{Q})$. ◀

For a rectangle $r = [x_1, x_2] \times [y_1, y_2]$, let $x(r) = [x_1, x_2]$ and let $y(r) = [y_1, y_2]$. Let $\mathcal{I}_x = \{x(r) \mid r \in \mathsf{R}\}$ and $\mathcal{I}_y = \{y(r) \mid r \in \mathsf{R}\}$.

▶ **Lemma 14.** *For any two elements $I = [x_1, x_2]$ and $I' = [x'_1, x'_2]$ in $\mathcal{I}_x$, we have $x_2 \leq x'_2$ if $x_1 < x'_1$, and $x_1 \leq x'_1$ if $x_2 < x'_2$. The same property holds for $\mathcal{I}_y$.*

Proof. Let q and q' be the squares in Q with $x(r(q)) = I$ and $x(r(q')) = I'$. Let $x(q) = [a_1, a_2]$ and $x(q') = [a'_1, a'_2]$. Since $x_1 < x'_1$, and q, q' are unit squares, we have $a_1 < a'_1$, and thus $a_2 < a'_2$. This implies that $x_2 \leq x'_2$ by the snap rounding method. The same can be shown for the other cases analogously. ◀

By Lemma 14, we obtain the following corollary. Note that $\mathsf{R} \subseteq \{\alpha \times \beta \mid \alpha \in \mathcal{I}_x, \beta \in \mathcal{I}_y\}$.

▶ **Corollary 15.** *Both $|\mathcal{I}_x|$ and $|\mathcal{I}_y|$ are $O(1/\varepsilon)$. $|R| = O(1/\varepsilon^2)$.*

Recall that $\mathcal{G}$ can be constructed in $O(n \log \min\{n, 1/\varepsilon\})$ time. After sorting the grid lines, we can use binary search to snap round every square in Q in $O(n \log \min\{n, 1/\varepsilon\})$ total time.

▶ **Theorem 16.** *Given a set P of n positively weighted points in the plane, we can compute in $O(n \log \min\{n, 1/\varepsilon\})$ time a set R of positively weighted rectangles in the plane such that $|R| = O(1/\varepsilon^2)$ and the dual of an optimal solution for 2-$\mathsf{Depth}(R)$ is a $(1-\varepsilon)$-approximate solution for 2-$\mathsf{Cover}(P)$.*

3.4 Algorithm

Let R be the set of $O(1/\varepsilon^2)$ positively weighted rectangles obtained by Theorem 16. Recall that each corner of a rectangle in R lies on a grid point of $\mathcal{G}$. As defined in Section 3.3, $\mathcal{G}$ is formed by a set L_v of $O(1/\varepsilon)$ vertical lines and a set L_h of $O(1/\varepsilon)$ horizontal lines. Let $L_v = \{\ell_1, \ell_2, \ldots, \ell_t\}$. For each i from 1 to $t-1$, ℓ_i and ℓ_{i+1}, form a vertical slab which we denote by Γ_i (See Figure 4(a)). Then there is a pair of index (i, j) such that 2-$\mathsf{Depth}(\mathsf{R})$ has an optimal pair of points (p_1, p_2) with $p_1 \in \Gamma_i$ and $p_2 \in \Gamma_j$. For every possible pair of indices (i, j) for $1 \leq i \leq j < t$, we compute an optimal pair of points, (p_1, p_2), for 2-$\mathsf{Depth}(\mathsf{R})$ while restricting that $p_1 \in \Gamma_i$ and $p_2 \in \Gamma_j$. Among all the pairs of points, the one (p_1, p_2) that maximizes the sum of weights of the rectangles in R containing p_1 or p_2 is an optimal solution for 2-$\mathsf{Depth}(\mathsf{R})$.

