Optimal Facility Location under Various Distance Functions *

Sergei Bespamyatnikh[1], Klara Kedem[2,3] and Michael Segal[2]

[1]Department of Computer Science
University of British Columbia, Vancouver, B.C. Canada V6T 1Z4

[2]Department of Mathematics and Computer Science
Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel

[3]Computer Science Department
Cornell University, Upson Hall, Cornell University, Ithaca, NY 14853

May 5, 1999

Abstract

We present efficient algorithms for two problems of facility location. In both problems we want to determine the location of a single facility with respect to n given sites. In the first we seek a location that *maximizes* a weighted distance function between the facility and the sites, and in the second we find a location that *minimizes* the sum (or sum of the squares) of the distances of k of the sites from the facility.

1 Introduction

Facility location is a classical problem of operations research that has also been examined in the computational geometry community. The task is to position a point in the plane (the *facility*) such that a distance between the facility and given points (*sites*) is minimized or maximized.

Most of the problems described in the facility location literature are concerned with finding a "desirable" facility location: the goal is to *minimize* a distance function between the facility (*e.g.*, a service) and the sites (*e.g.*, the customers). Just as important is the case of locating an "undesirable" or obnoxious facility. In this case instead of minimizing the largest distance between the facility and the destinations, we maximize the smallest distance. Applications for the latter version are, *e.g.*, locating garbage dumps, dangerous chemical factories or nuclear power plants. The latter problem is unconstrained if the domain of possible locations for the facility is the entire plane. Practically the location of the facility should be in a bounded region R, whose boundary may or may not have a constant complexity.

In this paper we consider the following two problems:

*Work by S. Bespamyatnikh was done while visiting Ben-Gurion University. K. Kedem was supported by a grant from the Israel Science Foundation founded by The Israel Academy of Sciences and Humanities, by a grant from the U.S.-Israeli Binational Science Foundation and by the Mary Upson Award from the College of Engineering at Cornell University. **E-mail: besp@cs.ubc.ca, kedem@cs.cornell.edu, segal@cs.bgu.ac.il.**

1. **Undesirable location.** Let S be a set of n points in the plane, enclosed in a rectangular region R. Let each point p of S have two positive weights $w_1(p)$ and $w_2(p)$. Find a point $c \in R$ which maximizes

$$\min_{p \in S}\{\max\{w_1(p) \cdot d_x(c,p), w_2(p) \cdot d_y(c,p)\}\},$$

   where $d_x(c,p)$ defines the distance between the x coordinates of c and p, and $d_y(c,p)$ defines the distance between the y coordinates of c and p.

2. **Desirable location.** Given a set S of n points and a number $1 \leq k \leq n-1$ find a point p such that sum of the $L_1(L_\infty)$ distances from p to all the subsets of S of size k is minimized.

   For this problem we consider two cases: the *discrete* case – where $p \in S$, and the *continuous* case where p is any point in the plane.

The first problem is concerned with locating an obnoxious facility in a rectangular region R under the weighted L_∞ metric, where each site has two weights, one for each of the axes. An application for two-weighted distance is, *e.g.*, an air pollutant which is carried further by south-north winds than by east-west winds. For the **unweighted** case of this problem, where R is a simple polygon with up to n vertices and under the Euclidean metric, Bhattacharya and Elgindy [4] present an $O(n \log n)$ time algorithm. For weighted sites one can construct the Voronoi diagram and look for the optimal location either on a vertex of this diagram or on the boundary of the region R. Unfortunately, for weighted sites, the Voronoi diagram is known to have quadratic complexity in the worst case, and it can be constructed in optimal $O(n^2)$ time [2]. Thus, the optimal location, using the Voronoi diagram, can be found in $O(n^2)$ time [9]. The first subquadratic algorithm for the weighted problem under L_∞ metric and a rectangular R region was presented by Follert et al. [10]. Their algorithm runs in $O(n \log^4 n)$ time. In this paper we present two algorithms for the two-weighted L_∞ metric problem in a rectangular region.. The first one has $O(n \log^3 n)$ running time and it is based on the parametric searching of Megiddo [11] which combines between the sequential and the parallel algorithms for the decision problem in order to solve the optimization problem. The second algorithm has $O(n \log^2 n)$ running time, and uses the different optimization approach that is described by Megiddo and Tamir [12].

