New Variations of the Maximum Coverage Facility Location Problem*

Bhaswar B. Bhattacharya[1], and Subhas C. Nandy[2]

[1]Department of Statistics, Stanford University, California, USA,bhaswar@stanford.edu
[2]Advanced Computing and Microelectronics Unit, Indian Statistical Institute,
Kolkata - 700108, India, nandysc@isical.ac.in

Abstract. Consider a competitive facility location scenario where, given a set $\mathcal{U}$ of n users and a set $\mathcal{F}$ of m facilities in the plane, the objective is to place a new facility in an appropriate place such that the number of users served by the new facility is maximized. Here users and facilities are considered as points in the plane, and each user takes service from its nearest facility, where the distance between a pair of points is measured in either L_1 or L_2 or L_∞ metric. This problem is also known as the maximum coverage (MaxCov) problem. In this paper, we will consider the k-MaxCov problem, where the objective is to place k (≥ 1) new facilities such that the total number of users served by these k new facilities is maximized. We begin by proposing an $O(n \log n)$ time algorithm for the k-MaxCov problem, when the existing facilities are all located on a single straight line and the new facilities are also restricted to lie on the same line. We then study the 2-MaxCov problem in the plane, and propose an $O(n^2)$ time and space algorithm in the L_1 and L_∞ metrics. In the L_2 metric, we solve the 2-MaxCov problem in $O(n^3 \log n)$ time and $O(n^2 \log n)$ space. Finally, we consider the 2-Farthest-MaxCov problem, where a user is served by its farthest facility, and propose an algorithm that runs in $O(n \log n)$ time, in all the three metrics.

1 Introduction

The main objective in any facility location problem is to judiciously place a set of facilities, serving a set of users, such that certain optimality criterion is satisfied. Facilities and users are generally modeled as points in the plane. A facility can be *attractive*, like hospitals, schools, and supermarkets; or *obnoxious*, like garbage dumps and chemical plants. On the other hand, the set of users is either discrete, consisting of finitely many points, or continuous, that is, a region where every point is considered to be a user. Provided all the facilities are equally equipped in all respects, a user always avails the service from its nearest facility. Consequently, each facility has its *service zone*, consisting of the set of users that are served by it. For a set $\mathcal{U}$ of users, finite or infinite, and a set $\mathcal{F}$ of facilities, define for every $f \in \mathcal{F}$, $\mathcal{U}(f, \mathcal{F})$ as the set of users in $\mathcal{U}$ that are served by the facility f. Many variations of facility location problem in both the discrete and continuous user category, under several optimality criteria, have been studied [13].

1.1 Competitive Facility Location: Prior Work

Competitive facility location is concerned with the favorable placement of facilities by competing market players, and has been studied in several contexts [15, 16, 31]. In such a scenario the optimization criteria is to maximize the cardinality or the area of the service zone depending on whether the demand region is discrete or continuous, respectively.

* A preliminary version of this paper has appeared in the *Proc. 22nd Canadian Conference on Computational Geometry (CCCG)*, Manitoba, Canada, 2010, 241-244.

For continuous demand region, Dehne et al. [12] addressed the problem of locating a new facility q amidst a set $\mathcal{F}$ of n existing facilities, such that the area of the region served by q is maximized. The problem reduces to placing a new point q amidst a set of n existing points $\mathcal{F}$ such that the Voronoi region of q is maximized. Dehne et al. [12] showed that, when the given points are in convex position, the area function has only a single local maximum inside the region where the set of Voronoi neighbors do not change. For the same problem, Cheong et al. [10] gave a near-linear time algorithm that locates the new optimal point approximately, when the points in $\mathcal{F}$ are in general position. Variations of this problem, involving maximization of the area of Voronoi regions of a set of points placed inside a circle, have been recently considered by Bhattacharya [7].

The game-theoretic analogue of such competitive problems for continuous demand regions is a situation where two players place two disjoint set of facilities in a demand region. A player p is said to own a part of the demand region that is closer to the facilities owned by p than to the other player, and the player which finally owns the larger area is the winner of the game. The area a player owns at the end of the game is called the payoff of the player. In the *one-round game* the first player places m facilities following which the second player places another m facilities in the demand region. In the *m-round game* the two players place one facility each alternately for m rounds in the demand region. Ahn et al. [3] studied a one-dimensional Voronoi game, where the demand region is a line segment. The one-round Voronoi game in $\mathbb{R}^2$ was studied by Cheong et al. [11], for a square-shaped demand region. Fekete and Meijer [17] studied the two-dimensional one-round game played on a rectangular demand region with aspect ratio ρ.

The version of these competitive problems in the discrete user case is the problem of placing a new facility amidst a set of existing ones such that the number of users served by the new facility is maximized. This problem has been recently addressed by Cabello et al. [8], and this is referred to as the MaxCov problem. They showed that in the L_1 and L_∞ metrics, the problem can be solved in $O(n \log n)$ time. In the L_2 metric, they proved that if the number of existing facilities $m \geq 2$, the MaxCov problem is 3SUM hard, and gave an algorithm for finding the set of all possible optimal placements of the new facility in $O(n^2)$ time. They also showed that for $m = 1$ the MaxCov problem in L_2 metric can be solved in $O(n \log n)$ time, and this is asymptotically optimal under the algebraic decision tree model.

1.2 The k-MaxCov Problem: Our Results

In this paper we consider a generalization of the MaxCov problem, where instead of placing one new facility, one may wish to place multiple facilities simultaneously such that these facilities together serve the maximum number of users. This leads to the following generalization of the MaxCov problem.

k-MaxCov Problem: Given a set $\mathcal{U}$ of n users, and a set $\mathcal{F}$ of m existing facilities with $m < n$, find the placement of a set F^* of k (≥ 1) new facilities such that the total number of users in $\mathcal{U}$ served by the facilities in F^* is maximized. In other words, we have to find the placement of a set F^* of k (≥ 1) new facilities such that $|\bigcup_{f \in F^*} \mathcal{U}(f, \mathcal{F} \cup F^*)|$ is maximized, for $F^* \subset \mathbb{R}^2 \backslash \mathcal{F}$.

Clearly, the 1-MaxCov problem is nothing but the MaxCov problem as discussed by Cabello et al. [8]. In this paper, we study the k-MaxCov problem. We begin with the case where the set of users is any arbitrary finite subset of $\mathbb{R}^2$, but the existing facilities are all on a single straight line, and the new facilities are to be placed on the same line such

that the number of users served by the new facilities is maximized. We give an $O(n \log n)$ time algorithm for solving the k-MaxCov-problem on a line. We then proceed to study the 2-MaxCov problem in the plane. The objective here is to place two new facilities f and f' such that the total number of users in $\mathcal{U}$ served by f and f', that is, $|\mathcal{U}(f, \mathcal{F} \cup \{f, f'\}) \cup \mathcal{U}(f', \mathcal{F} \cup \{f, f'\})|$, is maximized, for $f, f' \in \mathbb{R}^2 \backslash \mathcal{F}$. We propose an algorithm for solving the 2-MaxCov problem in the plane, in the L_1 and L_∞ metrics, which runs in $O(n^2)$ time and uses $O(n^2)$ space. We also show that, in the L_2 metric the 2-MaxCov problem when there is only one existing facility, can be easily solved in $O(n)$ time. For more than 2 existing facilities, an $O(n^4)$ time algorithm for the 2-MaxCov problem is easy to get. Using a result of Katz and Sharir [20], we improve the time and space complexities of the 2-MaxCov problem in L_2 metric to $O(n^3 \log n)$ and $O(n^2 \log n)$, respectively, when there are more than 2 existing facilities. Finally, we consider the 2-Farthest MaxCov problem, where an user takes services from one of its farthest facilities, and obtain an $O(n \log n)$ time algorithm for solving it in all the three different metrics.

1.3 Potential Applications

Competitive facility location addresses the problem of placing sites by competing market players, and the expected income the new facility will generate depends on the market share it will capture [15, 16, 27, 32]. In our situation, imagine that a set of already existing facilities serves the users of a town. A new company, with the aim to compete with the existing facilities, now wishes to establish k (≥ 1) outlets in the town simultaneously. The problem of maximizing the profit of the company, in the sense that it serves the maximum possible number of users, now reduces to the k-MaxCov problem.

One recent application of maximum coverage in competitive environment was considered by Abellanas et al. [2]. They considered a problem in political economics with two opposing parties. The goal of the parties is to capture the greatest number of voters of a discrete population of elements, and the objective is to determine the optimum positions for the party in terms of guaranteeing the maximum number of voters. More formally, the policy plane consists of k political parties and the location of the n voters are represented by points. A voter choses a party by proximity or affinity to its policy. Moreover, a party can change its policy within a certain neighborhood with the objective of obtaining the greatest possible number of voters. The objective considered by the authors was to obtain the optimum situations for the party in this environment, that is, the positions such that the region of corresponding Voronoi diagram contains the greatest number of voters. This model was later extended to encompass more practical situations in a follow-up paper by the Abellanas et al. [1]. These problems have roots in the maximum coverage problems discussed in this paper, and are generally solved by computational geometric techniques.