For two indices $1 \leq i \leq j < t$, we denote by 2-$\mathsf{Depth}(\Gamma_i, \Gamma_j)$, or simply 2-$\mathsf{Depth}(i, j)$, the case of 2-$\mathsf{Depth}(\mathsf{R})$ in which we seek a pair of points (p_1, p_2) maximizing the total weight of rectangles in R containing at least one of them, subject to the constraint that $p_1 \in \Gamma_i$ and $p_2 \in \Gamma_j$.

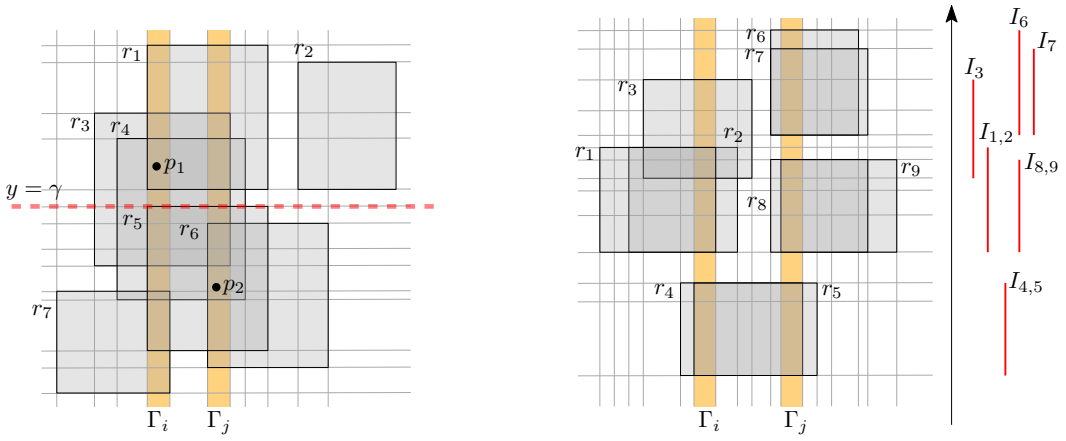


Figure 4 (a) $\mathsf{R}_{i\setminus j} = \{r_7\}$, $\mathsf{R}_{j\setminus i} = \{r_6\}$, $\mathsf{R}_{i,j} = \{r_1, r_3, r_4, r_5\}$. (p_1, p_2) for 2-$\mathsf{Depth}(\mathsf{R})$ with total weight of 5 for unit weight rectangles. $\mathsf{R}_{i,j}(\gamma) = \{r_1, r_3, r_4\}$, $\mathsf{R}^c_{i,j}(\gamma) = \{r_5\}$. $\mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma)) = 3$, $\mathsf{Depth}(\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma)) = 2$. (b) $\mathsf{R}^\ell = [r_1, r_2, r_3, r_4, r_5, r_8, r_7, r_6, r_9]$, $\mathsf{R}^r = [r_1, r_2, r_3, r_4, r_5, r_6, r_8, r_7, r_9]$, $\mathsf{R}^t = [r_4, r_5, r_8, r_9, r_1, r_2, r_3, r_7, r_6]$, $\mathsf{I}_{i\setminus j} = [I_{1,2}, I_3]$, $\mathsf{I}_{j\setminus i} = [I_{8,9}, I_7, I_6]$, $\mathsf{I}_{i,j} = [I_{4,5}]$.

Without loss of generality, we assume that 2-$\mathsf{Depth}(i, j)$ has an optimal pair $(p_1^\star, p_2^\star)$ such that $p_1^\star$ lies to the left of and above $p_2^\star$. The remaining cases can be handled analogously. For ease of description, we assume that $p_1^\star$ and $p_2^\star$ lie in $\mathsf{int}(\Gamma_i)$ and $\mathsf{int}(\Gamma_j)$, respectively. The cases where either p_1 or p_2 lies on the boundary can be handled in a similar manner.

