Output sensitive algorithms for covering many points

Hossein Ghasemalizadeh, Mohammadreza Razzazi

Discrete Mathematics and Theoretical Computer Science DMTCS vol. **17:1**, 2015, 309–316

Output sensitive algorithms for covering many points[*]

Hossein Ghasemalizadeh[†] Mohammadreza Razzazi[‡]

SSRD Lab, Computer Engineering and Information technology dep., Amirkabir Univ. of technology, Tehran, Iran

received 10th Nov. 2012, revised 10th Oct. 2014, accepted 27th Mar. 2015.

In this paper we devise some output sensitive algorithms for a problem where a set of points and a positive integer, m, are given and the goal is to cover a maximal number of these points with m disks. We introduce a parameter, ρ, as the maximum number of points that one disk can cover and we analyse the algorithms based on this parameter. At first, we solve the problem for $m = 1$ in $O(n\rho)$ time, which improves the previous $O(n^2)$ time algorithm for this problem. Then we solve the problem for $m = 2$ in $O(n\rho+\rho^3 \log \rho)$ time, which improves the previous $O(n^3 \log n)$ algorithm for this problem. Our algorithms outperform the previous algorithms because ρ is much smaller than n in many cases. Finally, we extend the algorithm for any value of m and solve the problem in $O(mn\rho + (m\rho)^{2m-1} \log m\rho)$ time. The previous algorithm for this problem runs in $O(n^{2m-1} \log n)$ time and our algorithm usually runs faster than the previous algorithm because $m\rho$ is smaller than n in many cases. We obtain output sensitive algorithms by confining the areas that we should search for the result. The techniques used in this paper may be applicable in other covering problems to obtain faster algorithms.

Keywords: covering with disks, output sensitive algorithm, computational geometry

1 Introduction

In the classic covering problem, a set of points are given, and the goal is to determine the minimum number of unit radius disks required to cover all of the points. This problem is NP-Hard [1] . In the maximum covering problem, the number of unit radius disks to be used is given as m, and the goal is to cover the most points possible. We call this problem $MostPoints(P, m)$. This problem is NP-Hard if we consider m as a generalized input because the classic covering problem can be solved by iteratively solving $MostPoints(P, m)$ to reach the minimum value of m such that m unit disks cover all points.
$MostPoints(P, 1)$ was first introduced by Drezner [2], and he solved it in $O(n^2 \log n)$ time. To solve the problem, he replaced every point with a disk centered at that point and obtained the maximum depth in the arrangement of disks. Later, Chazelle and Lee [3] solved $MostPoints(P, 1)$ in $O(n^2)$ time. Aronov and Har-Peled [5] showed that $MostPoints(P, 1)$ is a 3-SUM hard problem, which means that, the problem belongs to a class of problems for which no sub-quadratic algorithms are known [4]. However, some approximation algorithms have been given. In [5], Aronov and Har-peled gave a $(1 - \varepsilon)$-approximation

[*]This research was in part supported by a grant from IPM (No. CS1391-4-16)
[†]Email: ghasemalizadeh@aut.ac.ir
[‡]Corresponding author email: razzazi@aut.ac.ir

algorithm for the number of covered points and it runs in $O(n\varepsilon^{-2}\log n)$ time. Figueiredo and Fonesca [6] solved a different version of the $MostPoints(P,1)$ problem: Given a set of n points with positive real weights in d-dimensional space, they consider an approximation to the problem of placing a unit ball, such that the sum of the weights of the points enclosed by the ball is maximized. Given an approximation parameter $\varepsilon < 1$, they presented an $O(\frac{n}{\varepsilon^{d-1}})$ expected time algorithm that determines a ball of radius $1+\varepsilon$ enclosing a weight at least as large as the weight of the optimal unit ball.
As stated before, $MostPoints(P,m)$ for $m > 1$, is NP-Hard . Trivial greedy algorithm is a $(1-\frac{1}{e})$-approximation algorithm for it [7]. The greedy algorithm first finds a disk that covers a maximal number of points in $O(n^2)$ time. To pick up the second disk, it removes the points located in the first disk and finds the disk that covers a maximal number of points. It repeats this process until m disks are picked up. This yields a $(1-\frac{1}{e})$-approximation algorithm that runs in $O(mn^2)$ time. The first $(1-\varepsilon)$-approximation algorithm for this problem was given in [8] that runs in $O(n\log n+n\varepsilon^{-6m+6}\log(\frac{1}{\varepsilon}))$ time. We present a polynomial time approximation schema for this problem in [9], which runs in $O((1+\varepsilon)mn+\varepsilon^{-1}n^{4\sqrt{2}\varepsilon^{-1}+2})$ time.

