AUT Journal of Mathematics and Computing

AUT J. Math. Com., 1(1) (2020) 47-55
DOI: 10.22060/ajmc.2018.3044

A simple greedy approximation algorithm for the unit disk cover problem

Mahdi Imanparast[a], Seyed Naser Hashemi*[b]

[a]*Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran*
[b]*Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran*

ABSTRACT: Given a set $\mathcal{P}$ of n points in the plane, the unit disk cover problem, which is known as an NP-hard problem, seeks to find the minimum number of unit disks that can cover all points of $\mathcal{P}$. We present a new 4-approximation algorithm with running time $O(n \log n)$ for this problem. Our proposed algorithm uses a simple approach and is easy to understand and implement.

Review History:
Received:24 April 2018
Revised:06 July 2018
Accepted:08 August 2018
Available Online:1 February 2020

Keywords:
Computational geometry
Approximation algorithms
Unit disk cover problem
Facility location

1. Introduction

The unit disk cover (UDC) problem seeks to find a set of unit disks with a minimal cardinality, so that it covers a set of points given by $\mathcal{P}$ in the plane. The UDC problem is known as an NP-hard problem [1]. This problem occurs in applications such as facility location, motion planning, and image processing [1, 2]. One of the first methods to solve the UDC problem was to use the shifting strategy, which for a large arbitrary parameter $\ell \geqslant 1$, was able to approximate this problem. For points in $\mathbb{R}^d$ and any integer $\ell \geqslant 1$, it is possible to compute the UDC with the factor $(1 + \frac{1}{\ell})^d$ and time $O((d\ell)^{O(d)} n^{(O(d\ell)^d)})$ [2] or with factor $2(1 + \frac{1}{\ell})^{d-1}$ and time $O((d\ell)^{O(d)} n^{O(d^d)})$ [3] in the Euclidean space. Gonzalez [3] presented an 8-approximation algorithm which runs in $O(n \log S)$ time, where $S \leqslant n$ is the number of disks in the optimal solution. An approximation algorithm with running time $O(n^3 \log n)$ is also presented in [4], but no attempt has been made to obtain the exact amount of its approximation factor. By imposing disks center on a grid, a $3(1 + \frac{1}{\ell})^2$-approximation algorithm for each $\ell \geqslant 1$ with $O(Kn)$ time is presented in [5], in which K is a function of ℓ and the size of the approximation grid. A 2.8334-approximation algorithm with running time $O(n(\log n \log\log n)^2)$ is also presented in [6], which is partly difficult to implement. Beside this, it has a high constant factor at running time [8]. By dividing inputs into vertical strips, Liu and Lu [7] provided a $\frac{25}{6}$-approximation algorithm with running time $O(n \log n)$ for this problem. Finally, Biniaz et al. [8] proposed an approximation algorithm with approximation factor 4 and running time $O(n \log^2 n)$ for the UDC problem, and they also showed that by using the plane sweep technique; its running time can be improved to $O(n \log n)$ time. A list of all available approximation algorithms for the UDC problem, as well as their approximation factor and running time, are shown in Table 1.

2. Main results

In this paper, we present a new 4-approximation algorithm for the UDC problem. It should be noted that our proposed algorithm is completely different in computational techniques from [6], and [8]. Moreover, it uses a

*Corresponding author.
E-mail addresses: m.imanparast@aut.ac.ir, nhashemi@aut.ac.ir

Table 1: A summary on existing approximation algorithms for the UDC problem in the plane.

Reference	Approximation factor	Running Time	Year
[2]	$(1+\frac{1}{\ell})^2$	$O(\ell^4(2n)^{4\ell^2+1})$	1985
[3]	$2(1+\frac{1}{\ell})$	$O(\ell^2 n^{6\ell\sqrt{2}+1})$	1991
[3]	8	$O(n \log S)$	1991
[4]	O(1)	$O(n^3 \log n)$	1995
[5]	$3(1+\frac{1}{\ell})^2$	$O(Kn)$	2001
[6]	2.8334	$O(n(\log n \log\log n)^2)$	2007
[7]	$\frac{25}{6}$	$O(n \log n)$	2014
[8]	4	$O(n \log n)$	2017
Ours	4	$O(n \log n)$	2018

simpler approach for solving this problem. The remainder of our paper is organized as follows: in subsection 2.1, we first describe a simple 4-approximation algorithm with running time $O(n^2)$ for the UDC problem. Our analyses on the approximation factor and the running time of the proposed algorithm is presented in subsection 2.2. We then improve the running time of our algorithm to $O(n \log n)$ in subsection 2.3. Finally, our paper is concluded in section 3.

