Linear Time Approximation Schemes for Geometric Maximum Coverage

Jian Li[1(✉)], Haitao Wang[2], Bowei Zhang[1], and Ningye Zhang[1]

[1] Institute for Interdisciplinary Information Sciences (IIIS), Tsinghua University, Beijing 100084, China
lijian83@mail.tsinghua.edu.cn, {bw-zh14,zhangny12}@mails.tsinghua.edu.cn
[2] Department of Computer Science, Utah State University, Utah 84322, USA
haitao.wang@usu.edu

Abstract. We study approximation algorithms for the following geometric version of the maximum coverage problem: Let $\mathcal{P}$ be a set of n weighted points in the plane. We want to place m $a \times b$ rectangles such that the sum of the weights of the points in $\mathcal{P}$ covered by these rectangles is maximized. For any fixed $\varepsilon > 0$, we present efficient approximation schemes that can find a $(1-\varepsilon)$-approximation to the optimal solution. In particular, for $m = 1$, our algorithm runs in linear time $O(n \log(\frac{1}{\varepsilon}))$, improving over the previous result. For $m > 1$, we present an algorithm that runs in $O(\frac{n}{\varepsilon} \log(\frac{1}{\varepsilon}) + m(\frac{1}{\varepsilon})^{O(\min(\sqrt{m}, \frac{1}{\varepsilon}))})$ time.

Keywords: Maximum coverage · Geometric set cover · Polynomial-time approximation scheme

1 Introduction

The maximum coverage problem is a classic problem in theoretical computer science and combinatorial optimization. In this problem, we are given a universe $\mathcal{P}$ of weighted elements, a family of subsets and a number k. The goal is to select at most k of these subsets such that the sum of the weights of the covered elements in $\mathcal{P}$ is maximized. It is well-known that the most natural greedy algorithm achieves an approximation factor of $1 - 1/e$, which is essentially optimal (unless P=NP) [17,20,25]. However, for several geometric versions of the maximum coverage problem, better approximation ratios can be achieved (we will mention some of such results below). In this paper, we mainly consider the following geometric maximum coverage problem:

Jian Li, Bowei Zhang and Ningye Zhang's research was supported in part by the National Basic Research Program of China Grant 2015CB358700, 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61202009, 61033001, 61361136003. Haitao Wang's research was supported in part by NSF under Grant CCF-1317143.

© Springer International Publishing Switzerland 2015
D. Xu et al. (Eds.): COCOON 2015, LNCS 9198, pp. 559–571, 2015.
DOI: 10.1007/978-3-319-21398-9_44

Definition 1. *($\mathsf{MaxCov_R}(\mathcal{P}, m)$) Let $\mathcal{P}$ be a set of n points in a 2-dimensional Euclidean plane $\mathbb{R}^2$. Each point $p \in \mathcal{P}$ has a given weight $w_p \geq 0$. The goal of our geometric max-coverage problem (denoted as $\mathsf{MaxCov_R}(\mathcal{P}, m)$) is to place m $a \times b$ rectangles such that the sum of the weights of the covered points by these rectangles is maximized. More precisely, let S be the union of m rectangles we placed. Our goal is to maximize*

$$\mathsf{Cover}(\mathcal{P}, S) = \sum_{p \in \mathcal{P} \cap S} w_p.$$

We also study the same coverage problem with unit disks, instead of rectangles. We denote the corresponding problem as $\mathsf{MaxCov_D}(\mathcal{P}, m)$. One natural application of the geometric maximum coverage problem is the facility placement problem. In this problem, we would like to locate a certain number of facilities to serve the maximum number of clients. Each facility can serve a region (depending on whether the metric is L_1 or L_2, the region is either a square or a disk).

1.1 $m = 1$

Previous Results: We first consider $\mathsf{MaxCov_R}(\mathcal{P}, 1)$. Imai and Asano [21], Nandy and Bhattacharya [24] gave two different exact algorithms for $\mathsf{MaxCov_R}(\mathcal{P}, 1)$, both running in time $O(n \log n)$. It is also known that solving $\mathsf{MaxCov_R}(\mathcal{P}, 1)$ exactly in algebraic decision tree model requires $\Omega(n \log n)$ time [4]. Tao et al. [26] proposed a randomized approximation scheme for $\mathsf{MaxCov_R}(\mathcal{P}, 1)$. With probability $1 - 1/n$, their algorithm returns a $(1 - \varepsilon)$-approximate answer in $O(n \log(\frac{1}{\varepsilon}) + n \log \log n)$ time. In the same paper, they also studied the problem in the external memory model.

Our Results: For $\mathsf{MaxCov_R}(\mathcal{P}, 1)$ we show that there is an approximation scheme that produces a $(1 - \varepsilon)$-approximation and runs in $O(n \log(\frac{1}{\varepsilon}))$ time, improving the result by Tao et al. [26].

1.2 General $m > 1$

Previous Results: Both $\mathsf{MaxCov_R}(\mathcal{P}, m)$ and $\mathsf{MaxCov_D}(\mathcal{P}, m)$ are NP-hard if m is part of the input [22]. The most related work is de Berg, Cabello and Har-Peled [12]. They mainly focused on using unit disks (i.e., $\mathsf{MaxCov_D}(\mathcal{P}, m)$). They proposed a $(1 - \varepsilon)$-approximation algorithm for $\mathsf{MaxCov_D}(\mathcal{P}, m)$ with time complexity $O(n(m/\varepsilon)^{O(\sqrt{m})})$.

