Improved Algorithms for Placing Undesirable Facilities

Matthew J. Katz[1] Klara Kedem[1] Michael Segal[2]

[1]Department of Computer Science
Ben-Gurion University, Beer-Sheva 84105, Israel
[2]Department of Communication Systems Engineering
Ben-Gurion University, Beer-Sheva 84105, Israel

March 5, 2001

Abstract

We improve several existing algorithms for determining the location of one or more *undesirable* facilities amidst a set P of n demand points, under various constraints and distance functions. We assume that the demand points reside within some given bounded region R. Applying concepts and techniques from Computational Geometry, we provide efficient algorithms for the following problems:

1. **Maxmin multi-facility location:** Locate k undesirable facilities within R under the constraints that the smallest distance between each demand point and the facilities is at least a given r, and the distance between any two facilities is at least a given $\mathcal{D}$. Under the L_∞ (L_1) norm we present efficient algorithms for any k, and under the L_2 norm we can locate efficiently two such facilities. In all cases R is assumed to be an axis-parallel rectangle.
2. **Minsum coverage:** Given a set of weighted demand points contained in an axis-parallel rectangular region R, and given a smaller axis-parallel rectangle Q, place Q within R such that the sum of weights of the demand points contained in Q is minimized.

Keywords: Facility location, computational geometry, maxmin, minsum.

1 Introduction

Many location problems deal with *undesirable* or *obnoxious* facilities (see, e.g. Ben-Moshe et al. (1999), Bespamyatnikh et al. (1999), Brimberg and Mehrez (1994), Brimberg and Wesolowsky (1995), Drezner and Wesolowsky (1994), Konforty and Tamir (1997)). A facility is called undesirable or obnoxious if it may pose a danger to the individuals living nearby, may have an adverse effect on property values, or may cause lower quality of life through pollution. Examples of obnoxious facilities are nuclear power plants, garbage dump sites, mega-airports, and chemical plants.

In this paper we study two general problems concerned with locating an undesirable facility (or a number of facilities) amidst demand points. We consider various distance functions, a varying number of facilities, or weighted demand points, and varying facility constraints. In the rest of this section we describe the problems, survey previously known algorithms for them, and state our results which are based on concepts and techniques from Computational Geometry.

Problem 1: Maxmin multi-facility location. In their paper "Multi-facility location using a maximin criterion and rectangular distances", Brimberg and Mehrez (1994) solve the maxmin multiple facilities location problem, under the L_∞ norm. This problem is stated as follows. Given a set $P = \{p_1, \ldots, p_n\}$ of n points in a rectangular region R, and given a distance r and another distance $\mathcal{D}$, locate k undesirable facilities, $F = \{f_1, \ldots, f_k\}$, such that the smallest distance between each demand point and the facilities is at least r, and the distance between any pair of facilities is at least $\mathcal{D}$. Their approach involves a branch and bound algorithm, and their algorithm runs in time $O(n^{2k})$.

We twist the problem slightly by turning it into an *optimization problem*. We seek the *largest* r for which it is still possible to place k undesirable facilities under the distance constraints stated above. The algorithms that we present for this optimization problem run in time $O(n \log^2 n)$ for $k = 2$ or 3, and in time $O(n^{k-2} \log^2 n)$ for $k \geq 4$. In order to solve the optimization problem, we first solve the corresponding decision problem, which is exactly the problem of Brimberg and Mehrez, and then apply an optimization scheme due to Frederickson and Johnson (1994) (see below), which adds a logarithmic factor to the time bounds of the algorithms that we obtain for the decision problem.

Our algorithms for the decision problem can be extended, in a straightforward way and within the same time bounds, to handle different separation values r_i for the points $p_i \in P$, instead of one value r. In addition we show that under the Euclidean norm the decision problem for $k = 2$ can be solved in time $O(n \log n)$.

Problem 2: Minsum coverage. Another type of obnoxious facility location problems is described in a paper by Drezner and Wesolowsky (1994): Given a set P of weighted demand points contained within a large rectangular (or circular) domain R, and given a rectangle (or a circle) Q of a fixed size, find a placement of Q within R so that the sum of weights of the demand points that are contained in Q is minimal. In more detail, each point $p_i \in P$ has a weight w_i assigned to it, the goal is to find how to place Q within R such that the following sum is minimized

$$\sum_{\{i \mid p_i \in Q\}} w_i .$$

This kind of problem might arise, e.g., when an obnoxious facility which affects its close neighborhood has to be placed in a populated area. Assuming this neighborhood is of a known size rectangle (resp. circle), and that each population site has a weight assigned to it, which might be the number of people at that site, one would like to minimize the total number of people who are affected by this facility.

Drezner and Wesolowsky present $O(n^2)$-time algorithms for both the rectangular and the circular case. We improve the algorithm for the rectangular case and present an $O(n \log n)$-time algorithm. In addition, we obtain a lower bound of $\Omega(n \log n)$ for the unweighted rectangular case, thus proving the optimality of our algorithm.