2.1. The proposed algorithm

In this subsection, we propose a new approximation algorithm for solving the UDC problem. The main idea of the algorithm is to round the demand points of $\mathcal{P}$ to a grid, and then to use an iterative greedy approach for finding the maximum covering disk in each iteration of the algorithm. First, the algorithm rounds points of $\mathcal{P}$ to the central-points of a grid with side length $\sqrt{2}$. Let $\mathcal{D} = \{d_1, d_2, \ldots, d_m\}$ be the set of rounding points after rounding to the grid that we call it Potential Circles Set (PCS). Then, for each point of the PCS, that is generated by rounding to a grid, we shift it to four directions Up-Left (UL), Up-Right (UR), Down-Left (DL), and Down-Right (DR) by unit size. We do this process for each point d_i. For example, $UL(d_i) = (d_i - \sqrt{2}/2, d_i + \sqrt{2}/2)$ shifts point d_i to Up-Left direction by unit size. By rounding to a grid with side length $\sqrt{2}$, we can find a PCS of unit disks that covers all demand points. We increase the PCS domain for having more option to select by shifting rounded points in initial set $\mathcal{D}$ to four directions. For each point d_i of the new PCS, we count the number of demand points inside the unit circle C_{d_i} with center point d_i. Then, in an iterative process, we find a point of the PCS with maximum coverage and remove demand points inside it. Finally, we repeat this process for points in the remaining demand points until all points are covered. We can now give this simple approach in the following algorithm:

Algorithm 1: APPROXIMATE UNIT DISK COVER

Input : a point set $\mathcal{P}$ of n demand points in the plane.
Output: a solution for the UDC problem.

Round points of $\mathcal{P}$ to central-points of a grid with side length $\sqrt{2}$;
Let $\mathcal{D} = \{d_1, d_2, \ldots, d_m\}$ be the set of rounding points;
for $i := 1$ to m **do**
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{UL(d_i), UR(d_i), DL(d_i), DR(d_i)\}$;
end for
while $(\mathcal{P} \neq \phi)$
    Compute list $\varphi = \{\varphi_1, \varphi_2, \ldots, \varphi_{|\mathcal{D}|}\}$, where φ_i is the number of
    demand points inside the unit circle C_{d_i} with center point d_i;
    $j \leftarrow$ index of a point $d_j \in \mathcal{D}$, where φ_j is the maximum of the list φ;
    $\hat{\mathcal{C}} \leftarrow \hat{\mathcal{C}} \cup \{d_j\}$;
    $\mathcal{D} \leftarrow \mathcal{D} - \{d_j\}$;
    $\mathcal{P} \leftarrow \mathcal{P} -$ all points inside C_{d_j};
end while
Output $\hat{\mathcal{C}}$;

2.2. Analysis

In this subsection, we present some analyses on the approximation factor and running time of the proposed algorithm.

Theorem 1. The proposed algorithm provides an approximation solution which is within a factor of 4 of the optimal for the UDC problem.

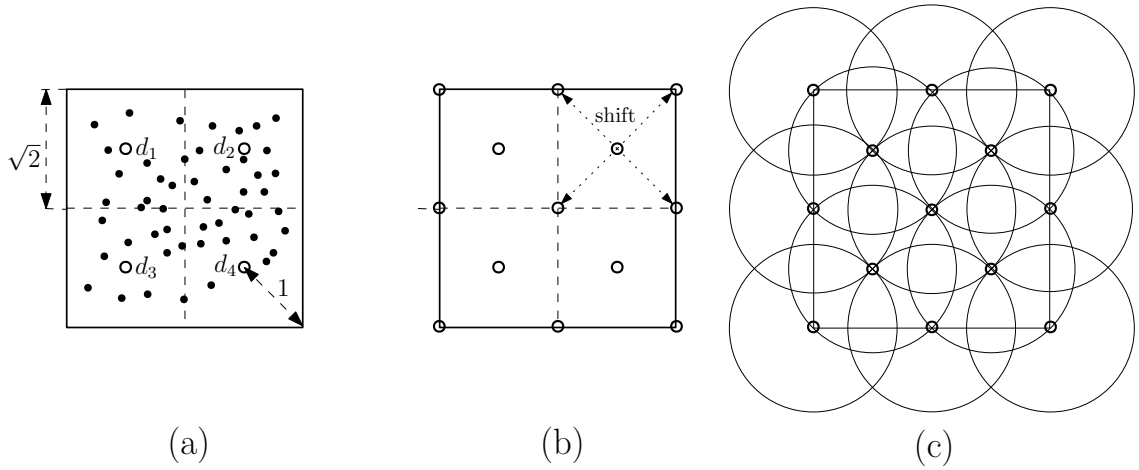

Figure 1: (a) Rounding points to the central-points (∘) of a grid with side length $\sqrt{2}$. (b) Shifting central-points to four directions by unit size. (c) The PCS set which is used by Algorithm 1.