For a rectangle r, let r_y denote the y-coordinate of its top side. For a set H of rectangles and a real value α, let $\mathsf{H}(\alpha) = \{r \in \mathsf{H} \mid \alpha < r_y\}$ and $\mathsf{H}^c(\alpha) = \mathsf{H} \setminus \mathsf{H}(\alpha)$. For a set H of weighted rectangles, let $\mathcal{I}_y(\mathsf{H}) = \{I_y(r) \mid r \in \mathsf{H}\}$. We abuse the notation and use $\mathsf{Depth}(\mathsf{H})$ to denote the total weight of rectangles in H hit by an optimal point for $\mathsf{Depth}(\mathsf{H})$. See Figure 4(a) for the following definition.

▶ **Definition 17.** *Let $R_i = \{r \in R \mid r \cap \mathit{int}(\Gamma_i) \neq \emptyset\}$ for $1 \leq i < t$. For two indices i and j, $1 \leq i < j < t$, let $R_{i\setminus j} = R_i \setminus R_j$, $R_{j\setminus i} = R_j \setminus R_i$, and $R_{i,j} = R_i \cap R_j$. For $i = j$, let $R_{i\setminus i} = \emptyset$ and $R_{i,i} = R_i$.*

For a point $p \in \mathbb{R}^2$, let $x(p)$ and $y(p)$ denote its x and y-coordinates.

▶ **Lemma 18.** *For any fixed i, j with $1 \leq i \leq j < t$, let $\mathsf{opt}(i, j)$ be the total weight of rectangles in R hit by an optimal solution for 2-$\mathit{Depth}(i, j)$. If there is an optimal pair of points (p_1, p_2) for 2-$\mathit{Depth}(i, j)$ with $p_1 \in \Gamma_i, p_2 \in \Gamma_j$ and $y(p_1) > y(p_2)$, there is a real value γ satisfying $\mathit{Depth}(R_{i\setminus j} \cup R_{i,j}(\gamma)) + \mathit{Depth}(R_{j\setminus i} \cup R^c_{i,j}(\gamma)) = \mathsf{opt}(i, j)$.*

Proof. Let $\mathsf{Q}_1 \subset \mathsf{R}$ be the set of rectangles hit by p_1. Let $\mathsf{Q}_2 \subset \mathsf{R}$ be the set of rectangles hit by p_2 but not by p_1. Note that $w(\mathsf{Q}_1) + w(\mathsf{Q}_2) = \mathsf{opt}(i, j)$ as $\mathsf{Q}_1 \cap \mathsf{Q}_2 = \emptyset$.

Let $\gamma = \max\{r_y \mid r \in (\mathsf{Q}_2 \cap \mathsf{R}_{i,j})\}$. See Figure 4(a). Observe that $y(p_2) \leq \gamma < y(p_1)$. We will prove that $w(\mathsf{Q}_1) = \mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$ and $w(\mathsf{Q}_2) = \mathsf{Depth}(\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma))$.

Every rectangle $r \in \mathsf{Q}_1$ has $y(p_1) \leq r_y$ by the definition. Every rectangle in Q_1 is from either $\mathsf{R}_{i\setminus j}$ or $\mathsf{R}_{i,j}$. (Note that $\mathsf{R}_{i\setminus j}$ and $\mathsf{R}_{i,j}$ are disjoint.) For a rectangle of the latter case, that is $r \in \mathsf{Q}_1 \cap \mathsf{R}_{i,j}$, observe that $r \in \mathsf{R}_{i,j}(\gamma)$ as we have $\gamma < y(p_1) \leq r_y$. Therefore $\mathsf{Q}_1 \subset (\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$.

Every rectangle in Q_2 is from either $\mathsf{R}_{j\setminus i}$ or $\mathsf{R}_{i,j}$. (Note that $\mathsf{R}_{j\setminus i}$ and $\mathsf{R}_{i,j}$ are disjoint.) For a rectangle of the latter case, that is $r \in \mathsf{Q}_2 \cap \mathsf{R}_{i,j}$, observe that $r_y \leq \gamma$ by the definition of γ. Therefore such r is contained in $\mathsf{R}^c_{i,j}(\gamma)$. Therefore $\mathsf{Q}_2 \subset (\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma))$.