The second problem deals with locating a desirable facility under the *min-sum* criterion. Some applications for this problem are locating a component in a VLSI chip or locating a welding robot in an automobile manufacturing plant. Elgindy and Keil [8] consider a slight variation of the problem under the L_1 metric: Given a positive constant D, locate a facility c that maximizes the number of sites whose sum of distances from c is not greater than D. They also consider the discrete and continuous cases. The runtimes of their algorithms are $O(n \log^4 n)$ for the discrete case and $O(n^2 \log n)$ for the continuous case. Our algorithm for the discrete case runs in time $O(n \log^2 n)$, and for the continuous case in $O((n-k)^2 \log^2 n + n \log n)$ time.

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 $\sqrt{2}$. In what follows we alternate between these metrics to suit our algorithms.

An outline of the paper is as follows. In Section 2 we present two algorithm for solving the first problem. Section 3 describes a data structure and an algorithm for solving the discrete case of Problem 2. Using this data structure and some more observations, we show in Section 4 how to solve the continuous case of Problem 2.

2 Undesirable facility location

In this section we first present a sequential algorithm that answers a decision query of the form: given $d > 0$, determine whether there exists a location $c \in R$ whose x-distance from each point $p_i \in S$ (the distance between the x coordinates of c and p_i) is $\geq d \cdot w_1(p_i)$, and whose y-distance to the points of S is $\geq d \cdot w_2(p_i)$. We will use this sequential algorithm in order to obtain two different algorithms for solving our problem.

The first is based on the parametric search optimization scheme [11] and, tus, we provide a parallel version of the decision algorithm in order to use it. Let T_s denote the runtime of the sequential decision algorithm, and T_p, resp. P, the time and number of processors of the parallel algorithm; then the optimal solution (a point c that maximizes d) can be computed in sequential time $O(PT_p + T_sT_p \log P)$ [11].

The second uses another optimization approach, proposed in [12]. The main idea is to represent a set of potential solutions in a compact, efficient way, use a parallel sorting scheme and then look for our solution by some kind of a binary search. The running time of the algorithm is $O(T_s \log n)$.

2.1 The sequential algorithm

The formulation of the decision problem above implies that each point $p_i \in S$ defines a *forbidden* rectangular region

$$R_i = \{r \in R^2 | d_x(r, p_i) < d \cdot w_1(p_i), d_y(r, p_i) < d \cdot w_2(p_i)\}$$

where c cannot reside. Denote by U_R the union of all the R_i. An *admissible location* for c exists if and only if $R \cap U_R \neq \emptyset$. In other words, we are given a set of n rectangles R_i and want to find whether U_R *covers* R. When each point has the same weight in both axes then the combinatorial complexity of the boundary of U_R is linear. In our case the boundary of U_R has $O(n^2)$ vertices in the worst case.

The problem of finding whether a set of n rectangles covers a rectangular region R has been solved in $O(n \log n)$ time using the *segment tree* T [13]. We outline this well known sequential algorithm for the sake of clarity of our parallel algorithm.

Denote by $L = \{x_1, \ldots x_{2n}\}$ the x coordinates of the endpoints of the horizontal sides of the rectangles. We call the elements of L the *instances* of T. Similarly, let $M = \{y_1, \ldots y_{2n}\}$ be the list of y coordinates of the endpoints of the vertical sides of the rectangles. Assume each list is sorted in ascending order. The leaves of the segment tree T contain *elementary segments* $[y_i, y_{i+1})$, $i = 1, \ldots, 2n-1$, in their *range* field. The range at each inner node in T contains the union of the ranges in the nodes of its children.