The maximum coverage problem is also related to the discrete version of the Voronoi game which is played on a given graph instead of on a continuous space. Given an undirected graph $G = (V, E)$ and k players, every player has to choose a vertex (facility) from V, and every vertex (customer) is assigned to the closest facilities. A player's payoff (utility) is the number of vertices assigned to his facility. Dürr and Thang [14] showed that deciding the existence of a Nash equilibrium for a given graph is NP-hard. More hardness results was later proved by Teramoto et al. [30].

Moreover, as mentioned by Cabello et al. [8], the k-MaxCov problem, has relevance in several problems of database management. Given a database, a *reverse nearest neighbor* query retrieves those objects that have a query object as their nearest neighbor. This concept

was introduced by Korn et al. [21, 22], and has several applications in marketing and decision support systems. The *reverse nearest neighbor* query has several variants ranging from monochromatic or bi-chromatic versions to static or dynamic versions and has been widely studied in the literature [5, 9, 23, 25, ?,29]. In the bichromatic case, the point set consists of red and blue points, and the problem is to compute those points belonging to one of the two colors for which a query point is a bichromatic nearest neighbor. The k-MaxCov problem is a geometric optimization problem related to the planar bi-chromatic static variant of the reverse nearest neighbor queries. The setup consists of two sets of point, some of which are designated as facilities, and others as customers. In this setting, a reverse nearest neighbor query asks for the set of customers affected by the opening of a set of k new facilities, with the assumption that all customers choose the nearest facility.

Similarly, the k-Farthest MaxCov problem relates the *reverse farthest neighbor* query in databases. A reverse farthest neighbor query is to retrieve the objects in an object set, whose furthest neighbor is the query point. The monochromatic version of finding the set of all reverse farthest neighbors for a query point under the L_2 distance is known to be an open problem [21, 32]. The k-Farthest MaxCov problem is the bi-chromatic version of reverse farthest neighbor problem for a set of k query points. The bi-chromatic reverse farthest neighbor problem and its application to spatial databases has been recently studied by Yao et al. [33] and Liu et al. [24].

2 The k-MaxCov Problem on a Line

Let $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ be any fixed set of n users in $\mathbb{R}^2$, and $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$ be the set of m existing facilities all lying on a straight line ℓ. This variation of the k-MaxCov problem asks for the placement of k (≥ 1) new facilities $F' = \{f'_1, f'_2, \ldots, f'_k\}$ on ℓ such that the total number of users in $\mathcal{U}$ served by the facilities in F' is maximized.

Assume that the positions of the facilities along the line ℓ are distinct, that is, no two points in $\mathcal{F}$ are coincident on ℓ. Suppose $f_1 < f_2 < \ldots < f_m$ be the sorted order of facilities along the line ℓ. Define auxiliary facilities $f_0 = -M$ and $f_{m+1} = M$, where M is such that $\mathcal{U} \cup \mathcal{F} \subset [-M, M] \times [-M, M]$. For each user $u_i \in \mathcal{U}$, let the foot of the perpendicular drawn from u_i on ℓ be u'_i. Let $\mathcal{U}' = \{u'_i | u_i \in \mathcal{U}\}$. Denote by $\ell_i^{\perp}$ the line passing through f_i perpendicular to the line ℓ. Let S_i denote the open strip bounded by the parallel lines $\ell_i^{\perp}$ and $\ell_{i+1}^{\perp}$. (see Figure 1). Note that if a new facility f is placed on ℓ in the interior of the interval $J_i = [f_i, f_{i+1}]$, for some $i \in \{0, 1, \ldots, m\}$, then f can serve only the users in $\mathcal{U} \cap S_i$. Moreover, observe that the users in $\mathcal{U}$ which lie on the perpendicular line $\ell_i^{\perp}$ are always served by the facility f_i.

We now have the following simple observation:

Observation 1 *For every $f_i \in \mathcal{F}$, two new facilities f and f' can be placed in the interior of the interval $[f_i, f_{i+1}]$, such that $\mathcal{U}(f, \mathcal{F} \cup \{f, f'\}) \cup \mathcal{U}(f', \mathcal{F} \cup \{f, f'\}) = \mathcal{U} \cap S_i$.*

Proof. Let u'_a and u'_b be points in $\mathcal{U}'$ in the interior of the interval $[f_i, f_{i+1}]$ nearest to the facilities f_i and f_{i+1}, respectively. Note that if there is only one user in the interior of the interval $[f_i, f_{i+1}]$, then $u'_a = u'_b$. The result now follows from placing the two new facilities f and f' at the midpoints of the segments $[f_i, u'_a]$ and $[u'_b, f_{i+1}]$, respectively. □

This observation implies that two new facilities can be placed in the interior of an interval J_i such that all the users in S_i are served by the two new facilities. Therefore, no more users

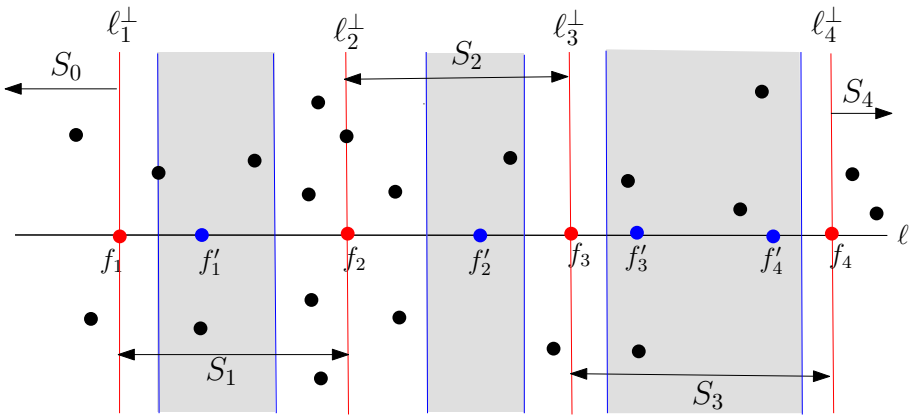

Fig. 1. The k-MaxCov Problem on a line: f_1, f_2, f_3, f_4 are existing facilities and f_1', f_2', f_3', f_4' are new facilities. The users in the shaded vertical strips will be served by the new facilities f_1', f_2', f_3', f_4'.

can be served by placing more than two new facilities in the interior of some interval J_i, for $i \in \{0, 1, \ldots, m\}$.

For every user $u_i \in \mathcal{U}$, we denote by R_i the interval on ℓ with midpoint at the point u_i' and length $2|d(u_i', \phi(u_i'))|$, where $\phi(u_i')$ is the facility in $\mathcal{F}$ which is closest to u_i', and $d(a, b)$ is the distance between a pair of points a, b on ℓ. Clearly, the interior of R_i does not contain any other facility in $\mathcal{F}$.

Now, associated with each interval J_i, we define the following two quantities:

n_i: The number of users in $\mathcal{U}$ which lie in the region S_i. That is, $n_i = |\mathcal{U} \cap S_i|$. Note that two new facilities can be placed in the interior of the interval J_i such that together they serve n_i users of $\mathcal{U}$ (Observation 1).

γ_i: The maximum number of users a single new facility placed on ℓ can acquire from the strip S_i. Thus, γ_i is the maximum depth (i.e., the maximum number of mutually intersecting intervals) in the arrangement $\mathcal{R}_i = \{R_q | u_q' \in J_i\} = \{R_q | u_q \in S_i\}$. Note that, $\gamma_0 = n_0$ and $\gamma_{m+1} = n_{m+1}$. The optimum position of a single new facility in the interior of the interval J_i is a point in the region (cell) where the maximum depth is attained; at this position the new facility serves γ_i users in $\mathcal{U}$.

