Placing an obnoxious facility in geometric networks *

Michael Segal

Communication Systems Engineering Department
Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel

November 14, 2002

Abstract

In this paper we consider several different problems of placing an obnoxious facility on geometric networks. In particular, our main results show how to obtain efficient polynomial time algorithms for locating an obnoxious facility on the given network under various distance functions such as maximizing the total sum of distances or maximizing the smallest of the distances from the facility to the nodes of the network. Our algorithms are obtained by applying concepts and techniques from Computational Geometry such as range searching, constructing spanners and other optimization schemes.

Keywords: Geometric networks, facilities, optimization techniques, spanners, range searching.

1 Introduction

Many location problems deal with *undesirable* or *obnoxious* facilities on the networks [5, 6, 10, 12]. 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 thru pollution. Examples of obnoxious facilities are garbage dumps, nuclear reactors, prisons, and military installations.

In this paper we consider a number of problems, all of which are concerned with locating an undesirable facility on a geometric network (or in the interior of a network) under varying constraints. We survey previously known algorithms for these problems and propose more efficient algorithms which are based on concepts and techniques from Computational Geometry.

Problem 1: Obnoxious facility in the interior of a planar network. Drezner and Wesolowsky [12] solve the following problem: Consider a planar connected network $G = (V, E)$ consisting of n nodes

*E-mail: segal@cse.bgu.ac.il

and m arcs connecting some pairs of nodes such that each arc $e \in E$ is a straight line between the end nodes in V; distances are Euclidean. Assume also that the boundary of the network is connected with arcs. A positive weight w_v is associated with each node $v \in V$ and a positive weight w_e is associated with each edge arc $e \in E$. Denote by $CH(G)$ the convex hull of the nodes in V. Our goal is to find a point $c \in CH(G)$ which maximizes

$$\min\{\min_{v \in V}\{w_v d_V(c, v)\}, \min_{e \in E}\{w_e d_E(c, e)\}\},$$

where $d_V(c, v)$ defines the Euclidean distance between c and v, and $d_E(c, e)$ defines the Euclidean distance between c and e. In other words, our goal is to find a point inside of the convex hull of V that will maximize the minimum weighted distance between the point and the nodes and arcs of the network.

The main motivation comes from the urban setting when the network may consist of noisy or polluting roads, transportation corridors, or rail lines. In [12] an $O(m^3 \log(1/\varepsilon))$ time algorithm is presented for an ε-approximate solution in the weighted case and it is shown how to deal with the unweighted case ($w_v = w_e = 1$) in $O(mn^2)$ time.

We show how to reduce the running time for the ε-approximate solution to $O(m^2 \log n \log(1/\varepsilon))$ time. We slightly change the definition of the problem and consider the case of the rectilinear network: the arcs between nodes are either horizontal or vertical. The rectilinear paths in such networks usually represent urban routes in grid city models. Instead of demanding the facility to lie inside of the $CH(V)$ we bound our network by the smallest enclosing rectangle $B(V)$ of V and require the location of the facility to be inside of $B(V)$. The distances measured between the facility and nodes (edges) are under L_∞ metric. We show how to solve the weighted instance of the rectilinear network in $O(n \log^2 n)$ time significantly improving the previous result of Drezner and Wesolowsky for this type of networks.

Problem 2: Obnoxious facility: maximizing the minimum distance from all demand nodes or maximizing the sum of distances from all demand nodes. Tamir [23] and later, Berman and Drezner [5] considered a problem where we are given a network $G = (V, E)$ with n weighted nodes and m edges and we wish to find a facility c that is allowed to be placed on an edge or node that maximizes

$$\min_{v \in V} w_v dist(c, v),$$

where $dist(c, v)$ defines the length of path between c and v. They [23, 5] show how to solve the unweighted case in $O(m)$ time and weighted case in $O(n^3)$ time. In fact, Tamir [23] presents

$O(mn)$ time algorithm when the distance matrix between all nodes is given. Tamir [24] also gave a subquadratic algorithm for solving the problem on tree network. We note that there exist a simple $O(n \log n)$ time algorithm for the complete Euclidean network (unweighted case). The objective of maximizing the (weighted) sum of distances from all demand nodes has been also studied ([10, 14]). In this problem we are looking for a facility c that is allowed to be placed on an edge or node that maximizes

$$\sum_{v \in V} w_v dist(c, v).$$

The complexity of maximizing the weighted sum of distances is the same as for minimizing the weighted sum of distances. Yet, no subquadratic algorithm has been reported in literature, except the known folklore $O(n \log n)$ time algorithm that solves the problem for weighted complete Manhattan network (see [16]). We present an $O(n \log^2 n)$ time algorithm for the weighted rectilinear complete network when we also require the weighted distance between the facility and the demand nodes to be bounded by some value R and show how to find an ε-approximate solution for the unweighted ($w_v = 1$) Euclidean complete networks in $O(n \log n)$ time when the facility should be located at one of the nodes. As byproduct we obtain algorithms with the same running times for placing the *desirable* facility with the objective of *minimizing* the sum of distances from the demand nodes. Our results are based on the existence of the *Euclidean t-spanners* and their decomposition into a constant number of binary trees [2] (see also [3]). The full description of t-spanners and results obtained by Arya et al. [2] will be given later in Section 3.