The paper is organized as follows. In Section 2 we present our algorithms for Problem 1 above and for several of its special instances. Towards the end of the section, we show how to solve the optimization problem, by applying an optimization scheme devised by Frederickson and Johnson (1994), which involves a fast search in a collection of implicit sorted matrices whose entries consist of all potential solutions. Since applying this scheme is almost standard (see, e.g., Glozman et al. (1998)), we only discuss it briefly. In Section 3 we present our *event-driven* algorithm for Problem 2. Roughly speaking the events are of the type "Q meets a point of P", "a point leaves Q", or "Q touches the boundary of R". We employ a *segment tree* data structure (see Mehlhorn (1984)) in which we dynamically maintain the sum of weights of the points that are within Q at each event. The minimum of these sums of weights dictates the best location for Q. A lower bound for the minsum problem is also shown in Section 3.

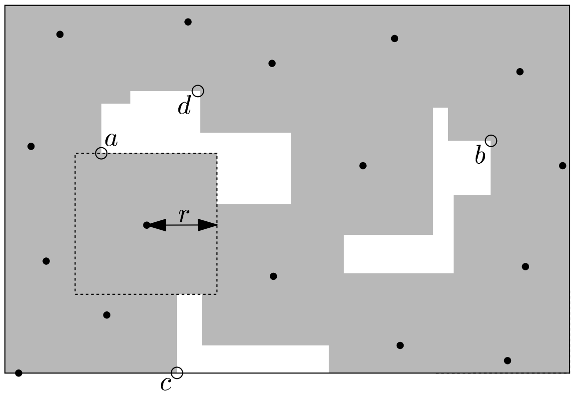

Figure 1: The rectangle R; the points P; around each point there is a grey square of size r; the white region is V; the two candidate solutions are $\{a, b\}$ and $\{c, d\}$.

2 Maxmin multi-facility location

Brimberg and Mehrez (1994) observed that under the maximum norm (L_∞) Problem 1 above can be restated as follows. (Recall that the L_∞ distance between points a and b is $max\{|a_x - b_x|, |a_y - b_y|\}$.) Draw around each of the points $p_i \in P$, a square of size r (where the size of a square is half its side length), and observe the union of the squares U. The facilities should be located in the (closed) region $V = R - U$, such that their pairwise distances are at least $\mathcal{D}$. This turns out to be our decision problem. We define the decision problem as the question: "For the given square size r, does there exist a set of k locations in V such that their pairwise distances are at least $\mathcal{D}$?" If there is one, we report the locations of the facilities and the answer *"yes"*. If not, we return *"no"*. The associated optimization problem will output the largest square size r^* for which the answer to the decision problem is *"yes"*. We will describe it towards the end of this section.

We solve several variants of the decision problem. Under the L_∞ norm, we first present $O(n \log n)$-time algorithms for $k = 2, 3$, and then present a general scheme for larger values of k that yields, for any $k \geq 4$, an algorithm that runs in time $O(n^{k-2} \log n)$. Thus, significantly improving the $O(n^{2k})$ (for any $k \geq 1$) solution proposed by Brimberg and Mehrez (1994). As we show, the space requirements of our algorithms vary between $O(n)$ and $O(n \log n)$, while the solution in the paper of Brimberg and Mehrez (1994) requires $O(n^2)$ space.

$k = 2$. It is well known that the combinatorial complexity of the boundary of the union of n squares is linear in n (see Preparata and Shamos (1985)). In other words, the boundary of this union consists of $O(n)$ vertices and edges and can be computed in time $O(n \log n)$ and space $O(n)$, using, e.g., a sweepline algorithm (see Mehlhorn (1984)). Thus the boundary of V can be computed in time $O(n \log n)$ and space $O(n)$. The problem of locating two facilities whose L_∞ inter-distance is at least $\mathcal{D}$, boils down to finding two pairs of points on the boundary of V (see Figure 1). One pair consisting of points in V with the smallest and largest x-coordinates, respectively (points a and b in Figure 1), and another pair consisting of points with the smallest and largest y-coordinates, respectively (points c and d in Figure 1). It is easy to see that we can choose these 4 points to be vertices of the boundary of V, thus, they can be found in time $O(n)$, by traversing the boundary vertices. If the inter-distance for both pairs of points is smaller than $\mathcal{D}$, then we cannot place the facilities as required, and the answer to the decision problem is *"no"*. If one of the pairs has inter-distance at least $\mathcal{D}$, then we place the facilities at these points and return their locations and the answer *"yes"*.