Let $N = \{n_i | i \in \{0, 1, \ldots, m+1\}\}$ and $\Gamma = \{\gamma_i | i \in \{0, 1, \ldots, m+1\}\}$. We now have the following simple observation:

Observation 2 *The values in the sets N and Γ in their respective sorted orders can be computed in $O(n \log n)$ time.*

Proof. The sorted order of the facilities in $\mathcal{F}$, along the line ℓ, can be obtained in $O(m \log m)$ time. Once the sorted order of the facilities is known, it takes $O(\log m)$ time for a user $u \in \mathcal{U}$ to determine in which one of the strips $S_0, S_1, \ldots, S_m$ it lies. Therefore, the values in N can be computed in $O(n \log m)$ time. Once it is known in which strip S_i a user $u_a \in \mathcal{U}$ lies, it takes constant time to find the interval R_a. Therefore, the n intervals $R_1, R_2, \ldots, R_n$ can be computed in $O(n \log m)$ time, and the sorted order of their end points can be obtained in $O(n \log n)$ time. Finally, a linear time scan of the end-points of $R_1, R_2, \ldots, R_n$ is required to compute the value of γ_i for each strip S_i. Thus, the overall time required to obtain the sorted order of the values of N and Γ is $O(n \log n)$. □

For every $i \in \{0, 1, \ldots, m+1\}$, let us denote $\delta_i = n_i - \gamma_i$; $\delta_0 = \delta_{m+1} = 0$. We now have the following observation:

Observation 3 *$\gamma_i \geq \delta_i$ for every $i \in \{1, 2, \ldots, m\}$.*

Proof. Since $n_i = \gamma_i + \delta_i$, to prove the above observation it suffices to show that $\gamma_i \geq n_i/2$. Let us split the strip S_i as $S_i^+ = (f_i, \frac{f_i+f_{i+1}}{2}] \times \mathbb{R}$ and $S_i^- = [\frac{f_i+f_{i+1}}{2}, f_{i+1}) \times \mathbb{R}$. Observe that either $|\mathcal{U} \cap S_i^+| \geq n_i/2$ or $|\mathcal{U} \cap S_i^-| \geq n_i/2$. Without loss of generality, assume that $|\mathcal{U} \cap S_i^+| \geq n_i/2$. Let $u' \in \mathcal{U}'$ be the point in (f_i, f_{i+1}) nearest to f_i. If a new facility is placed at $f' = \frac{f_i+u'}{2}$, then the number of users served by f' is clearly greater than $n_i/2$, as the Voronoi region of f' contains the set of all users in the strip S_i^+. Now, since γ_i is the maximum number of users that the new facility can acquire from the interval (f_i, f_{i+1}), it follows that $\gamma_i \geq n_i/2$, and the result follows. □

With this observation the problem of solving the k-MaxCov problem on a line now becomes straightforward. Let $\Delta = \{\delta_i | i \in \{0, 1, \ldots, m+1\}\}$. Consider the sorted order of the numbers in $\Gamma \cup \Delta$. Select the k largest numbers from this set. Since $\gamma_i \geq \delta_i$ for every $i \in \{0, 1, \ldots, m+1\}$, it can be ensured that no δ_i is selected without selecting the corresponding γ_i. If for some $i \in \{0, 1, \ldots, m+1\}$ only γ_i is selected, then we place only one new facility in the interior of the interval J_i at the region where the maximum depth of the intervals in the set $\mathcal{R}_i = \{R_q | u'_q \in J_i\}$ is attained. If for some $i \in \{0, 1, \ldots, m+1\}$ both γ_i and δ_i are selected, then we place two new facilities in the interior of the interval J_i (using Observation 1), and these two facilities together serve all the n_i users in the strip S_i. Since the sorted order of the elements in $\Gamma \cup \Delta$ can be found in $O(m \log m)$ time, it follows from Observation 2 that the k-MaxCov problem on a line can be solved in $O(n \log n)$ time.

Theorem 1. *The k-MaxCov-problem on a line can be solved in $O(n \log n)$ time.* □

3 The 2-MaxCov Problem in the Plane

In this section we present our results for solving the 2-MaxCov problem in the L_1, L_2, and L_∞ metrics in the plane. The distances between any two points p and q in the plane in the L_1, L_2, and L_∞ metrics are denoted by $d_1(p, q)$, $d_2(p, q)$, and $d_\infty(p, q)$, respectively. Let $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ be the set of users and $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$ be the set of existing facilities. For every user $u_i \in \mathcal{U}$, we denote by $\phi(u_i)$ the nearest facility of u_i in $\mathcal{F}$.

Definition 1. *The nearest facility disk of a user u_i is the region such that if another facility f is placed in that region, $\phi(u_i)$ will no longer remain the nearest facility for u_i, and f becomes the nearest facility of u_i. We use R_i to denote the nearest facility disk of a user u_i.*

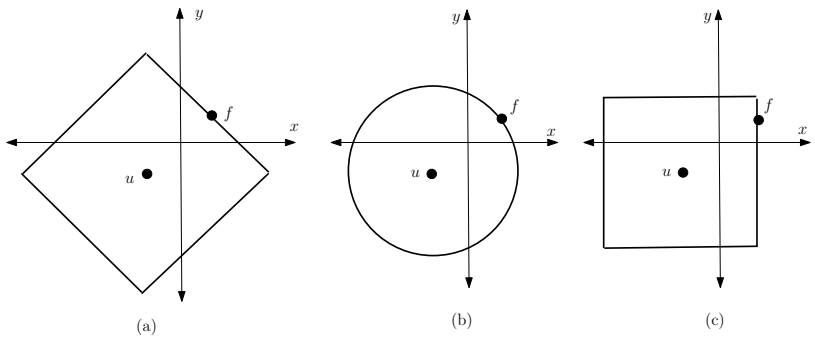

Fig. 2. Nearest facility disk for a user u and its nearest facility f in the (a) L_1, (b) L_2, and (c) L_∞ metrics.

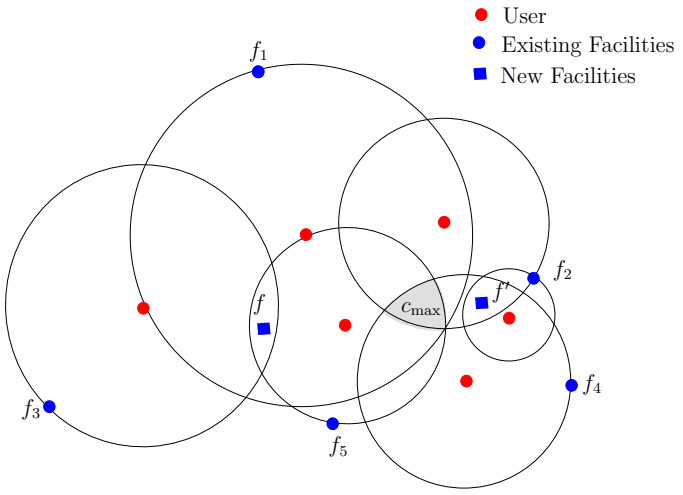

Fig. 3. Demonstration of the 1-MaxCov and the 2-MaxCov problems in the L_2 metric.

In L_1 metric, R_i is a square of side-length $2d_1(u_i, \phi(u_i))$, whose diagonals are axis-parallel and intersect at the user u_i (Figure 2(a)). In L_2 metric, R_i is a circle with radius $d_2(u_i, \phi(u_i))$ centered at the point u_i (Figure 2(b)). In L_∞ metric, R_i is a square of side-length $2d_\infty(u_i, \phi(u_i))$, whose edges are axis-parallel and the diagonals intersect at u_i (Figure 2(c)). Clearly, the interior of the nearest facility disk for each user in any metric does not contain any facility point. Note that the nearest facility of a user may not be unique but the nearest facility disk of a user is always unique. For the sake of convenience, we impose the restriction that a new facility must be placed in the interior (not on the boundary) of a nearest facility disk. This avoids the situation where a user has two equidistant facilities to choose from, one of which is existing and one of which is new. In order to explain our algorithms, let us define the following terms.

Definition 2. *An* arrangement $\mathcal{A}(\mathcal{R})$ *is the subdivision of space induced by a collection of geometric objects $\mathcal{R}$, like rectangles or circles. Each subdivision is a maximal connected region contained in a fixed subset of $\mathcal{R}$ and disjoint from the other subdivisions. These are called the* cells *of the arrangement $\mathcal{A}(\mathcal{R})$. The* depth *of a* cell *is the number of objects $\mathcal{R}$ that overlap on that cell.*

For our purpose, $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ is the collection of n nearest facility disks and $\mathcal{A}(\mathcal{R})$ is the arrangement induced by them. Note that if a new facility lies in the interior of r nearest facility disks in $\mathcal{R}$, then it serves exactly r users in $\mathcal{U}$, for any $r \in \{0, 1, \ldots, n\}$. Therefore, the 1-MaxCov problem can be solved by finding the cell $c_{\max}$ of maximum depth in the arrangement $\mathcal{A}(\mathcal{R})$. An instance of 1-MaxCov problem in L_2 metric is demonstrated in Figure 3. Here the solution is the shaded region $c_{\max}$ where the maximum number of nearest facility disks intersect.