Now we first show that $w(\mathsf{Q}_1) = \mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$. As $\mathsf{Q}_1 \subset (\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$, it is clear that $w(\mathsf{Q}_1) \leq \mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$. Now suppose $w(\mathsf{Q}_1) < \mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$. As every rectangle in $(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$ intersects $\mathsf{int}(\Gamma_i)$, there always exists an optimal point p' for $\mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$ such that $p' \in \Gamma_i$. Let R' be the set of rectangles in $(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$ hit by p', that is $w(\mathsf{R}') = \mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$. Observe that $\mathsf{R}' \cap \mathsf{Q}_2 = \emptyset$ because we have $\mathsf{R}' \subset (\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$, $\mathsf{Q}_2 \subset (\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma))$, and $(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma)) \cap (\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma)) = \emptyset$. Then the total weight of rectangles in R hit by a new pair (p', p_2) is $w(\mathsf{R}')+w(\mathsf{Q}_2) > w(\mathsf{Q}_1)+w(\mathsf{Q}_2)$, which contradicts that (p_1, p_2) is an optimal pair of points for 2-$\mathsf{Depth}(i, j)$.

In a similar way, we can show that $w(\mathsf{Q}_2) = \mathsf{Depth}(\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma))$. ◀

3.4.1 Data Structures and Invariants

Given an input set R of weighted rectangles, we store them in three lists. The first list, R^ℓ, stores the rectangles in increasing order of the x-coordinates of their left sides. The second list, R^r, stores the rectangles in increasing order of the x-coordinates of their right sides. In the case of a tie, the rectangles are further sorted by increasing order of the y-coordinates of their top sides. If the tie still remains unsolved, we sort them by increasing order of the y-coordinates of their bottom sides. See Figure 4(b). The third list, R^t, stores the rectangles in increasing order of the y-coordinates of their top sides. In the case of a tie, the rectangles are further sorted by increasing order of the y-coordinates of their bottom sides. If the tie still remains unsolved, we sort them by increasing order of the x-coordinates of their right sides.

We compute R^ℓ and R^r once at the beginning of the algorithm. Since $|\mathsf{R}| = O(1/\varepsilon^2)$ by Corollary 15, this can be done in $O((1/\varepsilon^2)\log(1/\varepsilon))$ time. For any fixed indices i and j with $1 \leq i \leq j < t$, recall that $\mathcal{I}_y(\mathsf{R}_{i\setminus j})$ denotes the set of weighted intervals induced by $\mathsf{R}_{i\setminus j}$ in the y-direction. By Lemma 15, there are $O(1/\varepsilon)$ intervals in $\mathcal{I}_y(\mathsf{R}_{i\setminus j})$.

Note that two or more rectangles in $\mathsf{R}_{i\setminus j}$ may have the same interval, ignoring their weights. In such a case, we merge them into a single interval. The weight of the merged interval is set to the total weight of those intervals. Let $\mathsf{I}_{i\setminus j}$ denote the resulting set of intervals, which satisfies $|\mathsf{I}_{i\setminus j}| = O(1/\varepsilon)$. In the same way, we define $\mathsf{I}_{j\setminus i}$ and $\mathsf{I}_{i,j}$ with respect to $\mathcal{I}_y(\mathsf{R}_{j\setminus i})$ and $\mathcal{I}_y(\mathsf{R}_{i,j})$, respectively. We always maintain the lists $\mathsf{I}_{i\setminus j}$, $\mathsf{I}_{j\setminus i}$, and $\mathsf{I}_{i,j}$ in increasing order of the right endpoints of the intervals. In the case of a tie, the intervals are further sorted by their left endpoints.