Theorem 2.1 *The two-facility location problem can be solved in $O(n \log n)$ time and $O(n)$ space.*

$k = 3$. As in the previous case compute the region V and claim:

Claim 2.2 *If there exists a solution for the three-facility location problem under the constraints above, then there exists a solution in which at least one of the three facilities is on a vertex of the boundary of V.*

Proof. Assume the points a, b and c are the locations of the facilities in the solution. Assume that a is the leftmost point (the point with the smallest x-coordinate among the three), b is the middle point (again, with respect to the x-coordinate), and c is the rightmost point. By pushing a and c horizontally leftwards and rightwards, respectively, until they reach vertical edges of the boundary of V, we only increase the distances among the solution points. Now, if b is the highest (resp. lowest) point, we can push the lower (resp. higher) point between a and c downwards (resp. upwards) until it reaches a vertex of the boundary of V. Otherwise, if b is not the highest nor the lowest among the three facilities, then we can push the higher between the points a and c upwards, and the lower between a and c downwards, until they reach vertices of V. ■

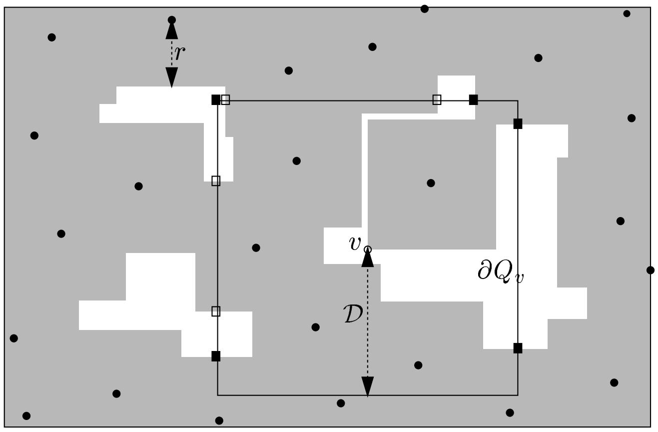

Figure 2: The square Q_v, and the vertices in B depicted as empty or black-filled tiny squares, where the black-filled squares are the extremal points of B.

Based on this claim we design our algorithm as follows. For each vertex v of the boundary of V, we assume that one of the facilities, say f_1, is located on v. In order to locate the remaining two facilities f_2 and f_3, we solve a two-facility location problem, but with a slightly different region than V. Let Q_v be a square of size $\mathcal{D}$ centered at v. It is easy to see that the facilities f_2 and f_3 should reside in $V - Q_v$ (see Figure 2).

Incorporating this observation in the two-facility algorithm in the naive way leads to a roughly quadratic algorithm. This is because the boundary, ∂Q_v, of Q_v might intersect the boundary, ∂V, of V in $O(n)$ points, and we must compute $V - Q_v$ for each vertex v of the boundary of V. However, we claim that it is not necessary to compute all the points of intersection between ∂Q_v and ∂V before applying the two-facility algorithm. Let us denote by B the vertices that are formed by subtracting Q_v from V. Notice (Figure 2) that B consists of the vertices that are both on $\partial(V - Q_v)$ and on ∂Q_v (depicted in the Figure by small squares on ∂Q_v). Define the *extremal points* of B as the pair of leftmost and rightmost points of B on each of the horizontal sides of ∂Q_v, and the pair of lower and upper points of B on each of the vertical sides of Q_v. There are at most 8 such points (in the Figure they are the black filled small squares).

Claim 2.3 *If there exists a solution for the two-facility location problem in*

$V - Q_v$, where one (resp. both) of the facilities f_2 and f_3 is (resp. are) in B, then there exists a solution in which one (resp. both) of the facilities is an extremal point of B.

Proof. Assume, without loss of generality, that f_2 is on a vertex $b \in B$ and b is not an extremal point of B. We show how f_2 can be moved to an extremal point of B without decreasing the distances between the facilities. Clearly, the distance between f_2 and f_1 remains $\mathcal{D}$ when f_2 is moved to a new location on ∂Q_v. (Recall that f_1 is on the center of Q_v.) As for the distance between f_2 and f_3 there are a number of cases, all are essentially similar and we mention only one: If b is on a horizontal edge of Q_v and f_3 is located to its left, then we can move b to the right extremal point of B on that edge without decreasing the distance between f_2 and f_3. Now if f_3 is also on a vertex $c \in B$ that is not an extremal point of B, then we move it to the extremal point of the edge to which c belongs for which the distance to f_2's new location increases. ■

This claim ensures that even if we do not compute all the new vertices that are formed by subtracting Q_v from V, i.e., we find only the extremal points of B, we can still find a solution to the three-facility location problem if one exists.

How do we efficiently find the vertices of $\partial(V - Q_v)$ where f_2 and f_3 may be placed? This process is divided into two parts. In one we find by *range searching* the vertices of ∂V that lie outside of Q_v, and thus are candidate placements for the two facilities. These vertices are not enumerated but are known implicitly, as will be explained below. In the other part we find by *ray shooting* the extremal points of B.