[1] We note that their algorithm can be easily extended to $\mathsf{MaxCov_R}$ with the same time complexity.

We are not aware of any explicit result for $\mathsf{MaxCov_R}(\mathcal{P}, m)$ for general $m > 1$.

---

[1] They were mainly interested in the case where m is a constant. So the running time becomes $O(n(1/\varepsilon)^{O(\sqrt{m})})$ (which is the bound claimed in their paper) and the exponential dependency on m does not look too bad for $m = O(1)$. Since we consider the more general case, we make the dependency on m explicit.

It is known [12] that the problem admits a PTAS via the standard shifting technique [19]. [2]

Our Results: Our main result is an approximation scheme for $\mathsf{MaxCov_R}(\mathcal{P}, m)$ which runs in time

$$O\left(\frac{n}{\varepsilon}\log\frac{1}{\varepsilon} + m\left(\frac{1}{\varepsilon}\right)^{\Delta}\right),$$

where $\Delta = O(\min(\sqrt{m}, \frac{1}{\varepsilon}))$. Our algorithm can be easily extended to other shapes. The algorithm for approximating approximating $\mathsf{MaxCov_D}(\mathcal{P}, m)$ can be found in the full version of this paper. [3] The running time of our algorithm is

$$O\left(n\Big(\frac{1}{\varepsilon}\Big)^{O(1)} + m\Big(\frac{1}{\varepsilon}\Big)^{\Delta}\right).$$

Following the convention of approximation algorithms, ε is a fixed constant. Hence, the second term is essentially $O(m)$ and the overall running time is essentially linear $O(n)$. Our algorithm follows the standard shifting technique [19], which reduces the problem to a smaller problem restricted in a constant size cell. The same technique is also used in de Berg et al. [12]. They proceeded by first solving the problem exactly in each cell, and then use dynamic programming to find the optimal allocation for all cells. [4]

Our improvement comes from another two simple yet useful ideas. First, we apply the shifting technique in a different way and make the side length of grids much smaller ($O(\frac{1}{\varepsilon})$, instead of $O(m)$ in de Berg et al.'s algorithm [12]). Second, we solve the dynamic program approximately. In fact, we show that a simple greedy strategy (along with some additional observations) can be used for this purpose, which allows us to save another $O(m)$ term.

1.3 Other Related Work

There are many different variants for this problem. We mention some most related problems here.

Barequet et al. [3], Dickerson and Scharstein [13] studied the max-enclosing polygon problem which aims to find a position of a given polygon to cover maximum number of points. This is the same as $\mathsf{MaxCov_R}(\mathcal{P}, 1)$ if a polygon is a rectangle. Imai et al. [21] gave an optimal algorithm for the max-enclosing rectangle problem with time complexity $O(n \log n)$.

$\mathsf{MaxCov_D}(\mathcal{P}, m)$ was introduced by Drezner [15]. Chazelle and Lee [9] gave an $O(n^2)$-time exact algorithm for the problem $\mathsf{MaxCov_D}(\mathcal{P}, 1)$. A Monte-Carlo

[2] Hochbaum and Maass [19] obtained a PTAS for the problem of covering given points with a minimal number of rectangles. Their algorithm can be easily modified into a PTAS for $\mathsf{MaxCov_R}(\mathcal{P}, m)$ with running time $n^{O(1/\epsilon)}$.

[3] The full version of this paper can be found on CS arXiv.

[4] In fact, their dynamic programming runs in time at least $\Omega(m^2)$. Since they focused on constant m, this term is negligible in their running time. But if $m > \sqrt{n}$, the term can not be ignored and may become the dominating term.

$(1-\varepsilon)$-approximation algorithm for $\mathsf{MaxCov_D}(\mathcal{P}, 1)$ was shown in [1], where $\mathcal{P}$ is an unweighted point set. Aronov and Har-Peled [2] showed that for unweighted point sets an $O(n\varepsilon^{-2}\log n)$ time Monte-Carlo $(1-\varepsilon)$-approximation algorithm exists, and also provided some results for other shapes. de Berg et al. [12] provided an $O(n\varepsilon^{-3})$ time $(1-\varepsilon)$-approximation algorithm.

For $m > 1$, $\mathsf{MaxCov_D}(\mathcal{P}, m)$ has only a few results. For $m = 2$, Cabello et al. [7] gave an exact algorithm for this problem when the two disks are disjoint in $O(n^{8/3}\log^2 n)$ time. de Berg et al. [12] gave $(1-\varepsilon)$-approximation algorithms that run in $O(n\varepsilon^{-4m+4}\log^{2m-1}(1/\varepsilon))$ time for $m > 3$ and in $O(n\varepsilon^{-6m+6}\log(1/\varepsilon))$ time for $m = 2, 3$.

The dual of the maximum coverage problem is the classical set cover problem. The geometric set cover problem has enjoyed extensive study in the past two decades. The literature is too vast to list exhaustively here. See e.g., [6,8,10,16,23,27] and the references therein.

