CHAPTER 3

Rectilinear Minimax Location Problem

1.1 Problem Formulation

Let $S = \{(a_i, b_i) : i = 1, 2, ..., n\}$ be a set of demand points in the two-dimensional Euclidean plane R^2. Assume $I = \{1, 2,..., n\}$. The problem we are going to consider here is the following:

$$\text{Minimize} \quad f(x, y) \qquad (1)$$
$$(x, y) \in R^2$$

where $$f(x, y) = \max_{i \in I} [\, w_i (|x - a_i| + |y - b_i|) + g_i \,] \qquad (2)$$

In (1) w_i denotes the weight associated with the i-th demand point and g_i is the nonnegative constant which may be interpreted as the time required by the user i to prepare to go to the centre.

The linear programming formulation of the problem (1) subject to the condition (2) can be written as [25]:

$$\text{Minimize} \quad z$$

subject to

$$x + y \le a_i + b_i + (z - g_i)/w_i$$
$$x + y \ge a_i + b_i - (z - g_i)/w_i$$
$$-x + y \le -a_i + b_i + (z - g_i)/w_i$$
$$-x + y \ge -a_i + b_i - (z - g_i)/w_i$$

where $i \in I$.

It is to be noted that for each demand point the linear programming model requires four inequality constraints. If there are n demand points then we have to consider four vectors each of which consists of 4n components. The algorithms we are going to develop will require only four vectors each containing n components. Consequently we can solve a class of problems having a large number of demand points, which are difficult to solve by using the linear programming model. We have also shown that the present algorithms require much less computer CPU time to solve a weighted rectilinear minimax location problem than the linear programming method.

In the next section we introduce some definitions and basic concepts which will be used

In the next section we introduce some definitions and basic concepts which will be used to develop our algorithms.

1.2 Some Fundamental Concepts

The following definitions, properties and notation are provided for convenience.

Definition 1.2.1. The Euclidean distance of any point P(x, y) from another point $P_i(a_i, b_i)$ is denoted by $ed(P, P_i)$ and is defined by

$$ed(P, P_i) = \sqrt{(x - a_i)^2 + (y - b_i)^2}$$

Definition 1.2.2. The generalized weighted rectilinear distance of P(x, y) from any demand point $P_i(a_i, b_i)$ is denoted by $rd(P, P_i)$ and is defined by

$$rd(P, P_i) = w_i (| x - a_i | + | y - b_i |) + g_i$$

Definition 1.2.3. Given two demand points $P_i(a_i, b_i)$ and $P_j(a_j, b_j)$, and a point P(x, y) we define the difference function, to be denoted by $\Delta(x, y; a_i, b_i; a_j, b_j)$, in the following manner:

$$\Delta(x, y ; a_i, b_i ; a_j, b_j) = rd(P, P_i) - rd(P, P_j)$$

Definition 1.2.4. SR is the smallest rectangle containing all demand points, $P(a_i, b_i)$, in the x-y plane. The boundary ∂SR of SR consists of the four straight lines, viz.,

$$x = \max_{i \in I} \{ a_i \}; x = \min_{i \in I} \{ a_i \}; y = \max_{i \in I} \{ b_i \}; y = \min_{i \in I} \{ b_i \}$$

Definition 1.2.5. The L-shaped path obtained by joining the points (a_i, b_i) to (a_i, b_j) to (a_j, b_j) by straight line segments is denoted by $L(P_i, P_j)$.

Definition 1.2.6. Consider the rectangle a pair of whose opposite vertices is

$$P_i(a_i, b_i) \text{ and } P_j(a_j, b_j)$$

having sides are parallel to the coordinate axes. We denote this rectangle by $R(P_i, P_j)$ and its boundary by $\partial R(P_i, P_j)$.

Given a point P(x, y) the value of the objective function f(x, y) is given by (2). Consider another point Q(x + dx, y + dy) in the neighbourhood of P(x, y).

Definition 1.2.7. If $f(x, y) \geq f(x + dx, y + dy)$, the direction obtained by joining the line segment from P to Q is said to be a descent direction.

Definition 1.2.8. The locus of P(x, y), for which

$$\Delta(x, y; a_i, b_i ; a_j, b_j) = 0,$$

will be called an equipolygon corresponding to the demand points P_i and P_j, and will be denoted by $EP(P_i, P_j)$.

The equation of the equipolygon $EP(P_i, P_j)$ can be written as

$$w_i[u_i(x - a_i) + v_i(y - b_i)] + g_i = w_j[u_j(x - a_j) + v_j(y - b_j)] + g_j$$

where u_k and v_k, $(k = i, j)$, are given by

$$u_k = \begin{cases} 1 & \text{if } x > a_k \\ 0 & \text{if } x = a_k \\ -1 & \text{if } x < a_k \end{cases}$$

$$v_k = \begin{cases} 1 & \text{if } y > b_k \\ 0 & \text{if } y = b_k \\ -1 & \text{if } y < b_k \end{cases}$$

We now mention some important properties, of an equipolygon, which follow immediately from the definition of the equipolygon. We will use these properties to develop of our algorithms.

(P1) Inclination of the edges of an equipolygon with x-axis

Let $u = u_i + u_j$ and $v = v_i + v_j$.

The edge(s) of $EP(P_i, P_j)$ for which $u = 2$ and $v = -2$ or $u = -2$ and $v = 2$, will be inclined at an angle of 45° with the positive direction of the axis of x.

When u and v are each equal to 2 or -2 the edges of $EP(P_i, P_j)$ will make an angle of 135° with the positive direction of x-axis.

For all other location of (x, y) the angle between the edges of $EP(P_i, P_j)$ and x-axis measured in the counter clockwise sense are

$$\arctan\left\{\frac{(w_i u_i - w_j v_j)}{(w_i v_j - w_j v_i)}\right\}$$

(P2) Number of edges of an equipolygon

For the sake of definiteness let us assume that $w_i > w_j$ and $\Delta(a_i, b_i; a_i, b_i; a_j, b_j) < 0$.

(i) If $\Delta(a_j, b_i; a_i, b_i; a_j, b_j) > 0$ and $\Delta(a_i, b_j; a_i, b_i; a_j, b_j) > 0$ then the equipolygon $EP(P_i, P_j)$ will have four sides.

In Figure-1, A and B are demand points, ACBD is R (A, B) and PQMZ is the equipolygon corresponding to the demand points A and B with four sides.

(ii) If $\Delta(a_j, b_i; a_i, b_i; a_j, b_j) > 0$ and $\Delta(a_i, b_j; a_i, b_i; a_j, b_j) < 0$

or $\Delta(a_j, b_i; a_i, b_i; a_j, b_j) < 0$ and $\Delta(a_i, b_j; a_i, b_i; a_j, b_j) > 0$

then the equipolygon $EP(P_i, P_j)$ will have six sides.

In Figure-2, PQMNTL denotes the equipolygon, corresponding to the demand points A and B, having six sides.

(iii) In all other situations number of sides of the equipolygon $EP(P_i, P_j)$ will be eight.

It is to be noted that for small values of g_i and g_j the equipolygon $EP(P_i, P_j)$ will have either four or six sides. In our subsequent discussions we assume that all g_i's are small.

(P3) In cases (i) and (ii) mentioned above equipolygons $EP(P_i, P_j)$ are all closed and the point possessing greater weight lies within them.

(P4) If $\Delta(a_i, b_i; a_i, b_i; a_j, b_j) < 0$, $\Delta(a_j, b_j; a_i, b_i; a_j, b_j) < 0$ and $w_i = w_j$ then the equipolygon is no longer closed. It then has three sides; one of them lies within $R(P_i, P_j)$ having inclination 45° or 135° with the positive direction of the x-axis and the other sides are semi-infinite lines parallel to x and y axes depending on the values of g_i, g_j, $|a_i - a_j|$ and $|b_i - b_j|$.

Definition 1.2.9. The point of intersection of two or more equipolygons will be called a **point of equality** with respect to the intersecting equipolygons. The point common to the equipolygons $EP(P_i, P_j)$ and $EP(P_i, P_k)$, i.e., the point of equality of $EP(P_i, P_j)$ and $EP(P_i, P_k)$ will be denoted by $EQ(P_i, P_j, P_k)$.