For the first part we compute the boundary of V and preprocess its vertices for orthogonal range searching (see de Berg et al. (1997)) with the fractional cascading technique (see Chazelle and Guibas (1986)). Now we have a data structure for range searching queries. When we place f_1 on v (for each boundary vertex $v \in V$), we perform a range search query with Q_v in this data structure, and find the vertices of V that lie outside of Q_v. This algorithm is standard (see e.g, de Berg et al. (1997)). However we give a rough sketch of it for completeness of the presentation.

We build a 2-dimensional *range tree* T which consists of two levels as described in de Berg et al. (1997). The main tree is a balanced binary search tree T_1 ordered by the x-coordinates of the vertices of ∂V. For each internal node or a leaf node $w \in T_1$, we associate a *canonical subset* $P(w)$ consisting of the vertices stored in the leaves of the subtree of T_1 rooted at w. The canonical subset $P(w)$ is stored in a balanced binary search tree T_w ordered by the y-coordinates of the points in $P(w)$. We call T_w the *associated structure* of w. At each of the nodes u of T_w, we store the leftmost, rightmost, topmost, and bottommost vertices among the vertices stored in the leaves of the subtree rooted at u. At the node w we store a pointer to the root of T_w. The whole structure requires $O(n \log n)$ storage space.

Assume that Q_v, the orthogonal range query, is given by $[x, x'] \times [y, y']$. The query algorithm first selects $O(\log n)$ canonical subsets that together contain the points whose x-coordinates lie in the range $[x, x']$. Of those subsets, the algorithm reports the points whose y-coordinates lie in the range $[y, y']$, as a collection of $O(\log^2 n)$ nodes of associated structures. The union of the canonical subsets of these nodes consists of the desired set of points. The runtime of the above query algorithm is $O(\log^2 n)$ which can be improved to $O(\log n)$ applying the fractional cascading technique of Chazelle and Guibas (1986).

In the ray shooting part we determine, for each of the edges of Q_v, its two extreme points in B (if they exist). We employ a ray-shooting algorithm (see de Berg et al. (1997)) which we sketch here. We first preprocess the horizontal edges of the boundary of V for logarithmic time vertical ray shooting queries. We perform a similar preprocessing step for horizontal ray shooting. We will have two data structures, and two types of queries, respectively, one for the vertical direction and one for the horizontal direction. We continue to describe just the vertical ray shooting. This data structure occupies $O(n \log n)$ space (see de Berg et al. (1997)).

Given the query rectangle Q_v, we check whether the endpoints of its vertical edge $e = \overline{ab}$ lie in V (i.e., in R but not in U). An endpoint that lies in V is already an extremal point of B. Assuming, say, a (resp. b) is not an extremal point of B, we perform an orthogonal ray shooting query with the ray emanating from a (resp. b) and containing e, to detect the point of B (on e) which is closest to a (resp. b), if such a point exists. This can be done in $O(\log n)$ time (see de Berg et al. (1997)).

In order to solve the two-facility location problem in $V - Q_v$ in logarithmic time, we consider the $O(\log n)$ extreme vertices stored in the $O(\log n)$ nodes of associated structures that were reported by the range searching, together with the at most 8 extreme vertices of B. Among all these vertices, we select the two farthest vertices in each of the directions x and y. If the distance of one of these farthest pairs is at least $\mathcal{D}$, then we have a solution to the two-facility location problem in $V - Q_v$, otherwise, there is no solution. Figure 3 shows that we may have to place one of the remaining two

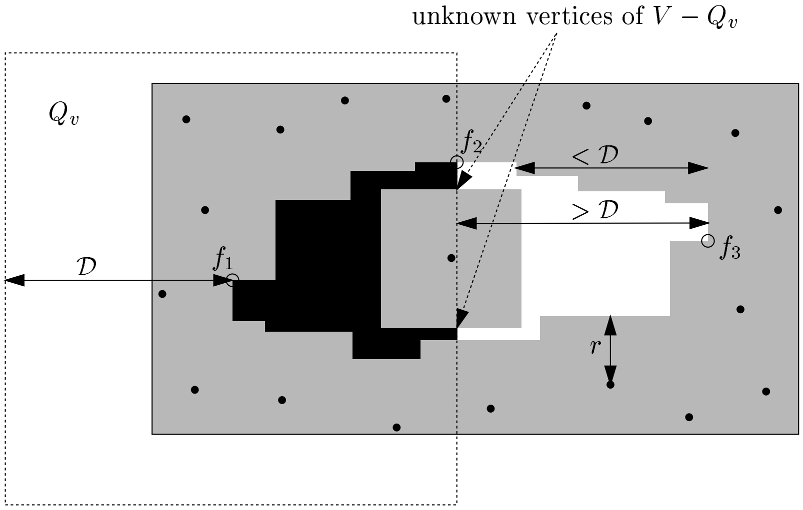

Figure 3: f_2 must be placed on the right edge of Q_v.