The trivial method to optimally solve $MostPoints(P,m)$ is to consider all subsets of size m of n^2 possible disks and find the subset that covers the most number of points. This method takes $O(n^{2m})$ time. In [8], de Berg et al. gave an improved algorithm for this problem. They first solved the problem for $m = 2$ in $O(n^3\log n)$ time. To solve the problem for $m > 2$, they fixed every subset of $m-2$ disks, and they found the best two disks after removing the points contained in the $m-2$ disks. This takes $O(n^{2m-1}\log n)$ time. We call this algorithm, $MostPointsBCH$.
In Section 2, we present an output sensitive algorithm for the $MostPoints(P,1)$ problem, which runs in $O(n\rho)$ time, where ρ is the maximum number of points that one disk can cover. In Section 3 we devise an algorithm for the $MostPoints(P,2)$ problem, which runs in $O(n\rho+\rho^3\log\rho)$ time. In Section 4 we extend the algorithm for $MostPoints(P,m)$ and in Section 5 we conclude the paper.

2 Output sensitive algorithm for MostPoints(P,1)

Consider ρ as the maximum number of points that one disk can cover in the point set P of n points. In this section, we devise an algorithm for $MostPoints(P,1)$ that solves the problem in $O(n\rho)$ time. Chazzle and Lee [3] provide the best known algorithm for this problem, which solves the problem in $O(n^2)$ time, and we refer to it as the $MostPointsCL$ algorithm. Now we explain our algorithm.
Consider the bounding box of P and extend to the left two units and then down two units. Draw a 4×4 cell-sized grid on this bounding box and call it G_o. Shift G_o two units upward and call the new grid G_u. Shift G_o two units to the right and call the new grid G_r. Shift G_u two units to the right and call the new grid G_{ur}. One can see that for any unit disk d there is a grid $G\in\{G_o,G_u,G_r,G_{ur}\}$ such that d is fully contained in a single cell of G as shown in Figure 1.

Now, for each non-empty grid cell C of every grid, compute the optimal disk on the set of points lying inside C using the $MostPointsCL$ algorithm. C contains at most $c\rho$ points for a constant value $c\leq 9$, as shown in Figure 2, and the time spent for cell C is $O(|C|^2)$, where $|C|$ denotes the number of points in C. The total number of points over all cells C is $4n$, since each point is contained in exactly one cell per grid. Assume that we have l cells in the grids. To obtain the running time of the algorithm, we sum the