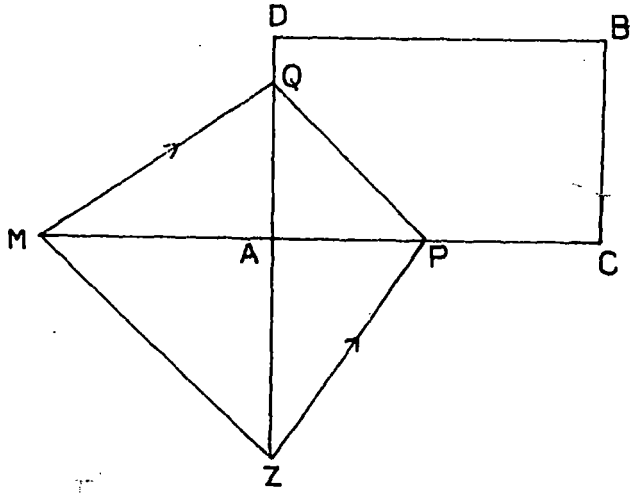

Figure - 1

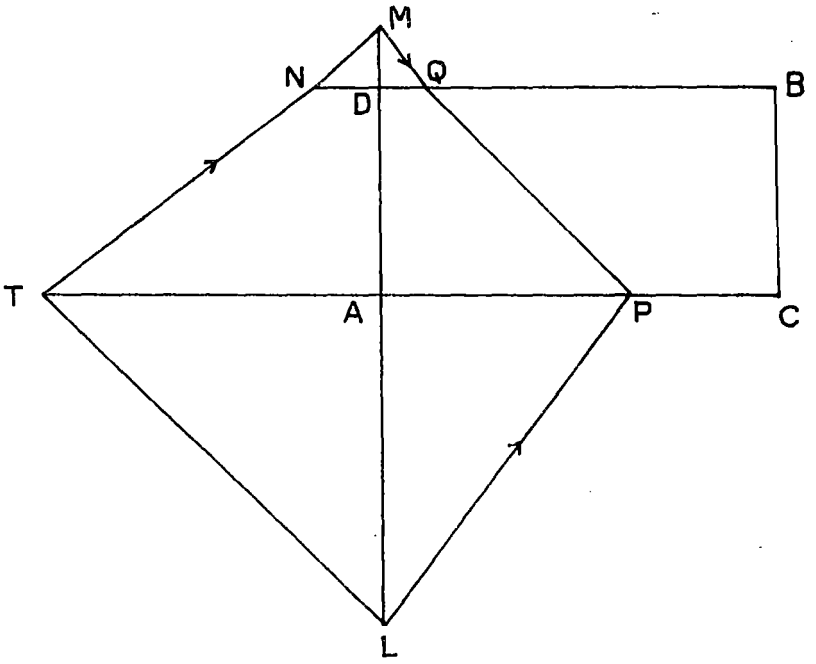

Figure – 2

An Efficient Algorithm for Rectilinear Minimax Location Problem

2.1 Some Lemmas Related to the Problem

In this section we are going to develop an algorithm of the problem (1) subject to the constraint (2), mentioned in section 1.1 of this chapter, based on the concept of dual feasibility technique. Elzinga and Hearn[23] used this technique to solve a Euclidean minimax location problem. Hearn and Vijoy explained this method in detail (see [29]). We now prove the following lemma which is directly related to our algorithm.

Lemma 2.1.1. The equipolygon corresponding to two different demand points $P_1(a_1, b_1)$ and $P_2(a_2, b_2)$ having weights w_1 and w_2 respectively, and small response parameters g_1, g_2 intersects $R(P_1, P_2)$.

Proof. Since g_1 and g_2 are small it follows immediately from the definition of the difference function that

$$\Delta(a_1, b_1; a_1, b_1; a_2, b_2) = g_1 - rd(P_1, P_2) < 0 \quad (1)$$

$$\Delta(a_2, b_2; a_1, b_1; a_2, b_2) = rd(P_2, P_1) - g_2 > 0 \quad (2)$$

Consider the point $P(a_2, b_1)$. Then, we have

$$\Delta(a_2, b_1; a_1, b_1; a_2, b_2) = rd(P, P_1) - rd(P, P_2) \quad (3)$$

If the expression (3) is positive then the linear function

$$F(x) = \Delta(x, b_1; a_1, b_1; a_2, b_2),$$

has opposite signs at $x = a_1$ and $x = a_2$, and being continuous, must have one and only one zero at $x = \alpha$ where α lies between a_1 and a_2. On the contrary expression (3) less than zero implies that the linear function

$$G(y) = \Delta(a_2, y; a_1, b_1; a_2, b_2),$$

must vanish at $y = \beta$ where β lies between b_1 and b_2. From the above discussion it follows that the equipolygon $EP(P_1, P_2)$ intersects either the boundary $y = b_1$ or $x = a_2$ of $R(P_1, P_2)$. Similarly it can be proved that the equipolygon $EP(P_1, P_2)$ intersects either the boundary $y = b_2$ or $x = a_1$ of $R(P_1, P_2)$. That is the equipolygon $EP(P_1, P_2)$ intersects $\partial R(P_1, P_2)$ exactly at two points provided $R(P_1, P_2)$ is not a degenerate rectangle in which case we get only one point of intersection.

The above lemma not only proves the existence of some portion of the equipolygon $EP(P_1, P_2)$ within $R(P_1, P_2)$ but also provides a method of finding the point of intersections of $EP(P_1, P_2)$ and $\partial R(P_1, P_2)$.

It follows from the definition of the equipolygon and Lemma 2.1.1 that the inclination of the portion of equipolygon, for two demand points P_i and P_j within $R(P_i, P_j)$, with the positive direction of x-axis is either 45^0 or 135^0.

In figure 1, for the demand points A and B, ACBD denotes R(A, B) and PQ represents the portion of EP(A, B) within R(A, B). The sides AC and DB are parallel to the axis of x. The inclination of PQ with the positive direction of x-axis is 135^0.

We now use these definitions, properties and Lemma 2.1.1 to develop our algorithm in the next section.

2.2 Algorithm of the Problem

We first note that the optimal solution of the minimax problem will occur on $\partial R(P_i, P_j)$ or within $R(P_i, P_j)$ for a given pair of demand points P_i, P_j; because any movement from a point on $EP(P_i, P_j)$, outside $R(P_i, P_j)$, toward the nearest boundary $\partial R(P_i, P_j)$ and perpendicular to it, will cause the value of the objective function to decrease.

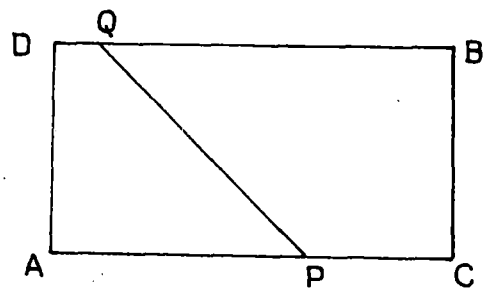

Figure - 1

Algorithm. Rectilinear Minimax Location Problem

Initial Step. Take any two demand points, say $A(a_1, b_1)$ and $B(a_2, b_2)$. Find the points of intersections of EP(A, B) and $\partial R(A, B)$. Denote these points by $P(u_1, v_1)$ and $Q(u_2, v_2)$. Let d = rd(A, P) and go to Step 1.

Step 1. If for all demand points i, both $rd(P_i, P)$ and $rd(P_i, Q)$ are less than or equal to d then stop; any point on the line segment PQ is a facility point

Else go to Step 2.

Step 2. If for some demand point P_i, both $rd(P_i, P)$ and $rd(P_i, Q)$ are greater than d then $A \leftarrow P_i$ and go to Step 3

Else go to Step 4.

Step 3. Find a demand point, say P_k, such that the generalized rectilinear distances of it from the points of intersections, $P(u_1, v_1)$ and $Q(u_2, v_2)$, of $EP(A, P_k)$ and $\partial R(A, P_k)$ is greater than d. Let $d = rd(A, P)$ and $B \leftarrow P_k$; repeat Step 1.