3.4.2 Algorithm for fixed indices i, j

We consider two fixed indices i and j with $1 \leq i \leq j < t$, and present an algorithm that finds an optimal pair of points for 2-$\mathsf{Depth}(i, j)$. For now, we assume that the sorted lists $\mathsf{I}_{i\setminus j}$, $\mathsf{I}_{j\setminus i}$, and $\mathsf{I}_{i,j}$ are given. We will later show how to compute them efficiently. By Lemma 18, there is a real value γ such that an optimal point $p_1 \in \Gamma_i$ for $\mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$, and an optimal point $p_2 \in \Gamma_j$ for $\mathsf{Depth}(\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma))$ together form an optimal pair (p_1, p_2) for 2-$\mathsf{Depth}(i, j)$. Thus, our goal is to find such γ and the corresponding optimal pair (p_1, p_2).

Since every rectangle in $\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma)$ intersects $\mathsf{int}(\Gamma_i)$, $\mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$ can be reduced to the one-dimensional problem $\mathsf{Depth}(\mathcal{I}_y(\mathsf{R}_{i\setminus j}) \cup \mathcal{I}_y(\mathsf{R}_{i,j}(\gamma)))$. Recall that for any fixed point p on the real line, the total weight of intervals in $\mathcal{I}_y(\mathsf{R}_{i\setminus j}) \cup \mathcal{I}_y(\mathsf{R}_{i,j}(\gamma))$ hit by p is the same as the total weight of intervals in $\mathsf{I}_{i\setminus j} \cup \mathsf{I}_{i,j}(\gamma)$ hit by p. (Here, we slightly abuse the notation: For a list I of intervals on the real line and a real value α, let $\mathsf{I}(\alpha)$ denote the set of intervals in I whose right endpoints are larger than α, and let $\mathsf{I}^c(\alpha)$ denote the remaining intervals in I. Let $\mathsf{I} \cup \mathsf{I}'$ denotes the set consisting of all intervals from I and I'.) Therefore, for any fixed γ, it suffices to compute $\mathsf{Depth}(\mathsf{I}_{i\setminus j} \cup \mathsf{I}_{i,j}(\gamma))$. Similarly, $\mathsf{Depth}(\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma))$ is reduced to $\mathsf{Depth}(\mathsf{I}_{j\setminus i} \cup \mathsf{I}^c_{i,j}(\gamma))$.

Thus, our objective is to find a real value γ that maximizes $\mathsf{Depth}(\mathsf{I}_{i\setminus j} \cup \mathsf{I}_{i,j}(\gamma)) + \mathsf{Depth}(\mathsf{I}_{j\setminus i} \cup \mathsf{I}^c_{i,j}(\gamma))$. Imagine that γ decreases from ∞. Then $\mathsf{Depth}(\mathsf{I}_{i\setminus j} \cup \mathsf{I}_{i,j}(\gamma))$ does not decrease, and we can easily maintain this using a simple incremental update since the intervals in $\mathsf{I}_{i\setminus j} \cup \mathsf{I}_{i,j}$ satisfy the property described in Lemma 14. For $\mathsf{Depth}(\mathsf{I}_{j\setminus i} \cup \mathsf{I}^c_{i,j}(\gamma))$, we increase γ from $-\infty$, and the update can be done in a similar manner. Since all the lists are already sorted, this entire process takes linear time, and we can find the value of γ that maximizes $\mathsf{Depth}(\mathsf{I}_{i\setminus j} \cup \mathsf{I}_{i,j}(\gamma)) + \mathsf{Depth}(\mathsf{I}_{j\setminus i} \cup \mathsf{I}^c_{i,j}(\gamma))$. As there are $O(1/\varepsilon)$ intervals in $\mathsf{I}_{i\setminus j}$, $\mathsf{I}_{j\setminus i}$, and $\mathsf{I}_{i,j}$, we have the following lemma.