Problem 3: Obnoxious facility: number of customers within a prespecified distance R is minimized. This problem is formulated as: Given a network $G = (V, E)$ with n weighted nodes and m edges of given lengths and given a number $R > 0$, find a facility c that is allowed to be placed on an edge or node in order to minimize the number of nodes v such that

$$w_v dist(c, v) \leq R.$$

Berman et al. [6] show how to solve this problem in $O(mn \log n)$ time. Plastria and Carrizosa [20] considered the problem of placing a facility in (a) the plane or (b) on a planar network such that the total number of points lying within the influence radius from the facility should be minimized. They [20] were able to produce all candidate solutions for the (a) case in $O(n^3)$ time with overall $O(n^3 \log n)$ time complexity for the problem, while spending total $O(n^3)$ time for the (b) case. In fact, they were able to solve both cases in $O(\log n)$ time for any given radius R after initial

Problems	Previous result	Our result
Problem 1, Euclidean, weighted, ε-appr.	$O(m^3 \log(1/\varepsilon))$ [12]	$O(m^2 \log n \log(1/\varepsilon))$
Problem 1, Rectilinear, weighted	none	$O(n \log^2 n)$
Problem 2 (max-sum, discrete)	$O(n^2 \log n)$ [10] $O(n \log n)$ to all demand points	$O(n \log n)$, Euclidean, ε-appr. unweighted $O(n \log^2 n)$, rectilinear bounded distance
Problem 3	$O(mn \log n)$ [6] continuous weighted $O(n^3)$, Euclidean [20]	$O(n^{7/5}\text{polylog}\, n)$, Euclidean discrete $O(n \log^2 n)$, rectilinear unweighted, discrete

Table 1: Previous best results compared to our results.

$O(n^3 \log n)$ preprocessing time. The advantage of their method is by enabling the execution of an interactive procedure (over the values of R) in real time. We consider the case of the unweighted complete Euclidean and rectilinear networks when the facility must be placed at one of the nodes. We provide an $O(n^{7/5}\text{polylog}\, n)$ time algorithm for the Euclidean case and $O(n \log^2 n)$ time algorithm for the rectilinear network case.

We summarize the previous known results and our new results in Table 1. The main contribution of this paper is to provide a number of efficient algorithms to some well known problems in location theory on networks. Using new ideas and techniques from Computational Geometry we show how one can tackle these problems in geometric settings. Our paper is organized as follows. We focus on the problem 1 and explain the concept of a hippodrome in the next section. Section 3 discovers properties of spanners and to use of them in a solution of problem 2. Algorithm by Agarwal and Sen [1] and orthogonal range queries are applied for solving problem 3 in Section 4. Finally we conclude and mention some possible future work at Section 5.

2 Obnoxious facility in the interior of the network

Our objective is to maximize the minimal weighted distance from the facility to a node or an arc. As it was observed in [12], without loss of generality, we may assume that the weights of nodes are not larger than the weights of arcs to which they belong. Thus, we can ignore the weights of the nodes. First we show how to deal with the Euclidean case and then turn our attention to the rectilinear networks.

2.1 Euclidean case

We first consider the decision problem: given a real value R determine whether there exists a facility in the interior of the network $G = (V, E)$ whose distance from each edge $e_i \in E, 1 \leq i \leq m$ is greater or equal than R/w_{e_i} where w_{e_i} is the weight of the edge e_i. The formulation of the decision problem above implies that each edge e_i defines a *forbidden* region where the facility c cannot reside. We call this region a *hippodrome* $H(e_i, R)$ (see also Efrat et al. [13]). In Figure 1 the concept of a hippodrome is illustrated.

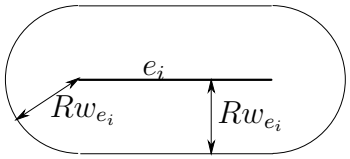

Figure 1: The hippodrome $H(e_i, R)$.

Denote by U_R the union of all $H(e_i, R)$, $1 \leq i \leq m$. An *admissible location* for the facility c exists if and only if the interior of the convex hull enclosing the network is not covered by U_R. Since the total number of the intersection points of all hippodromes is at most $O(m^2)$ we can detect whether U_R covers the convex hull by walking along the contour of U_R in $O(m^2 \log m)$ time [21].

Applying a bisection method similar to the one described in [12] requires $O(\log(1/\varepsilon))$ iterations to find a solution within an ε of the optimum. Thus we conclude by

Theorem 1 *The ε-approximate solution for Problem 1 can be found in $O(m^2 \log n \log(1/\varepsilon))$ time.*

Remark 1. If we assume that $w_e = 1, \forall e \in E$ we can design a faster runtime algorithm. The idea is to generate Voronoi diagram for a set of m edges. It is known, how to compute such diagram in $O(m \log m)$ time [27]. All possible candidate solutions for the facility are the vertices of this diagram since they are equidistant from the triplets (or more) of the segments. We can check all these $O(m)$ vertices in order to find the location of the facility.