Step 4. If for some demand point P_i, either $d1 = rd(P, P_i)$ or $d2 = rd(Q, P_i)$ is greater than d then obtain the point of intersection of $EP(A, P_i)$ with the line segment PQ. Let T(a, b) be this point.

If $d1 > d$, then $P \leftarrow T$

Else $Q \leftarrow T$

and go to Step 1.

Remarks

1. If we do not encounter Step3 of the algorithm then the optimum solution of the minimax problem can be obtained in a linear time. On the other hand, in the worst case, the complexity of the algorithm is $O(n^2)$.

2. Points of intersection of the equipolygon EP(A, B) and $\partial R(A, B)$ are given by the following expressions:

Let the demand points $A(a_1, b_1)$ and $B(a_2, b_2)$ have weights w_1 and w_2, respectively, and small response parameters g_1, g_2. If

$\Delta (a_1, b_1; a_1, b_1; a_2, b_2) < 0$,

$\Delta (a_2, b_1; a_1, b_1; a_2, b_2) > 0$,

then u, the x-coordinate of the point of intersection, is given by

$u = (w_1a_1 + w_2a_2 + kw_2|b_1 - b_2| + g_1 - g_2) / (w_1 + w_2)$, and $v = b_1$

where k =1 if $b_1 < b_2$ and k = -1 if $b_1 \geq b_2$.

On the other hand if

$\Delta(a_2, b_1; a_1, b_1; a_2, b_2) < 0$,

$\Delta(a_2, b_2; a_1, b_1; a_2, b_2) > 0$,

then v, the y-coordinate of the point of intersection, is given by

$v = (w_1b_1 + w_2b_2 + kw_2|a_1 - a_2| + g_1 - g_2) / (w_1 + w_2)$, and $u = a_2$

where k =1 if $a_1 < a_2$ and k = -1 if $a_1 \geq a_2$.

We now consider the following problem (see, [25]) to explain our algorithm.

Example 1. The four points with coordinates, associated weights and response parameters are:

(3, 3, 2, 1), (3, 6, 3, 0), (6, 3, 4, 0) and (7, 8, 2, 0).

In each pair of parentheses the first pair of numbers represents the coordinates of the demand points while the third and fourth numbers represent a weight and response parameter of the demand point, respectively.

In Initial Step we take the demand points (3, 3) and (3, 6). The corresponding coordinates of the points P and Q are (3, 4.6) and (3, 4.6). Following Steps 1 through 3 the demand points for the next iterations are (6, 3) and (3, 3) and the coordinates of P and Q are (4.835, 3) and (4.835, 3) respectively. Following Steps 1 through 3 the demand points for the next iterations are (3, 6) and (6, 3). Using Steps 1, 2 and 4 it follows that any point of the line segment joining the points (5.14, 4.71) and (5.54, 5.11) may be a facility point.

In the next section we discuss the computational aspect of the algorithm for small as well as large set of data points and compare the performance of the present algorithm with that of the simplex method.

2.3 Computational Experience

The PASCAL code of the present algorithm has been developed and programs run on a PC 486 DX2 66 MHz. As actual data is not readily available, we found it convenient to work with n data points generated by standard Turbo PASCAL procedure **Randomize** and function **Random**. Four sets of n real numbers each were generated- two for the coordinates of the demand points, one for the associated weight and one for the response parameter.

It is clear from the simplex formulation of the present problem, mentioned in Section 1, that the dual simplex method [30] is most suitable for the present problem. This method requires only three vectors for nonbasic variables and one vector for the right-hand side of the constraint.

For n demand points each vector requires 4n components to be stored. In addition to this we must have information about the index of the basic variables. This can be achieved by using a vector having 4n integer components.

From the above discussion it follows immediately that the present algorithm is capable of solving problems having data points approximately five times larger than the problem that can be solved by the dual simplex method. We can use T-transfomation (see Francis and White [25]) for the details) to solve the generalized rectilinear minimax location problem. This method is very easy to implement in a computer. The present algorithm and T-transformation can solve problems having large number of data points. But T-transfomation method is not efficient from the computational point of view. We have developed Turbo PASCAL codes of the present algorithm, the dual simplex method and T-transformation. We have considered five sets containing 100, 200, 300, 400 and 500 data points distributed at random in a two dimensional Euclidean plane. Each of the above sets was randomly generated twenty five times. The CPU times of the algorithms are given in Table 1, where the first column represents the number of data points and the second through fourth represent the average CPU times in seconds of the present algorithm, the simplex method and T-transformation.

Table 1: Average CPU times for different algorithms.

1	2	3	4
100	0.04	0.08	0.55
200	0.10	0.17	2.25
300	0.13	0.25	4.95
400	0.16	0.32	8.90
500	0.20	0.43	13.90

Table 2: Average CPU time in seconds for convergence

Points	1000	1250	1500	1750	2250	2000	2500
Average time	0.44	0.57	0.73	0.91	1.25	1.08	1.32

By keeping n fixed at 1000 the program was executed 25 times. The CPU time and the number of iterations required by the present algorithm to converge were recorded each time. The

number n was next incremented by 250 and this process was continued until n attained the value 2500. The results have been summarized in Table 2. In the second row of the Table 2, the average time is expressed in secs.

2.4 Conclusion

In this paper we have studied an alternative algorithm to solve a weighted single facility rectilinear minimax location problem when a small response parameter is added to the distance function. The algorithm in the worst case is O (n^2) complex. The present algorithm is based on the concept of equipolygon-which is the locus of a point such that the generalized weighted rectilinear distance of it from two demand points are equal- and the points of intersection, P and Q, of the equipolygon with the boundary of the smallest rectangle, through the two demand points, whose sides are parallel to the coordinate axes. We denote the generalized weighted rectilinear distance from either demand point by d. Whenever we get a demand point whose generalized weighted rectilinear distance from either P or Q is greater than d, we update the position of P or Q. If for a demand point the generalized weighted rectilinear distance from both P and Q are greater than d then update the demand points. This process is continued until we obtain the optimal solution. In each iteration the objective value either remains the same or strictly increases from one iteration to another. It is to be noted that whenever we update a pair of demand points the objective value strictly increases.

For n given demand points the algorithm requires only four vectors each having n components. But in simplex method we must know five vectors each having 4n components. Consequently, this algorithm can solve problems approximately five times larger than can be solved by the simplex method. When both the algorithms can be used, the present method requires less computer CPU time. It is worth mentioning that the idea of the present algorithm can be used to solve the weighted rectilinear minimax location problem when weight function depends on the direction (see, [6] and [10]).