2 Preliminaries

We first define some notations and mention some results that are needed in our algorithm. Denote by $G_\delta(a, b)$ the square grid with mesh size δ such that the vertical and horizontal lines are defined as follows

$$G_\delta(a,b) = \{(x,y) \in \mathbb{R}^2 \mid y = b + k \cdot \delta, k \in \mathbb{Z}\} \cup \{(x,y) \in \mathbb{R}^2 \mid x = a + k \cdot \delta, k \in \mathbb{Z}\}$$

Given $G_\delta(a, b)$ and a point $p = (x, y)$, we call the integer pair $(\lfloor x/\delta \rfloor, \lfloor y/\delta \rfloor)$ the *index* of p (the index of the cell in which p lies in).

Perfect Hashing: Dietzfetbinger et al. [14] shows that if each basic algebraic operation (including $\{+, -, \times, \div, \log_2, \exp_2\}$) can be done in constant time, we can get a perfect hash family so that each insertion and membership query takes $O(1)$ expected time. In particular, using this hashing scheme, we can hash the indices of all points, so that we can obtain the list of all non-empty cells in $O(n)$ expected time. Moreover, for any non-empty cell, we can retrieve all points lies in it in time linear in the number of such points.

Linear Time Weighted Median and Selection: It is well known that finding the weighted median for an array of numbers can be done in deterministic worst-case linear time. The setting is as follows: Given n distinct elements $x_1, x_2, ..., x_n$ with positive weights $w_1, w_2, ..., w_n$. Let $w = \sum_{i=1}^{n} w_i$. The *weighted median* is the element x_k satisfying $\sum_{x_i < x_k} w_i < w/2$ and $\sum_{x_i > x_k} w_i \leq w/2$. Finding the kth smallest elements for any array can also be done in deterministic worst-case linear time. See e.g., [11].

An Exact Algorithm for $\mathsf{MaxCov_R}(\mathcal{P}, 1)$: As we mentioned previously, Nandy and Bhattacharya [24] provided an $O(n \log n)$ exact algorithm for the $\mathsf{MaxCov_R}(\mathcal{P}, 1)$ problem. We are going to use this algorithm as a subroutine in our algorithm.

3 A Linear Time Algorithm for $\mathsf{MaxCov_R}(\mathcal{P}, 1)$

Notations: Without loss of generality, we can assume that $a = b = 1$, i.e., all the rectangles are 1×1 squares, (by properly scaling the input). We also assume that all points are in general positions. In particular, all coordinates of all points are distinct. For a unit square r, we use $w(r)$ to denote the sum of the weights of the points covered by r. We say a unit square r is located at (x, y) if the top-left corner of r is (x, y).

Now we present our approximation algorithm for $\mathsf{MaxCov_R}(\mathcal{P}, 1)$.

3.1 Grid Shifting

Recall the definition of a grid $G_\delta(a, b)$ (in Section 2). Consider the following four grids: $G_2(0,0)$,$G_2(0,1)$,$G_2(1,0)$,$G_2(1,1)$ with $\delta = 2$. We can easily see that for any unit square r, there exists one of the above grids that does not intersect r (i.e., r is inside some cell of the grid). This is also the case for the optimal solution.

Now, we describe the overall framework, which is similar to that in [26]. Our algorithm differs in several details. MaxCovCell(c) is a subroutine that takes a 2×2 cell c as input and returns a unit square r that is a (1-ε)-approximate solution if the problem is restricted to cell c. We present the details of MaxCovCell in the next subsection.

Algorithm 1. $\mathsf{MaxCov_R}(\mathcal{P}, 1)$

$w_{\max} \leftarrow 0$
for each $G \in \{G_2(0,0), G_2(0,1), G_2(1,0), G_2(1,1)\}$ **do**
  Use perfect hashing to find all the non-empty cells of G.
  for each non-empty cell c of G **do**
    $r \leftarrow$ MaxCovCell(c).
    If $w(r) > w_{\max}$, **then** $w_{\max} \leftarrow w(r)$ and $r_{\max} \leftarrow r$.
  end for;
end for;
return $r_{\max}$;

As we argued above, there exists a grid G such that the optimal solution is inside some cell $\mathsf{c}^\star \in G$. Therefore, MaxCovCell($\mathsf{c}^\star$) should return a (1-ε)-approximation for the original problem $\mathsf{MaxCov_R}(\mathcal{P}, 1)$.

3.2 MaxCovCell

In this section, we present the details of the subroutine MaxCovCell. Now we are dealing with the problem restricted to a single 2×2 cell c. Denote the number of point in c by n_c, and the sum of the weights of points in c by W_c. We distinguish two cases, depending on whether n_c is larger or smaller than $\left(\frac{1}{\varepsilon}\right)^2$. If $n_\mathsf{c} < \left(\frac{1}{\varepsilon}\right)^2$, we simply apply the $O(n \log n)$ time exact algorithm. [24]

Algorithm 2. PARTITION($\{x_1, x_2, ..., x_n\}$)

Find the weighted median x_k (w.r.t. w-weight);
$\mathcal{L} = \mathcal{L} \cup \{x_k\}$;
Generate $S = \{x_i \mid w_i < x_k\}$, $L = \{x_i \mid w_i > x_k\}$;
If the sum of the weights of the points in S is lager than w_d, run PARTITION(S);
If the sum of the weights of the points in L is lager than w_d, run PARTITION(L);

The other case requires more work. In this case, we further partition cell c into many smaller cells. First, we need the following simple lemma.