In the 2-MaxCov Problem, we have to place two new facilities f and f' such that $|\mathcal{U}(f, \mathcal{F} \cup \{f, f'\}) \cup \mathcal{U}(f', \mathcal{F} \cup \{f, f'\})|$ is maximized, for $f, f' \in \mathbb{R}^2 \backslash \mathcal{F}$. Suppose one of the new facilities, say f, is placed at some cell c of $\mathcal{A}(\mathcal{R})$ such that $c = \bigcap_{j=1}^{k} R_{i_j}$ for some $\{i_1, i_2, \ldots, i_k\} \subset \{1, 2, \ldots, n\}$. Then the best possible position for another facility f' is the region of maximum depth in the arrangement of $\{R_1, R_2, \ldots, R_n\} \backslash \{R_{i_1}, R_{i_2}, \ldots, R_{i_k}\}$, that is, where $|\mathcal{U}(f', \mathcal{F} \cup \{f, f'\})|$ is maximized. Therefore, the optimum placement of the two facilities f and f' in the 2-MaxCov problem can be obtained by checking each cell $c \in \mathcal{A}(\mathcal{R})$ as the possible position of f, and then computing the best position of f' as mentioned above. In Figure

3, the optimal positions of f and f' are also shown using boxes ($\square$). Note that neither f nor f' are in the cell $c_{\max}$. Moreover, the complexity of the arrangement of circles $\mathcal{A}(\mathcal{R})$ is solely determined by the number of users n.

3.1 The 2-MaxCov Problem in L_1 and L_∞ Metrics

In the L_∞ metric the nearest facility disks $R_1, R_2, \ldots, R_n$ are axis-parallel squares. In the L_1 metric, we can rotate the axes of the coordinate system by an angle 45^o to make the nearest facility disks axis-parallel. Thus, in both L_1 and L_∞ metrics, we may consider each $R_i \in \mathcal{R}$ as an axis-parallel square with center (intersection of two diagonals) at u_i. The arrangement of these n squares (see Definition 2 may have $O(n^2)$ number of cells in the worst case. However, the cell with maximum depth can be found in $O(n \log n)$ time [18, 26]. A naive approach for this version of the 2-MaxCov problem is as follows: Construct the arrangement $\mathcal{A}(\mathcal{R})$ of the nearest facility disks; and consider each cell of the arrangement $\mathcal{A}(\mathcal{R})$ for the placement of f. Then remove the disks $\mathcal{R}' \subset \mathcal{R}$ that overlap on that cell, and find the cell with maximum depth in the *revised arrangement* of disks $\mathcal{R} \setminus \mathcal{R}'$. The time complexity of this naive approach solves the problem in $O(n^3 \log n)$ time. We now propose an efficient algorithm for this problem by dynamically maintaining the *revised arrangements*.

The Hanan grid [19] for a finite set S of points in the plane is obtained by constructing vertical and horizontal lines through each point in S. For the collection $\mathcal{R}$ of axis-parallel rectangles, we consider the Hanan grid of the set of all the vertices of the rectangles in $\mathcal{R}$. This can be obtained by drawing vertical lines along the vertical edges, and horizontal lines along the horizontal edges of all the members in $\mathcal{R}$ (see Figure 4). As there are n rectangles in $\mathcal{R}$, we have $O(n^2)$ grid cells in the Hanan grid. Note that each grid cell is a rectangle, and we take the point of intersection of the diagonals of a grid cell as the representative point of that cell and refer to it as *site*. By definition each *site* lies in at most one cell in the arrangement $\mathcal{A}(\mathcal{R})$, and the *sites* are arranged in n horizontal lines. We consider each *site* separately and repeat the steps (i) and (ii) as stated below until all the rectangles are removed.

(i) observe the number n_1 of rectangles in $\mathcal{R}$ that contain it,
(ii) delete all those rectangles from $\mathcal{R}$,
(iii) find the *site* that is contained in maximum number n_2 of rectangles in $\mathcal{R}$, and then
(iv) insert all the rectangles deleted in Step (ii) again in $\mathcal{R}$.

Finally, we report the *site* for which $n_1 + n_2$ is maximum. The algorithm for executing steps (i) and (ii) and the necessary data structures are explained below.

We construct a leaf-search 2D tree $\mathcal{T}$ with the *sites* [6]. The *leaf node* of $\mathcal{T}$ contain the *site*s, and internal nodes contain the discriminant values. Each leaf is attached with an integer location χ indicating the number of rectangles containing that *site*. Each internal node of $\mathcal{T}$ is attached with two integer fields max_χ and θ and a pointer field max_ptr; the max_χ of an internal node v contains the maximum χ-value of a leaf node in the subtree rooted at v, and max_ptr of v points to that leaf node. The role of θ field will be explained later. Moreover, there is also an efficient point location structure in the Hanan grid whose each cell points to the corresponding *site* (leaf) in $\mathcal{T}$.

Note that, the counting problem (the number of points inside a rectangle) in a 2D tree with n nodes can be solved in $O(\sqrt{n})$ time [6]. We use that algorithm as a preprocessing step for computing the χ values of the $O(n^2)$ leaf nodes in $\mathcal{T}$ as follows:

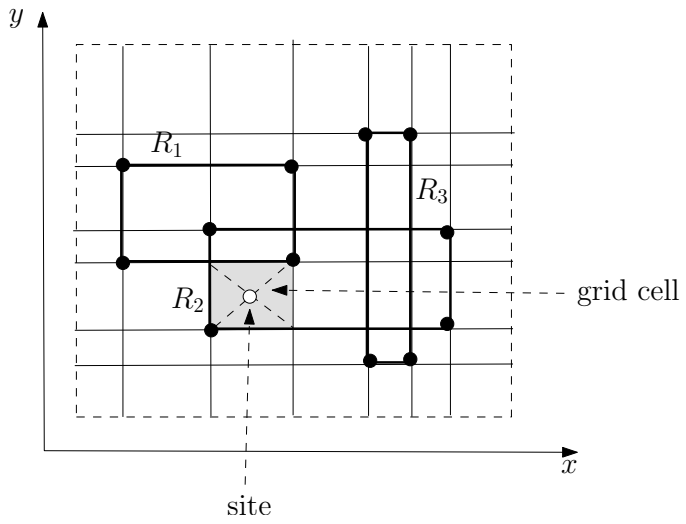

Fig. 4. Constructing the Hanan grid from a collection of axis-parallel rectangles.

Initially set the χ values of all the leaf-nodes and the θ values of all the internal nodes to zero. Consider each rectangle $R \in \mathcal{R}$, and execute the algorithm for the counting query in $\mathcal{T}$ [6]. On the search path,

(i) if a leaf is reached, its χ field is incremented by 1,
(ii) if a non-leaf node is observed such that all the sites in the subtree rooted at that node lies inside the rectangle R, then its θ field is incremented by 1, and search does not progress further in that subtree.

In the subsequent processing of some other rectangle $R' \in \mathcal{R}$, if the search visits a node u having non-zero θ, it is (i) propagated to the θ or χ field of both the children of u depending on whether the recipient nodes are non-leaf or leaf, (ii) added to the max_χ field of both the children of u if they are non-leaves, and then (iii) the θ-field of node u is set to zero. After incrementing the θ or χ fields of the nodes on the search path up to the leaf level, the same path is traced back to set the max $\_\chi$ and max_ptr fields of the nodes on those paths. Note that, after processing all the rectangles in $\mathcal{R}$, there may exist some non-leaf nodes in $\mathcal{T}$ having non-zero θ value. These are propagated towards the leaves by performing a pre-order traversal in $\mathcal{T}$. Finally, a post-order traversal of $\mathcal{T}$ is also required to adjust the max_χ and max_ptr fields of all the non-leaf nodes in $\mathcal{T}$.

Next, we execute the following procedure to solve the 2-MaxCov problem in L_1 or L_∞ metric by sweeping a horizontal line from top to bottom. At this point of time, we mention that, we need to attach one more integer field χ^* with each leaf. Similarly, each non-leaf node is also attached with two more integer fields θ^*, max_χ^* and one more pointer field max_ptr^*. Initially, these integer fields of each node in $\mathcal{T}$ are set to 0.

During the horizontal line sweep, when it hits the top boundary of a member $R \in \mathcal{R}$, it is conceptually deleted from the floor. In other words, the χ-value of all the leaves inside R are decremented by 1, and χ^*-value of those leaves are incremented by 1. During this process, when the χ-value of a *site* s becomes zero, the subset of rectangles $\mathcal{R}_s \subset \mathcal{R}$ containing s are deleted. However, the χ^* value of s contains its original χ-value ($= |\mathcal{R}_s|$). Now, we need to identify another *site* s' having maximum χ-value after deleting the rectangles in $\mathcal{R}_s$. When the sweep line reaches the bottom boundary of a rectangle $R \in \mathcal{R}$, it is inserted on the floor,

or in other words, the χ-value of all the leaves inside R are incremented by 1, and χ^*-value of those leaves are decremented by 1.