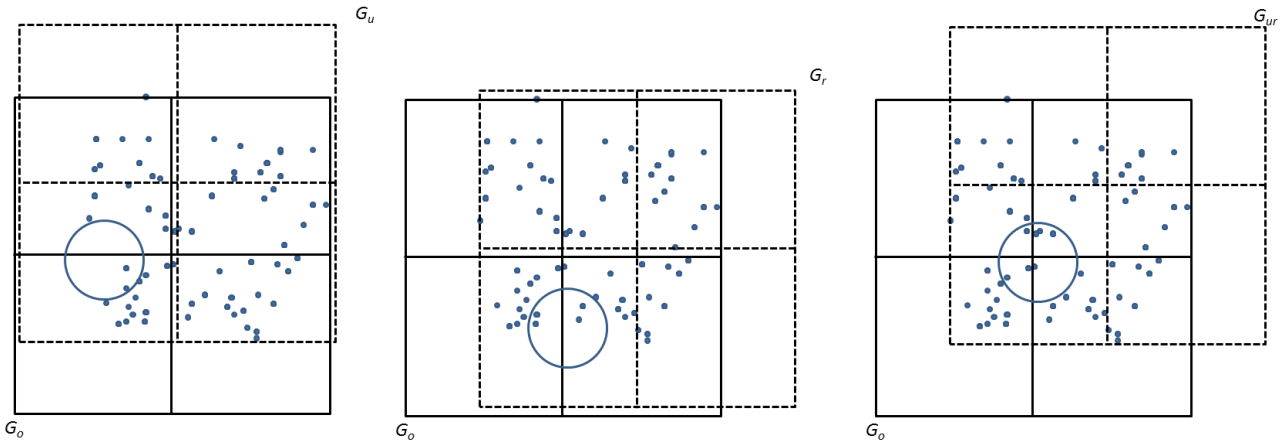

Fig. 1: Every unit disk is fully contained in a single cell of a grid $G \in \{G_o, G_u, G_r, G_{ur}\}$.

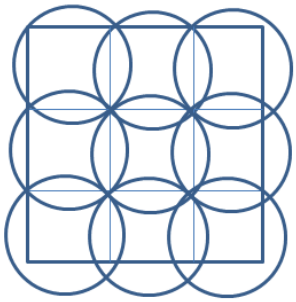

Fig. 2: Every 4×4 square can be covered by at most 9 unit disks.

running time of the algorithm on all cells of the grids. The total running time of the algorithm is:

$$\sum_{i=1}^{l} |C_i|^2 \leq \sum_{i=1}^{l} |C_i| \times 9\rho \leq 9\rho \times 4n = O(\rho n)$$

We call this algorithm $MostPointsShiftedGrids(P)$, which solves the $MostPoints(P, 1)$ problem in $O(n\rho)$ time.

3 Output sensitive algorithm for MostPoints(P,2)

In this section, we describe our algorithm for $MostPoints(P, 2)$. Define g_1 as one of the disks that covers a maximal number of points from the point set P. Define g_2 as one of the disks that covers a

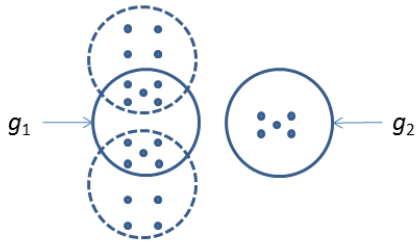

Fig. 3: This figure shows the greedy solution and the optimal solution for $MostPoints(P, 2)$ in a sample point set. In this example, the greedy algorithm returns g_1 and g_2 which cover 15 points together, whereas the disks in the optimal solution, the two disks specified with dashed lines, cover 18 points together.

maximal number of points after removing the points located in g_1. g_1 and g_2 can be obtained using the $MostPointsShiftedGrids$ algorithm of Section 2. g_1 and g_2 are the result of the greedy algorithm for $MostPoints(P, 2)$. An example is given in Figure 3. In this example, the optimal solution covers 18 points, whereas the greedy algorithm covers 15 points. As illustrated in this example, the two disks of the optimal solution have common points with g_1. Intuitively, either the disks of the optimal solution have common points with g_1 or the greedy algorithm obtains the optimal solution. Lemma 1 proves this claim. By the best two disks, we mean the optimal solution for $MostPoints(P, 2)$. Let D be a set of disks. The function $Cover(D)$ denotes the set of points covered by the disks in D and $|Cover(D)|$ refers to the number of these points.

Lemma 1 *Let g_1 be one of the disks that covers a maximal number of points in the point set P and g_2 be one of the disks that covers a maximal number of points after removing the points located in g_1. Let o_1 and o_2 be the two disks that together cover a maximal number of points among any combination of two disks. Either both o_1 and o_2 have common points with g_1, or g_1 and g_2 cover a maximal number of points.*

Proof: Suppose that at least one of o_1 and o_2, say o_2, does not have any common point with g_1. Then, we have $|Cover(\{g_1, g_2\})| \geq |Cover(\{g_1, o_2\})|$, because g_2 covers a maximal number of points of the points not covered by g_1. We also have $|Cover(\{g_1, o_2\})| \geq |Cover(\{o_1, o_2\})|$ because g_1 covers at least the same number of points as o_1 (since it is a disk that covers the maximum number of points) and g_1 does not have any point in common with o_2. Thus $|Cover(\{g_1, g_2\})| \geq |Cover(\{o_1, o_2\})|$, which implies that the total number of the points covered by g_1 and g_2 is maximal. □

Based on Lemma 1, we should look for the best two disks in the disks that have common points with g_1. All such disks are contained within a circle having the same center as g_1 and a radius of 3. We call this region Ng_1 as shown on the left side of Figure 4. Lemma 2 bounds the number of points in Ng_1.