2.5 Pascal Code of the Rectilinear Minimax Location Problem

In this section we have included the Pascal program of the algorithm given in section 2.2.

```
program L1_minimax(input, output, infile);
{This programm uses concept of dual feasibility and properties of the spherical triangle to solve
a rectilinear minimax location problem}
uses dos,crt;
```

```
const n=2500;
type
 list=array [1..2500] of real;
var
 infile:text;
 x,y,w,g:list; {These four vectors supply information regarding demand points}
 i,i1,k,k1,k2,l,l1,m,m1:integer;
 u1,u2,v1,v2,x1,y1,d0,dd:real;
 flag:boolean;
 hh,mm,ss,hs:word;
procedure distance(var d:real;a,b:real;jj:integer);
 { This procedure finds generalized weighted rectilinear distance of a demand point from a
given point }
 begin
 d:=w[jj]*(abs(a-x[jj])+abs(b-y[jj]))+g[jj];
 end; {end of the procedure distance}
procedure difference(var p:real; a1,b1:real; j1,j2:integer);
  {This procedure obtains difference of rectilinear distances of two different demand points
from a given point}
 var
  d1,d2:real;
  begin
   distance(d1,a1,b1,j1);
   distance(d2,a1,b1,j2);
   p:=d1-d2
  end; {end of difference}
procedure x_coordinate(var uu:real; j1,j2:integer);
{this procedure obtains x-coordinate of the point of intersection of the equipolygon,
corresponding to two demand points, and minimum rectangle defined by these demand points}
 var
  a1:integer;
```

```
 begin
  if x[j1]<x[j2] then a1:=1 else a1:=-1;
  uu:=w[j1]*x[j1]+w[j2]*x[j2]+a1*(w[j2]*abs(y[j1]-y[j2])+g[j2]-g[j1]);
  uu:=uu/(w[j1]+w[j2])
 end; {end of the procedure x_coordinate}
procedure y_coordinate(var vv:real; j1,j2:integer);
{this procedure finds y-coordinate of the point of intersection of the equipolygon
corresponding to two demand points, and minimum rectangle defined by these demand points}
 var
  a1:integer;
 begin
  if y[j1]<y[j2] then a1:=1 else a1:=-1;
  vv:=w[j1]*y[j1]+w[j2]*y[j2]+a1*(g[j2]-g[j1]-w[j1]*abs(x[j1]-x[j2]));
  vv:=vv/(w[j1]+w[j2])
 end; {end of the procedure y_coordinate}
procedure coordinate(i1,i2:integer);
{this procedure finds coordinates of the point of intersection of the equipolygon,
corresponding to two demand points, and minimum rectangle defined by these two demand
points }
 var
  q1,q2:real;
 begin
  difference(q1,x[i2],y[i2],i1,i2);
  difference(q2,x[i1],y[i2],i1,i2);
  if q1*q2<0 then
   begin
    x_coordinate(u2,i2,i1);v2:=y[i2]
   end
  else
   begin
    y_coordinate(v2,i2,i1);u2:=x[i1]
   end;
```

```
 difference(q1,x[i1],y[i1],i1,i2);
 difference(q2,x[i2],y[i1],i1,i2);
 if q1*q2<0 then
  begin
   x_coordinate(u1,i1,i2);v1:=y[i1]
  end
 else
  begin
   y_coordinate(v1,i1,i2);u1:=x[i2]
  end
end; {end of the procedure coordinate}
procedure next(var u:real;i1,i2:integer);
 {This procedure determines the rectilinear distance of a demand point from the point of intersection of the equipolygon defined by this  point and another demand point and the smallest rectangle defined by  these two demand points}
 var
  q1,q2:real;
 begin
  difference(q1,x[i1],y[i1],i1,i2);
  difference(q2,x[i2],y[i1],i1,i2);
  if q1*q2<0 then
   begin
    x_coordinate(u,i1,i2);u:=w[i1]*abs(u-x[i1])+g[i1]
   end
  else
   begin
    y_coordinate(u,i1,i2);u:=w[i2]*abs(u-y[i2])+g[i2]
   end
 end; {end of the procedure next}
procedure l_m(var ll,mm:integer;a,b,a1,b1:real;ii:integer);
 begin
  ll:=0;mm:=0;
```

```
 if x[ii]>(a+a1)/2 then ll:=-1
 else ll:=1;
 if y[ii]>(b+b1)/2 then mm:=-1
 else mm:=1
 end; {end of the procedure l_m}
procedure update;
{This procedure finds the point of intersection of the equipolygons defined by the pair of points i & k1 and k1 & k2}
 var
 p,r,s:real;
procedure convex (varr1,z:real;c,a,b,a1,b1:real);{This sub-procedure determines the coordinates of the point dividing the line segment joining two given points in a given ratio}
 begin
 r1:=(c-a)/(b-a);
 z:=r1*(b1-a1)+a1;
 end; {end of sub-procedure convex}
 begin  {action block of update}
 r:=-1;s:=-1;
 if ((u1<x[i]) and (x[i]<u2)) or ((u2<x[i]) and (x[i]<u1)) then
  convex(r,y1,x[i],u1,u2,v1,v2);
 if ((v1<y[i]) and (y[i]<v2)) or ((v2<y[i]) and (y[i]<v1)) then
  convex(s,x1,y[i],v1,v2,u1,u2);
 if (r>-1) and (s>-1) then
  begin
  if r<s then
   begin
    difference(p,x[i],y1,i,k1);
    if p<0 then
     if (i1=-1) then l_m(l1,m1,x1,y[i],u2,v2,i)
     else l_m(l1,m1,x[i],y1,u1,v1,i)
    else {p>0}
     if (i1=-1) then l_m(l1,m1,x[i],y1,u1,v1,i)
```

```
    else l_m(l1,m1,x1,y[i],u2,v2,i)
   end {r<s}
  else {r>s}
   begin
    difference(p,x1,y[i],i,k1);
    if p<0 then
     if (i1=-1) then l_m(l1,m1,x[i],y1,u2,v2,i)
     else l_m(l1,m1,x1,y[i],u1,v1,i)
    else {p1>0}
     if (i1=-1) then l_m(l1,m1,x1,y[i],u1,v1,i)
     else l_m(l1,m1,x[i],y1,u2,v2,i)
   end
 end {end of r>-1 and s>-1}
else if r>-1 then
 begin
  difference(p,x[i],y1,i,k1);
  if p<0 then
   if (i1=-1) then l_m(l1,m1,x[i],y1,u2,v2,i)
   else l_m(l1,m1,x[i],y1,u1,v1,i)
  else {p>0}
   if (i1=-1) then l_m(l1,m1,x[i],y1,u1,v1,i)
   else l_m(l1,m1,x[i],y1,u2,v2,i)
 end
else if s>-1 then
 begin
  difference(p,x1,y[i],i,k1);
  if p<0 then
   if (i1=-1) then l_m(l1,m1,x1,y[i],u2,v2,i)
   else l_m(l1,m1,x1,y[i],u1,v1,i)
  else {p>0}
   if (i1=-1) then l_m(l1,m1,x1,y[i],u1,v1,i)
   else l_m(l1,m1,x[i],y1,u2,v2,i)
```

```
   end
  else l_m(l1,m1,u1,v1,u2,v2,i);
 end; {end of procedure update}
procedure test;
 var
  count:boolean;
  a1,b1,p1,p2:real;
procedure selection; {this procedure selects the next two demand points}
 var
  j:integer;
  begin
   j:=1;
   while count and (j<=n) do
    begin
     if (j<>i) then
      begin
       next(dd,i,j);
       if dd>d0 then
        begin
         k:=j;count:=false
        end
      end;
     j:=j+1
    end {end of while}
 end; {end of sub-procedure selection}
procedure newx_y(var a,b:real);
{This sub-procedure obtains the new values of (u1,v1) or (u2,v2)}
 var
  c:real;
   begin
    c:=w[i]*(l1*(u1-x[i])+m1*(v1-y[i]))-w[k1]*(l*(u1-x[k1])+m*(v1-y[k1]));
    c:=c+g[i]-g[k1];
```

```
    c:=c/(w[k1]*(l*(u2-u1)+m*(v2-v1))-w[i]*(l1*(u2-u1)+m1*(v2-v1)));
    a:=c*(u2-u1)+u1;
    b:=c*(v2-v1)+v1
   end; {end of newx_y}
  begin {action block of test}
  i:=1;count:=true;
  coordinate(k1,k2);
  distance(d0,u1,v1,k1);
  while (count) and (i<=n) do
   begin
    if (i<>k1) and (i<>k2) then
     begin
      difference(p1,u1,v1,i,k1);
     { p1>0 implies distance of the point i from (u1,v1) is greater than that of the point k1}
      difference(p2,u2,v2,i,k1);
      { p2>0 implies distance of the point i from (u2,v2) is greater than that of the point k1}
      if (p1>0) and (p2>0) then
          {This condition implies that we have to find two new demand points for the next
iteration}
       begin
        selection;k1:=i;k2:=k;d0:=dd
       end
      else if (p1>0) and (p2<0) then {This condition implies that we have to upgrade (u1,v1)}
       begin
        i1:=1;update;l_m(l,m,u1,v1,u2,v2,k1);
        newx_y(a1,b1);u1:=a1;v1:=b1
       end
      else if (p1<0) and (p2>0) then  {This condition implies that we have to upgrade (u2,v2)}
       begin
        i1:=-1; update;l_m(l,m,u1,v1,u2,v2,k1);
        newx_y(a1,b1);u2:=a1;v2:=b1
       end
```

```
    end;{i<>k1 and i<>k2}
   i:=i+1
  end;{end of while}
 if (count=true) then flag:=false
end; {end of the procedure test}
begin {main action block}
 clrscr;
 assign(infile,'file.dat');
 reset(infile);
 for i:=1 to n do
  readln(infile,x[i],y[i],w[i],g[i]);
 gettime(hh,mm,ss,hs);
 writeln(hh,':',mm,':',ss,'.',hs);
 flag:=true;k1:=1;k2:=2;d0:=0;
 while flag do
  test;
 writeln('k1, k2 ',k1,' ',k2);
 write('Stretch extends from  (',u2:2:2,' , ',v2:2:2,' )  to  ');
 writeln('( ',u1:2:2,', ',v1:2:2, ')');
 gettime(hh,mm,ss,hs);
 writeln(hh,':',mm,':',ss,'.',hs);
 close(infile)
end.
```