A vertical line is swept over the plane from left to right stopping at the instances of T. At each instance x, 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 from) T (v is stored in $O(\log n)$ nodes and is equal to the disjoint union of the ranges of these nodes). The update of T at instance x involves maintaining a *cover number* in the nodes. The cover number at a node counts how many vertical rectangle sides cover the range of this node and do not cover the range of its parent. If at deleting a rectangle the height of R is not wholly covered by all the vertical segments that are currently in T, then the answer to the decision problem is "yes". Namely, we found a point in R which is not in U_R, and we are done. If the answer is "no" then we update T and proceed to the next instance. Thus

Lemma 1 *Given a fixed $d > 0$ we can check in $O(n \log n)$ time, using $O(n)$ space, whether there*

exists a point $c \in R$, such that for every point $p_i \in S$ the following holds: $d_x(c, p_i) \cdot w_1(p_i) \geq d$ and $d_y(c, p_i) \cdot w_2(p_i) \geq d$.

2.2 The parallel version and the optimization

Next we present the parallel algorithm which we believe is of independent interest. In order to produce an efficient parallel algorithm for the decision problem we add some information into the nodes of T. This information encaptures the cover information at each node, as will be seen below.

Let $L = \{x_1, x_2, \ldots, x_{2n}\}$ be the list of instances as above. Let the projection of a rectangle R_j on the x axis be $[x_i, x_k]$. We associate with R_j a *life-span* integer interval $l_j = [i, k]$. Let v_j be the projection of R_j on the y axis. The integer interval l_j defines the instances at which the segment v_j is stored in T during the sequential algorithm. We augment T by storing the life-span of each vertical segment v_j in the $O(\log n)$ nodes of T that v_j updates. We further process each node in T so that it contains a list of *cover two* life-ranges. This is a list of intervals consisting of the pairwise intersections of the life-spans in the node. For example, assume that a node s contains the life-spans $[1, 7], [3, 4]$ and $[5, 6]$. The list of cover two at s is $[3, 4]$ and $[5, 6]$. If a vertical segment is to be deleted from s at instances x_1, x_2 or x_7, then s will be exposed after the deletion. But if the deletion occurs at instance x_3, x_4, x_5 or x_6 then, since the cover of s is 2 at this instance, s will not be exposed by deleting v_j.

Our parallel algorithm has two phases: phase I constructs the augmented tree T and phase II checks whether R gets exposed at any of the deletion instances.

Phase I The segment tree T can be easily built in parallel in time $O(\log n)$ using $O(n \log n)$ processors [1]. Unlike in Lemma 1 above, where we store in each node just the cover number, here we store for each segment its life-span in $O(\log n)$ nodes. Thus T occupies now $O(n \log n)$ space [13]. Adding the cover two life-span intervals is performed as follows.

We sort the list of life-spans at each node according to the first integer in the interval that describes a life-span. We merge the list of life-spans at each node as follows. If two consecutive life-spans are disjoint we do not do anything. Assume the two consecutive life-spans $[k_1 k_2]$ and $[g_1, g_2]$ overlap. We produce two new *life-ranges*:

(a) the life-range of cover at least one – $[k_1, \max(k_2, g_2)]$ and

(b) the life-range of cover at least two – $[g_1, \min(k_2, g_2)]$.

We continue to merge the current life-range of cover at least one from item (a) above with the next life-span in the list till the list of life-spans is exhausted. We next merge the cover two life-ranges into a list of disjoint intervals by taking the unions of overlapping intervals. At this stage each node has two lists of life-ranges. But this does not suffice for phase II. For each node we have to accumulate the cover information of its descendants. Starting at the leaves we recursively process the two lists of life-ranges at the nodes of T separately. We describe dealing with the list of cover one. Assume the two children nodes of a node s contain intersecting life-ranges, then this intersection interval is an interval of instances where all the range of s is covered. We copy the intersection interval into the node s. When we are done copying we merge the copied list with the node's life-span list as described above, and then merge the list of cover two with the copied list of cover two, by unioning overlapping intervals.

Phase II. Our goal in this phase is to check in parallel whether, upon a deletion instance, the height of R is still fully covered or a point on it is exposed. We do it as follows. Assume that the vertical segment v is deleted from T at the j^{th} instance. We go down the tree T in the nodes that

store v and check whether the life-span lists at **all** these nodes contain the instance j in their list of cover two. If they do then (the height of) R is not exposed by deleting v.