As we did earlier for computing χ, here also, both incrementing and decrementing of χ and χ^* (during the deletion and insertion of rectangles in $\mathcal{T}$) are done in an aggregative manner using θ and θ^* respectively. The role of max_χ and max_ptr remain the same as earlier; max_χ^* of a node v contains the maximum χ^* value of a leaf in the subtree rooted at v and max_ptr^* points to that leaf node.

Instead of specifically observing the instances when the χ-value of a *site* becomes zero, when the bottom boundary of a rectangle $R \in \mathcal{R}$ is reached, prior to inserting it on the floor, the following actions are taken in order:

(i) Two leaves v and v^* with maximum χ and χ^* values are identified using max_ptr and max_ptr^* pointers of the root node. If the facilities are positioned at v and v^*, then the total number of users can be served is $(max\_\chi + max\_\chi^*)$. We maintain a global integer counter MAX and two pointers π and π^*, where MAX stores the maximum value of $(max\_\chi + max\_\chi^*)$; π and π^* store the corresponding leaves in $\mathcal{T}$.

(ii) Then the χ-values of all the leaves in $\mathcal{T}$ that are inside R are incremented and χ^* values of all the leaves in $\mathcal{T}$ that are inside R are decremented in an aggregative manner using θ and θ^* fields respectively along the search path of R. The max_ptr and max_ptr^* are also set appropriately.

Though the number of sites is $O(n^2)$, the construction of $\mathcal{T}$ requires $O(n^2)$ time and space, since the sites are arranged in n horizontal and n vertical lines. The reason is that, at each internal node the discriminant value (partition line) can be obtained in $O(1)$ time without computing the median of the set of points with respect to their respective coordinates (corresponding to the level of the tree). During the preprocessing for computing the χ-values of the leaves in $\mathcal{T}$, processing each rectangle $R \in \mathcal{R}$ needs $O(\sqrt{n^2}) = O(n)$ time. Thus the entire preprocessing needs $O(n^2)$ time. Similarly, during the execution of 2-MaxCov algorithm, processing each boundary of each rectangle $R \in \mathcal{R}$ needs $O(n)$ time. Thus, this phase also takes $O(n^2)$ time in total.

Theorem 2. *In the L_1 and L_∞ metrics, the* 2-MaxCov *problem can be solved in $O(n^2)$ time and space.* □

3.2 2-MaxCov problem in L_2 metric

In the L_2 metric, the nearest facility disks are circles of different radii. The number of cells in an arrangement of n circles can be $O(n^2)$, and the region where the maximum number of circles overlap can also be computed in $O(n^2)$ time [8]. Thus, the naive approach for solving the 2-MaxCov problem in L_2 metric needs $O(n^4)$ time and $O(n^2)$ space. In the following, we show that

(i) if the number of existing facilities is only one, then the 2-MaxCov problem can be easily solved in linear time, and

(ii) if the number of existing facilities is more than one, then the 2-MaxCov problem can be easily solved in $O(n^3 \log n)$ time and $O(n^2 \log n)$ space.

Theorem 3. *The* 2-MaxCov *problem with one existing facility can be solved in $O(n)$ time.*

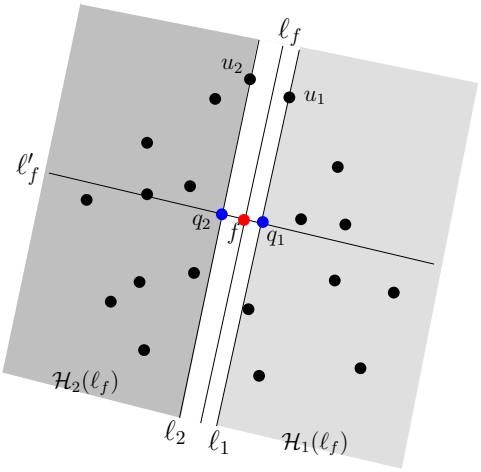

Fig. 5. Demonstration of 2-MaxCov problem with one facility in the L_2 metric.

Proof. Consider a set $\mathcal{U}$ of n users and one existing facility f. Let ℓ_f be a line passing through f which contains no point of the user set $\mathcal{U}$, and ℓ'_f be the line perpendicular to ℓ_f passing through the facility f. Denote by $\mathcal{H}_1(\ell_f)$ and $\mathcal{H}_2(\ell_f)$ the two open halfspaces on the two sides of the line ℓ_f. Let $u_1 \in \mathcal{U} \cap \mathcal{H}_1(\ell_f)$ be the point which is closest to the line ℓ_f. Similarly, let $u_2 \in \mathcal{U} \cap \mathcal{H}_2(\ell_f)$ be the point which is closest to the line ℓ_f. Denote by ℓ_1 and ℓ_2 the lines through u_1 and u_2 respectively, which are parallel to ℓ_f (see Figure 5). If the lines ℓ_1 and ℓ_2 intersect ℓ'_f at the points q_1 and q_2, then it is easy to see that $|\mathcal{U}(q_1, \{f, q_1, q_2\}) \cup \mathcal{U}(q_2, \{f, q_1, q_2\})| = |\mathcal{U}| = n$. This implies that $|\mathcal{U}(f, \{f, q_1, q_2\})| = 0$, and the best location of the two new facilities are at the points q_1 and q_2. Since the nearest neighbors u_1 and u_2 of ℓ_f in its two sides can be computed in $O(n)$ time, the theorem follows. □

When there are more than one existing facilities, consider the arrangement $\mathcal{A}(\mathcal{R})$ of the n nearest facility disks $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$. A geometric clique of the disks in $\mathcal{R}$ is a cell c in the arrangement $\mathcal{A}(\mathcal{R})$ such that the depth of the cell c is greater than or equal to the depth of all its neighboring cells. Observe that the optimum location of the two new facilities must lie in two different geometric cliques of $\mathcal{R}$. Since the dual graph of the arrangement $\mathcal{A}(\mathcal{R})$ can be computed in $O(n^2)$ time [4], all the geometric cliques of $\mathcal{R}$ can be computed in $O(n^2)$ time as well. Suppose we place a new facility at a geometric clique C of $\mathcal{R}$, where C is the intersection of the disks $R_{i_1}, R_{i_2}, \ldots, R_{i_k}$. Then the best possible location of the next new facility lies in a cell having maximum depth in the arrangement $\mathcal{R} \backslash \{R_{i_1}, R_{i_2}, \ldots, R_{i_k}\}$. As the maximum depth in an arrangement of circles can be found in $O(n^2)$ time and the location of the first new facility needs to be checked for all the geometric cliques of $\mathcal{R}$, the 2-MaxCov problem can be solved in $O(Kn^2)$ time, where K denotes the total number of geometric cliques in $\mathcal{R}$. Since K can be $O(n^2)$, the worst case running time of this algorithm is $O(n^4)$, which is same as the complexity of the naive approach.

We shall now improve this by using a data-structure introduced by Katz and Sharir [20]. They proved that given a collection of n circles in the plane, it is possible to construct a data structure in $O(n^2 \log n)$ time and space, so that the set of circles containing a query point in their interior can be reported in $O(\log n + B)$ time, where B is the number of circles containing the query point.

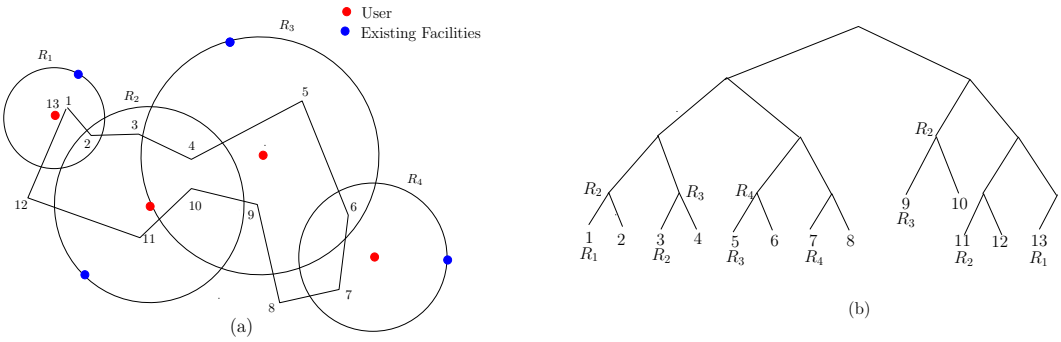

Fig. 6. 2-MaxCov problem in the L_2 metric: (a) An Eulerian walk in the arrangement of circles $\mathcal{A}(\mathcal{R})$, (b) The corresponding segment tree $\mathcal{T}$.

In the following, we recall the data-structure used by Katz and Sharir [20] and suitably modify it for solving the 2-MaxCov problem in L_2 metric. Consider the arrangement of the n nearest facility disks $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$. The points of intersection between pairs of circles in $\mathcal{R}$ are called the *vertices* of $\mathcal{A}(\mathcal{R})$. An *edge* of the arrangement $\mathcal{A}(\mathcal{R})$ is a maximal connected portion of the boundary of a circle in $\mathcal{R}$ that does not contain any vertex of the arrangement in its relative interior.