Remark 2. Notice that the optimal interior point has a property that its weighted distances to some pair or triplet of nodes and edges are identical. Thus, we can found an exact solution by applying a binary search over the set of values corresponding to the set of planar points satisfying the above balancing property between triplets of edges and nodes. Unfortunately, generating all values explicitly will lead to $O(m^3)$ additional runtime factor which is worse in factor $O(m/\log m)$

than our result for the decision problem. Another possibility is to try to generate all the candidate solution values implicitly. In order to do so, we can apply a parametric search of Megiddo [19]. This technique requires an efficient parallel algorithm for the decision problem. Assuming that $O(T_s)$ ($O(T_p)$) is the running time of the sequential (resp. parallel) algorithm for the decision problem and P is the number of processors used in the parallel algorithm, Megiddo [19] shows how to solve the optimization problem in $O(PT_p + T_sT_p \log P)$ time. The parallel implementation of the sequential algorithm above is not straightforward, since the sequential algorithm employ a sequential iteration over the contour of the union of hippodromes. Nevertheless, this iteration can be parallelized, using persistent trees and related techniques. Combining this with Megiddo's technique we can obtain $O(m^2 \text{polylog}\, n)$ time algorithm for the optimization problem. For the sake of brevity, we do not provide any further details, and leave them to the interested reader.

2.2 Rectilinear case

As in the Euclidean case we first solve the decision problem and then show how to deal with the optimization stage. Notice, that from the definition, any rectilinear network with n nodes has only $O(n)$ edges as shown in Figure 2. Let B be the smallest axis-parallel rectangle enclosing the network. Since we are working under the L_∞ metric all hippodromes $H(e_i, R)$ become axis-parallel rectangles. Our problem, thus, is to find whether the union of these axis-parallel rectangles covers B. This problem can be efficiently solved in $O(n \log n)$ time using the data structure called *segment tree* T [18]. For the sake of completeness we sketch this algorithm below.

The leaves of T correspond to the ascending sorted order of the y-coordinates of the endpoints of the vertical sides of rectangles $H(e_i, R), 1 \leq i \leq m$. We define the range of a leaf l to be $[y_i, y_{i+1})$, $i = 1, \ldots, O(n)$, where y_i is the y-coordinate that corresponds to l, and y_{i+1} is the y-coordinate that corresponds to the next consecutive leaf after l. The range at each internal node in T contains the union of ranges in the nodes of its children and the range of the root of T corresponds to the vertical side of B. A vertical line is swept over the plane from left to right stopping at the endpoints of the horizontal sides of the $H(e_i, R), 1 \leq i \leq m$. At each stop, either a rectangle is added to the union or it is deleted from it. The vertical side v of this rectangle is inserted to (or deleted) the segment tree T and is stored (removed) in (from) $O(\log n)$ nodes of T (v is stored in $O(\log n)$ nodes and equals to the disjoint union of the ranges of these nodes). The update of T involves maintaining a counter for each node. The counter for node $u \in T$ counts how many vertical rectangle sides cover the range of this node and do not cover the range of its parent. At

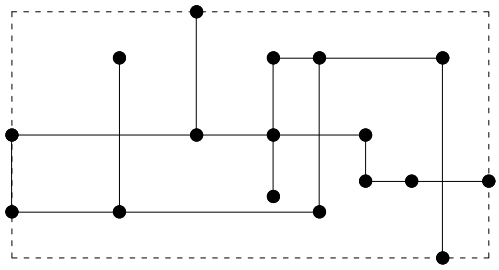

Figure 2: Rectilinear network with 17 nodes and enclosing rectangle.

each stop we check whether B is covered by the current union of rectangles using the counter of the root of T by checking whether the counter is greater than 0. If at deleting a rectangle the height of B is not wholly covered by all the vertical segments that are currently in T, then the answer to the decision problem is "no", namely the union of the axis-parallel rectangles does not cover B. If the height of B is fully covered, then we update T by deleting a segment from at most $O(\log n)$ nodes and recomputing the counter value for these nodes and their $O(\log n)$ ancestors. Since the number of events is linear the whole process takes $O(n \log n)$ time.