Lemma 2 *Let ρ be the maximum number of points that can be covered by a disk. The number of points in Ng_1 is $O(\rho)$.*

Proof: We can cover Ng_1 with at most 16 unit disks. We can divide Ng_1 into four quarters and each quarter can be covered with four disks as shown on the right side of Figure 4. To cover each quarter, consider four points with equal distance, located on the perimeter of each quarter. Every two points out of these four points determines one disk center inside the quarter. This gives us the centers of three disks out

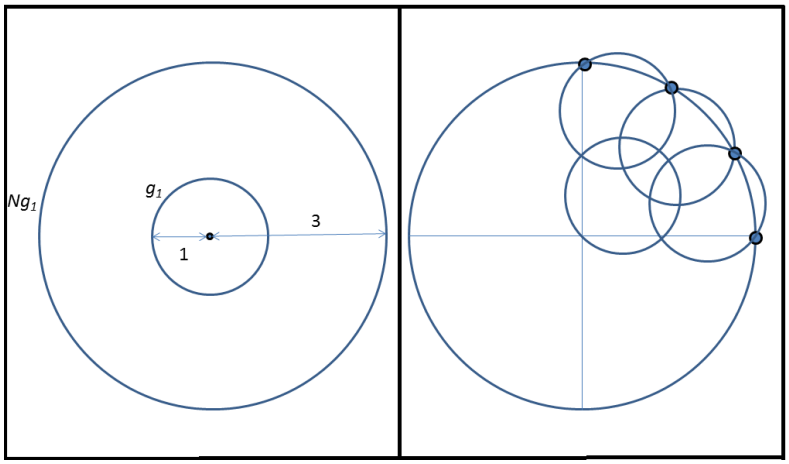

Fig. 4: The disks which have common points with g_1 are located in a disk with the same center as g_1 and a radius of 3, which is shown on the right side of this figure. We call this region Ng_1. We can cover the Ng_1 region with at most 16 unit disks as depicted in the left side of this figure.

of these four disks. The fourth disk cuts through the center of the quarter and the intersections of the first disk and last disk with the edges of the quarter. Every one of these disks covers no more than ρ points. So the number of points in Ng_1 is at most 16ρ points, which is $O(\rho)$ points. □

Using Lemma 1 and Lemma 2, the best two disks can be g_1 and g_2, or the two disks around g_1 that cover a maximal number of points. To find the best two disks in P, first we find g_1 and g_2 in $O(n\rho)$ time. Then, we determine the points in Ng_1, and we run $MostPointsBCH$ [8] on the points located in Ng_1. This algorithm obtains the best two disks in Ng_1, in $O(\rho^3 \log \rho)$ time. Finally, we compare the best two disks in Ng_1 with g_1 and g_2 and consider the one that covers more points as the optimal solution for $MostPoints(P, 2)$. Since ρ can be much smaller than n in practice, our algorithm which runs in $O(n\rho + \rho^3 \log \rho)$ time, is an improvement over the algorithm of [8] that runs in $O(n^3 \log n)$ time.