With these definitions, we present our algorithm for the 2-MaxCov problem in the L_2 metric. The algorithm has three phases.

Phase **1:** From the arrangement $\mathcal{A}(\mathcal{R})$ we construct the dual graph $\mathcal{D}(\mathcal{A}(\mathcal{R}))$ of $\mathcal{A}(R)$. Take a representative point in the proper interior of each cell of $\mathcal{A}(R)$. These are the nodes of $\mathcal{D}(\mathcal{A}(\mathcal{R}))$. Between a pair of nodes, there is an edge if the corresponding two cells share an edge of $\mathcal{A}(R)$. Next, we compute a spanning tree of $\mathcal{D}(\mathcal{A}(\mathcal{R}))$ and duplicate each edge of the tree to obtain an Eulerian tour π of $\mathcal{D}(\mathcal{A}(\mathcal{R}))$. The corresponding Euler tour is shown in Figure 6(a). As the dual $\mathcal{D}(\mathcal{A}(\mathcal{R}))$ is planar, it can have at most $O(n^2)$ edges, and so, the overall number of edges in π is $O(n^2)$. Now, for every circle $R_i \in \mathcal{R}$, mark all the edges of π that cross the boundary of R_i. These marked edges partition π into subsequences, so that each subsequence either lies completely inside or completely outside R_i. Note that the number of edges of π crossing R_i is $O(n)$; so the number of subsequences of π inside R_i is also $O(n)$ in the worst case. These will be considered as segments of π inside R_i. The total number of segments generated for all the members in $\mathcal{R}$ is $O(n^2)$ in the worst case.

Phase **2:** In this phase, we build a balanced binary leaf-search tree $\mathcal{T}$ with the nodes in π, in their order along π in its leaves. Each node $v \in \mathcal{T}$ is attached with a list L, two integer fields χ and max_χ, and a pointer field max_ptr. For each $R_i \in \mathcal{R}$, we insert all the segments inside it in $\mathcal{T}$ as follows:

Let I be a segment inside R_i. We search $\mathcal{T}$ from its root with I to find a node v whose discriminant value lies in I. This will be considered as the *fork node*, and R_i is stored in the list L attached to v. From node v, the search proceeds towards left and right with the two end points of I. At a node u on the left (resp. right) path, if the search proceeds towards left (resp. right), then R_i is stored in the list L attached with the right (resp. left) child of u. The reason is that all these nodes lie inside the interval I.

Next, we perform a pre-order traversal in $\mathcal{T}$ to propagate the list L attached to each node to the leaves of $\mathcal{T}$ rooted at that node. The χ, max_χ and max_ptr fields are set as we have done for the 2-MaxCov problem in L_1 and L_∞ metrics in the previous section.

Phase **3:** Now, we consider each leaf $u \in \mathcal{T}$ in order. Let $\mathcal{R}_u = \{R_{u_1}, R_{u_2}, \ldots, R_{u_p}\}$ be the set of disks stored in the list L attached to it. We consider each member of $R_{u_j} \in \mathcal{R}_u$, $j = 1, 2, \ldots, p$, and delete all the segments in R_{u_j} from $\mathcal{T}$. Let $p' = \max\_\chi$ value attached to the root of $\mathcal{T}$, and its max_ptr points to a leaf u', after the deletion. Now, if the new facilities f and f' are positioned at u and u', then it covers $p+p'$ users. u' will be referred to as the *associate* of u. Thus, we can identify an appropriate leaf u^* such that it along with its *associate* covers the maximum number of users. While processing the next node w (adjacent to u) in π, either only one disk is inserted in or deleted from $\mathcal{R}_u$ to get $\mathcal{R}_w$. In other words, all the intervals attached to that disk are inserted/deleted in $\mathcal{T}$. Since insertion/deletion of an interval in $\mathcal{T}$ needs $O(\log n)$ time and the number of intervals in a disk can be $O(n)$ in the worst case, processing each leaf of $\mathcal{T}$ needs $O(n \log n)$ time in the worst case.

Thus, we have the following result:

Theorem 4. *In the L_2 metric, for $m \geq 2$, the* 2-MaxCov *problem can be solved in $O(n^3 \log n)$ time and $O(n^2 \log n)$ space.*

Proof. The construction of $\mathcal{A}(R)$, the dual graph $\mathcal{D}(\mathcal{A}(\mathcal{R}))$, its spanning tree, and the computation of the Eulerian cycle π needs $O(n^2 \log n)$ time. The computation of the intervals inside all the disks need $O(n^3)$ time since it needs traversal of π for each member in $\mathcal{R}$. Thus, Phase 1 needs $O(n^3)$ time. In Phase 2, we need to insert $O(n^2)$ intervals corresponding to all the members in $\mathcal{R}$. This needs $O(n^2 \log n)$ time. Finally, in Phase 3, we need to consider all the leaves of $\mathcal{T}$ in order. Since the time required for processing each leaf is $O(n \log n)$, the time complexity result follows.

The space required for storing the tree $\mathcal{T}$ is $O(n^2)$. In Phase 2, the intervals attached with each circle $R_i \in \mathcal{R}$ is inserted in $\mathcal{T}$. Since the number of intervals in a circle is $O(n)$, and for each interval, R_i is stored in the list L of at most $O(\log n)$ nodes of $\mathcal{T}$, the space required for the list L of all the nodes in $\mathcal{T}$ is $O(n^2 \log n)$ in the worst case. Finally, after propagating the list L of an internal node v of $\mathcal{T}$ to the leaves of the subtree rooted at v, the list L of v is deleted. Thus in this step of Phase 2, the space complexity does not increase.

n Phase 3, while processing the leaves of $\mathcal{T}$, the insertion and deletion of segments corresponding to a disk R_i in $\mathcal{T}$ does not affect the space complexity. □

Remark 1: Given the solution of the 2-MaxCov problem, the k-MaxCov problem can be solved inductively, for $k \geq 3$. In the k-MaxCov Problem, we have to place k new facilities $F' = \{f'_1, f'_2, \ldots, f'_k\}$ such that $\bigcup_{j=1}^{k} |\mathcal{U}(f'_j, \mathcal{F} \cup F')|$ is maximized, for $F' \subset \mathbb{R}^2 \backslash \mathcal{F}$. Now, suppose one of the new facilities, say $f' \in F'$, is placed at some cell c of $\mathcal{A}(\mathcal{R})$ such that $c = \bigcap_{j=1}^{r} R_{i_j}$ for some $\{i_1, i_2, \ldots, i_r\} \subset \{1, 2, \ldots, n\}$. Then the best possible position of the remaining $k-1$ facilities $F' \backslash \{f'\}$ is obtained by solving the $(k-1)$-MaxCov problem for the arrangement $\{R_1, R_2, \ldots, R_n\} \backslash \{R_{i_1}, R_{i_2}, \ldots, R_{i_r}\}$ of the nearest facility disks. Therefore, the optimum placement of the k new facilities in the k-MaxCov problem can be obtained by checking each cell $c \in \mathcal{A}(\mathcal{R})$ as the possible position of f', and then solving the $(k-1)$-MaxCov problem as mentioned above. Hence, Theorem 2 implies that the k-MaxCov problem in the L_1 and L_∞ metrics can be solved in $O(n^{2k-2})$ time and $O(n^2)$ space. Similarly, from Theorem 4 it follows that the k-MaxCov problem in the L_2 metric can be solved in $O(n^{2k-1} \log n)$ time and $O(n^2 \log n)$ space.

4 The Farthest MaxCov Problem

In the preceding sections, we assumed that every user avails the service from its nearest facility. In this section we shall deal with the situation where each user is served by its farthest facility. Such a situation may arise when the facilities are obnoxious, and a customer no longer finds a facility desirable and wants to stay as far away from it as possible.

We are given a set $\mathcal{U}$ of n users and a set $\mathcal{F}_o$ of m obnoxious facilities. For every facility $f \in \mathcal{F}_o$ let $\mathcal{U}_o(f, \mathcal{F}_o) = \{u \in \mathcal{U} | d(u, f) \geq d(u, f'), \forall\ f' \in \mathcal{F}_o \backslash \{f\}\}$, where $d(a, b)$ is the distance measure in the desired metric between the points a and b. We now introduce the farthest version of the k-MaxCov problem as follows:

k-Farthest MaxCov problem: Given a set $\mathcal{U}$ of n users, a set $\mathcal{F}_o$ of m existing obnoxious facilities with $m \ll n$, inside a bounded region $C \subset \mathbb{R}^2$, find the placement of a set F^* of k (≥ 1) new facilities inside C such that $|\bigcup_{f \in F^*} \mathcal{U}_o(f, \mathcal{F}_o \cup F^*)|$ is maximized, for all possible placements of F^* inside C avoiding the positions of the members in $\mathcal{F}_o$.