Now, we turn our attention to the optimization problem. As one can observe, assuming that the optimal solution is not attained on the boundary of B, there is an optimal facility c and two edges e_i and e_j such that either $w_{e_i} d_x(c, e_i) = w_{e_j} d_x(c, e_j) =$ optimal value, or $w_{e_i} d_y(c, e_i) = w_{e_j} d_y(c, e_j) =$ optimal value, where d_x (d_y) is the horizontal (vertical) distance between c and an edge of the network. Denote by $x(e_i)$ (resp. $y(e_i)$) the x-coordinate (y-coordinate) of the endpoints of edge $e_i \in E$. Notice that $x(e_i)$ (resp. $y(e_i)$) is defined uniquely for vertical (horizontal) edge e_i, and may attain one of two values for horizonal (vertical) edge e_i. This implies (with the given assumption) that the optimal distance R^* from the facility to the nodes and edges in the network is an element in one of the following four sets:

$S_1 = \{(x(e_i) - x(e_j))/(1/w_{e_i} + 1/w_{e_j}) : e_i, e_j \in E\}$,

$S_2 = \{|(x(e_i) - x(e_j))/(1/w_{e_i} - 1/w_{e_j})| : e_i, e_j \in E\}$,

$S_3 = \{(y(e_i) - y(e_j))/(1/w_{e_i} + 1/w_{e_j}) : e_i, e_j \in E\}$,

$S_4 = \{|(y(e_i) - y(e_j))/(1/w_{e_i} - 1/w_{e_j})| : e_i, e_j \in E\}$.

In order to locate R^* in the above four sets we can, as in [4], directly apply the search procedure of Megiddo and Tamir [17] with the modification described in Cole [11]. Each one of the four sets is of the form: $S' = \{(a_i + b_j)/(c_i + d_j) : 1 \leq i, j \leq n\}$, for an appropriate set of $4n$ numbers $a_i, b_j, c_i, d_j, (1 \leq i, j \leq n)$. The goal is then to find two elements $s, t \in S'$ such that $s < R^* \leq t$ and

no other element of S' is strictly between s and t. The total running time of the above scheme is $O((n + T_s) \log n)$, where T_s is the runtime of the decision algorithm. Since our decision algorithm runs in $O(n \log n)$ time we conclude that if the optimal solution is not attained on the boundary of B then the optimal solution can be found in $O(n \log^2 n)$ time.

The case where the optimal solution is attained on the boundary of the rectangle can be solved by looking at four one-dimensional problems, corresponding respectively to the four edges of the rectangle. The time needed to solve each one-dimensional problem is $O(n \log n)$, as shown by Tamir [23].

Theorem 2 *The optimal solution for Problem 1 for the rectilinear network under L_∞ metric can be found in $O(n \log^2 n)$ time and $O(n)$ space.*

It should be noted that the same approach does not work for every block norm (norm with a polyhedral unit ball in $\mathbb{R}^2$). The main reason is the dependency of our algorithm on the use of the segment tree data structure which keeps track of only one type (vertical, in our case) segments. The difficulty is that in order to check the covering of the network we need a data structure that can keep a combined track of different types of segments; each type is parallel to one of the sides of a convex polyhedron that defines our norm.

3 Obnoxious facility: maximizing the minimum distance or the sum of the distances

The problem is to locate a facility c on a given network $G = (V, E)$ with n nodes and m edges in order to (a) maximize the minimal distance from all demand nodes or (b) maximize the sum of distances from all demand nodes. We note that when G represents the complete Euclidean graph the unweighted version of the case (a) can be easily solved in $O(n \log n)$ by determining the diameter for the set of points corresponding to V, see e.g. [21]. In this section we mainly consider the case (b). We first present our approximate solution when the facility c is restricted to be one of the $|V|$ nodes for the Euclidean complete network and then show how to solve this problem exactly for the rectilinear network.

3.1 Euclidean case

Let $G = (V, E)$ be a graph, where each edge $e \in E$ has weight w_e and let $d_G(u, v)$ be the length of a shortest path between vertices u and v in G. Let $t > 1$ be a given constant. A subgraph G'

is a *t-spanner* for G if, for all pairs of vertices u and v, $d_{G'}(u,v)/d_G(u,v) \leq t$. When V is a set of points in $\mathbb{R}^d$, G is a complete graph, and the length of edge (u,v) is the Euclidean distance between these points, we call G' an *Euclidean t-spanner*. It is known how to construct such spanners having $O(n)$ edges in $O(n \log n)$ time [7, 22, 25]. Moreover, Arya et al. [2] (see also [3]) present a new construction of such spanners with a remarkable property that such spanner can be decomposed into a constant number of binary trees, so that each of the $O(n^2)$ spanner paths is mapped entirely to a path in one of these trees. More precisely, Arya et al. [2] proved the following theorem.

Theorem 3 (Arya et al.[2]) *Given a set of points $V \in \mathbb{R}^d$, and given $t > 1$, a forest consisting of $O(1)$ rooted binary trees T_i can be built in $O(n \log n)$ time and $O(n)$ space, having the following properties:*

1. *For each tree T_i in the forest, there is a $1-1$ correspondence between the leaves of this tree and the points of V.*
2. *Each internal node has a unique representative point, which can be selected arbitrarily from the points in any of its descendent leaves.*
3. *Given any two points u, v there is a tree T of the forest, so that the path formed by walking from representative to representative along the unique path in T between these nodes, is a t-spanner path for u and v.*