Complexity of the algorithm.

It is easy to show that the life-range lists do not add to the amount of required storage. The number of initial life-span intervals is $O(n \log n)$. The number of initial life-ranges of cover two cannot be greater than that. It has been shown [7] that copying the lists in the nodes in the segment tree to their respective ancestors does not increase the asymptotic space requirement. The augmentation of T is performed in parallel time $O(\log n)$ with $O(n \log n)$ processors as follows. We allocate a total of $O(n \log n)$ processors to merge the life-span ranges in the nodes of T, putting at each node a number of processors which is equal to the number of life-span ranges in the node. Thus the sorting and merging of the life-span ranges is performed in parallel in time $O(\log n)$.

The checking phase is performed in parallel by assigning $O(\log n)$ processors to each deletion instance. For the deletion of a vertical segment v, one processor is assigned to each node that stores v. These processors perform in parallel a binary search on the cover two life-ranges of these nodes. Thus the checking phase is performed in time $O(\log n)$.

Summing up the steps of the parallel algorithm, we get a total of $O(\log n)$ parallel runtime with $O(n \log n)$ processors and $O(n \log n)$ space. Plugging this algorithm to the parametric search paradigm [11] we get

Theorem 2 *Given a set S of n points in the plane, enclosed in a rectangular region R, and two positive weights $w_1(p)$ and $w_2(p)$ for each point $p \in S$, we can find, in $O(n \log^3 n)$ time, a point $c \in R$ which maximizes*

$$\min_{p \in S}\{\max\{w_1(p) \cdot d_x(c,p), w_2(p) \cdot d_y(c,p)\}\}.$$

2.3 Another approach

By carefully looking at the respective Voronoi diagram we have the following crucial observation.

Observation 3 *Assume that the optimal solution is not attained on the boundary of the rectangle. Then, w.l.o.g., there is an optimal point c, and two points p and q such that either*

$$w_1(p)d_x(c,p) = w_1(q)d_x(c,q) = optimalvalue,$$

or

$$w_2(p)d_y(c,p) = w_2(q)d_y(c,q) = optimalvalue.$$

The above observation with a given assumption implies that the optimal value is an element in one of the following four sets:
$S_1 = \{(p_x + q_x)/(1/w_1(p) + 1/w_1(q)) : p, q \in S\}$, $S_2 = \{(p_x - q_x)/(1/w_1(p) - 1/w_1(q)) : p, q \in S\}$, $S_3 = \{(p_y - q_y)/(1/w_2(p) - 1/w_2(q)) : p, q \in S\}$, $S_4 = \{(p_y + p_y)/(1/w_2(p) + 1/w_2(q)) : p, q \in S\}$.

Megiddo and Tamir [12] describe how to search for the optimal value r^* within a set of the form: $S' = \{(a_i + b_j)/(c_i + d_j) : 1 \leq i, j \leq n\}$. Thus there will be given $4n$ numbers $a_i, b_j, c_i, d_j (1 \leq i, j \leq n)$, and we will have to find two elements $s, t \in S'$ such that $s < r^* \leq t$ and no element of S' is strictly between s and t. We briefly describe their [12] approach.

Set S' consists of the points of intersection of straight lines $y = (c_i x - a_i) + (d_j x - b_j)$ with the x-axis. The search will be conducted in two stages. During the first stage we will identify an interval $[s_1, t_1]$ such that $s_1 < r^* \leq t_1$ and such that the linear order induced on $\{1, \ldots, n\}$ by the numbers $c_i x - a_i$ is independent of x provided $x \in [s_1, t_1]$. The rest of work is done in Stage 2.