▶ **Lemma 19.** *For any fixed i and j with $1 \le i \le j < t$, once we have $\mathsf{I}_{i\setminus j}$, $\mathsf{I}_{j\setminus i}$, and $\mathsf{I}_{i,j}$, we can solve 2-$\mathsf{Depth}(i,j)$ in $O(1/\varepsilon)$ time.*

3.4.3 Final algorithm

For any fixed i with $1 \le i < t$, we show how to compute 2-$\mathsf{Depth}(i,j)$ for every $j = i, i+1, \ldots, t-1$ in total time $O((1/\varepsilon^2)\log(1/\varepsilon))$. Since $t = O(1/\varepsilon)$, this can be done by running the algorithm from Lemma 19 for each j in the range $(t-i+1) = O(1/\varepsilon)$, resulting in a $O((1/\varepsilon^2)\log(1/\varepsilon))$ time in total.

However, it remains to show that the lists $\mathsf{I}_{i\setminus j}$, $\mathsf{I}_{j\setminus i}$, and $\mathsf{I}_{i,j}$ are properly updated as we enumerate j from i to $t-1$. To this end, we introduce new notations S_i and T_i, and provide inductive relations for $\mathsf{R}_{i\setminus j}$, $\mathsf{R}_{j\setminus i}$, and $\mathsf{R}_{i,j}$.

▶ **Definition 20.** *For an index $1 \le i < t$, let $R_i = \{r \in R \mid r \cap \mathit{int}(\Gamma_i) \neq \emptyset\}$. Let $R_0 = R_t = \emptyset$. We then define $S_i = R_i \setminus R_{i-1}$ and $T_i = R_i \setminus R_{i+1}$ for $1 \le i < t$.*

For any fixed i with $1 \le i < t$, the rectangles of S_i appear as a contiguous sublist in R^ℓ, and they appear before those of S_{i+1} in R^ℓ. Symmetrically, the rectangles of T_i appear as a contiguous sublist in R^r, and they appear before those of T_{i+1} in R^r.

▶ **Observation 21.** *For i and j with $1 \le i \le j < t$, $R_{i\setminus j+1} = R_{i\setminus j} \cup (R_{i,j} \cap T_j)$, $R_{j+1\setminus i} = (R_{j\setminus i} \setminus T_j) \cup S_{j+1}$, and $R_{i,j+1} = R_{i,j} \setminus T_j$.*

Using the inductive relation above, we prove the following lemma.

▶ **Lemma 22.** *For any fixed i with $1 \le i < t$, the lists $\mathsf{I}_{i\setminus j}$, $\mathsf{I}_{j\setminus i}$, and $\mathsf{I}_{i,j}$ for all $j = i, \ldots, t-1$ can be computed in time $O(1/\varepsilon^2)$ in total.*

Proof. For the base case where $i = j$, recall that $\mathsf{R}_{i\setminus i} = \emptyset$ and $\mathsf{R}_{i,i} = \{r \in \mathsf{R} \mid r \cap \Gamma_i \neq \emptyset\}$. Thus, we initialize $\mathsf{I}_{i\setminus j}$ and $\mathsf{I}_{j\setminus i}$ as empty lists. For $\mathsf{I}_{i,i}$, we find the rectangles in $\mathsf{R}_{i,i}$ and store them in increasing order of the y-coordinates of their top sides. This can be done by scanning the list R^t. Note that two or more rectangles in $\mathsf{R}_{i\setminus j}$ may have the same interval, ignoring their weights. In such a case, we merge them into a single interval. The weight of the merged interval is set to the total weight of those intervals. This process takes $O(|\mathsf{R}^t|) = O(1/\varepsilon^2)$ time, producing the correct $\mathsf{I}_{i,j}$.

We maintain two pointers, one each for R^ℓ and R^r. As j increases, the pointer in R^ℓ moves to the first element contained in S_{j+1}. and the pointer in R^r moves to the first element contained in T_j.