Our goal is to compute, for each one of the nodes (representing points) in the forest of trees, the sum of distances to all other nodes and then choose a node that gives a maximal sum. Since we have a number of such trees T_i, computing the sum of the distances between each leaf in T_i and all other leaves will not provide a solution yet. The main problem that for a given nodes $v, u, w \in V$ corresponding to leaves v, u, w in each tree, a t-spanner path between v and u may be defined by the leaves v and u in some tree T_i while a t-spanner path between v and w may be defined by the leaves v and w in some other tree T_j. Fortunately, we can deal with this problem. We need to distinguish between the t-spanner paths and non t-spanner paths in each tree T_i. In order to do so, we look carefully at the definition of each T_i. Arya et al. [3] showed that each T_i contains different types of nodes; one of them is called a *dumbbell node*. They [3] observe that if two leaves u and v lie on opposite sides of a dumbbell node (which is always exist) then the path from u to v is a t-spanner path. Our idea is to count, for each tree T_i and every leaf $u \in T_i$ the total sum of the lengths of the t-spanner paths between u and other leaves. Then, for each point $u \in V$ (that

corresponds to a constant number of leaves in a small collection of binary trees) we compute the total sum of the lengths of all t-spanner paths starting from u using the values computed for the corresponded leaves. We choose as our solution a point $u \in V$ with a maximal computed sum S_{max} of distances from u to all other nodes in G'. Let S_{opt} be a sum of the distances from the optimal solution node to all nodes in G and let S_u be a sum of the distances from u to all nodes in G. Clearly $S_{opt} \geq S_u$. Choosing $t = 1 + \varepsilon$, we obtain: $(1+\varepsilon)S_{opt} \geq (1+\varepsilon)S_u \geq S_{max} \geq S_{opt}$. Thus, $S_{opt} \geq S_u \geq S_{opt}/(1+\varepsilon) \geq (1-\varepsilon)S_{opt}$.

It remains to show how to compute, for a given tree T and every leaf $u \in T$ the total sum SUM_u of the lengths of the t-spanner paths between u and all other leaves in T. We proceed in two stages. At the first stage, for each node w in T we compute five values: C_w^l, the sum of the lengths of the paths between w and all leaves at the left subtree of w, C_w^r, the sum of the lengths of the paths between w and all leaves at the right subtree of w, N_w^1, the number of leaves in T excluding the tree rooted at w, N_w^2, the number of leaves in the left subtree of w and N_w^3, the number of leaves in the right subtree of w. All these values can be evaluated in linear time starting at leaves and processing towards the root of T.

At the second stage we proceed to compute the actual SUM_u values for every leaf $u \in T$. We recursively traverse T starting at root towards the leaves. For a currently traversed node w we compute a value A_w which is defined as follows. If w is the root of T_i then $A_w = 0$. If a dumbbell node v is a father of w (and suppose w is the left son of v) then $A_w = C_v^r + dist_G(w,v)N_v^3 + dist_G(w,v)A_v$. If w is the right son of a dumbbell node v then $A_w = C_v^l + dist_G(w,v)N_v^2 + dist_G(w,v)A_v$. In all other cases $A_w = dist_G(w,v)A_v$. In this fashion we ignore the non t-spanner paths and count only the needed information. At the end, $SUM_u = A_u$ for every leaf $u \in T$. The whole process can be accomplished in linear time using, e.g. preorder tree traversal. We conclude with the following theorem.

Theorem 4 *Given a set of n points in the plane, using the results of Arya et al. [3] we can ε-approximately compute the sum of unweighted Euclidean distances from each point to all other points in $O(n \log n)$ time using $O(n)$ space. In particular, we can find an ε-approximate solution (i.e. a node that produces the sum of the distances within factor $(1-\varepsilon)$ of the optimal solution) for Problem 2 in $O(n \log n)$ time using $O(n)$ space.*

Remark 3. Using the same approach we can find an approximate *median* of the complete Euclidean network $G = (V, E)$ (i.e. a node that produces the sum of the distances that within factor

$(1+\varepsilon)$ of the optimal minimal sum of the distances) in $O(n \log n)$ time using $O(n)$ space.

Remark 4. We note that when G is a planar graph in the theorem above the optimal solution can be computed using the algorithm of Frederickson [15] that evaluates the distances between all pairs of nodes of G in $O(n^2)$ total time.

One may think about approximating the optimal solution within some constant factor $\delta > 0$, i.e. find a node that produces the sum of the distances S', such that $S_{opt} + \delta \leq S' \leq S_{opt}$. We use a single bootstrapping approach in order to achieve this bound. Note, that for a given $\varepsilon > 0$, we can find S' such that $(1-\varepsilon)S_{opt} \leq S' \leq S_{opt}$. Since we need to choose $\varepsilon S_{opt} < \delta$, without knowing S_{opt} we first run our algorithm with, say, $\varepsilon = \frac{1}{2}$, and obtain $\frac{1}{2}S_{opt} \leq S'$. Then we choose $\varepsilon = \frac{\delta}{2S'} < \frac{\delta}{S_{opt}}$ and run the algorithm to obtain the desired precision.