2.6 Pascal Program of the Dual Simplex Method for Rectilinear Minimax

In this section we develop the Pascal program of the rectilinear minimax problem given in section 1.1 of this chapter.

```
program linearmax(input,output,infile); {this program obtains the optimum solution of the
rectilinear minimax location by Dual Simplex method }
 uses dos,crt;
 type list=array [1..2000] of real;
    list1=array[1..500] of real;
```

```
    list2=array[1..2010] of integer;
    list3=array[1..3] of integer;
    list4=array[1..3] of real;
    mat=array [1..2000,1..3] of real;
var
  infile:text;
  x,y,w,g:list1; {these four vectors supply the information about the demand points}
  b:list;
  c:list4;
  ba:list2;
  nb:list3;
  a:mat;{from this matrix we get the information about the nonbasic variables}
  i,j,k,i1,j1,n,m:integer;
  a1,b1,u,x1,x2,x3,e:real;
  flag:boolean;
  hh,mm,ss,hs:word;
procedure index; {this procedure initializes the index of the basic variable}
 begin
  for i:= 1 to m do
   ba[i]:=i+3
 end; {end of the procedure index}
procedure obj; {this procedure initializes the objective row}
 begin
  c[1]:=0;c[2]:=0;c[3]:=-1
 end;{end of the procedure obj}
procedure inirhs; { this procedure finds the initial value of the R.H.S}
 var d1:real;
 begin
  for i:= 1 to n do
    begin
      k:=4*(i-1)+1;d1:=g[i]/w[i];
      b[k]:=x[i]+y[i]-d1;b[k+1]:=x[i]-y[i]-d1;
```

```
      b[k+2]:=-x[i]+y[i]-d1;b[k+3]:=-x[i]-y[i]-d1;
     end
   end;{ end of the procedure inirhs}
 procedure inimat; {this procedure initializes the nonbasic constrained matrix}
  var
   d1:real;
  begin
   for i:=1 to n do
    begin
     j:=4*(i-1)+1;
     d1:=-1/w[i];
     a[j,1]:=1;a[j+1,1]:=1;a[j+2,1]:=-1;a[j+3,1]:=-1;
     a[j,2]:=1;a[j+1,2]:=-1;a[j+2,2]:=1;a[j+3,2]:=-1;
     a[j,3]:=d1;a[j+1,3]:=d1;a[j+2,3]:=d1;a[j+3,3]:=d1;
    end;
  end;{end of the procedure matrix}
 procedure rhs; {this procedure determines which row will be the next pivoting row}
  var
   mx:real;
  begin
   mx:=0;i1:=0; {i1 determines a pivot row}
   for i:=1 to m do
    if b[i]<mx then
      begin
       mx:=b[i];i1:=i;
      end;
   if mx=0 then flag:=false {flag=false implies optimal solution of the problem}
  end;{end of rhs}
  procedure nextbasic; {this procedure obtains a basic index for the next entering basic
variable}
  var
   d1,d2:real;
```

```
count:boolean;
begin
 count:=true;j:=1;d1:=2100.0;j1:=0;  {j1 determines the next entering variable}
 while (j<=3) and count do
  begin
   if a[i1,j]<0 then
    begin
     if c[j]=0 then
      begin
       count:=false;j1:=j
      end
     else
      begin
       d2:=c[j]/a[i1,j];
       if d2<d1 then
        begin
         d1:=d2;j1:=j
        end
      end
    end;
   j:=j+1
  end;{end of while}
end;{end of procedure nextbasic}
procedure update; {this procedure updates matrix, objective row and R.H.S of the
constraints}
 var
  d1,d2:real;
 begin
  d1:=1/a[i1,j1];
  b[i1]:=b[i1]*d1;       {updates  an i1th element of the R.H.S}
  u:=u-b[i1]*c[j1];      {updates objective value}
  for j:=1 to 3 do       {updates an i1th nonbasic row of the matrix}
```

```
    a[i1,j]:=a[i1,j]*d1;
   for i:=1 to m do        {updates R.H.S}
    if (i<>i1) then
     b[i]:=b[i]-b[i1]*a[i,j1];
   for i:=1 to m do        {updates the nonbasic coefficient matrix}
    if (i<>i1) then
    begin
     d2:=a[i,j1];
     for j:=1 to 3 do
      if j<>j1 then
       a[i,j]:=a[i,j]-d2*a[i1,j];
    end;
   d2:=c[j1];
   for j:=1 to 3 do     {update an objective row}
    if j<>j1 then
     c[j]:=c[j]-d2*a[i1,j];
   c[j1]:=-d1*d2;
   for i:=1 to m do
    if i<>i1 then
     a[i,j1]:=-a[i,j1]*d1;
   a[i1,j1]:=d1;
   j:=nb[j1];nb[j1]:=ba[i1];ba[i1]:=j;
 end;{end of update}
procedure alter(j:integer);{this procedure obtains alternative optimum}
 var d1,d2:real;
 begin
  d1:=2245.0;k:=0;
  for i:=1 to m do
   if a[i,j]>0 then
    begin
     d2:=b[i]/a[i,j];
     if d2<d1 then
```

```
        begin
          d1:=d2;k:=i
        end
      end;
    b[k]:=b[k]/a[k,j];
    for i:=1 to m do
     if i<>k then
      b[i]:=b[i]-b[k]*a[i,j]
  end;{end of alter}
procedure solution;{this procedure obtains the optimal solution}
 begin
   for i:=1 to m do
    begin
     if ba[i]=1 then x1:=b[i];
     if ba[i]=2 then x2:=b[i];
     if ba[i]=3 then x3:=b[i]
    end
 end;
begin {main action block}
  clrscr;
  assign(infile,'file.dat'); {this file contains coordinates of the demand points}
  reset(infile);
  writeln('supply the value of no. of points');
  readln(n);
  for i:=1 to n do
   readln(infile,x[i],y[i],w[i],g[i]);
  gettime(hh,mm,ss,hs);
  writeln(hh,':',mm,':',ss,'.',hs);
  i1:=0;j1:=0;u:=0;nb[1]:=1;nb[2]:=2;nb[3]:=3;e:=0.0000005;
  flag:=true;
  m:=4*n;
  index;
```

```
    obj;
    inirhs;
    inimat;
    while flag do
     begin
      rhs;
      if flag=true then
       begin
        nextbasic;
        update;
       end
     end; {end of while}
    writeln('value of the objective = ',u:2:2);
    solution;
    writeln('c[1], c[2], c[3] ',c[1],' ',c[2],' ', c[3]);
    write('stretch extends from  (',x1:2:2,' , ',x2:2:2,')');
    if abs(c[2])<e then alter(2)
    else if abs(c[1])<e then alter(1)
    else if abs(c[3])<e then alter(3);
    solution;
    write(' to  (',x1:2:2,' , ',x2:2:2, ')');
    writeln(' ');
    gettime(hh,mm,ss,hs);
    writeln(hh,':',mm,':',ss,'.',hs);
    close(infile)
  end.{end of action block}
```