Assuming we already have $\mathsf{I}_{i\setminus j}$, by Observation 21, we need to find rectangles in $(\mathsf{R}_{i,j} \cap \mathsf{T}_j)$ to obtain $\mathsf{I}_{i\setminus j+1}$. This can be done by scanning the elements in R^r starting from its pointer, which takes $O(|\mathsf{T}_j|)$ time. Note that these rectangles share the same right-side x-coordinate and are sorted by the y-coordinates of their top sides. Hence, we can convert them into intervals, and merge the same intervals (ignoring their weights) as done in the previous

paragraph. This also takes $O(|\mathsf{T}_j|)$ time. By Lemma 15, the resulting set of intervals, say $\mathsf{I}_{\mathsf{new}}$, has size $O(1/\varepsilon)$. We then merge $\mathsf{I}_{\mathsf{new}}$ into $\mathsf{I}_{i\setminus j}$. We merge the same intervals as done in the previous paragraph if necessary. Since both $\mathsf{I}_{i\setminus j}$ and $\mathsf{I}_{\mathsf{new}}$ are sorted by their right endpoints, this merge can be done in $O(|\mathsf{I}_{i\setminus j}| + |\mathsf{I}_{\mathsf{new}}|) = O(1/\varepsilon)$ time.

Similarly, we need to find the rectangles in $(\mathsf{R}_{j\setminus i} \cap \mathsf{T}_j) \cup \mathsf{S}_{j+1}$ to obtain $\mathsf{I}_{j+1\setminus i}$ from $\mathsf{I}_{j\setminus i}$. The rectangles in $(\mathsf{R}_{j\setminus i} \cap \mathsf{T}_j)$ can be found by scanning the elements in R^r, as before. We then convert these rectangles into intervals and merge the same intervals, which can be done in $O(|\mathsf{T}_j|)$ time. After that, we update $\mathsf{I}_{j\setminus i}$ by removing or decreasing the weights of the corresponding intervals. In the same way, we find the rectangles in S_{j+1} by scanning the elements in R^ℓ from its pointer, convert them into intervals, and process them similarly. Therefore, the total time required is $O(|\mathsf{T}_j| + |\mathsf{S}_{j+1}| + 1/\varepsilon)$. The update for $\mathsf{I}_{i,j+1}$ can be handled in the same manner.

It takes $O(|\mathsf{T}_j|+1/\varepsilon)$ time for computing $\mathsf{I}_{i\setminus j+1}$, $O(|\mathsf{T}_j|+|\mathsf{S}_{j+1}|+1/\varepsilon)$ time for computing $\mathsf{I}_{j+1\setminus i}$, and $O(|\mathsf{T}_j| + 1/\varepsilon)$ time from computing $\mathsf{I}_{i,j+1}$. Summing over $j = i$ to $t-1$, the total time is $\sum_{j=i}^{t-1} O\big(|\mathsf{S}_{j+1}| + |\mathsf{T}_j| + 1/\varepsilon\big) = O(|\mathsf{R}|) + O(1/\varepsilon^2) = O(1/\varepsilon^2)$. ◀

By combining Lemmas 19 and 22, we can conclude that for any fixed index i with $1 \le i < t$, we can compute 2-$\mathsf{Depth}(i, j)$ for all $j = i, \ldots, t-1$ in $O(1/\varepsilon^2)$ total time. Repeating this process for all $i = 1, \ldots, t-1$ takes $O(1/\varepsilon^3)$. Since it takes $O(n \log \min\{n, 1/\varepsilon\})$ time to compute the set R by Theorem 16, we obtain the following result.

▶ **Theorem 23.** *Given a set P of n positively weighted points in the plane, we can compute a $(1-\varepsilon)$-approximate pair of squares for 2-$\mathsf{Cover}(P)$ in $O(n \log \min\{n, 1/\varepsilon\} + 1/\varepsilon^3)$ time.*

3.5 Extension to higher dimensions

Our algorithm extends naturally to higher dimensions. For any fixed $d \ge 3$, an optimal pair to the type (1) case can be computed in $O((n/\varepsilon^{d-2}) \log(1/\varepsilon))$ time by extending the algorithm of Lemma 7.