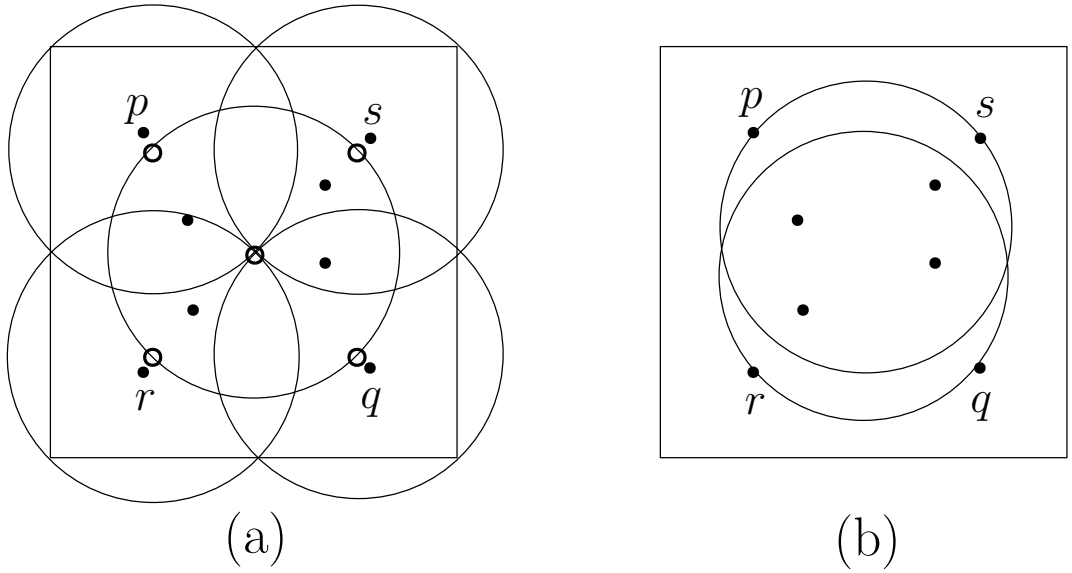

Figure 2: (a) Algorithm 1 has to use five disks for covering all points. (b) An optimal solution with two disks which can cover all points.

Proof. Let $\hat{\mathcal{C}}$ be the produced solution by Algorithm 1, and $\mathcal{C}_{opt}$ be the optimal solution for a point set. To simplify, assume that the set of points is inside a square Q with the side length $2\sqrt{2}$, as shown in Figure 1 (a). After rounding these points to the central-points on a grid with side length $\sqrt{2}$, four points d_1, d_2, d_3 and d_4 will be obtained which are shown with small circle points (∘) in Figure 1 (a). Then, $\mathcal{D} = \{d_1, d_2, d_3, d_4\}$ is the first Potential Circle Set (PCS) for the point set $\mathcal{P}$. Subsequently, by shifting these points into four directions with unit size, the number of points of the set PCS expands to 13 points (see Figure 1 (b)). Note that the proposed algorithm is a deterministic algorithm and it iteratively chooses a disk with maximum coverage in the remaining point set, until all points are covered. Therefore, in the following examples, it should be noted that the algorithm selects disks in order of maximum coverage, respectively. Moreover, a disk is used by Algorithm 1, if points inside it can not be covered by any disk with more coverage of the PCS set.

It is obvious that every set of points inside the square Q can be covered by one to five disks of the PCS set in Figure 1 (c). Now, in order to show how is the relationship between the set of disks which is selected by the algorithm, and the set of disks in an optimal solution for points inside a square Q, we first consider the following two observations.

Observation 1. For a set of points inside the square Q, it is not possible to cover a set of points that the proposed algorithm covers with five adjacent disks in the optimal solution with one disk.