2.7 Pascal Program of T-transform

In this section we give the Pascal code of the T-transform of the problem given in section 1.1 of this chapter.

```
program t_transform(input,output,infile);
{Francis and White [25] have discussed, in detail, the algorithm of this program}
```

```
uses crt,dos;
const n=200;
type
  list=array[1..2500] of real;
var
  infile:text;
  x,y,w,g:list;
  i,j,p1,p2,q1,q2:integer;
  z,z1,z2,r,r1,r2,s,x1,x2,s1,s2,m1,m2:real;
  flag:boolean;
  hh,mm,ss,hs:word;
procedure max(var i1,i2:integer;var m:real;a1,b1:real);
 var
  a:real;
  begin
   a:=(w[i]*w[j]*abs(a1-b1)+w[i]*g[j]+w[j]*g[i])/(w[i]+w[j]);
   if a>m then
    begin
     m:=a;i1:=i;i2:=j
    end
  end;{end of the procedure max}
procedure stretch1(var n1,n2:real;k:integer);
{this procedure determines the maximum and minimum of two given sets}
 var a,b,a1,b1:real;
  begin
   n1:=-1100.00;n2:=maxint;
   for i:=1 to n do
    begin
     a1:=k*x[i]+y[i];b1:=z-g[i];
     a:=a1-b1/w[i];
     b:=a1+b1/w[i];
     if a>n1 then n1:=a;
```

```
        if b<n2 then n2:=b
      end
  end;{end of procedure stretch1}
begin{begin of main action block}
  clrscr;
  assign(infile,'file.dat');
  reset(infile);
  for i:=1 to n do
   readln(infile,x[i],y[i],w[i],g[i]);
  gettime(hh,mm,ss,hs);
  writeln(hh:2,': ',mm:2,':',ss:2,'.',hs);
  p1:=0;p2:=0;q1:=0;q2:=0;z1:=0;z2:=0;
  for i:=1 to n-1 do
   begin
    r1:=x[i]+y[i];
    s1:=-x[i]+y[i];
    for j:=i+1 to n do
     begin
      r2:=x[j]+y[j];
      max(p1,p2,z1,r1,r2);
      s2:=-x[j]+y[j];
      max(q1,q2,z2,s1,s2);
     end {end of j loop}
   end;{end of i loop}
  if z1>z2 then z:=z1
  else z:=z2;
  r1:=x[p1]+y[p1];r2:=x[p2]+y[p2];
  m1:=w[p1]*r1+w[p2]*r2;
  m2:=g[p1]-g[p2];
  x1:=w[p1]+w[p2];
  if (r1<=r2) then r:=(m1-m2)/x1
  else r:=(m1+m2)/x1;
```

```
s1:=-x[q1]+y[q1];s2:=-x[q2]+y[q2];
m1:=w[q1]*s1+w[q2]*s2;
m2:=g[q1]-g[q2];
x1:=w[q1]+w[q2];
if (s1<=s2) then s:=(m1-m2)/x1
else s:=(m1+m2)/x1;
if z1=z2 then
 begin
  x1:=(r-s)/2;x2:=(r+s)/2;
  writeln('the coordinates of the required unique facility point are:');
  writeln('( ',x1:2:2,', ',x2:2:2,')')
 end
else if (z1>z2) then
 begin
  stretch1(s1,s2,-1);
  x1:=(r-s1)/2;x2:=(r+s1)/2;
  write('the stretch extends from (',x1:2:2,' , ',x2:2:2, ')  to  ');
  x1:=(r-s2)/2;x2:=(r+s2)/2;
  writeln('(',x1:2:2,', ',x2:2:2,')')
 end
else
 begin
  stretch1(r1,r2,1);
  writeln(' math ',r1:2:2,' ',r2:2:2); x1:=(r1-s)/2;x2:=(s+r1)/2;
  write('the stretch extends from (',x1:2:2,' , ',x2:2:2, ')  to ');
  x1:=(r2-s)/2;x2:=(s+r2)/2;
  writeln('(',x1:2:2,', ',x2:2:2,')')
 end;
gettime(hh,mm,ss,hs);
writeln(hh:2,': ',mm:2,':',ss:2,'.',hs);
close(infile)
end.{end of the action block}
```

Solution for Weighted Rectilinear Minimax Problem

3.1 Background

In this section we introduce some basic concepts which will be needed for the solution of the problem. The following lemmas are directly related to the algorithm mentioned in section 3.2.

Lemma 3.1.1. For any two demand points $P_i(a_i, b_i)$ and $P_j(a_j, b_j)$ the following results hold.

(a) If $w_i > w_j$ and $\Delta(a_i, b_i; a_i, b_i; a_j, b_j) > 0$ then there exists no point P(x, y) in the x-y plane such that $rd(P, P_i) = rd(P, P_j)$.

(b) If $w_i > w_j$ and $\Delta(a_i, b_i; a_i, b_i; a_j, b_j) < 0$ then the locus of the point (x, y) satisfying

$$\Delta(x, y; a_i, b_i; a_j, b_j) = 0$$

will be a closed polygon.

Proof. $\Delta(a_i, b_i; a_i, b_i; a_j, b_j) > 0$ implies that

$$g_i > rd(P_i, P_j) \tag{1}$$

The inequality (1), the condition $w_i > w_j$ and triangle inequality imply that

$$\begin{aligned} rd(P, P_j) &\le w_j(|x - a_i| + |y - b_i|) + rd(P_i, P_j) \\ &< g_i + w_j(|x - a_i| + |y - b_i|) \\ &< rd(P, P_i) \end{aligned}$$

This establishes the result given in (a).

Again $w_i > w_j$ and definition 1.2.3 imply

$$\lim_{|x| \to \infty} \Delta(x, b_i; a_i, b_i; a_j, b_j) \to \infty \tag{2}$$

Since $\Delta(x, b_i; a_i, b_i; a_j, b_j)$ is a linear function of x, it follows from conditions (b) of Lemma 3.1.1 and relation (2) that

$$\Delta(\alpha_1, b_i; a_i, b_i; a_j, b_j) = 0$$

and $$\Delta(\alpha_2, b_i; a_i, b_i; a_j, b_j) = 0$$

where $\alpha_1 < a_i < \alpha_2$. In other words $\Delta(x, b_i; a_i, b_i; a_j, b_j)$ has exactly two different zeros. Similarly it follows that $\Delta(a_i, y; a_i, b_i; a_j, b_j)$ vanishes at β_1 and β_2 where $\beta_1 < b_i < \beta_2$.

Again by virtue of definition 1.2.3 and continuity of $\Delta(x, y; a_i, b_i; a_j, b_j)$ we get the following results:

(i) if $\beta \in (\beta_1, \beta_2)$ then $\Delta(x, \beta; a_i, b_i; a_j, b_j) = 0$ exactly at two values of x lying in (α_1, α_2).

(ii) if $\beta \notin (\beta_1, \beta_2)$ then $\Delta(x, \beta; a_i, b_i; a_j, b_j) \neq 0$ for all x.

(iii) if $\alpha \in (\alpha_1, \alpha_2)$ then $\Delta(\alpha, y; a_i, b_i; a_j, b_j) = 0$ exactly at two values of y lying in (β_1, β_2).

(iv) if $\alpha \notin (\alpha_1, \alpha_2)$ then $\Delta(\alpha, y; a_i, b_i; a_j, b_j) \neq 0$ for all y.

Results (i) through (iv) prove the last part of Lemma 3.1.1. □

Lemma 3.1.2. Let $A(a, b) \in R^2$ be any point such that $rd(A, P_i) > rd(A, P_j)$ and $\Delta(a_i, b_i; a_i, b_i; a_j, b_j) < 0$ for a pair of demand points P_i, P_j. Then there exists a point $P(x, y) \in L(P_i, A)$ such that $rd(P, P_i) = rd(P, P_j)$.

Proof. The proof of Lemma 3.1.2 follows immediately from the continuity of the distance function $\Delta(x, y; a_i, b_i; a_j, b_j)$. □

Consider a point P on an edge of the equipolygon $EP(P_i, P_j)$. A diamond [24] through P with centre at either P_i or P_j is drawn. Let Q be a point on the edge of the equipolygon, in which P lies. If Q is not outside the diamond then the direction from P to Q is the direction of descent at P. In Figure 1, $D_1D_2D_3D_4$ denotes the diamond corresponding to the demand point A, PQ is the direction of descent at P.