facilities f_2 and f_3 on the boundary of Q_v, otherwise, there does not exists a solution. The above deliberations lead to

Theorem 2.4 *The three-facility location problem can be solved in $O(n \log n)$ time and $O(n \log n)$ space.*

$k \geq 4$**.** In this case we claim that

Lemma 2.5 *We may assume that at least one of the facilities is on a vertex of the boundary of V.*

Proof. Let us consider the rectilinear free space $V = R - U$. Assume that there is an initial positioning for k facilities such that none of them is on a vertex of V. Our approach is to move the facilities, maintaining the $\geq \mathcal{D}$ distance requirement, such that at least one of them will be on a vertex. Denote the facilities by $F = \{f_1, \ldots, f_k\}$. About each facility f draw an axis-parallel square c with side length $\mathcal{D}$ (f being the square's center). Clearly in the given initial positioning the squares do not intersect. See Figure 4 (a). We push all the squares as much to the left as possible (Figure 4 (b)), so that they still do not intersect, and their centers remain in V throughout the motion. If at some point during this stage, one of the facilities coincides with a vertex of V, then we are done. If not, then, at

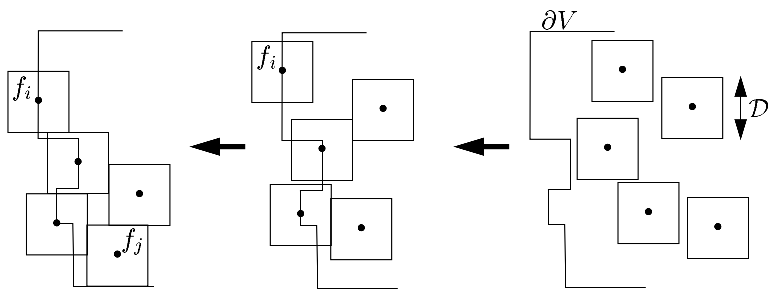

Figure 4: Proof of Lemma 2.5.

the end of this stage, the leftmost facility f_i must lie on a vertical edge of the boundary of V (if there are several leftmost facilities then f_i is taken to be the lowest among them). Next we push the squares as much down as possible (Figure 4 (c)), again, not letting any pair of them to intersect and not letting the facilities to penetrate into U nor leave R, and stopping if at some point a facility passes through a vertex of V. At the end of this stage, the bottommost facility f_j must lie on a horizontal edge of the boundary of V (and if there are several bottommost facilities f_j is the leftmost among them).

Assuming we have not stopped with a facility on a vertex, then we know that $i \neq j$, since otherwise the corresponding facility lies on a vertex and we would have stopped. We now check whether we can slide the square c_j to the left, under the same limitations, so that its center f_j coincides with a vertex of V. If we can, then we are done. Otherwise we proceed as follows. Consider the south west quarter plane defined by the line through the bottom edge of c_i and the line through the left edge of c_j. Notice that there exists at least one square that is fully contained in this quadrant. (This is true since there exists a square x blocking c_i from below, and there exists a square y blocking c_j from the left. Now, if $x = y$ then this square is such a square. Otherwise, if x lies to the left of the left edge of c_j, then x is such a square. And if x does not lie to the left of the left edge of c_j, then c_j is necessarily below the bottom edge of x, and therefore y which cannot be higher than x is fully contained in the above quarter plane.) We remove all the squares that are not fully contained in this quadrant, thus removing at least two squares (i.e., the squares c_i and c_j), and repeat the whole process for the remaining set of squares, etc. (Notice that there is no fear that the

$\mathcal{D}$ clearance property will be violated since the remaining squares are moved only left and down.) Eventually, if we do not stop earlier we are left with a single square, and this square can clearly be moved (left and down) so that its center lies on a vertex. ∎

Similarly to the three facility case we will position a square Q_v of side length $2\mathcal{D}$ centered on each vertex v of the boundary of V. For each such positioning of Q_v we compute the boundary of $V' = V - Q_v$. The solution for the $(k-1)$-facility location problem is applied to the vertices of the boundary of V', and so on, recursively. At any point in the computation, the boundary of V' is of complexity $O(n)$, and can be computed in $O(n)$ time from V.

Theorem 2.6 *The k-facility location problem, for $k \geq 4$, can be solved in $O(n^{k-2} \log n)$ time and $O(n \log n)$ space.*

Proof. We first compute $V = R - U$ in time $O(n \log n)$. Notice that adding a square Q_{v_i} at vertex v_i and updating the boundary of the free space can be done in $O(n)$ time. Thus, the running time for k facilities is $T(k) = n(O(n) + T(k-1))$. Recalling that $T(3) = O(n \log n)$, we obtain that $T(k) = O(n^{k-2} \log n)$. ∎

Remark 1. If, instead of one separation value, r, each point $p_i \in P$ has its own separation value, r_i, then the same algorithms apply without additional cost in time or space. This is because the boundary of the union of these squares is also linear in the number of squares (see Preparata and Shamos (1985)), and the rest follows immediately.