Stage 1. We search for r^* among the points of intersections of lines $y = c_i x - a_i$ with each other. The method is based on parallel sorting scheme. Imagine that we sort the set $\{1, \ldots, n\}$ by the $(c_i x - a_i)$'s, where x is not known yet. Whenever a processor has to compare some $c_i x - a_i$ with $c_j x - a_j$, we will in our algorithm compute the critical value $x_{ij} = (a_i - a_j)/(c_i - c_j)$. We use Preparata [14] parallel sorting scheme with $n \log n$ processors and $O(\log n)$ steps. Thus, a single step in Preparata scheme gives rise to the production of $n \log n$ points of intersection of lines $y = c_i x - a_i$ with each other. Given these $n \log n$ points and an interval $[s_0, t_0]$ which contains r^*, we can in $O(n \log n)$ time narrow down the interval so that it will still contain r^* but no intersection point in its interior. This requires the finding of medians in sets of cardinalities $n \log n$, $\frac{1}{2} n \log n$, $\frac{1}{4} n \log n, \ldots$ plus $O(\log n)$ evaluations of the sequential algorithm for the decision problem. Since the outcomes of the comparisons so far are independent of x in the updated interval, we can proceed with the sorting even though x is not specified. The effort per step is hence $O(n \log n)$ and the entire Stage 1 takes $O(n \log^2 n)$ time.

Stage 2. When the second stage starts we can ssume w.l.o.g. that for $x \in [s_1, t_1]$ $c_x - a_i \leq c_{i+1} - a_{i+1}$, $i = 1, \ldots, n-1$. Let $j (1 \leq j \leq n)$ be fixed and consider the set S_j of n lines $S_j = \{y = c_i x - a_i + d_j x - b_j, i = 1, \ldots, n\}$. Since S_j is "sorted" over $[s_1, t_1]$, we can find in $O(\log n)$ evaluations of the sequential algorithm for the decision problem a subinterval $[s_1^j, t_1^j]$ such that $s_1^j < r^* \leq t_1^j$, and that no member of S_j intersects the x-axis in the interior of this interval. We work on the S_j's in parallel. Specifically, there will be $O(\log n)$ steps. During a typical step, the median of the remainder of every S_j is selected (in $O(1)$ time) and its intersection point with the x-axis is computed. The set of these n points is then searched for r^* and the interval is updated accordingly. This enablesus to discard a half from each S_j. Clearly a single step lasts $O(n \log n)$ time and the entire stage is carried out in $O(n \log^2 n)$ time.

At the end of second stage we have the values $\{s_1^j\}$ and $\{t_1^j\}, j = 1, \ldots, n$. Defining $s = \max_{1 \leq j \leq n} \{s_1^j\}$ and $t = \min_{1 \leq j \leq n} \{t_1^j\}$ we obtain $s < r^* \leq t$, and no element of S' is strictly between s and t.

The case with the optimal solution attained on the boundary of the rectangle can be treated as subcase of a previous case. Thus we conclude by a theorem.

Theorem 4 *Given a set S of n points in the plane, enclosed in a rectangular region R, and two positive weights $w_1(p)$ and $w_2(p)$ for each point $p \in S$, we can find, in $O(n \log^2 n)$ time, a point $c \in R$ which maximizes*

$$\min_{p \in S} \{\max\{w_1(p) \cdot d_x(c,p), w_2(p) \cdot d_y(c,p)\}\}.$$

3 The discrete desirable facility location problem

The discrete min-sum problem is defined as follows. Given a set S of n points in the plane and a number k. Find a point in S such that the sum of distances from it to its k nearest neighbors in S is minimized. Our algorithms compute, for each point of S, the sum of distances from it to its k nearest neighbors in S, and output a point which minimizes the sum. First we deal with the special case of the discrete min-sum problem when $k = n - 1$.

3.1 The discrete min-sum problem for $k = n - 1$

This min-sum problem appears in [3] with an $O(n^2)$ trivial solution. Below we present an algorithm that solves this problem for the L_1 metric in $O(n \log n)$ time.