The monochromatic version of finding the set of all reverse farthest neighbors for a query point under the L_2 distance is known to be an open problem [21, 32]. The k-Farthest MaxCov problem is the bi-chromatic version of reverse farthest neighbor problem for a set of k query points. For a practical perspective of bi-chromatic reverse farthest neighbor problem and its application to spatial databases, refer to Yao et al. [33].

Cabello et al. [8] studied the 1-Farthest MaxCov problem. They referred to it as the Farthest MaxCov problem and proved the following result:

Theorem 5. *[8] Given a closed region $C \subset \mathbb{R}^2$ with constant complexity, containing a set $\mathcal{U}$ of n users and a set $\mathcal{F}_o$ of m obnoxious facilities, the* Farthest MaxCov *problem in the L_1, L_2, and L_∞ metrics can be solved in $O(n \log n)$ time.*

We study the 2-Farthest MaxCov problem in all the L_1, L_2, and L_∞ metrics. For every user $u_i \in \mathcal{U}$ we denote by $\phi_o(u_i)$ its farthest facility. Let R_i be the farthest facility disk with center at the point u_i and radius $d(u_i, \phi_o(u_i))$. Let $\mathcal{A}_o(\mathcal{R})$ denote the arrangement produced by the set of n such disks $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$.

Observation 4 *If a new facility $f^* \notin \mathcal{F}_o$ is placed in the proper interior of some cell $A_i \in \mathcal{A}_o(\mathcal{R})$, then the number of users that are served by f^* is the number of disks that do not contain the cell A_i.* □

Lemma 1. *If the region C containing the set of users $\mathcal{U}$ and the set of existing facilities $\mathcal{F}_o$ is bounded then both the new facilities f and f' of the* 2-Farthest MaxCov *problem will lie on the boundary of C.*

Proof. Observe that all the facilities in $\mathcal{F}_o$ are contained in every circle R_i; this implies that $\bigcap_{i=1}^{n} R_i$ contains all the facilities $\mathcal{F}_o$. Let f and f' be any two points in C and α be a point in $\bigcap_{i=1}^{n} R_i$. Now, as the rays $\overrightarrow{\alpha f}$ and $\overrightarrow{\alpha f'}$ progress, they exit from the disks in $\mathcal{R}$ one by one. Therefore, as the two new facilities move along these rays, the number of users that are served by these two facilities cannot decrease. If both the directed lines $\overrightarrow{\alpha f}$ and $\overrightarrow{\alpha f'}$ intersect the boundary of C at the points a and b respectively, then we have $|\mathcal{U}_o(a, \mathcal{F}_o \cup \{a, b\}) \cup \mathcal{U}_o(b, \mathcal{F}_o \cup \{a, b\})| \geq |\mathcal{U}_o(f, \mathcal{F}_o \cup \{f, f'\}) \cup \mathcal{U}_o(f', \mathcal{F}_o \cup \{f, f'\})|$. □

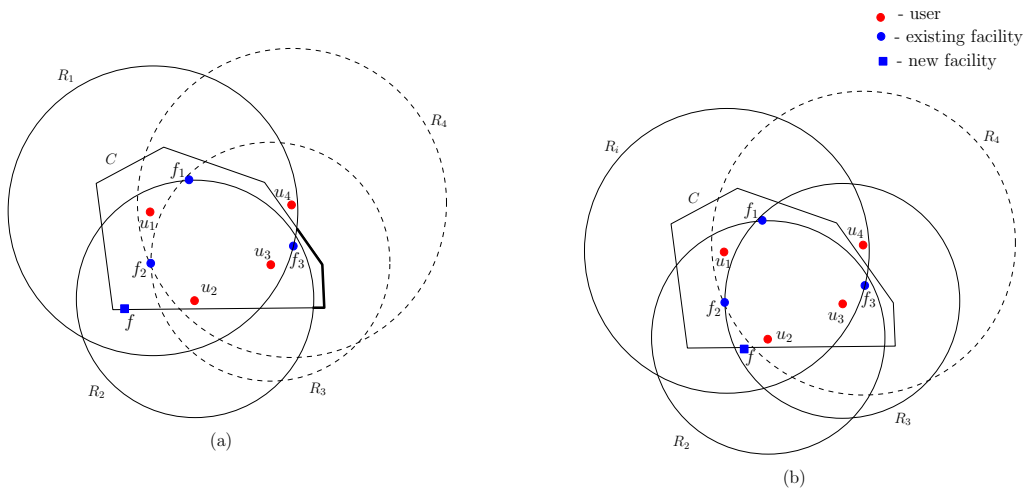

Fig. 7. 2-Farthest MaxCov problem: (a) The facility f serves the users u_3 and u_4 and $\partial C \backslash (R_1 \cup R_2) \neq \emptyset$, (b) The facility f serves the user u_4 and $C \subset \bigcup_{i=1}^{3} R_i$.

Now, suppose a new obnoxious facility f is already placed somewhere on the boundary ∂C of the region C. The circles R_i which do not contain the facility f, will correspond to the users which will be served by f. Suppose the point f lies inside the circles $R_{i_1}, R_{i_2}, \ldots, R_{i_p}$, for some $\{i_1, i_2, \ldots, i_p\} \subset \{1, 2, \ldots, n\}$. Here the following two cases can arise:

- $\partial C \backslash \bigcup_{j=1}^{p} R_{i_j} \neq \emptyset$: Here a region on ∂C exists where no circle of $\{R_{i_1}, R_{i_2}, \ldots, R_{i_p}\}$ overlaps (see the bold portion of ∂C in Figure 7(a)). In this case, the best possible location of the next new facility f', given the placement of f, is any point on $\partial C \backslash \bigcup_{j=1}^{p} R_{i_j}$, and the facilities f and f' serve all the users in $\mathcal{U}$.
- $\partial C \backslash \bigcup_{j=1}^{p} R_{i_j} = \emptyset$: Here $f \in \bigcap_{j=1}^{p} R_{i_j}$ and $C \subset \bigcup_{j=1}^{p} R_{i_j}$ (see Figure 7(b)). In other words, every point on ∂C is covered by at least one of the circles in $\{R_{i_1}, R_{i_2}, \ldots, R_{i_p}\}$. Here, we need to place f' on ∂C where minimum number of members in $\{R_{i_1}, R_{i_2}, \ldots, R_{i_p}\}$ overlap.

Note that ∂C is a closed curve of constant complexity s. We compute the points of ∂C intersected by R_i for $i = 1, 2, \ldots, n$. Each edge of ∂C can intersect R_i in at most one arc-segment (in each of the aforesaid three metrics). Therefore, the number of points where R_i intersects ∂C is at most $2s$. Thus, the total number of points of intersection on ∂C by the set of circles $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ is at most $2sn$. We have an arrangement of sn arc-segments on ∂C, which has at most $2sn$ cells. Note that each cell is covered by at least one circle, otherwise an obnoxious facility placed on an uncovered cell can serve all the users in $\mathcal{U}$. We need to find two cells for the placement of f and f' such that the total number of distinct circles covering these two cells is minimum.

Consider a point q in the interior of C, and choose a halfline ℓ from q that does not pass through any vertex of ∂C. Now, consider the arc-segments on ∂C that intersect the line ℓ once, and split each of them into two arc-segments by the line ℓ. Thus, we have a set $\mathcal{S}$ of $r = O(n)$ intervals along ∂C. The arrangement of $\mathcal{S}$ has at most $2r$ cells. We take a representative point in each cell and sort the points along ∂C in the circular order starting from the point of intersection of the line ℓ and ∂C in the counter-clockwise direction. We construct a leaf-search height-balanced binary tree $\mathcal{T}$ with those representative points at its leaves. The internal nodes contain the discriminant values such that none of them have

value same as that of any leaf node. Each leaf node is attached with an integer degree field χ that indicates the number of arc-segments in $\mathcal{S}$ that do not contain that leaf node.

At first, we consider each arc-segment and increment the χ value of each leaf node that lie inside that arc-segment by one in an aggregative manner as we did in Section 3.1 for the 2-MaxCov problem in L_∞ metric. This needs $O(n \log n)$ time. We attach a pointer field *min_ptr* and an integer field *min_*χ with each internal node of $\mathcal{T}$. For an internal node v in $\mathcal{T}$, *min_*χ contains the minimum χ value of the nodes at the subtree rooted at v, and the *min_ptr* pointer points to a leaf having minimum degree in the sub-tree rooted at v.