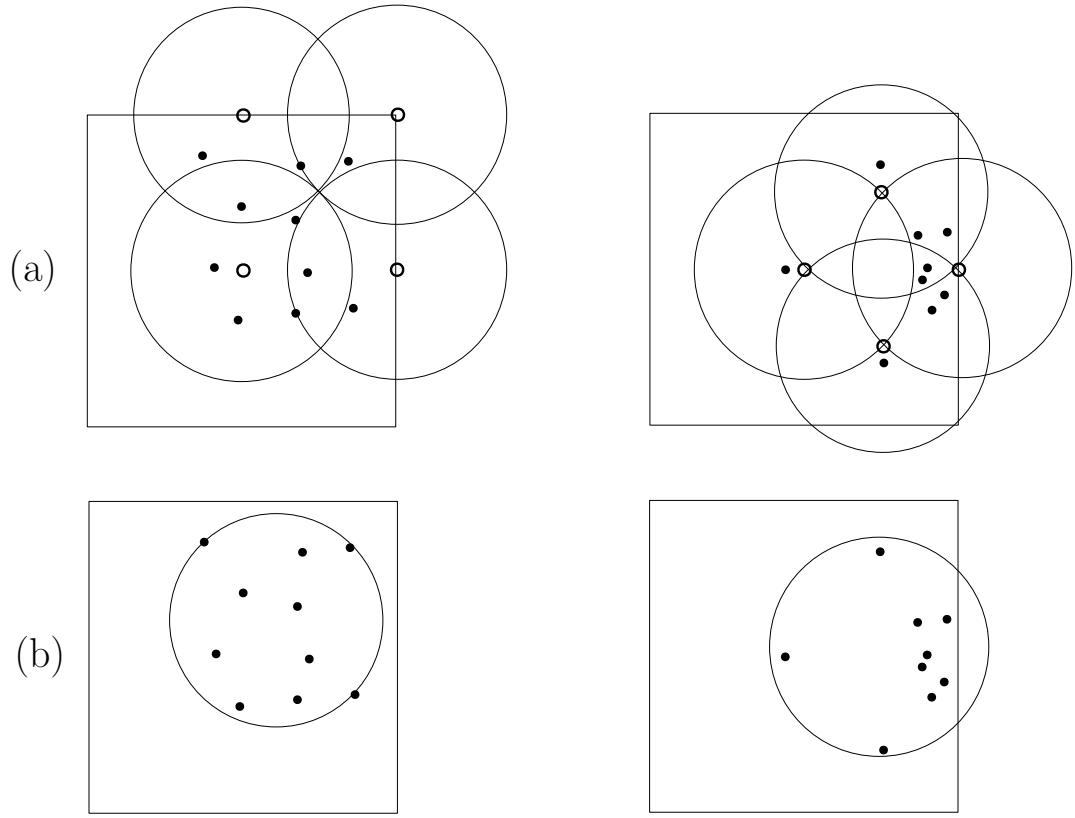

Figure 3: (a) Algorithm 1 has to use four disks to cover all points. (b) An optimal solution that can cover all points with one disk.

For this purpose, consider the set of points presented in Figure 2. In fact, this example shows a case where five disks can be adjacent as well as an extreme case that the proposed algorithm has to use five disks to cover them. But it can be clearly seen that the optimal solution for this point set needs at least two disks to cover this point set in the square Q, because the distance between two points p and q, and the distance between two points r and s are more than 2. And so, they cannot be covered with a unit disk with the radius of one. It should be noted that if two points p and r, or two points q and s go near to each other, then, the number of disks required by Algorithm 1 to cover them is reduced to four or three disks, respectively.

Observation 2. There are some point sets so that, while the proposed algorithm needs four adjacent disks to cover the points, an optimal solution can cover these points with one disk.

The given examples in Figure 3, represent this observation. Therefore, two observations 1 and 2 show that in the worst case for a set of points in Q (a square with side length $2\sqrt{2}$), that is covered by Algorithm 1 with four adjacent disks, it is possible to cover occasionally this point set with one disk. So, let $|\hat{\mathcal{C}}|$ be the number of disks in a solution $\hat{\mathcal{C}}$ produced by Algorithm 1, and $|\mathcal{C}_{opt}|$ be the number of disks in an optimal solution $\mathcal{C}_{opt}$, then, for a set of points in a square Q, we have: $|\hat{\mathcal{C}}| \leqslant 4 \cdot |\mathcal{C}_{opt}|$.

Now check if this ratio is also held in the general case. To this end, we examine below different situations. First, assume that there are two adjacent squares Q and Q', so that in each of them there are sets of points. In Figure 4, we illustrate two examples where the proposed algorithm covers points in both squares with six or five disks (see Figure 4 (a)). In this case, an optimal solution can cover these point sets by two disks (see Figure 4 (b)). Note that in both examples of Figure 4, we consider the most extreme cases that the algorithm has to use six or five disks for covering all points.

However, the example of Figure 5 shows two adjacent squares Q and Q' such that after running the proposed algorithm, square Q requires three disks and square Q' requires one disk to cover their interior points. In addition, an optimal solution can cover all points inside these two squares with one disk. Therefore, this example shows that the ratio $|\hat{\mathcal{C}}| \leqslant 4 \cdot |\mathcal{C}_{opt}|$ can also occur for two adjacent squares.

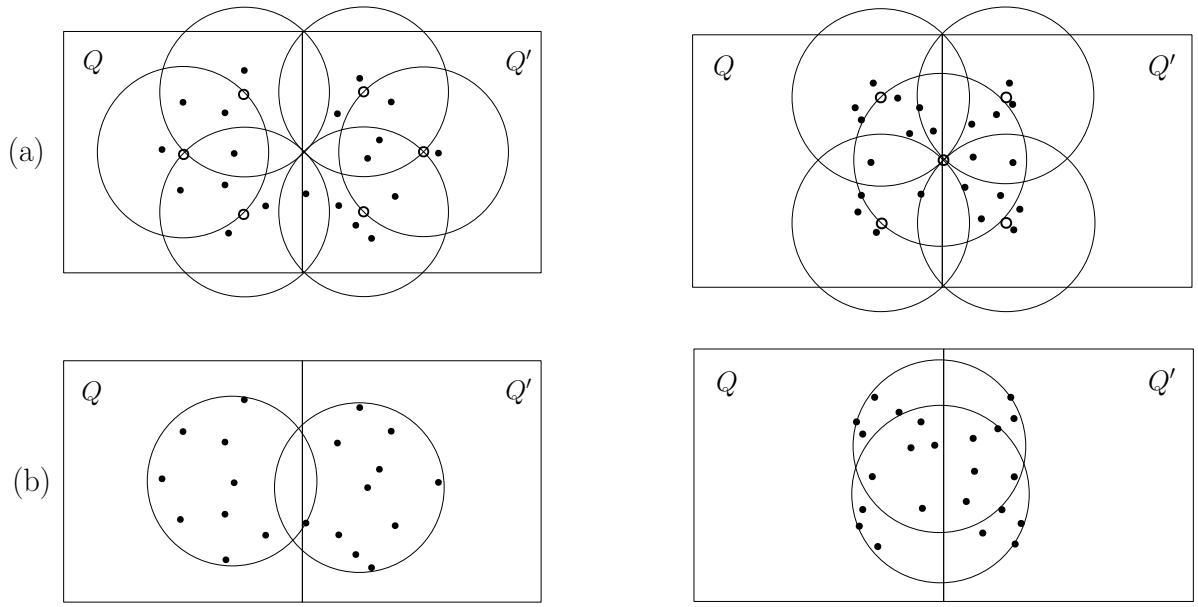

Figure 4: (a) Algorithm 1 has to use six or five disks to cover all points inside two square Q and Q'. (b) An optimal solution which covers all points with two disks.