Remark 2. As far as we know nothing has been done considering the Euclidean norm (L_2). In this case instead of n squares of size r we have n discs of radius r. For $k = 2$, it is easy to show that if there exists a solution, then there exists a solution in which the two facilities are on vertices of the boundary of $V = R - U$. The combinatorial complexity of U and V is $O(n)$ (see Kedem et al. (1986)), and, after computing V, we can compute the farthest pair of vertices in $O(n \log n)$ time, using the corresponding algorithm in Preparata and Shamos (1985) for computing the diameter of a set of points. Thus, the two-facility location problem under the Euclidean norm can also be solved in $O(n \log n)$ time and $O(n)$ space.

The optimization scheme

In order to find the largest value $r*$ for which there still exists a solution to the k-facility location problem (keeping $\mathcal{D}$ fixed), we employ the optimization technique of Frederickson and Johnson (1984) (see, e.g., Glozman et al. (1998)). Each pair p_i, p_j of demand points determines eight critical values, four for each dimension. We list the critical values for the x-difference d_x between p_i and p_j: (i) $d_x/2$, (ii) d_x, (iii) $(d_x - \mathcal{D})/2$, and (iv) $(\mathcal{D} - d_x)/2$. In addition, each demand point p determines four critical values; the two horizontal distances between p and the boundary of R and the two vertical distances between p and the boundary of R.

We can represent all these distances as a constant collection of sorted matrices, and then perform a binary search on these values using the decision algorithm as an "oracle". As it was shown in the paper of Frederickson and Johnson (1984), the above scheme adds a multiplicative $O(\log n)$ factor to the running time of the decision algorithm.

3 Minsum coverage

Let $P = \{p_1, \ldots, p_n\}$ be a set of n weighted points within an axis-parallel rectangle R. Denote the weights by $\{w_1, \ldots, w_n\}$, respectively. Let Q be another axis-parallel rectangle which is smaller than R (both in width and in height). The goal is to place Q within R such that the sum of the weights of the points of P that are contained in Q is minimal.

Denote by c the width of Q and by d its height. Below we describe the main ideas of our algorithm and the data structure that we employ. Assume we put at each point p_i a horizontal segment s_i of length c, centered at p_i. Assume we have two more segments, b and t, assigned to the bottom and top edges of R respectively. The segments b and t coincide with the corresponding sides of R but are shorter on each side by $c/2$ (see Figure 5). This is because we are looking for the best location of the centerpoint of Q such that Q is fully contained in R.

The data structure we employ for better efficiency of our algorithm is the *segment tree*. We briefly outline the structure of the segment tree (for more details see Mehlhorn (1984)). To construct the segment tree we need to first define the *elementary intervals*. We orthogonally project the endpoints of all the segments s_i onto b. They subdivide b into small, elementary intervals. More precisely, the elementary intervals are the maximally connected segments, starting and ending at projected endpoints and not having a projected endpoint in their interiors. In Figure 6 we show the projections

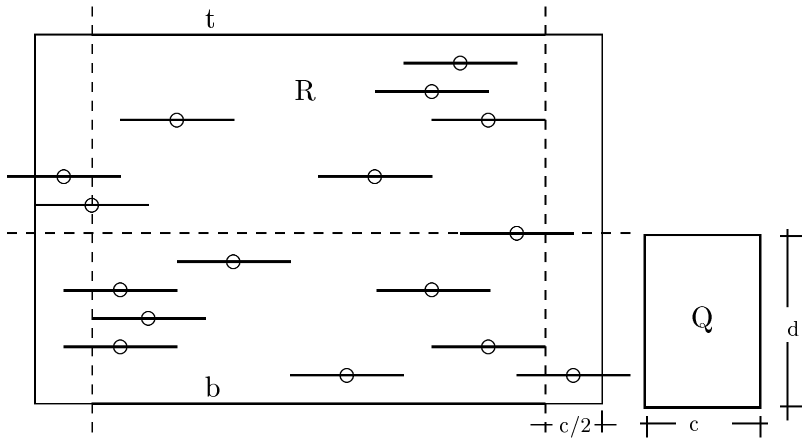

Figure 5: Data for the initial segment tree.

of all the endpoints, and denote below R the 14 elementary intervals.