Now, we consider each leaf u of $\mathcal{T}$ in order. Let $R_u = \{R_{u_1}, R_{u_2}, \ldots, R_{u_p}\}$ be the set of disks not containing u. We delete the arc segments corresponding to these discs from $\mathcal{T}$ and find the cell of minimum depth by updating the *min_*χ and *min_ptr* of the tree as was done in Section 3.1. Since insertion/deletion of an arc-segment $\mathcal{T}$ needs $O(\log n)$ time and the number of arc-segments not containing u can be $O(n)$ in the worst case, processing the first leaf of $\mathcal{T}$ needs $O(n \log n)$ time. While we move from one leaf to the next, a constant number of arc-segment is inserted/deleted and so the update can be done $O(\log n)$ time. As there are $O(n)$ leaves in $\mathcal{T}$, the entire process takes $O(n \log n)$ time in the worst case.

Theorem 6. *The* 2-Farthest MaxCov *problem in any one of L_1, L_2 and L_∞ metric can be solved in $O(n \log n)$ time.* □

Remark 2: By similar inductive arguments as in Remark 1, Theorem 6 can be used to obtain an $O(n^{k-1} \log n)$ time algorithm for the k-Farthest MaxCov problem in any one of L_1, L_2 or L_∞ metric.

5 Conclusions

In this paper, we study the k-MaxCov problem, which is the generalization of the MaxCov problem introduced by Cabello et al. [8]. We show that the k-MaxCov problem on a line ℓ can be solved in $O(n \log n)$ time, where n is the number of users on the 2D plane and m number of facilities already exist on the line ℓ. We have also considered the nearest and the farthest versions of the 2-MaxCov problem in the plane, in L_1, L_2 and L_∞ metrics. For the nearest 2-MaxCov problem in L_1 and L_∞ metrics, we proposed an algorithm that needs $O(n^2)$ time and space. If the number of existing facilities is one, then the nearest 2-MaxCov in the L_2 metric can easily be solved in linear time. For arbitrary number of existing facilities, we proposed an algorithm for the nearest 2-MaxCov problem in L_2 metric, which runs in $O(n^3 \log n)$ time and uses $O(n^2 \log n)$ space.

We have also shown that the farthest version of the 2-MaxCov problem can be solved in $O(n \log n)$ time for each of the L_1, L_2 and L_∞ metrics.

Obtaining efficient algorithms for both the nearest and the farthest k-MaxCov problems in the plane, for $k \geq 3$ is a natural problem for future research. Extending the results to dimensions greater than 2 is another possible generalization.

Acknowledgment. The authors wish to thank Aritra Banik and Sandip Das of Indian Statistical Institute, Kolkata for very useful discussions about maximum coverage problems in competitive facility location. The authors wish to thank the anonymous referees for valuable comments which improved the quality and the presentation of the paper.

References

1. M. Abellanas, M. Dolores López, and J. Rodrigo, *Searching for equilibrium positions in a game of political competition with restrictions*, European Journal of Operational Research, Vol. 201 (3), 892–896, 2010.
2. M. Abellanas, I. Lillo, M. Dolores López, and J. Rodrigo, *Electoral strategies in a dynamical democratic system: Geometric models*, European Journal of Operational Research, Vol. 175 (2), 870–878, 2006.
3. H.-K. Ahn, S.-W. Cheng, O. Cheong, M. Golin, and R. van Oostrum, Competitive facility location: the Voronoi game, *Theor. Comput. Sci.*, Vol. 310, 457–467, 2004.
4. N. M. Amato, M. T. Goodrich, and E. A. Ramos, *Computing the arrangement of curve segments: divide-and-conquer algorithms via sampling*, Proc. 11th ACM-SIAM Sympos. Discrete Algorithms (SODA), 705–706, 2000.
5. R. Benetis, C.S. Jensen, G. Karčiauskas, and S. Šaltenis, *Nearest neighbor and reverse nearest neighbor queries for moving objects*, The VLDB Journal, Vol. 15, 229–250, 2006.
6. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwartzkopf, Computational Geometry - Algorithms and Applications, Springer, 1997.
7. B. B. Bhattacharya, *Maximizing Voronoi regions of a set of points enclosed in a circle with applications to facility location*, J. Math. Model. Algor., Vol. 9(4), 375–392, 2010.
8. S. Cabello, J. Miguel Diáz-Báñez, S. Langerman, C. Seara, and I. Ventura, *Facility location problems in the plane based on reverse nearest neighbor queries*, Eur. J. Operations Research, Vol. 202(1), 99–106, 2010.
9. O. Cheong, A. Vigneron, and J. Yon, *Reverse nearest neighbor queries in fixed dimension*, arxiv:0905.4441, June 5, 2009.
10. O. Cheong, A. Efrat, and S. Har-Peled, *On finding a guard that sees most and a shop that sells most*, Discrete Computational Geometry, Vol. 37, 545–563, 2007.
11. O. Cheong, S. Har-Peled, N. Linial, and J. Matoušek, The one-round Voronoi game, *Discrete and Computational Geometry*, Vol. 31(1), 125–138, 2004.
12. F. Dehne, R. Klein, and R. Seidel, *Maximizing a Voronoi region: the convex case*, Int. Journal of Computational Geometry and Applications, Vol. 15, 463–475, 2005.
13. Z. Drezner and H. W. Hamacher (Eds.), Facility Location: Applications and Theory, Springer, 2002.
14. C. Dürr and N. K. Thang, *Nash Equilibria in Voronoi Games on Graphs*, Proc. European Symposium on Algorithms (ESA), 17–28, 2007.
15. H. A. Eiselt and G. Laporte, *Competitive spatial models*, European Journal of Operational Research, Vol. 39, 231–242, 1989.
16. H. A. Eiselt, G. Laporte, and J. F. Thisse, *Competitive location models: A framework and bibliography*, Transportation Science, Vol. 27, 44–54, 1993.
17. S. P. Fekete and H. Meijer, The one-round Voronoi game replayed, *Comput. Geom. Theory Appl.*, Vol. 30 (2), 81–94, 2005.
18. H. Imai and T. Asano. *Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane*, Journal of Algorithms, Vol. 4(4), 310–323, 1983.
19. M. Hanan, *On Steiner's problem with rectilinear distance*, SIAM J. in Appl. Math., Vol. 14, 255–265, 1966.
20. M. J. Katz and M. Sharir, *An expander-based approach to geometric optimization*, SIAM J. Comput., Vol. 26, 1384–1408, 1997.
21. F. Korn and S. Muthukrishnan, *Influence sets based on reverse nearest neighbor queries*, Proc. ACM SIGMOD International Conference on Management of Data, SIGMOD Record, Vol. 29(2), 201–212, 2000.
22. F. Korn, S. Muthukrishnan, and D. Srivastava, *Reverse nearest neighbor aggregates over data streams*, Proc. 28th VLDB Conference, 814–825, 2002.
23. K.-I. Lin, M. Nolen, and C. Yang, *Applying bulk insertion techniques for dynamic reverse nearest neighbor problems*, Proc. 7th Int. Database Engineering and Applications Symposium, 290, 2003.
24. J. Liu, H. Chen, K. Furuse, and H. Kitagawa, *An Efficient Algorithm for Arbitrary Reverse Furthest Neighbor Queries*, Proc. Asia-Pacific Web Conference (APWeb), 60–72, 2012.
25. A. Maheshwari, J. Vahrenhold, and N. Zeh, *On reverse nearest neighbor queries*, Proc. 14th Canadian Conference on Computational Geometry, 128–132, 2002.
26. S. C. Nandy and B. B. Bhattacharya, *A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids*, Computers and Mathematics with Applications, Vol. 29, 45–61, 1995.
27. F. Plastria, *Static competitive location: An overview of optimisation approaches*, European Journal of Operational Research, Vol. 129, 461–470, 2001.
28. A. Singh, H. Ferhatosmanoglu and A. Aman Tosun, *High dimensional reverse nearest neighbor queries*, Proc. 12th Int. Conf. on Information and Knowledge Management, 91–98, 2003.

29. Y. Tao, D. Papadias, and X. Lian, *Reverse kNN search in arbitrary dimensionality*, Proc. 30th VLDB Conference, 744–755, 2004.
30. S. Teramoto, E. D. Demaine, and R. Uehara, *The Voronoi game on graphs and its complexity*, J. Graph Algorithms Appl., Vol. 15(4), 485–501, 2011.
31. R. Tobin, T. Friesz, and T. Miller, Existence theory for spatially competitive network facility location models, *Annals of Operations Research*, Vol. 18, 267–276, 1989.
32. Q. Wang, R. Batta, and C. M. Rump, *Algorithms for a facility location problem with stochastic customer demand and immobile servers*, Annals of Operations Research, Vol. 111, 17–34, 2002.
33. B. Yao, F. Li, and P. Kumar, *Reverse furthest neighbors in spatial databases*, IEEE International Conference on Data Engineering, 664–675, 2009.