3.2 Rectilinear case

With any node u of the rectilinear network $G = (V, E)$ we associate the x and y coordinates of the point in the plane that corresponds to u and refer to them as $x(u)$ and $y(u)$. A path $p =< v_1, v_2, \ldots, v_k >$ in G is called *xy-monotone* (increasing or decreasing) if $x(v_1) \leq x(v_2) \leq \ldots \leq x(v_k)$ and either $y(v_1) \leq y(v_2) \leq \ldots \leq y(v_k)$ or $y(v_1) \geq y(v_2) \geq \ldots \geq y(v_k)$. Since we are interested to work with the rectilinear distances but on the other hand we don't require our network to be complete we make an assumption that for any pair of nodes u, v in G such that $x(u) \leq x(v)$ there is an xy-monotone path connecting u and v. Such network is called a *Manhattan network*, see [16]. Using this assumption our problem can be restated as follows. Given a set V of n points in the plane and given a positive value R, find a point $c \in V$ such that the sum of the weighted L_1 distances from c to the points in V at distance at most R from c is maximized. We found that working under L_∞ metric is easier from the reasons that will be shown later. We rotate the entire graph by 45 degrees, obtaining the network $G' = (V', E')$ and now measure the L_∞ distances. It is well known that the metrics L_1 and L_∞ are dual in the plane, in the sense that nearest neighbors under L_1 in a given coordinate system are also nearest neighbors under L_∞ in a 45 degrees rotated coordinate system (and vice versa). The distances, however, are different by a multiplicative factor of $\sqrt{2}$. Thus, after finding a point producing the maximal sum under L_∞ metric we multiply the sum by $\sqrt{2}$ and obtain the desired sum.

In order to compute efficiently the sums of distances for all points $v' \in V'$ we apply the orthogonal range searching algorithm for weighted points of Willard and Lueker [26] which is defined as

follows. Given n weighted points in d-space and a query d-dimensional rectangle Q, compute the accumulated weight of the points in Q. The data structure in [26] is of size $O(n \log^{d-1} n)$, it can be constructed in time $O(n \log^{d-1} n)$, and a range query can be answered in time $O(\log^d n)$. We show how to apply their data structure and algorithm to our problem (see also a similar approach in [4]).

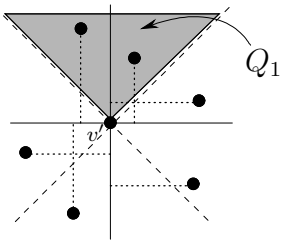

Figure 3: Division into four wedges. The wedge Q_1 is shaded.

Let $v' \in V'$ be the point for which we want to compute the sum of distances from it to all points $v'_i \in V'$. Let T be the square centered at v' with side length $\sqrt{2}R$. Clearly T can be decomposed into four triangles by its diagonals such that the L_∞ distance between all points of V' within one triangle is, without loss of generality, the sum of y coordinates of the points of V' in this triangle minus the y coordinate of v' times the number of points of V' in this triangle. More precisely, let Δ_u be the closed triangle whose base is the upper side of R and whose apex is v'. Denote by σ_u the sum of the weighted L_∞ distances between the points in Δ_u and v', and by N_u the number of points in $V'_u = \{V' - v'\} \cap \Delta_u$. Then

$$\sigma_u = \sum_{v'_i \in V'_u} w_{v'_i} y(v'_i) - y(v') \cdot N_u.$$

Our goal in what follows is to prepare two data structures for orthogonal range search for weighted points, as in [26], one with the weights being the x coordinates of the points of V' multiplied by a weight of a corresponding point and one with the y coordinates as weights multiplied by a weight of a corresponding point, and then to define orthogonal ranges, corresponding to the triangles in T for which the sums of x (y) coordinates multiplied by the weights of points will be computed.

We proceed with computing σ_u. Let l_1 a line whose slope is 45° passing through the origin and l_2 be a line whose slope is 135° passing through the origin. These lines define 4 wedges (see Figure 3 above). The first type of wedges: Q_1 and Q_2 – the wedges of points whose absolute value of y coordinates are larger than their absolute value of x coordinates, (2) The second type of wedges: Q_3

and Q_4 – the wedges of points whose absolute value of x coordinates are larger than their absolute value of y coordinates.

The wedges Q_1 and Q_2 and the wedges Q_3 and Q_4 define two data structure, as in [26]. Observe, *e.g.*, the wedge Q_1 (that contains Δ_u). We transform l_1 and l_2 into corresponding axes of an orthogonal coordinate system, and apply the same transformation on all the points $v'_i \in V'$. We construct the orthogonal range search data structure for the transformed points with the original y coordinates multiplied by the weights of points as weights. (Similarly we construct data structures for the points of V transformed according to Q_3 (Q_4) for the x sums.) Using this data structure it is easy to compute $\sum_{v'_i \in V'_u} w_{v'_i} y(v'_i)$

To compute N_u we can use the same wedge range search scheme, but with unit weights on the data points (instead of coordinates multiplied by the weights of points). In a similar way we compute the sum σ_d for the lower triangle in T (σ_l and σ_r for the left and right triangles in R respectively) and the corresponding number of points N_d (N_l and N_r). Our formula for the sum of the L_∞ distances from all the points of V' to v' is $S_{v'} = \sigma_u + \sigma_d + \sigma_l + \sigma_d$.