Lemma 1. *Given n points in $\mathbb{R}^2$ with positive weights $w_1, w_2, ..., w_n$, $\sum_{i=1}^{n} w_i = w$. Assume that $x_1, x_2, ..., x_n$ are their distinct x-coordinates. We are also given a value w_d such that* $\max(w_1, w_2, ..., w_n) \le w_d \le w$, *Then, we can find at most $2w/w_d$ vertical lines such that the sum of the weights of points strictly between (we do not count the points on these lines) any two adjacent lines is at most w_d in time $O(n \log(w/w_d))$.*

Proof. See Algorithm 2. In this algorithm, we apply the weighted median algorithm recursively. Initially we have a global variable $\mathcal{L} = \emptyset$, which upon termination is the set of x-coordinates of the selected vertical lines. Each time we find the weighted median x_k and separate the point with the vertical line $x = x_k$, which we add into $\mathcal{L}$. The sum of the weights of points in either side is at most half of the sum of the weights of all the points. Hence, the depth of the recursion is at most $\lceil \log(w/w_d) \rceil$. Thus, the size of $\mathcal{L}$ is at most $2^{\lceil \log(w/w_d) \rceil} \le 2w/w_d$, and the running time is $O(n \log(w/w_d))$. □

Now, we describe how to partition cell c into smaller cells. First we partition c with some vertical lines. Let $\mathcal{L}_v$ to denote a set of vertical lines. Initially, $\mathcal{L} = \emptyset$. Let $w_d = \frac{\varepsilon \cdot W_\mathsf{c}}{16}$. We find all the points whose weights are at least w_d. For each such point, we add to $\mathcal{L}_v$ the vertical line that passes through the point. Then, we apply Algorithm 2 to all the points with weights less than w_d. Next, we add a set $\mathcal{L}_h$ of horizontal lines in exactly the same way.

Lemma 2. *The sum of the weights of points strictly between any two adjacent lines in $\mathcal{L}_v$ is at most $w_d = \frac{\varepsilon \cdot W_\mathsf{c}}{16}$. The number of vertical lines in $\mathcal{L}_v$ is at most $\frac{32}{\varepsilon}$. Both statements hold for $\mathcal{L}_h$ as well.*

Proof. The first statement is straightforward from the description of the algorithm. We only need to prove the upper bound of the number of the vertical lines. Assume the sum of the weights of those points considered in the first (resp. second) step is W_1(resp. W_2), $W_1 + W_2 = W_\mathsf{c}$. The number of vertical lines in $\mathcal{L}_v$ is at most

$$W_1 / \left(\frac{\varepsilon \cdot W_\mathsf{c}}{16} \right) + 2W_2 / \left(\frac{\varepsilon \cdot W_\mathsf{c}}{16} \right) \le \frac{32}{\varepsilon}.$$

The first term is due to the fact that the weight of each point we found in the first step has weight at least $\frac{\varepsilon \cdot W_\mathsf{c}}{16}$, and the second term directly follows from Lemma 1. □

We add both vertical boundaries of cell c into $\mathcal{L}_v$ and both horizontal boundaries of cell c into $\mathcal{L}_h$. Now $\mathcal{L} = \mathcal{L}_v \cup \mathcal{L}_h$ forms a grid of size at most $(\frac{32}{\varepsilon}+2) \times (\frac{32}{\varepsilon}+2)$. Assume $\mathcal{L} = \{(x, y) \in \mathbb{R}^2 \mid y = y_j, j \in \{1, ..., u\}\} \cup \{(x, y) \in \mathbb{R}^2 \mid x = x_i, i \in \{1, ..., v\}\}$, with both $\{y_i\}$ and $\{x_i\}$ are sorted. $\mathcal{L}$ partitions c into *small cells*. The final step of our algorithm is simply enumerating all the unit squares located at $(x_i, y_j), i \in \{1, ..., u\}, j \in \{1, ..., v\}$, and return the one with the maximum coverage. However, computing the coverage exactly for all these unit squares is expensive. Instead, we only calculate the weight of these unit square approximately as follows. For each unit square r, we only count the weight of points that are in some small cell fully covered by r. Now, we show this can be done in $O\left(n_{\mathsf{c}} \log\left(\frac{1}{\varepsilon}\right) + \left(\frac{1}{\varepsilon}\right)^2\right)$ time.

After sorting $\{y_i\}$ and $\{x_i\}$, we can use binary search to identify which small cell each point lies in. So we can calculate the sum of the weights of points at the interior, edges or corners of all small cells in $O(n_{\mathsf{c}} \log\left(\frac{1}{\varepsilon}\right))$ times.

Thus searching the unit square with the maximum (approximate) coverage can be done with a standard incremental algorithm in $O\left(\frac{1}{\varepsilon}\right)^2$ time. Due to space constraints, we omit the details which can be found in the full version of this paper.

Putting everything together, we conclude that if $n_{\mathsf{c}} \geq \left(\frac{1}{\varepsilon}\right)^2$, the running time of MaxCovCell(c) is $O\left(n_c \log\left(\frac{1}{\varepsilon}\right) + \left(\frac{1}{\varepsilon}\right)^2\right)$. We can conclude the main result of this section with the following theorem.

Theorem 1. *Algorithm 1 returns a (1-ε)-approximate answer for* $\mathsf{MaxCov_R}(\mathcal{P}, 1)$ *in* $O(n \log\left(\frac{1}{\varepsilon}\right))$ *time.*

Proof. We only show the proof of the approximation guarantee of the algorithm. The complete proof can be found in the full version of this paper. We only need to prove that MaxCovCell(c) returns a (1-ε)-approximation for cell c. The case $n_{\mathsf{c}} < \left(\frac{1}{\varepsilon}\right)^2$ is trivial since we apply the exact algorithm. So we only need to prove the case of $n_{\mathsf{c}} \geq \left(\frac{1}{\varepsilon}\right)^2$.