The L_1 metric is separable, in the sense that the distance between two points is the sum of their x and y-distances. Therefore we can solve the problem for the x and y-coordinates separately. We regard the x coordinates part. We sort the points according to their x-coordinates. Let $\{p_1, \ldots, p_n\}$ be the sorted points. For each $p_i \in S$ we compute the sum σ_i^x of the x-distances from p_i to the rest of the points in S. This is performed efficiently as follows. For the point p_1 we compute σ_1^x by computing and summing up each of the $n-1$ distances. For $1 < i \leq n$ we define σ_i^x recursively: assume the x-distance between p_{i-1} and p_i is δ, then $\sigma_i^x = \sigma_{i-1}^x + \delta \cdot (i-1) - \delta \cdot (n-i+1)$. Clearly the sums σ_i^x (for $i = 1, \ldots n$) can be computed in linear time when the points are sorted. We compute σ_i^y analogously. Assume the point $p \in S$ is i^{th} in the x order and j^{th} in the y order. The sum of distances from p to the points in S is $\sigma_{ij} = \sigma_i^x + \sigma_j^y$. The point which minimizes this sum is the sought solution.

Theorem 5 *Given a set S of n points in the plane sorted in x direction and in y directions, we can find in linear time a point $p \in S$ which minimizes the sum of the L_1 distances to the points in S.*

We can extend this theorem to the case where the distance to be minimized is the sum of squared L_2 distances from a point to the rest of the points of S, since the separability property holds for this case as well. Assume we have computed $\{\sigma_1^x, \ldots, \sigma_n^x\}$ above and let $\tau_i^x = \sum_{j=1}^n (x_j - x_i)^2$. The recursion formula for computing all the squared x-distances is easily computed to be

$$\tau_i^x = \tau_{i-1}^x - 2\delta\sigma_{i-1}^x - n\delta^2$$

where the x-distance between p_{i-1} and p_i is δ.

Corollary 6 *Given a set S of n points in the plane, sorted in x direction and in y direction, we can find in linear time a point p of S which minimizes the sum of squared L_2 distances to the points in S.*

3.2 The general case

We turn to the discrete min-sum problem for $1 \leq k \leq n-1$. We describe the algorithm for the L_∞ metric. It has two phases. In the **first phase** we find, for each point $p_i \in S$, the smallest square R_i centered at p_i which contains at least $k+1$ points of S. We also get the *square size* λ_i which is defined as half the side length of R_i. In the **second phase** we compute for each p_i, $i = 1, \ldots, n$, the sum of the distances from it to the points of S in R_i and pick i for which this sum is minimized.

For the **first phase** we apply a simple version of parametric searching. Assume $q = (q_x, q_y) \in S$ is the query point for which we want to find the smallest square R which contains at least $k+1$ points of S. For a parameter λ, denote by $R(\lambda)$ a square of size λ centered at q. We test whether $R(\lambda)$ contains at least $k+1$ points of S by applying Chazelle's [5, 6] orthogonal range counting. Namely, given a set of n points in the plane and an orthogonal range, find the number of points contained in the range. 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 region can be answered in time $O(\log n)$.

Clearly the minimum value of λ is the distance from the query point to its k^{th} nearest neighbor. Thus candidate values for λ are $|q_x - p_x|$ and $|q_y - p_y|$ for all $p = (p_x, p_y) \in S$. By performing a binary search in the sets $\{p_x \mid p \in S, p_x > q_x\}$, $\{p_x \mid p \in S, p_x < q_x\}$, $\{p_y \mid p \in S, p_y > q_y\}$ and $\{p_y \mid p \in S, p_y < q_y\}$, we find the smallest λ such that $R(\lambda)$ contains at least $k+1$ points of S.

Lemma 7 *Given a set S of n points and a positive integer $k < n$. We can find for each point $p_i \in S$ the smallest square centered at p_i that contains at least $k+1$ points of S in total time $O(n \log^2 n)$.*

In the **second phase** we compute, for each point $p_i \in S$, the sum of distances from p_i to its k nearest neighbors, namely, the points of S which are contained in R_i. In order to compute efficiently the sums of distances in all the squares R_i, we apply the orthogonal range searching algorithm for weighted points of Willard and Lueker [15] which is defined as follows. Given n weighted points in d-space and a query d-rectangle Q, compute the accumulated weight of the points in Q. The data structure in [15] 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.