The algorithm described above requires $O(n \log n)$ preprocessing time and space, and then $O(\log^2 n)$ query time per point $v'_i \in V'$ to determine the sum of distances to the points in V lying at distance $\sqrt{2}R$ from v'_i. Thus, in total $O(n \log^2 n)$ time we can determine a node v in the rectilinear network $G = (V, E)$ which maximizes (or minimizes) the sum of the distances from it to the other nodes in the network lying at distance R from v. Thus, we have

Theorem 5 *Given a rectilinear network $G = (V, E)$ with a property that every two nodes are connected by xy-monotone path and given a value R we can find a node that produces the maximal sum of the weighted distances for Problem 2 in $O(n \log^2 n)$ time using $O(n \log n)$ space.*

4 Obnoxious facility: minimizing the number of customers within a prespecified distance R

Given a network $G = (V, E)$ with n nodes and m edges of given lengths, find a node on the network such that the number of nodes within a prespecified distance R from it is minimized. First we solve the case of the complete Euclidean network and then present more efficient algorithm for the rectilinear network.

4.1 Euclidean case

The following lemma is due to Agarwal and Sen [1].

Lemma 6 ([1]) *Let S be a set of n points in the plane, and let s be a parameter, with $n \leq s \leq n^3$. S can be preprocessed in time $O(s \log n)$ into a data structure of size $O(s)$, so that the number of points lying in a query disk can be counted in $O((n^{3/4}/s^{1/4}) \log^c n)$ time, for some constant $c > 0$.*

Since we are dealing with the complete Euclidean network the above lemma is applicable for our purposes. More precisely, we set $s = n^{7/5}$, preprocess S as in the Lemma 6 and query it with disk of radius R centered at each node $v_i \in V$. Then we choose the node that produces the minimal answer. Therefore, we have

Theorem 7 *Given a complete Euclidean network $G = (V, E)$ in time $O(n^{7/5} \log^c n)$, for some constant $c > 0$, we can find a node in G such that the number of nodes within a prespecified distance R from it is minimized.*

4.2 Rectilinear case

We can do much better for the rectilinear networks with xy-monotone paths between each pair of nodes. The idea is similar to one used in the previous section: we rotate the network by 45 degrees, obtaining $G' = (V', E')$ and now work with the L_∞ distances. Using the fact that the distances in these metrics are different by a multiplicative factor of $\sqrt{2}$ we use the following approach. For each node $v_i \in V$ we perform an orthogonal range query: we count the number of nodes of the network inside of the square with side length $\sqrt{2}R$ centered at v_i, by applying Chazelle's [8, 9] orthogonal range counting technique. Chazelle proposes a data structure that can be constructed in time $O(n \log n)$ and occupies $O(n)$ space, such that a range-counting answer for a query square can be answered in time $O(\log n)$. We report a node with the minimal number of nodes inside of its square. We conclude by

Theorem 8 *Given a rectilinear network $G = (V, E)$ with xy-monotone paths between each pair of nodes, in time $O(n \log n)$ we can find a node in G such that the number of nodes within a prespecified distance R from it is minimized.*

Remark 5. Using the same approach described in Theorem 8 we can solve the following problem: Given a rectilinear network $G = (V, E)$ with xy-monotone paths between each pair of nodes and

a number $1 \leq k \leq n-1$, find a node v in G such that the distance to the kth nearest node of v is maximized. By considering a rotated graph $G' = (V', E')$ we observe that the candidate values for such distance are $\sqrt{2}|x(v'_i) - x(v'_j)|$ and $\sqrt{2}|y(v'_i) - y(v'_j)|$ for all $v'_i, v'_j \in V'$. Notice that nearest neighbors under L_1 in a given coordinate system are also nearest neighbors under L_∞ in a 45 degrees rotated coordinate system (and vice versa). For each node $v'_i \in V'$ we perform a binary search with orthogonal range counting (in each step) in the sets $\{x(v'_j) \mid x(v'_j) > x(v'_i)\}$, $\{x(v'_j) \mid x(v'_j) < x(v'_i)\}$, $\{y(v'_j) \mid y(v'_j) > y(v'_i)\}$ and $\{y(v'_j) \mid y(v'_j) < y(v'_i)\}$, in order to find the smallest square centered at v'_i that contains $k+1$ (including v'_i) nodes of G'. We check the size of the square produced for each node of G' and then report the node contributing the largest square (the side length of this square multiplied by $\sqrt{2}$ is the maximal distance to the kth nearest node in G). The whole process takes $O(n \log^2 n)$ time.