Suppose the optimal unit square is r. Denote by Opt the weight of the optimal solution. The size of c is 2×2, so we can use 4 unit squares to cover the entire cell. Therefore, $\mathsf{Opt} \geq (\frac{W_{\mathsf{c}}}{4})$. Suppose r is located at a point p, which is in the strict interior of a small cell B separated by $\mathcal{L}$. [5] Suppose the index of B is (i, j). We compare the weight of r with $I(i, j)$ (which is the approximate weight of the unit square located at the top-left corner of B). By the rule of our partition, the weight difference is at most 4 times the maximum possible weight of points between two adjacent parallel lines in $\mathcal{L}$. So $I(i, j) \geq \mathsf{Opt} - 4 \cdot \frac{\varepsilon \cdot W_{\mathsf{c}}}{16} \geq (1 - \varepsilon)\mathsf{Opt}$. This completes the proof. □

[5] If p lies on the boundary of B, the same argument still works.

4 Linear Time Algorithms for $\mathsf{MaxCov}_{\mathsf{R}}(\mathcal{P}, m)$

4.1 Grid Shifting

For general m, we need the shifting technique [19]. Consider grids with a different side length: $G_{6/\varepsilon}(a, b)$. We shift the grid to $\frac{6}{\varepsilon}$ different positions: $(0, 0), (1, 1), ..., (\frac{6}{\varepsilon} - 1, \frac{6}{\varepsilon} - 1)$. (For simplicity, we assume that $\frac{1}{\varepsilon}$ is an integer and no point in $\mathcal{P}$ has an integer coordinate, so points in $\mathcal{P}$ will never lie on the grid line. Let

$$\mathbb{G} = \left\{G_{6/\varepsilon}(0, 0), ..., G_{6/\varepsilon}(6/\varepsilon - 1, 6/\varepsilon - 1)\right\}.$$

The following lemma is quite standard. The proof can be found in the full version of this paper.

Lemma 3. *There exist $G^\star \in \mathbb{G}$ and a $(1-\frac{2\varepsilon}{3})$-approximate solution R such that none of the unit squares in R intersects $G^\star$.*

We present a subroutine in section 4.4 which can approximately solve the problem for a grid, and apply it to each non-empty grid in $\mathbb{G}$. Then, in order to compute our final output from those obtained solutions, we apply a dynamic programming algorithm or a greedy algorithm which are shown in the next two sections.

4.2 Dynamic Programming

Now consider a fixed grid $G \in \mathbb{G}$. Let $\mathsf{c}_1, \ldots, \mathsf{c}_t$ be the cells of grid G and Opt be the optimal solution that does not intersect G. Obviously, $(\frac{6}{\varepsilon})^2$ unit squares are enough to cover an entire $\frac{6}{\varepsilon} \times \frac{6}{\varepsilon}$ cell. Thus the maximum number of unit squares we need to place in one single cell is $m_c = \min\{m, (\frac{6}{\varepsilon})^2\}$.

Let $\mathsf{Opt}(\mathsf{c}_i, k)$ be the maximum weight we can cover with k unit squares in cell c_i. For each nonempty cell c_i and for each $k \in [m_c]$, we find a $(1 - \frac{\varepsilon}{3})$-approximation $\mathsf{F}(\mathsf{c}_i, k)$ to $\mathsf{Opt}(\mathsf{c}_i, k)$. We will show how to achieve this later. Now assume that we can do it.

Let $\mathsf{Opt}_{\mathsf{F}}(m)$ be the optimal solution we can get from the values $\mathsf{F}(\mathsf{c}_i, k)$. More precisely,

$$\mathsf{Opt}_{\mathsf{F}}(m) = \max_{k_1,\ldots,k_t \in [m_c]} \left\{ \sum_{i=1}^{t} \mathsf{F}(\mathsf{c}_i, k_i) \;\middle|\; \sum_{i=1}^{t} k_i = m \right\} \tag{1}$$

We can see that $\mathsf{Opt}_{\mathsf{F}}(m)$ must be a $(1-\frac{\varepsilon}{3})$-approximation to Opt. We can easily use dynamic programming to calculate the exact value of $\mathsf{Opt}_{\mathsf{F}}(m)$. Denote by $A(i, k)$ the maximum weight we can cover with k unit squares in cells $\mathsf{c}_1, \mathsf{c}_2, ..., \mathsf{c}_i$. We have the following DP recursion:

$$A(i, k) = \begin{cases} \max_{j=0}^{\min(k, m_c)} \{A(i-1, k-j) + \mathsf{F}(\mathsf{c}_i, j)\} & \text{if} \quad i > 1 \\ \mathsf{F}(c_1, k) & \text{if} \quad i = 1 \end{cases}$$

The running time of the above simple dynamic programming is $O(m^2 \cdot m_c)$. One may notice that each step of the DP is computing a $(+, \max)$ convolution. However, existing algorithms (see e.g., [5,28]) only run slightly better than quadratic time. So the improvement would be quite marginal. But in the next section, we show that if we would like to settle for an approximation to $\mathsf{Opt}_\mathsf{F}(m)$, the running time can be dramatically improved to linear.