Let $q \in S$ be the point for which we want to compute the sum of distances from it to its k nearest neighbors. Let R be the smallest square found for q in the first phase. Clearly R can be decomposed into four triangles by its diagonals such that the L_∞ distance between all points of S within one triangle is, wlog, the sum of x coordinates of the points of S in this triangle minus the x coordinate of q times the number of points of S in this triangle. More precisely, let Δ_u be the closed triangle whose base is the upper side of R and whose apex is q. Denote by σ_u the sum of the L_∞ distances between the points in Δ_u and q, and by N_u the number of points of $S_u = \{S - q\} \cap \Delta_u$, then

$$\sigma_u = \sum_{p_j \in S_u} p_j^y - q_y \cdot N_u.$$

Our goal in what follows is to prepare six data structures for orthogonal range search for weighted points, as in [15], three with the weights being the x coordinates of the points of S and three with the y coordinates as weights, and then to define orthogonal ranges, corresponding to the triangles in R for which the sums of x (y) coordinates will be computed.

We proceed with computing σ_u. Let l_1 be the x axis, l_2 be a line whose slope is 45° passing through the origin, l_3 be the y axis and l_4 a line whose slope is 135° passing through the origin. These lines define wedges (see Figure 1(a)): (1) Q_1–the wedge of points between l_1 and l_2 whose x coordinates are larger than their y coordinates, (2) Q_2–the wedge of points between l_2 and l_3 whose y coordinates are larger than their x coordinates, and (3) Q_3–the wedge of points between l_3 and l_4 whose y coordinates are larger than their x coordinates.

Each of these wedges defines a data structure, as in [15]. Observe, *e.g.*, the wedge Q_1. We transform l_1 and l_2 into corresponding axes of an orthogonal coordinate system, and apply the same transformation on all the points $p_i \in S$. We construct the orthogonal range search data structure for the transformed points with the original y coordinates as weights. (Similarly we construct data structures for the points of S transformed according to Q_2 and Q_3, respectively, for the y sums, and another set of three data structures for the x sums.)

We denote by $Q_i(q)$ the wedge Q_i translated by q. Denote by $Y_i(q)$ the sum of the y coordinates of the points of S in $Q_i(q)$, $i = 1, 2, 3$. Then

$$\sum_{p_j \in S_u} p_j^y = (Y_2(q) + Y_3(q)) - (Y_2(q_1) + Y_3(q_1)) - Y_1(q_1) + Y_1(q_2),$$

where $q_1 = (q_x - \lambda, q_y + \lambda)$ and $q_2 = (q_x + \lambda, q_y + \lambda)$ (see Figure 1(b)). If the segment $[q_1, q_2]$ contains points of S we define q_1 and q_2 as $q_1 = (q_x - \lambda - \epsilon, q_y + \lambda + \epsilon)$ and $q_2 = (q_x + \lambda + \epsilon, q_y + \lambda + \epsilon)$ for some sufficiently small $\epsilon > 0$.

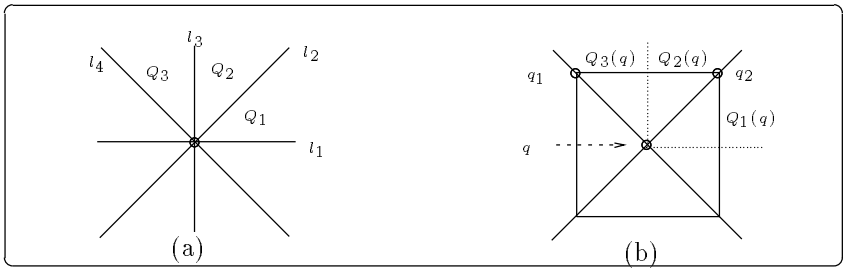

Figure 1: (a) The regions Q_i and (b) $Q_i(q)$

To compute N_u we can use the same wedge range search scheme, but with unit weights on the data points (instead of coordinates). In a similar way we compute the sum σ_d for the lower triangle in R (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).