We can easily calculate the direction of descent at any point P if we use **P(1).**

We now use these definitions, properties and lemmas to develop our algorithm in section 3.2.

3.2 Solution of the Problem

We first note that the optimal solution of the minimax problem will occur on $\partial R(P_i, P_j)$ or within $R(P_i, P_j)$ for a given pair of demand points P_i, P_j; because any movement from a point on $EP(P_i, P_j)$ perpendicular to and toward the nearest boundary $\partial R(P_i, P_j)$ and outside the rectangle will cause the value of the objective function to decrease.

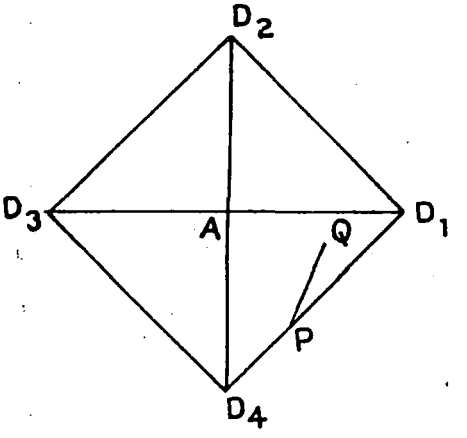

Figure - 1

We take a point P, which is not optimal, on $EP(P_i, P_j)$ such that $rd(P, P_k) \leq rd(P, P_i)$ for all demand points P_k ($k \neq i$ or j). We next continue moving away from this point in the direction of descent until optimality is reached. Otherwise we leave the current $EP(P_i, P_j)$, choose the edge of another equipolygon first of all encountered, being determined by a well-defined selection rule, and repeat the above procedure. It follows from **(P1)**, that corresponding to every nonoptimal point, on an equipolygon, there exists at least one direction of movement such that the objective will monotonically decrease (see, Figures 1 and 2 of section 1.2).

Selection Criterion. Consider the three distinct demand points P_i P_j and P_k and assume that the point of equality $EQ(P_i, P_j, P_k)$ is not optimal. To determine whether the direction of next movement is along the descent direction of $EP(P_j, P_k)$ or $EP(P_i, P_k)$ from $EQ(P_i, P_j, P_k)$, we must know which way the objective value decreases. With this end in view, let us draw lines through $EQ(P_i, P_j, P_k)$ parallel to the coordinate axes. These lines will divide the x-y plane into four quadrants. Since the equipolygons $EP(P_i, P_j)$ and $EP(P_i, P_k)$ intersect at $EQ(P_i, P_j, P_k)$ it follows from **(P1)** that not all three demand points, P_i, P_j and P_k are located in the same quadrant with respect to $EQ(P_i, P_j, P_k)$-rather any two of them are in one quadrant and the remaining one in an adjacent quadrant. From the properties of the equipolygon one has the following selection rule:

SC: *From points belonging to the same quadrant drop the point having a greater weight.*

Using this selection criterion we can state our algorithm as follows:

Algorithm. Rectilinear Minimax Location Problem

Initial Step. Take any point Q, say a vertex of SR, as initial point. Let $A \in S$ be such that

$rd(A, Q) = \max \{ rd(P_i, Q) : i \in I \}$

If $rd(A, Q) \geq rd(P_i, A)$ for all $i \in I$ then A is the required facility point, stop.

Else find a point $P \in L(A, Q)$ such that

$rd(B, P) = rd(A, P)$, $B \in S$ and $ed(Q, P)$ is a minimum; go to Step 1.

Step 1. If $P \in \partial R(A, B)$ then go to Step 4.

Else find the vertex V adjacent to the edge of EP(A, B), which goes through P, situated in the descent direction with respect to P and go to Step 2.

Step 2. If $rd(A, V) \geq rd(P_i, V)$ for all $i \in I$ then $P \leftarrow V$ and repeat Step 1.

Else find a point $C \in S$ such that $rd(A, Q) = rd(C, Q)$ where Q lies on the line segment PV and ed(P, Q) is a minimum. $P \leftarrow Q$ and go to Step 3.

Step 3. If neither $R(A, C) \cup \partial R(A, C)$ nor $R(B, C) \cup \partial R(B, C)$ contains P then use **SC** to find the

two points for the next iteration, denote these points by A, B and go to step 1;

If $P \in R(A, C) \cup \partial R(A, C)$ then $B \leftarrow C$ otherwise $A \leftarrow C$ and go to Step 4.

Step 4. Denote the points of intersection of EP(A, B) and $\partial R(A, B)$ by T_1 and T_2.

If $rd(A, T_1) \geq rd(P_i, T_1)$ for all $i \in I$ then $Q \leftarrow T_1$ and go to Step 6

Else if $rd(A, T_2) \geq rd(P_i, T_2)$ for all $i \in I$ then $Q \leftarrow T_2$ and go to Step 6

Else go to Step 5.

Step 5. Find a point $P_i \in S$ such that $rd(A, Q) = rd(P_i, Q)$ and ed(P, Q) is a minimum where Q lies on either PT_1 or PT_2; go to Step 6.

Step 6. Any point on the line segment PQ is a facility point.

Remarks

1. If the point of equality is nonoptimal then from the selection criteria we conclude that the objective value of the function will strictly decrease in the next iteration. Hence for a nonoptimal point the same edge of a given equipolygon cannot appear in any two iterations. It is also to be noted that the number of sides of the equipolygon is finite. Consequently in the worst case the number of arithmetic operations needed to get the optimal solution is $O(n^2)$.

2. If the number of demand points is n, then for a computational purpose we have to store four vectors each having n components.

We now apply our algorithm to solve the following problem [25] :

Example 1. The four points with coordinates, associated weights and response parameters are given by:

(3, 3, 2, 1), (3, 6, 3, 0), (6, 3, 4, 0) and (7, 8, 2, 0).

Within each pair of parentheses the first pair of numbers represents the coordinates of the demand point while the third and fourth numbers represent the weight and response parameter of the demand points, respectively.

The point $P(7, 3) \in \partial SR$ is taken as an initial approximation. The maximum weighted rectilinear distance occurs for the demand point A(3, 6). The point $Q(24/5, 3) \in L(A, P)$ is equidistant from A and another demand point B(7, 8). Following Step 1, we get the generalized weighted rectilinear distance of updated P(33/7, 36/7) from each of A(3, 6), B(7, 8) and C(6, 3) to be equal. Using **SC** we retain (7, 8) and (6, 3) for the next iteration. By Step1, P now becomes P(155/28, 143/28) whose generalized weighted rectilinear distance from (7, 8), (6, 3) and (3, 3) are equal. Since the point (155/28, 143/28) satisfies optimality condition, by steps 3 and 4 we find

that any point on the line segment joining (36/7, 33/7) and (155/28, 143/28) is a facility point.

In the next section we will compare the performances of the present algorithm with the Simplex Method.

3.3 Computational Experience

The PASCAL code of the present algorithm has been developed and the program run on a PC 486 DX2 66 MHz. As actual data is not readily available, we found it convenient to work with n data points generated by standard Turbo PASCAL (Borland) procedure **Randomize** and function **Random**. Four sets of n real numbers each were generated- two for the coordinates of the demand point, one for the associated weight and one for the response parameter.

It is clear from the simplex formulation of the present problem, mentioned in Section 1.1 that the dual simplex method [30] is most suitable for the present problem. This method requires only three vectors for nonbasic variables and one vector for the right-hand side of each constraint. For n demand points each vector requires 4n components to be stored. In addition to this we must have information about the index of the basic variables. This can be achieved by using a vector having 4n integer components.

From the above discussion it follows immediately that the present algorithm is capable of solving problems having data points approximately five times larger than the problem that can be solved by the dual simplex method. We have developed PASCAL code of the dual simplex method. We have considered five sets containing 100, 200, 300, 400 and 500 data points distributed at random. Each of the above sets was randomly generated twenty five times. The average times of both algorithms are given in Table 1.