4.3 A Greedy Algorithm

We first apply our $\mathsf{MaxCov}_\mathsf{R}(\mathcal{P}, 1)$ algorithm in Section 3 to each cell c_i, to compute a $(1-\frac{\varepsilon^2}{9})$-approximation of $\mathsf{Opt}(\mathsf{c}_i, 1)$. Let $f(\mathsf{c}_i, 1)$ be the return values. [6] This takes $O(n \log \frac{1}{\varepsilon})$ time. Then, we use the selection algorithm to find out the m cells with the largest $f(\mathsf{c}_i, 1)$ values. Assume that those cells are $\mathsf{c}_1, ..., \mathsf{c}_m, \mathsf{c}_{m+1}, ..., \mathsf{c}_t$, sorted from largest to smallest by $f(\mathsf{c}_i, 1)$.

Lemma 4. *Let* Opt' *be the maximum weight we can cover using* m *unit squares in* $\mathsf{c}_1, ..., \mathsf{c}_m$. *Then* $\mathsf{Opt}' \geq (1-\frac{\varepsilon^2}{9})\mathsf{Opt}$

Proof. Let k be the number of unit squares in Opt that are chosen from $\mathsf{c}_{m+1}, \ldots, \mathsf{c}_t$. This means there must be at least k cells in $\{\mathsf{c}_1, \ldots, \mathsf{c}_m\}$ such that Opt does not place any unit square. Therefore we can always move all k unit squares placed in $\mathsf{c}_{m+1}, \ldots, \mathsf{c}_t$ to these empty cells such that each empty cell contains only one unit square. Denote the weight of this modified solution by A. Obviously, $\mathsf{Opt}' \geq A$. For any i,j such that $1 \leq i \leq m < j \leq t$, we have $\mathsf{Opt}(\mathsf{c}_i, 1) \geq f(\mathsf{c}_i, 1) \geq f(\mathsf{c}_j, 1) \geq (1-\frac{\varepsilon^2}{9})\mathsf{Opt}(\mathsf{c}_j, 1)$. Combining with a simple observation that $\mathsf{Opt}(\mathsf{c}_i, k) \leq k\mathsf{Opt}(\mathsf{c}_i, 1)$, we can see that $A \geq (1-\frac{\varepsilon^2}{9})\mathsf{Opt}$. Therefore, $\mathsf{Opt}' \geq (1-\frac{\varepsilon^2}{9})\mathsf{Opt}$. $\square$

Hence, from now on, we only need to consider the first m cells $\{\mathsf{c}_1, ..., \mathsf{c}_m\}$. We distinguish two cases. If $m \leq 324(\frac{1}{\varepsilon})^4$, we just apply the dynamic program to $\mathsf{c}_1, ..., \mathsf{c}_m$. The running time of the above dynamic programming is $O((\frac{1}{\varepsilon})^{O(1)})$.

If $m > 324(\frac{1}{\varepsilon})^4$, we can use a greedy algorithm to find a answer of weight at least $(1-\frac{\varepsilon^2}{9})\mathsf{Opt}_\mathsf{F}(m)$.

Let $\mathsf{b} = (\frac{6}{\varepsilon})^2$. For each cell c_i, we find the upper convex hull of 2D points $\{(0, \mathsf{F}(\mathsf{c}_i, 0)), (1, \mathsf{F}(\mathsf{c}_i, 1)), \ldots, (\mathsf{b}, \mathsf{F}(\mathsf{c}_i, \mathsf{b}))\}$. See Figure 1. Suppose the convex hull points are $\{(t_{i,0}, \mathsf{F}(\mathsf{c}_i, t_{i,0})), (t_{i,1}, \mathsf{F}(\mathsf{c}_i, t_{i,1})), \ldots, (t_{i,s_i}, \mathsf{F}(\mathsf{c}_i, t_{i,s_i}))\}$, where $t_{i,0} = 0, t_{i,s_i} = \mathsf{b}$. For each cell, since the above points are already sorted from left to right, we can compute the convex hull in $O(\mathsf{b})$ time by Graham's scan[18]. Therefore, computing the convex hulls for all these cells takes $O(m\mathsf{b})$ time.

For each cell c_i, we maintain a value p_i representing that we are going to place t_{i,p_i} squares in cell c_i. Initially for all $i \in [m]$, $p_i = 0$. In each stage, we find the cell c_i such that current slope (the slope of the next convex hull edge)

$$\frac{\mathsf{F}(\mathsf{c}_i, t_{i,p_i+1}) - \mathsf{F}(\mathsf{c}_i, t_{i,p_i})}{t_{i,p_i+1} - t_{i,p_i}}$$

[6] Both $f(\mathsf{c}_i, 1)$ and $\mathsf{F}(\mathsf{c}_i, 1)$ are approximations of $\mathsf{Opt}(\mathsf{c}_i, 1)$, with slightly different approximation ratios.