4 The algorithm for MostPoints(P,m)

The algorithm of Section 3 can be extended to cover the maximum number of points with m disks. From now on, by the best j disks, we mean a set of j disks that cover the maximum number of points. The proposed algorithm is an iterative algorithm and uses the following fact, which will be proven in Lemma 3: the best j disks are among the disks that have some common points with the best $j-1$ disks; otherwise, the best $j-1$ disks plus one disk that covers a maximal number of points after removing the points located in the best $j-1$ disks, are the best j disks. The algorithm starts from $j=2$ and repeats this process until the best m disks are found. We call this algorithm $MostPointsIterative(P, m)$, which returns a set of m disks that cover the maximum number of points in the point set P. In this algorithm we use $MostPointsBCH(Q, j)$ to refer to the algorithm of [8], which returns a set of j disks that cover a maximal number of points in the point set Q. We also use $MostPointsShiftedGrids(P)$ to refer to the algorithm of Section 2 for covering the maximum number of points with one disk in the point set P. Let

D be a set of unit disks. As stated before, $Cover(D)$ refers to the points covered by the disks in D. We also define $Neighbour(D)$ as the points in the disks with the same centers as the disks in D and a radius of 3. Let OPT_i denote the best i disks. The outline of the algorithm is as follows:

Algorithm 1 $MostPointsIterative(P, m)$

Require: A set P of n points in the plane, and a positive integer m
  $g_1 = MostPointsShiftedGrids(P)$
  $OPT_1 = g_1$
  for $i = 2$ to m **do**
    $rp = P - Cover(OPT_{i-1})$
    $g_i = MostPointsShiftedGrids(rp)$
    $O_i = MostPointsBCH(Neighbor(OPT_{i-1}), i)$
    if $|Cover(OPT_{i-1} \cup g_i)| > |Cover(O_i)|$ **then**
      $OPT_i = OPT_{i-1} \cup g_i$
    else
      $OPT_i = O_i$
    end if
  end for
  return OPT_m

In Lemma 3 we prove the correctness of the above algorithm.

Lemma 3 *The **MostPointsIterative** algorithm obtains m disks that cover the maximum number of points.*

Proof: We prove the correctness of the algorithm by induction on the number of iterations. Assume that at the end of iteration i we have correctly computed OPT_i. Consider OPT_{i+1}, which consists of $i + 1$ disks. These disks may or may not have common points with $Cover(OPT_i)$. We consider both cases. In case 1 we consider that all $i + 1$ disks of OPT_{i+1} have common points with $Cover(OPT_i)$, and in case 2 we consider that some of these i disks do not have any common point with $Cover(OPT_i)$.

Case1: All $i + 1$ disks of OPT_{i+1} have common points with $Cover(OPT_i)$:

All disks that have common points with $Cover(OPT_i)$, have all points in $Neighbour(OPT_i)$ (refer to the definition of the $Neighbour$ function) . In line 6, the algorithm obtains the best $i + 1$ disks in the points of $Neighbour(OPT_i)$ using $MostPointsBCH$. So, in this case, the algorithm exactly computes OPT_{i+1}.
Case 2: Some of the disks in OPT_{i+1} do not have any common point with $Cover(OPT_i)$:

Consider that $F \in OPT_{i+1}$ is a disk that does not have any common point with $Cover(OPT_i)$. We can partition OPT_{i+1} into $\{OPT_{i+1} - F\}$ and F. By the definition of the $Cover$ function, we have the following equation:

$$\begin{aligned} OPT_{i+1} &= \{OPT_{i+1} - F\} \cup \{F\} \\ |Cover(OPT_{i+1})| &\leq |Cover(OPT_{i+1} - F)| + |Cover(F)| \end{aligned} \quad (1)$$

$\{OPT_{i+1} - F\}$ is a set of i disks. As OPT_i is a set of i disks that covers a maximal number of points, we have:

$$|Cover(OPT_{i+1} - F)| \leq |Cover(OPT_i)| \tag{2}$$

Furthermore, g_{i+1} is the disk that covers a maximal number of points after removing the points in $Cover(OPT_i)$. Therefore, g_{i+1} covers a maximal number of points among the disks that have no common points with OPT_i. Thus, we have:

$$|Cover(\{F\})| \ \leq \ |Cover(g_{i+1})| \tag{3}$$

By adding up two sides of inequality (2) and inequality (3) we have:

$$|Cover(\{OPT_{i+1} - F\})| + |Cover(\{F\})| \ \leq \ |Cover(OPT_i)| + |Cover(g_{i+1})| \tag{4}$$

From (1) and (4) we have:

$$|Cover(OPT_{i+1})| \leq \ |Cover(OPT_i)| + |Cover(g_{i+1})|$$

Thus in case 2, OPT_i and g_{i+1} cover the maximum number of points.
The algorithm compares the result of case 1 with the result of case 2, and considers the best one as OPT_{i+1}. So, the algorithm correctly computes OPT_{i+1}. The base case, $i = 1$, is correct and can be satisfied using the $MostPointsShiftedGrids(P)$ algorithm of Section 2. □

Lemma 4 obtains the running time of the ***MostPointsIterative*** algorithm.