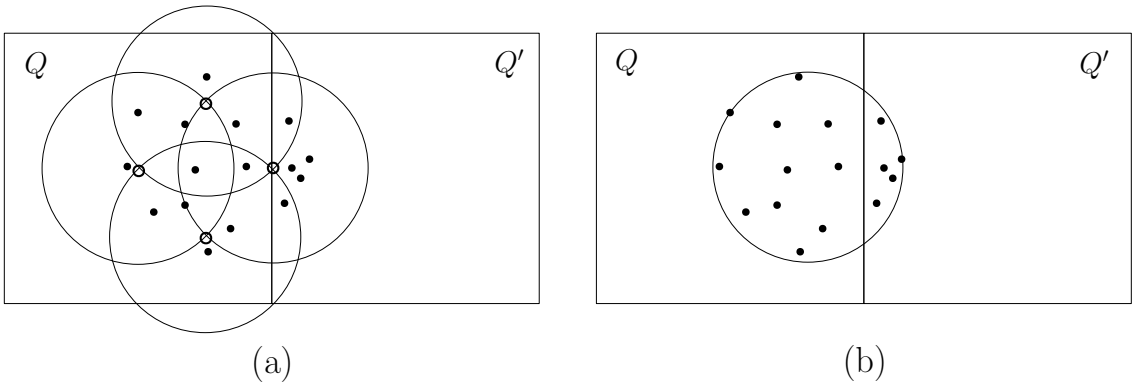

Figure 5: (a) Algorithm 1 has to use four disks to cover all points inside two square Q and Q'. (b) An optimal solution which covers all points with a disk.