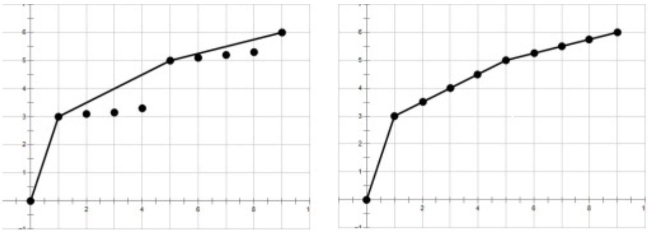

Fig. 1. $\mathsf{F}(\mathsf{c}_i, k)$ (left) and $\widehat{\mathsf{F}}(\mathsf{c}_i, k)$ (right)

is maximized. Then we add 1 to p_i, or equivalently we assign $t_{i,p_i+1} - t_{i,p_i}$ more squares into cell c_i. We repeat this step until we have already placed at least $m - \mathsf{b}$ squares. We can always achieve this since we can place at most b squares in one single cell in each iteration. Let m' the number of squares we have placed $(m = \mathsf{b} \le m' \le m)$. For the remaining $m - m'$ squares, we allocate them arbitrarily. We denote the algorithm by Greedy and let the value obtained be $\mathsf{Greedy}(m')$. Having the convex hulls, the running time of the greedy algorithm is $O(m)$.

Now we analyze the performance of the greedy algorithm.

Lemma 5. *The above greedy algorithm computes an* $(1 - \varepsilon^2/9)$*-approximation to* $\mathsf{Opt}_{\mathsf{F}}(m)$.

Proof. Define an auxiliary function $\widehat{\mathsf{F}}(\mathsf{c}_i, k)$ as follows: If $k = t_{i,j}$ for some j, $\widehat{\mathsf{F}}(\mathsf{c}_i, k) = F(\mathsf{c}_i, k)$. Otherwise, suppose $t_{i,j} < k < t_{i,j+1}$, then

$$\widehat{\mathsf{F}}(\mathsf{c}_i, k) = F(\mathsf{c}_i, t_{i,j}) + \frac{F(\mathsf{c}_i, t_{i,j+1}) - F(\mathsf{c}_i, t_{i,j})}{t_{i,j+1} - t_{i,j}} \times (k - t_{i,j}).$$

Intuitively speaking, $\widehat{\mathsf{F}}(\mathsf{c}_i, k)$(See Figure 1) is the function defined by the upper convex hull at integer points. [7] Thus, for all $i \in [m]$, $\widehat{\mathsf{F}}(\mathsf{c}_i, k)$ is a concave function. Obviously, $\widehat{\mathsf{F}}(\mathsf{c}_i, k) \ge \mathsf{F}(\mathsf{c}_i, k)$ for all $i \in [m]$ and all $k \in [\mathsf{b}]$.

Let $\mathsf{Opt}_{\widehat{\mathsf{F}}}(i)$ be the optimal solution we can get from the values $\widehat{\mathsf{F}}(\mathsf{c}_i, k)$ by placing i squares. By the convexity of $\widehat{\mathsf{F}}(\mathsf{c}_i, k)$, the following greedy algorithm is optimal: as long as we still have budget, we assign 1 more square to the cell which provides the largest increment of the objective value. In fact, this greedy algorithm runs in almost the same way as Greedy. The only difference is that Greedy only picks an entire edge of the convex hull, while the greedy algorithm here may stop in the middle of an edge (only happen for the last edge). Since the marginal increment never increases, we can see that $\mathsf{Opt}_{\widehat{\mathsf{F}}}(i)$ is concave.

By the way of choosing cells in our greedy algorithm, we make the following simple but important observation:

$$\mathsf{Greedy}(m') = \mathsf{Opt}_{\widehat{\mathsf{F}}}(m') = \mathsf{Opt}_{\mathsf{F}}(m').$$

[7] At first sight, it may appear that $\mathsf{F}(\mathsf{c}_i, k)$ should be a concave function. However, this is not true. A counter-example is provided in the full version of this paper.

So, our greedy algorithm is in fact optimal for m'. Combining with $m - m' \leq \mathsf{b}$ and the concavity of $\mathsf{Opt}_{\widehat{\mathsf{F}}}$, we can see that

$$\mathsf{Opt}_{\widehat{\mathsf{F}}}(m') \geq \frac{m-\mathsf{b}}{m}\mathsf{Opt}_{\widehat{\mathsf{F}}}(m) \geq \left(1 - \frac{\varepsilon^2}{9}\right)\mathsf{Opt}_{\mathsf{F}}(m).$$

The last inequality holds because $\mathsf{Opt}_{\widehat{\mathsf{F}}}(i) \geq \mathsf{Opt}_{\mathsf{F}}(i)$ for any i. □

4.4 Computing $\mathsf{F}(\mathsf{c}, k)$

Now we show the subroutine MaxCovCellM for computing $\mathsf{F}(\mathsf{c}, k)$. We use a similar partition algorithm as Section 3.2. The only difference is that this time we need to partition the cell finer so that the maximum possible weight of points between any two adjacent parallel partition lines is $(\frac{\varepsilon^3 W_{\mathsf{c}}}{864})$. After partitioning the cell, we enumerate all the possible ways of placing k unit squares at the grid point. Similarly, for each unit square r, we only count the weight of points that are in some cell fully covered by r.

We can adapt the algorithm in [12] to enumerate these possible choices in $O((\frac{1}{\varepsilon})^{\Delta})$ time where $\Delta = O(\min(\sqrt{m}, \frac{1}{\varepsilon}))$. The details can be found in the full version of this paper. Now we prove the correctness of this algorithm.

Lemma 6. MaxCovCellM *returns a* $(1 - \frac{\varepsilon}{3})$ *approximate answer for* $\mathsf{Opt}(\mathsf{c}_i, k)$.