Lemma 4 *The $MostPointsIterative$ algorithm runs in $O(mn\rho + (m\rho)^{2m-1} \log m\rho)$ time.*

Proof:
The algorithm runs in m iterations. At the start of the iteration i, we have found the best $i-1$ disks, and we are trying to find the best i disks. To find the best i disks, we obtain g_i in line 5, which takes $O(n\rho)$ time. We also look for the i disks covering the maximum number of points in $Neighbour(OPT_{i-1})$, in line 6. $Neighbour(OPT_{i-1})$ consists of $i-1$ disks, and based on Lemma 2, the neighbourhood of each disk can be covered with at most 16 disks. So the maximum number of points in $Neighbour(OPT_{i-1})$ is $16\rho(i-1)$, which is $O(i\rho)$ points. We search for the i disks that cover the maximum number of points in $O(i\rho)$ points. $MostPointsBCH$ is applied on these points, which takes $O((i\rho)^{2i-1} \log(i\rho))$ time. The algorithm runs in m iterations, so the running time is bounded by $O(n\rho) + \sum_{i=2}^{m}(O(n\rho) + (i\rho)^{2i-1} \log(m\rho))$, which is $O(mn\rho + (m\rho)^{2m-1} \log(m\rho))$. □

5 Conclusion and future works

In this paper we present two exact output sensitive algorithms for the problem of covering many points. In the first algorithm, for covering many points with one disk, we use shifted grids to confine the search area to several 4×4 squares. Each 4×4 square contains at most 9ρ points, where ρ is the maximum number of points that one disk can cover. The overall running time of the algorithm on all squares becomes $O(n\rho)$, which in many cases improves the $O(n^2)$ running time of the best previous known algorithm for this problem. In the second algorithm, for covering many points with m disks, we confine the regions

in which the resultant disks may reside using the greedy algorithm, and we search in these regions only. This leads to an output sensitive algorithm for this problem which runs in $O(mn\rho + (m\rho)^{2m-1} \log m\rho)$ time. This algorithm improves the $O(n^{2m-1} \log n)$ running time of the previous algorithm[8], in the cases where $m\rho$ is smaller than n.
Our algorithm can be improved by improving the running time of the algorithm in [8]. If some better algorithms for covering the most points with two disks are found, the running time of our algorithm and the algorithm of [8] will improve. Furthermore, by considering ρ as a parameter and using similar techniques, it may be possible to improve the running time of other covering algorithms.

6 Acknowledgement

We thank the referees for their helpful comments, which improved the paper structure and content. The authors also would like to thank 'Institute for Research in Fundamental sciences' (IPM) for their support of this project.

References

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

[2] Z. Drezner On a modified one-center model. *Management Science*, 27(7):848-851, 1981.

[3] B. M. Chazelle and D. T. Lee On a circle placement problem. *Computing*, 36(1):1-16, 1986.

[4] A. Gajentaan and M. H. Overmars On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry*, 5(3):165-185, 1995.

[5] B. Aronov and S. Har-Peled On approximating the depth and related problems. *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, 886-894, 2005.

[6] C. M. H. de Figueiredo and G. D. da Fonseca Enclosing weighted points with an almost-unit ball *Computational Geometry*, 109(21-22):1216-1221, 2009.

[7] D. S. Hochbaum and A. Pathria Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics*, 45(6):615-627, 1998.

[8] M. de Berg, S. Cabello, and S. Har-Peled Covering many or few points with unit disks. *Theory of Computing Systems*, 45(3):446-469, 2009.

[9] H. Ghasemalizadeh and M. Razzazi An Improved Approximation Algorithm for the Most Points Covering Problem. *Theory of Computing Systems*, 50(3): 545-558, 2012.