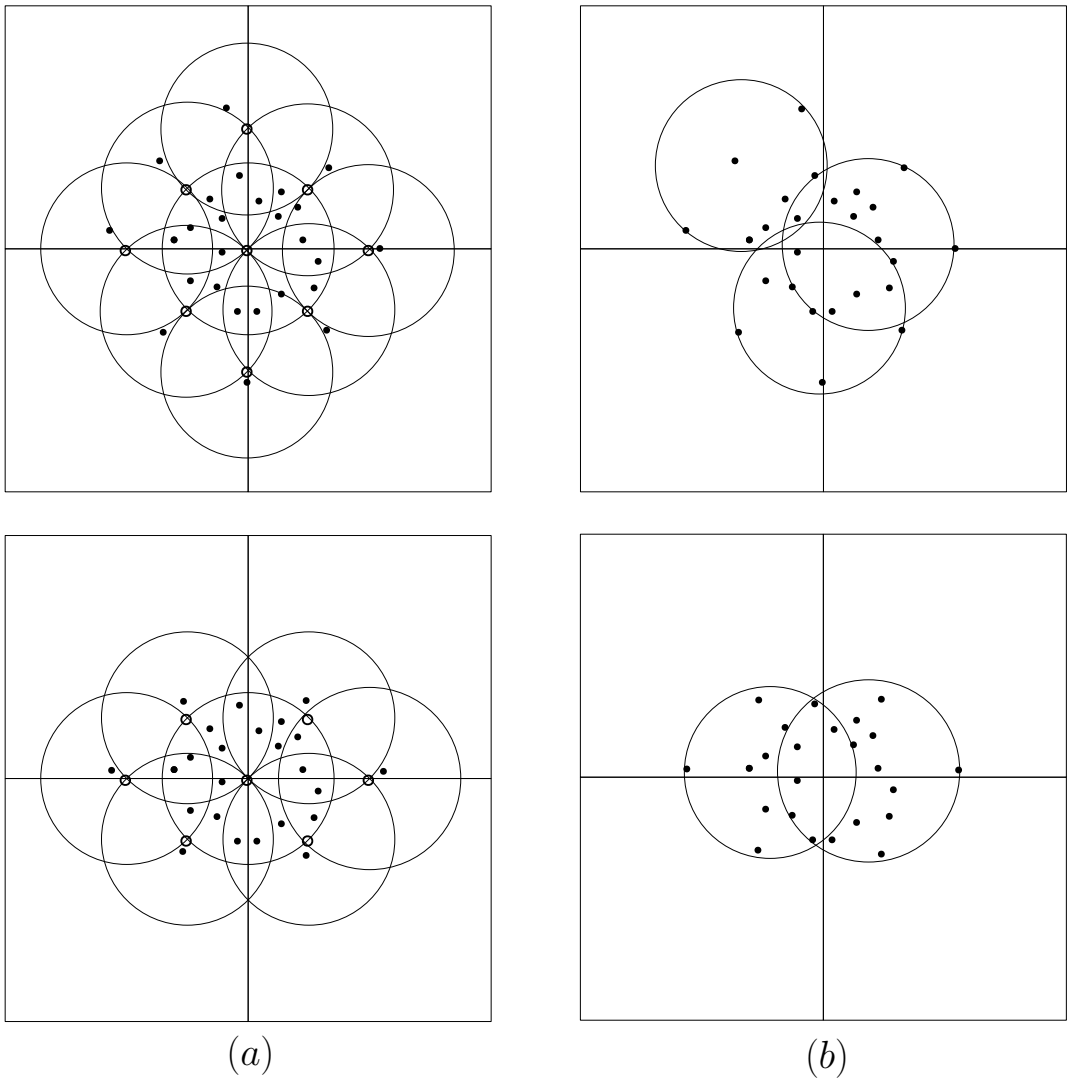

Figure 6: (a) Algorithm 1 has to use nine or seven disks to cover all points. (b) Optimal solutions with three and two disks which can cover all points.

Now, let us check the points inside four adjacent squares. Let us consider two worst case examples as shown in Figure 6 (a), when a set of points is gathered in the corner of the four squares. Then, the proposed algorithm needs nine or seven disks to cover these point sets, while the optimal solutions use two or three disks, respectively (see Figure 6 (b)), and so, there is a worst case ratio of $|\hat{\mathcal{C}}| \leqslant \frac{7}{2} \cdot |\mathcal{C}_{opt}|$ for four adjacent disks. Overall, these above extreme examples show that for each set of points, the ratio $|\hat{\mathcal{C}}| \leqslant 4 \cdot |\mathcal{C}_{opt}|$ is established, between the solution obtained by the proposed algorithm and the optimal solution. Therefore, this completes the proof. □

Theorem 2. The running time of the proposed algorithm is $O(n^2)$.

Proof. Rounding the set of n points to a grid takes $O(n)$ time. After rounding to the grid, we have $m = |\mathcal{D}|$ unit disk, which cover the point set, and by shifting the points of $\mathcal{D}$ into four directions, its size increases to a constant $d = |\mathcal{D}| \leqslant 5m$. What remains to check is the runtime of the **while** loop. There is a set $\mathcal{P}$ of n points, and a set PCS of size $d = |\mathcal{D}|$. We know that in each iteration of the **while** loop a disk which covers the maximum number of points is found and points inside it left out from the set $\mathcal{P}$. Let $|\mathcal{P}_i|$ be the number of points of $\mathcal{P}$ that remains at the beginning of the ith iteration. On the other hand, finding a disk with maximum coverage in the ith iteration takes $(d-(i-1))|\mathcal{P}_i|$ time. If the **while** loop takes t iterations, we have:

$$T(n) = T(|\mathcal{P}_2|) + d|\mathcal{P}_1|,$$

$$
\begin{aligned}
&= T(|\mathcal{P}_3|) + (d-1)|\mathcal{P}_2| + d|\mathcal{P}_1|,\\
&\vdots\\
&= (d-t)|\mathcal{P}_t| + \cdots + (d-1)|\mathcal{P}_2| + d|\mathcal{P}_1|,\\
&\leqslant d|\mathcal{P}_t| + \cdots + d|\mathcal{P}_2| + d|\mathcal{P}_1|,
\end{aligned}
$$

We know that in the worst case $d = O(n)$, and $|\mathcal{P}_i| < |\mathcal{P}_{i-1}|$, and $|\mathcal{P}_i| < n$, for $i > 1$. Moreover, we know that $|\mathcal{P}_1| = n$, and $|\mathcal{P}_1| + \sum_{i=2}^{t} |\mathcal{P}_i| \leqslant cn$, where c is a small constant. So, we have:

$$
\begin{aligned}
T(n) &\leqslant d(|\mathcal{P}_t| + \cdots + |\mathcal{P}_2| + |\mathcal{P}_1|),\\
&\leqslant n(|\mathcal{P}_t| + \cdots + |\mathcal{P}_1|) \leqslant cn^2 \in O(n^2). \qquad (2.1)
\end{aligned}
$$

and the theorem is resulted. □

2.3. Improving the running time

In this subsection, we improve the running time of our proposed algorithm in the previous subsection by using some more complex data structures. One of the factors increasing the running time of Algorithm 1 is the phase of calculating the number of points in the disks of the PCS set, which takes $O(n)$ time for each disk because we have to check all of the n points. Therefore, the main challenge is to reduce the running time of this part of the algorithm. Our solution to this is to restrict the search to a square with center point d_i and side length 2 around each point $d_i \in \mathcal{D}$ (see Figure 7). Clearly, in this case, the running time of calculating the number of points in the unit disk is limited to the number of points in the square, which is less time-consuming than the search in all points. What helps us with solving this problem is using range trees. Thus, we use an improved version of the range trees which uses the fractional cascading. This version of the range tree is known as a layered range tree. The layered range tree is a data structure that has the following properties.