5 Conclusions

In this paper we have presented a number of efficient algorithms to the well known problems in location theory on graphs. The efficiency of our algorithms is based on novel ideas and techniques from Computational Geometry. There is a wide range of open questions which are posed by this paper. The most intriguing one is how to generalize our algorithms for problems 2 and 3 (Euclidean case) for weighted case? Another interesting question concerns the optimizations stage of problem 1 (Euclidean case): how to generate a set of all possible solutions of cardinality $O(m^3)$ in less than a cubic time. Our parametric search algorithm is rather complicated and, therefore, one should look for other, much simpler algorithm. Finally, removing the assumption of existence of xy-monotone paths in problem 2 (rectilinear case) can generalize our solution for L_1 metric. In addition, we strongly believe that using similar approaches one can tackle various location problems on the networks in geometric settings.

References

[1] P. Agarwal and S. Sen, "Selection in monotone matrices and computing kth nearest neighbors", *Journal of Algorithms*, Vol. 20, pp. 581–601, 1996.

[2] S. Arya and G. Das and D. M. Mount and J. S. Salowe and M. Smid, "Euclidean spanners: short, thin, and lanky", *Proc. 27th Annu. ACM Sympos. Theory Comput.*, pp. 489–498, 1995.

[3] S. Arya and G. Das and D. M. Mount and J. S. Salowe and M. Smid, "Dumbbell trees: basic construction", private communication.

[4] S. Bespamytnikh, K. Kedem, M. Segal and A. Tamir, "Optimal Facility Location Under Various Distance Functions", *International Journal of Computational Geometry and Applications*, 10, pp. 523–534, 2000.

[5] O. Berman and Z. Drezner, "A note on the location of an obnoxious facility on a network", *European Journal of Operational Research*, Vol. 120, pp. 215–217, 2000.

[6] O. Berman, Z. Drezner and G. Wesolowsky, "Minimum covering criterion for obnoxious facility location on a network", *Networks*, Vol. 28, pp. 1–5, 1996.

[7] P. Callahan and R. Kosaraju "Faster Algorithms for Some Geometric Graph Problems in Higher Dimensions", in *Proc. SODA'93*, pp. 291–300, 1993.

[8] B. Chazelle, "Filtering search: A new approach to query-answering", *SIAM J. Comput.*, Vol. 15, pp. 703–724, 1986.

[9] B. Chazelle, "A functional approach to data structures and its use in multidimensional searching", *SIAM J. Comput.*, Vol. 17, pp. 427–462, 1988.

[10] R. Church and R. Garfinkel, "Locating an obnoxious facility on a network", *Transportation Science*, Vol. 12, pp. 107–118, 1978.

[11] R. Cole, "Slowing down sorting networks to obtain faster sorting algorithms", *J. ACM*, 34, pp. 200–208, 1987.

[12] Z. Drezner and G. Wesolowsky, "Obnoxious facility location in the interior of a planar network", *Journal of Regional Science*, Vol. 35(4), pp. 675–688, 1996.

[13] A. Efrat, M. Sharir, A. Ziv "Computing the smallest k-enclosing circle and related problems", *Computational Geometry: Theory and Applications* 4, pp. 119-136, 1994.

[14] E. Erkut and S. Newman, "Analytical models for locating undesirable facilities", *European Journal of Operation Research*, Vol. 40, pp. 275–291, 1989.

[15] G. Frederickson, "Fast algorithms for shortest paths in planar graphs, with applications", *SIAM J. Comput.*, Vol. 16, pp. 1004–1022, 1987.

[16] J. Gudmundsson, C. Levcopoulos and G. Narasimhan "Approximating a minimum manhattan network", *Nordic Journal of Computing*, 8(2), pp. 219–232, 2001.

[17] N. Megiddo and A. Tamir, "New results on the complexity of p-center problems", *SIAM J. Comput.*, 12(4), pp. 751–758, 1983.

[18] K. Mehlhorn, "Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry", *Springer-Verlag*, 1984.

[19] N. Megiddo "Applying parallel computation algorithm in the design of serial algorithms", *Journal of ACM*, 30, pp. 852–865, 1983.

[20] F. Plastria and E. Carrizosa "Undesirable facility location with minimal covering objectives", *European Journal of Operational Research*, 119, pp. 158–180, 1999.

[21] F. Preparata and M. Shamos "Computational Geometry: An Introduction", Springer-Verlag, New York, NY, 1985.

[22] J. Salowe, "Constructing multidimensional spanner graphs", *International Journal of Computational Geometry and Applications*, Vol. 1(2), pp. 99–107, 1991.

[23] A. Tamir, "Improved complexity bounds for center location problems on networks by using dynamic data structures", *SIAM J. Disc. Math.*, 1, pp. 377-396, 1988.

[24] A. Tamir, "Obnoxious facility location on graphs", *SIAM J. Disc. Math.*, 4, pp. 550–567, 1991.

[25] P. Vaidya, "A sparse graph almost as good as the complete graph on points in K dimensions", *Disrete and Computational Geometry*, Vol 6., pp. 369–381, 1991.

[26] D. Willard and G. Lueker, "Adding range restriction capability to dynamic data structures", *Journal of ACM*, Vol. 32, pp. 597–617, 1985.

[27] C. K. Yap "An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments", *Discrete and Computational Geometry*, 2, pp. 365–393, 1987.