At the initial phase of our algorithm we check where to locate Q if its lower side is constrained to coincide with b. As Q can slide left and right touching b, this defines a slab S in R of height d. We define the weight of an elementary interval e to be the sum of weights of all the intervals s_i of the points of $P \cap S$ that contain e. Observe that the x-coordinate of the center of Q can be anywhere along b, and that the sum of weights of the points that Q covers when its center is at a certain elementary interval, is exactly the weight of this interval. The claim is that at this phase the best location for the centerpoint of Q is when its x-coordinate is anywhere within the elementary interval which has the smallest weight (this interval is not necessarily unique, in our example it can be either interval 7, 8 or 10). In the Figure we put Q's center in interval 7.

We describe the segment tree T and its construction. Initially we compute all the elementary intervals and construct a binary tree whose leaves correspond to the elementary intervals sorted from left to right. We assign two attributes to each node in T: 1. The interval that the node *covers*, and 2. The weight of the node. The weight of each leaf is initially zero, and the interval is its elementary interval. Recursively, the interval of an inner node v is the union of the intervals of its two children-nodes, and the weight of an inner node v is the minimum of the weights of its two children-nodes plus the weights of the segments that are stored in v. (Thus, initially, the weight of all inner nodes is also zero.)

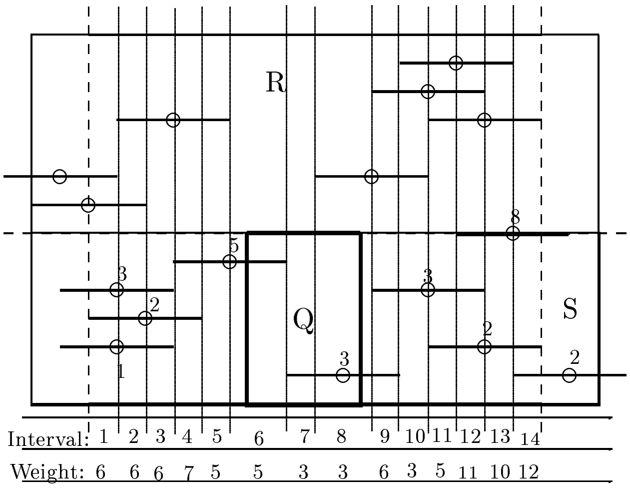

Figure 6: Elementary segments and their weights.

We now insert all the segments s_i that are contained in S (when S is in the initial phase, namely, touching b). The trick of the segment tree is that each segment can be inserted in at most $O(\log n)$ nodes and the weight attribute of the nodes can therefor be updated in the same number of nodes (please refer to Mehlhorn (1984) for details). At each insertion the weight at the root of T is the smallest weight of all the elementary intervals. Once all the segments in S have been inserted, the root contains the minsum weight for Q when it is constrained to touch b. Finding the elementary interval(s) that achieve this weight is easily done by peeking at the two children of the root and continuing down the tree in the direction of the child with smaller weight. (If both children are of equal weight, we pick one of them arbitrarily.)

We keep the weight of the root at this phase and continue to the next phases of the algorithm in our search for a better location for Q. The next phases are caused by the events of moving the slab S upwards. Each event is either (i) a point p_i gets on the lower side of S and is about to leave S, or (ii) a point p_i gets on the upper side of S and is about to enter S, or (iii) the top of S coincides with t.

Each of these cases is easily handled using the segment tree. For a type (i) event we delete the segment s_i from T and update the weights on the tree nodes that were affected by deleting s_i. We store this phase if the weight at the root is smaller than the minimum root weight that we achieved before.

For a type (ii) event we insert a segment s_i, and update T. Event of type (iii) terminates the algorithm. The best location of Q and the phase it has been found in are stored and we retrieve them.

The tree T has n leaves and its depth is $O(\log n)$. Each segment update, be it insertion or deletion, takes $O(\log n)$ time (this is the most important property of the segment tree). There are about n events, $n+2$ if we count the events when the slab touches the bottom and top of R. Thus, we obtain the following theorem

Theorem 3.1 *Given a set P of n weighted points within an axis-parallel rectangle R, and another axis-parallel rectangle Q which is smaller than R, it is possible to locate Q within R in $O(n \log n)$ time, such that the sum of the weights of the points lying in Q is minimized.*

A lower bound

A lower bound is obtained on a much simpler problem and thus it applies to the weighted minsum problem. Assume each point p_i has weight 1 and assume they are all on the x-axis and that Q is a zero height rectangle, namely a segment.