Proof. We can use $(\frac{6}{\varepsilon})^2$ unit squares to cover the entire cell, so $\mathsf{Opt}(\mathsf{c}_i, k) \geq \frac{k\varepsilon^2 W_{\mathsf{c}}}{72}$. By the same argument as in Theorem 1, the difference between $\mathsf{Opt}(\mathsf{c}_i, k)$ and the answer we got are at most $4k$ times the maximum possible weight of points between two adjacent parallel partition lines. Therefore, the algorithm returns a $(1 - \frac{\varepsilon}{3})$-approximate answer of $\mathsf{Opt}(\mathsf{c}_i, k)$. □

Now we can conclude the following theorem.

Theorem 2. *Let P be a set of n weighted point, for any $0 < \varepsilon < 1$ we can find a $(1 - \varepsilon)$-approximate answer for $\mathsf{MaxCov}_{\mathsf{R}}(\mathcal{P}, m)$ in time*

$$O\left(\frac{n}{\varepsilon}\log\frac{1}{\varepsilon} + m\left(\frac{1}{\varepsilon}\right)^{\Delta}\right),$$

where $\Delta = O(\min(\sqrt{m}, \frac{1}{\varepsilon}))$.

The proof is similar to Theorem 1, and it is given in the full version of this paper.

References

1. Agarwal, P.K., Hagerup, T., Ray, R., Sharir, M., Smid, M., Welzl, E.: Translating a planar object to maximize point containment. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, p. 42. Springer, Heidelberg (2002)
2. Aronov, B., Har-Peled, S.: On approximating the depth and related problems. SICOMP **38**(3), 899–921 (2008)
3. Barequet, G., Dickerson, M., Pau, P.: Translating a convex polygon to contain a maximum number of points. Computational Geometry **8**(4), 167–179 (1997)
4. Ben-Or, M.: Lower bounds for algebraic computation trees. In: STOC, pp. 80–86. ACM (1983)
5. Bremner, D., Chan, T.M., Demaine, E.D., Erickson, J., Hurtado, F., Iacono, J., Langerman, S., Taslakian, P.: Necklaces, convolutions, and $X + Y$. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 160–171. Springer, Heidelberg (2006)
6. Brönnimann, H., Goodrich, M.: Almost optimal set covers in finite vc-dimension. DCG **14**(1), 463–479 (1995)
7. Cabello, S., Díaz-Báñez, J.M., Seara, C., Sellares, J.A., Urrutia, J., Ventura, I.: Covering point sets with two disjoint disks or squares. Computational Geometry **40**(3), 195–206 (2008)
8. Chan, T.M., Grant, E., Könemann, J., Sharpe, M.: Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In: SODA, SODA 2012, pp. 1576–1585. SIAM (2012)
9. Chazelle, B.M., Lee, D.-T.: On a circle placement problem. Computing **36**(1–2), 1–16 (1986)
10. Clarkson, K.L., Varadarajan, K.: Improved approximation algorithms for geometric set cover. DCG **37**(1), 43–58 (2007)
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press (2009)
12. de Berg, M., Cabello, S., Har-Peled, S.: Covering many or few points with unit disks. Theory of Computing Systems **45**(3), 446–469 (2009)
13. Dickerson, M., Scharstein, D.: Optimal placement of convex polygons to maximize point containment. Computational Geometry **11**(1), 1–16 (1998)
14. Dietzfelbinger, M., Hagerup, T., Katajainen, J., Penttonen, M.: A reliable randomized algorithm for the closest-pair problem. Journal of Algorithms **25**(1), 19–51 (1997)
15. Drezner, Z.: Noteon a modified one-center model. Management Science **27**(7), 848–851 (1981)
16. Even, G., Rawitz, D., Shahar, S.M.: Hitting sets when the vc-dimension is small. IPL **95**(2), 358–362 (2005)
17. Feige, U.: A threshold of ln n for approximating set cover. JACM **45**(4), 634–652 (1998)
18. Graham, R.L.: An efficient algorith for determining the convex hull of a finite planar set. Information processing letters **1**(4), 132–133 (1972)
19. Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and vlsi. JACM **32**(1), 130–136 (1985)
20. Hochbaum, D.S., Pathria, A.: Analysis of the greedy approach in problems of maximum k-coverage. Naval Research Logistics **45**(6), 615–627 (1998)
21. Imai, H., Asano, T.: 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)

22. Megiddo, N., Supowit, K.J.: On the complexity of some common geometric location problems. SICOMP **13**(1), 182–196 (1984)
23. Mustafa, N.H., Ray, S.: Ptas for geometric hitting set problems via local search. In: SCG, pp. 17–22. ACM (2009)
24. Nandy, S.C., Bhattacharya, B.B.: A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. Computers & Mathematics with Applications **29**(8), 45–61 (1995)
25. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions. Mathematical Programming **14**(1), 265–294 (1978)
26. Tao, Y., Hu, X., Choi, D.-W., Chung, C.-W.: Approximate maxrs in spatial databases. Proceedings of the VLDB Endowment **6**(13), 1546–1557 (2013)
27. Varadarajan, K.: Weighted geometric set cover via quasi-uniform sampling. In: STOC, pp. 641–648. ACM (2010)
28. Williams, R.: Faster all-pairs shortest paths via circuit complexity. In: STOC, pp. 664–673. ACM (2014)