Theorem 3. ([9]) Suppose, $\mathcal{P}$ be a set of n points in the dimension d, for $d \geqslant 2$. A layered range tree for $\mathcal{P}$ uses $O(n \log^{d-1} n)$ storage and can be constructed in $O(n \log^{d-1} n)$ time. With this range tree, one can report points of $\mathcal{P}$ inside a rectangular query range in $O(\log^{d-1} n + k)$ time, where k is the number of reported points.

Now, suppose that for the PCS set $\mathcal{D}$, corresponding to each disk $d_i \in \mathcal{D}$, we put a square Q_i with side length 2, where its center is in point d_i. If we assume that the number of points inside square, Q_i is represented by k_i. In this case, using Theorem 3, in $O(n \log n)$ time a layered range tree can be created on the point set $\mathcal{P}$, and for each rectangular domain Q_i, the number of points inside it can be reported in $O(\log n + k_i)$ time, where k_i is the number of reported points. Then, the running time of computing the number of points inside a unit disk with center point d_i, is limited to the number of points inside the square Q_i, and it can be computed in $O(\log n + k_i)$ time. But, after computing the number of points in each disk in the list $\mathcal{D}$ by using the layered range tree, we need to find a disk d_m with maximum points, as well as left out points inside it from the point set $\mathcal{P}$. Suppose, corresponding to the list $\mathcal{D} = \{d_1, d_2, \cdots, d_{m'}\}$ of the PCS set, we compute a list $\varphi = \{\varphi_1, \varphi_2, \ldots, \varphi_{|\mathcal{D}|}\}$ for the number of points in each disk by using the layered range tree, such that φ_i is the number of points inside the disk d_i in the present. In order to easily find the disk with the maximum number of points in the list φ in each step of the algorithm, we use the max-heap tree data structure and store elements of the set φ in it. On the other hand, we know that a max-heap tree for n elements can be created in $O(n)$ time, and actions such as insertion and deletion can be done in time $O(\log n)$ on it. In this case, it is easy to have the maximum value of the set φ at the root of the max-heap tree in each step of the algorithm in constant time. However, by removing the points inside the disk d_m with the maximum number of points from the point set $\mathcal{P}$, we may have to update the number of points inside the adjacent disks to the disk d_m with the distance less than 2, which can be done by using the max-heap tree in $O(\log n)$ time. Therefore, the following result can be found.

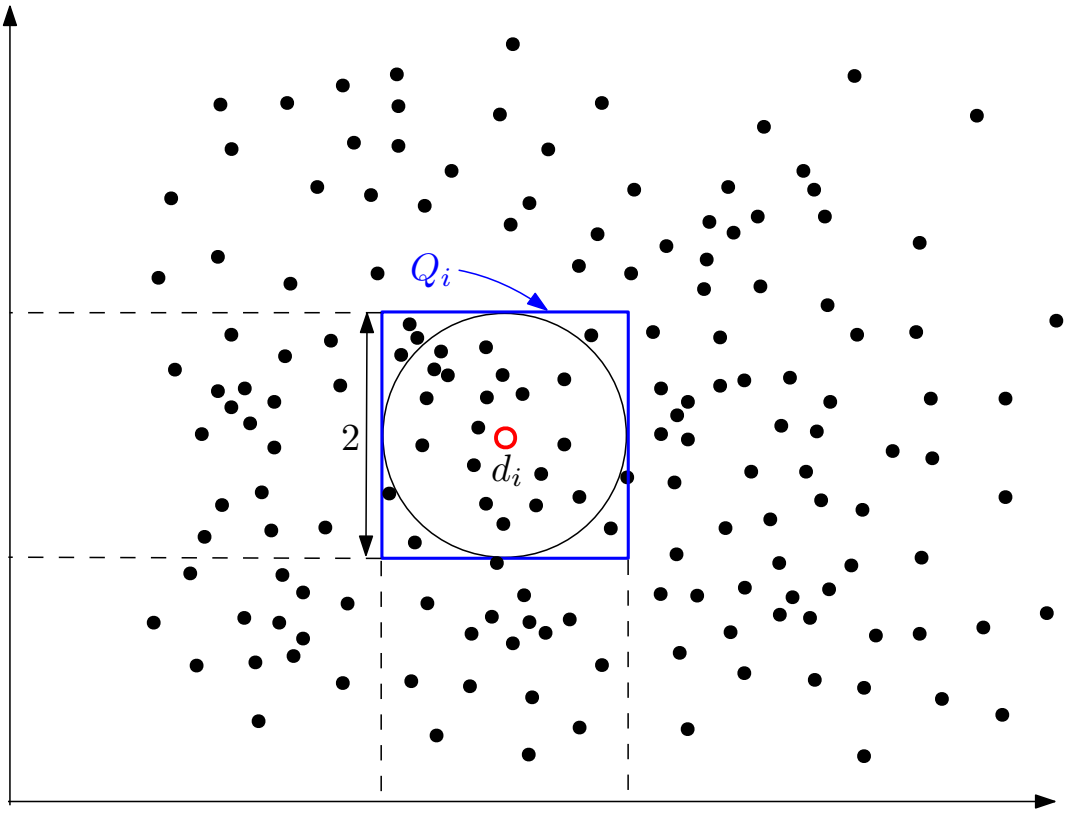

Figure 7: Using the layered range tree structure, leads to searching the points inside the disk d_i to points within the rectangular range Q_i, instead of searching the entire points in the plane.