It is possible that R contains more than $k+1$ points – this happens when more than one point of S is on the boundary of R. Our formula for the sum of the L_∞ distances should be

$$D = \sigma_u + \sigma_d + \sigma_l + \sigma_d - \lambda \cdot (N_u + N_d + N_l + N_r - k - 1).$$

Hence, the second phase of the algorithm, requires $O(n \log n)$ preprocessing time and space, and then $O(\log^2 n)$ query time per point $p_i \in S$ to determine the sum of distances to its k nearest points. Thus, for both phases, we conclude

Theorem 8 *The discrete min-sum problem in the plane for $1 \leq k \leq n-1$ and under L_∞-metric, can be solved in time $O(n \log^2 n)$ occupying $O(n \log n)$ space.*

4 The continuous desirable facility location problem

The continuous desirable facility location problem is defined as follows. Given a set S of n points and a parameter $1 \leq k \leq n-1$. Find a point c in the plane such that the sum of distances from c to its k nearest points from S is minimized. We consider the problem where the distances are measured by the L_1 metric.

We create a grid M by drawing a horizontal and a vertical line through each point of S. Assume the points of S are sorted according to their x coordinates and according to their y coordinates. Denote by $M(i,j)$ the grid point that was generated by the i^{th} horizontal line and the j^{th} vertical line in the y and x orders of S respectively. Bajaj [3] observed that the solution to the *continuous* min-sum problem with $k = n-1$ should be a grid point. As a matter of fact it has been shown that for this problem the point $M(\lfloor n/2 \rfloor, \lfloor n/2 \rfloor)$ is the required point. (Where for an even n the solution is not unique and there is a whole grid rectangle whose points can be chosen as the solution.)

For $k < n-1$, we can pick the solution from $O((n-k)^2)$ grid points, since the smallest x-coordinate that c might have is $x_{\lfloor k/2 \rfloor}$, and the largest $x_{n-\lceil k/2 \rceil}$ (similarly for y). This is true since in the extreme case where all the k points are the lowest leftmost points then according to Bajaj the solution to this k points problem is at $M(\lfloor k/2 \rfloor, \lfloor k/2 \rfloor)$. Similarly if the k points are located at any other corner of M. Thus we remain with $(n-k+1)^2$ grid points which are candidates for the solution c. Applying the discrete algorithm of Section 3.2, with the query points being the candidate solutions, we obtain the following theorem.

Theorem 9 *The continuous min-sum problem can be solved in* $(n \log n + (n-k)^2 \log^2 n)$ *time for any positive* $k \leq n-1$.

References

[1] M. Attalah, R. Cole, M. Goodrich, "Cascading divide and conquer: a technique for designing parallel algorithms", *SIAM Journal on Computing*, 18(3)(1989), pp. 499-532.

[2] F. Aurenhammer and H. Edelsbrunner "An optimal algorithm for for constructing the weighted Voronoi diagram in the plane", *Pattern Recognition*, 17(2), 1984, pp. 251–257.

[3] C. Bajaj, "Geometric optimization and computational complexity", Ph.D. thesis, Tech. Report TR-84-629, Cornell University, 1984.

[4] B. Bhattacharya, H. Elgindy, "An efficient algorithm for an intersection problem and an application", Tech. Report 86-25, Dept. of Comp. and Inform. Sci., University of Pennsylvania, 1986.

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

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

[7] B. Chazelle and H. Edelsbrunner and L. Guibas and M. Sharir "Algorithms for bichromatic line segment problems and polyhedral terrains", *Algorithmica*, 11 (1994), pp. 116–132.

[8] H. Elgindy, M. Keil, "Efficient algorithms for the capacitated 1-median problem", *ORSA J. Comput*, 4(1982), pp. 418–424.

[9] F. Follert "Lageoptimierung nach dem Maximin-Kriterium", Diploma Thesis, Univ. d. Saarlandes, Saarbrucken, 1984.

[10] F. Follert, E. Schömer, J. Sellen, "Subquadratic algorithms for the weighted maximin facility location problem", in *Proc. 7th Canad. Conf. Comput. Geom.*, 1-6, 1995.

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

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

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

[14] F. Preparata "New parallel-sorting schemes", *IEEE Trans. Comput.*, C-27, pp. 669-673, 1978.

[15] D.E. Willard, G.S. Lueker, "Adding range restriction capability to dynamic data structures", in *J. ACM*, 32(1985), pp. 597–617.