Consider the type (2) case. As in Section 3.2, the search space can be reduced to a constant-size d-dimensional box, and the same duality transform can also be applied. By applying a lemma analogous to Lemma 12, we can construct a grid such that there are $O(1/\varepsilon)$ grid hyperplanes along each axis x_i for $i = 1, \ldots, d$. After performing snap rounding for each dual hypercube into hyperrectangle, the algorithm proceeds in exactly the same manner as in the two-dimensional case.

Let $\Gamma_1, \ldots, \Gamma_t$ denote the slabs formed by two consecutive grid hyperplanes along the x_1-axis. For any fixed $1 \le i \le j \le t$, we solve 2-$\mathsf{Depth}(i, j)$. We can observe that once i and j are fixed, all the hyperrectangles of our interest intersect either Γ_i or Γ_j. We then project the problem onto the plane orthogonal to the x_1-axis and recursively solve the lower-dimensional instance.

Thus, we obtain the following theorem.

▶ **Theorem 24.** *Given a set P of n positively weighted points in d-dimensional space, we can compute a $(1-\varepsilon)$-approximate pair of hypercubes for 2-$\mathsf{Cover}(P)$ in $O((n/\varepsilon^{d-2}) \log(1/\varepsilon) + 1/\varepsilon^{2d-1})$ time for any fixed $d \ge 3$.*

References

1 Michael Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of 15th Annual ACM Symposium on Theory of Computing*, pages 80–86, New York, NY, USA, 1983. Association for Computing Machinery.

2 Sergio Cabello, J. Miguel Díaz-Báñez, Carlos Seara, J. Antoni Sellès, Jorge Urrutia, and Inmaculada Ventura. Covering point sets with two disjoint disks or squares. *Computational Geometry*, 40(3):195–206, 2008. doi:10.1016/J.COMGEO.2007.10.001.

3 Timothy M. Chan. Klee's measure problem made easy. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 410–419, 2013. doi:10.1109/FOCS.2013.51.

4 Mark de Berg, Sergio Cabello, and Sariel Har-Peled. Covering many or few points with unit disks. *Theory of Computing Systems*, 45(3):446–469, 2009. doi:10.1007/S00224-008-9135-9.

5 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

6 Dorit S. Hochbaum and Anu Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics*, 45(6):615–627, 1998.

7 Hiroshi Imai and Takao Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of Algorithms*, 4(4):310–323, 1983. doi:10.1016/0196-6774(83)90012-3.

8 Kai Jin, Jian Li, Haitao Wang, Bowei Zhang, and Ningye Zhang. Near-linear time approximation schemes for geometric maximum coverage. *Theoretical Computer Science*, 725:64–78, 2018. doi:10.1016/J.TCS.2017.11.026.

9 Priya Ranjan Sinha Mahapatra, Partha P. Goswami, and Sandip Das. Placing two axis-parallel squares to maximize the number of enclosed points. *International Journal of Computational Geometry & Applications*, 25(04):263–282, 2015. doi:10.1142/S0218195915500156.

10 Nimrod Megiddo and Kenneth J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, 1984. doi:10.1137/0213014.

11 Ali Mostafavi and Ali Hamzeh. High-dimensional axis-aligned bounding box with outliers. In *Proceedings of 34th Canadian Conference on Computational Geometry*, pages 264–269, 2022.

12 Subhas C. Nandy and Bhargab B. Bhattacharya. A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Computers & Mathematics with Applications*, 29(8):45–61, 1995.

13 George L. Nemhauser, Laurence Wolsey, and M.L. Fisher. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*, 14(1):265–294, 1978. doi:10.1007/BF01588971.

14 Yufei Tao, Xiaocheng Hu, Dong-Wan Choi, and Chin-Wan Chung. Approximate MaxRS in spatial databases. In *Proceedings of the VLDB Endowment*, pages 1546–1557. VLDB Endowment, 2013. doi:10.14778/2536258.2536266.