Bespamyatnikh et al. (1999) obtained an $\Omega(n \log n)$ lower bound for the following problem. Given n positive real numbers and a number γ, determine whether there exist two consecutive numbers in their sorted sequence $a_1, \ldots, a_n$, whose difference is greater than γ. The reduction of our problem to theirs is as follows. Let R be the segment $[a_1, a_n]$, and let Q be a segment of length γ. Every number corresponds to a 1-dimensional point with weight 1. If we can place Q within R so that the sum of the weights of the points lying in Q is 0, then two such numbers exist. Otherwise, we cannot find such a pair. We conclude that

Theorem 3.2 *Given a set P of n weighted points within an axis-parallel rectangle R, and another axis-parallel rectangle Q which is smaller than R, it is possible to locate Q within R in $\Theta(n \log n)$ time, such that the sum of the weights of the points lying in Q is minimized.*

4 Implementation

We have implemented the algorithms for both problems. The algorithm for the maxmin multi-facility problem has been implemented in Java under Windows NT. The main non trivial part of the algorithm was implementing the advanced data structures, such as the 2-dimensional range trees and the ray-shooting data structure. We used Red-Black trees (see Cormen, Leiserson, Rivest (1990)) in the implementation of the two data structures. We have implemented and applied a matrix search algorithm for the optimization step. The code for the latter algorithm is simple and short - less than 1000 lines, and runs very fast. We performed a number of tests, one of them, for example, with 20 facilities which are supposed to be at least 100 pixels apart pairwise, and with 180 input data points which are required to be at least 50 pixels away form the facilities. On Pentium 3 (700 MHZ) the computation was completed in 3 seconds. The code has a nice graphics interface and can be sent on CDROM upon request.

Regarding the second algorithm we should note that our code finds all the possible locations for the given rectangle. This algorithm has been implemented in C++ using the graphics library GL on Silicon Graphics platform. It can be downloaded from

`http://www.cs.bgu.ac.il/~segal/locate.`

We implement and maintain a segment tree. One difficulty in the implementation was dealing with all the end-cases, e.g., when two segments have the same endpoint. Still, the code is very short (less than 400 lines) and extremely fast. For an input consisting of 400 points the solution is found in 1.4 seconds on the SGI.

5 Conclusion

The norm that we use in this paper (L_∞) lends itself to some very fast algorithmic tools, by which we could improve existing algorithms for placing undesirable facilities. This is because a "unit-circle" under L_∞ is actually an axis-parallel square of size one. Under the Euclidean norm, the unit-circle is a circle (of course), and the tools above do not apply for circles. Therefore, it is probably more difficult to obtain solutions to the Euclidean versions of the two problems considered in this paper, which are as efficient as those proposed here. Thus, we are asking (1) Can the maxmin multi-facility location problem, under the Euclidean norm, be solved efficiently for values of k greater than 2? and (2) Can the $O(n^2)$-time result of Drezner and Wesolowsky (1994) for placing a small disk within a large disk be improved?

Acknowledgments – Work by M. Katz and K. Kedem has been supported by the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities, and by the Israel Ministry of Industry and Trade, LSRT consortium of the MAGNET program.

References

Bajaj C., (1984) Geometric optimization and computational complexity. Ph.D. thesis, Tech. Report TR-84-629, Cornell University.

Ben-Moshe B., Katz M., Segal M., (2000) Obnoxious facility location: complete service with minimal harm. International Journal of Computational Geometry & Application, 10, 581–592.

de Berg M., van Kreveld M., Overmars M., Schwarzkopf O., (1997) Computational Geometry: Algorithms and Applications. Springer-Verlag.

Bespamyatnikh S., Kedem K., Segal M., (1999) Optimal facility location under various distance functions. Workshop on Algorithms and Data Structures, 318–329.

Brimberg J., Mehrez A., (1994) Multi-facility location using a maximin criterion and rectangular distances. Location Science, 2(1), 11–19.

Brimberg J., Wesolowsky G.O., (1995) The rectilinear distance minsum problem with minimum distance constraints. Location Science, 3(3), 203–215.

Chazelle B., Guibas L., (1986) Fractional cascading: I. A data structuring technique. Algorithmica, 1, 133–162.

Chazelle B., Guibas L., (1986) Fractional cascading: II. Applications. Algorithmica, 1, 163–191.

T. Cormen, C. Leiserson, R. Rivest (1990) Introduction to algorithms. The MIT Press.

Drezner Z., Wesolowsky G.O., (1994) Finding the circle or rectangle containing the minimum weight of points. Location Science, 2(2), 83–90.

Frederickson G.N., Johnson D.B., (1984) Generalized selection and ranking: sorted matrices. SIAM Journal of Computing, 13, 14–30.

Glozman A., Kedem K., Shpitalnik G., (1998) Efficient solution of the two-line center problem and other geometric problems via sorted matrices. Computational Geometry: Theory and Applications, 11, 17–28.

Kedem K., Livne R., Pach J., Sharir M., (1986) On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. Discrete Computational Geometry, 1, 59–71.

Konforty Y., Tamir A., (1997) The single facility location problem with minimum distance constraints. Location Science, 5(3), 147–163.

Mehlhorn K., (1984) Multi-dimensional Searching and Computational Geometry. Data Structures and Algorithms, 3, Springer-Verlag.

Preparata F.P., Shamos M.I., (1985) Computational Geometry: An Introduction. Springer-Verlag, New York.