Theorem 4. For a set $\mathcal{P}$ of n points on the plane, the proposed algorithm can provide a 4-approximation for the UDC problem in $O(n \log n)$ time.

Proof. As mentioned above, rounding the points to the central-points of a grid takes $O(n)$ time. We construct a layered range tree on $\mathcal{P}$ at time $O(n \log n)$. Let $\mathcal{D} = \{d_1, d_2, \cdots, d_{m'}\}$ be the PCS set which is the set of rounding points after shifting to four directions with unit size. Then, for each square Q_i, by side length 2 and center point d_i, the number of points inside it can be found by using the layered range tree structure in $O(\log n + k_i)$ time, where k_i represents the number of reported points. So, the number of points inside the unit disk with center d_i, can be computed in $O(\log n + k_i)$ time. Consider list $\varphi = \{\varphi_1, \varphi_2, \ldots, \varphi_{|\mathcal{D}|}\}$, where φ_i is the number of points inside the disk $d_i \in \mathcal{D}$. Creating a max-heap tree on set φ takes $O(n)$ time. A disk with the maximum number of points can also be found in $O(1)$ time by using the max-heap tree. Then, in the ith iteration of the algorithm, for the disk d_m with the maximum number of points, we left out the points inside it from $\mathcal{P}$. Moreover, using the max-heap tree, we can update the number of points in the adjacent disks of disk d_m, which may have common points with disk d_m. This operation in the max-heap tree can be done in $O(\log |\mathcal{D}_i|)$ time, when $|\mathcal{D}_i|$ is the size of the PCS, $\mathcal{D}$, in the ith iteration of the algorithm. Overall, the running time of the algorithm involves the computation of a layered range tree, the creating of a max-heap tree, and the running time associated with the first iteration of the **while** loop:

$$T_1(n) = O(n \log n) + O(n) + [\sum_{i=1}^{|\mathcal{D}|}(O(\log n + k_i) + O(1))], \tag{2.2}$$

in addition to the running time of the other iterations of the **while** loop includes finding maximum covering disk using max-heap tree and updating the elements of the max-heap tree, which is:

$$T_2(n) = \sum_{i=2}^{t}(O(1) + O(\log |\mathcal{D}_i|)). \tag{2.3}$$

Since, the size of $\mathcal{D}$ in the worst case is $|\mathcal{D}| = O(n)$, and $\sum_{i=1}^{|\mathcal{D}|} O(k_i) = O(n)$. So, for relation (2), we have: $T_1(n) \leqslant O(n \log n)$. Similarly, since $|\mathcal{D}_i| < O(n)$, for $i > 1$. So, for relation (3), we have: $T_2(n) \leqslant O(|\mathcal{D}| \log |\mathcal{D}|) \leqslant O(n \log n)$. Therefore, the overall running time of the algorithm is: $T(n) = T_1(n) + T_2(n) \leqslant O(n \log n)$, and this completes the proof. □

3. Conclusion

We considered the UDC problem in the plane, which is known as an NP-hard problem. We proposed a simple 4-approximation algorithm with running time $O(n \log n)$ for this problem. This is not too hard to show that this approach can be extended to a constant factor approximation algorithm for covering a set of points by balls in the three dimensions, by using the layered range tree with running time $O(n \log^2 n)$.

References

[1] R. J. Fowler, M. S. Paterson, S. L. Tanimoto, Optimal packing and covering in the plane are NP-complete, Information Processing Letters, 12(3) (1981) 133-137.

[2] D. S. Hochbaum, W. Maass, Approximation schemes for covering and packing problems in image processing and VLSI, Journal of ACM, 32 (1985) 130-136.

[3] T. F. Gonzalez, Covering a set of points in multidimensional space, Information Processing Letters, 40(4) (1991) 181-188.

[4] H. Bronnimann, M. T. Goodrich, Almost optimal set covers in finite VC-dimension, Discrete and Computational Geometry, 14(4) (1995) 463-479.

[5] M. Franceschetti, M. Cook, J. Bruck, A geometric theorem for approximate disk covering algorithms, Technical report, (2001).

[6] B. Fu, Z. Chen, M. Abdelguerfi, An almost linear time 2.8334-approximation algorithm for the disc covering problem, In Proceedings of the 3rd International Conference on Algorithmic Applications in Management (AAIM'07), (2007) 317-326.

[7] P. Liu, D. Lu, A fast 25/6-approximation for the minimum unit disk cover problem, CoRR, abs/1406.3838, (2014).

[8] A. Biniaz, P. Liu, A. Maheshwari, M. Smid, Approximation algorithms for the unit disk cover problem in 2D and 3D, Computational Geometry: Theory and Applications, 60 (2017) 8-18.

[9] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, Springer Berlin Heidelberg, (2008) 95-115.

Please cite this article using:

Mahdi Imanparast, Seyed Naser Hashemi*, A simple greedy approximation algorithm for the unit disk cover problem, AUT J. Math. Com., 1(1) (2020) 47-55
DOI: 10.22060/ajmc.2018.3044