Table 1: CPU time for convergence of the present algorithm and the simplex method

Data points	CPU time for present Algorithm (in secs.)	CPU time for Dual Simplex Method (in secs)
100	0.00	0.08
200	0.05	0.16
300	0.10	0.32
400	0.15	0.48
500	0.20	0.56

Keeping n fixed at 1000 the program was executed 25 times. The CPU time and the number of iterations required by the present algorithm to converge were recorded each time. The number n was incremented by 250 and this process was continued until n attained the value 2500. The results have been summarized in Tables 2 and 3. The first column of Table 2 represents the number of iterations whereas the second through the eighth column show the frequency of convergence in twenty-five runs of the program for the same n. Table 3 shows the average and the maximum time in seconds for the data in Table 2

Table 2: The number of iterations required to converge

No. of Points	1000	1250	1500	1750	2000	2250	2500
1	15	16	14	18	16	12	14
2	9	9	10	7	8	12	9
3	1	0	1	0	1	1	2

In Table 3 the first row represents the number of data points. The second row denotes the average CPU time in seconds to converge and the third row denotes the maximum CPU time in seconds corresponding to these data points.

Table 3: Maximum and average CPU time in seconds for convergence

Points	1000	1250	1500	1750	2000	2250	2500
Average time	0.16	0.19	0.23	0.26	0.30	0.35	0.38
Maximum time	0.22	0.27	0.28	0.33	0.44	0.49	0.55

3.4 Conclusion

In this paper we have developed an alternative algorithm to solve a weighted rectilinear minimax facility location problem when a small response parameter is added to the distance function. The algorithm is based on the concept of equipolygon, which is the locus of a point such that the generalized weighted rectilinear distance of it from two given demand points are equal. If we find a point P on the edge of the equipolygon defined by two demand points A and B such that rd (A, P) $\geq$ rd (P, P_i) for all i $\in$ I, then we call this condition primal feasibility. We now move along the edge of this equipolygon such that the primal feasibility is satisfied and the value of rd(A, P) decreases monotonically. During this movement we may either reach the boundary of

R(A, B) or get a point Q on EP(A, B) such that rd (A, Q) = rd (Q, P_i). In the former case we get the optimal solution and in the latter case we use SC to choose two demand points, from among the three, for the next iteration and repeat this process until the optimal solution is attained. For n given demand points the algorithm requires only four vectors each having n components. But in simplex method we must have complete information about five vectors having 4n components each. Consequently, this algorithm can solve problems approximately five times larger than those which can be solved by the simplex method. When both methods can solve a problem, the present method requires less computer CPU time. It is worth mentioning that the idea of the present algorithm can be used to solve the weighted rectilinear minimax location problem when weight function depends on the direction ([6], [10]).

Comparing Table 3 of section 3.3 and Table 2 of section 2.3 we see that the present algorithm is faster than the algorithm given in section 2.2. The reason is - the algorithm given in section 3.2 depends on unidirectional search technique whereas the algorithm given in 2.2 depends on a bidirectional search.

3.5 Pascal Code of the Algorithm

In this section we are going to develop the Pascal code of the algorithm given in section 3.2.

```
program minimax (input, output,infile);
{this program uses the concept of primal feasibility to determine the optimum solution of the
rectilinear minimax location problem}
 uses crt,dos;
 const n=500;
 type list = array[1..2500] of real;
  var
   infile,outfile:text;
   x,y,w,g:list;
{these four vectors give the coordinates, weights and response parameters of the demand points}
   ic,l,m,l1,l2,m1,m2,i,i1,i2,i3:integer;
   e,md,dis,dist,u1,v1,u2,v2,x1,x2,y1,y2,p,f:real;
```

```
 flag :boolean;
 hh,mm,ss,hs:word;
procedure maximum(var xmax,ymin :real);
{this procedure finds the maximum and minimum of two sets}
 begin
 xmax := x[1]; ymin := y[1];
 for i := 2 to n do
  begin
   if xmax <x[i] then xmax := x[i];
   if ymin > y[i] then  ymin:= y[i]
  end;
end;{end of procedure maximum}
procedure distance1(xmax,ymin :real); {this procedure determines the minimum weighted
rectilinear distance of a set from a given point}
 var maxd, d : real;
 begin
  maxd:=-maxint;
  for i :=1 to n do
   begin
    d := w[i]*(( xmax -x[i])+(y[i] - ymin)) + g[i];
    if maxd < d then
     begin
      maxd := d; i1 := i;
     end
   end
 end;{end of the procedure distance1}
 procedure differ(var f11:real;t1,t:integer;a,b:real);
{this procedure finds the difference of generalized distances of a point from two given points}
 var f1,f2: real;
 begin
   f1:= w[t1]*(abs(x[t1]-a)+abs(y[t1]-b)) + g[t1];
```

```
    f2:= w[t]*(abs(x[t]-a)+abs(y[t]-b)) + g[t];
    f11:=f1-f2;
 end;{end of the procedure differ}
procedure value (var u11 :real;ymin:real);
 var k: real;
 begin
  k:= (w[i1]*(x[i1] - y[i1])) - (w[i]*(x[i]-y[i])) + g[i]-g[i1];
  u11:=  (k/(w[i1]-w[i])) +ymin
 end ;{end of the procedure value}
procedure product(var k:real;a1,a2,b1,b2:real);
 var k1,k2:real;
 begin
  differ(k1,i1,i2,a1,a2);
  differ(k2,i1,i2,b1,b2);
  k:=k1*k2
 end;{ end of the procedure product}
procedure maxmin(var xmax, xmin:real; x1,x2:real);
 begin
  if x1<x2 then
   begin
    xmax:= x2; xmin:= x1
   end
  else
   begin
    xmax:= x1; xmin:= x2
   end;
 end;{end of the procedure maxmin}
procedure findlm(var ll,mm: integer; a,b,x,y:real);
 begin
  ll:=0; mm:=0;
  if x>a then ll:=1;
  if x<a then ll:= -1;
```

```
  if y>b then mm:=1;
  if y<b then mm:=-1;
end;{end of the procedure findlm}
procedure lmsum;
begin
 l:=l1+l2; m:=m1+m2
end;
procedure xx(var uu:real; vy: real);
 var n1,n2:real;
 begin
  n1:=w[i2]*m2*(vy-y[i2])-w[i1]*m1*(vy - y[i1])+g[i2]-g[i1];
  n2:= l1*w[i1]*x[i1]-l2*w[i2]*x[i2];
  uu:= (n1+n2) /(w[i1]*l1-w[i2]*l2)
 end; { end of the procedure xx}
procedure yy(var vv :real;ux:real);
 var n1,n2:real;
 begin
  n1:= w[i2]*l2*(ux-x[i2])-w[i1]*l1*(ux-x[i1]);
  n2:= m1*w[i1]*y[i1]-m2*w[i2]*y[i2]+g[i2]-g[i1];
  vv:=(n1+n2)/(w[i1]*m1-w[i2]*m2)
 end;{ end of the procedure yy}
procedure finduv; {this procedure determines the point (u, v) for the next iteration}
var k,uu,vv,u,v:real;
begin
 maxmin(x2,x1,x[i1],x[i2]);
 maxmin(y2,y1,y[i1],y[i2]);
 findlm(l1,m1,x[i1],y[i1],u1,v1);
 findlm(l2,m2,x[i2],y[i2],u1,v1);
 lmsum;
 uu:=maxint; vv:=maxint;
  if (l=2) and (m=-2) then  {zone 7}
   begin
```

```
 product(k,x1,y1,x2,y1);
 if k<0 then uu:=x2 else vv:=y1
end;
if (l=0) then  {zone 4 & 6}
 begin
 if (m=-2) then  {zone-4}
  begin
   product(k,x1,y1,x2,y1);
   if k<0 then vv := y1
   else
    if w[i1] > w[i2] then uu := x[i2]
    else uu := x[i1]
  end
 else  {zone-6}
  begin
   product(k,x1,y2,x2,y2);
   if k<0 then vv := y2
   else
    if w[i1] > w[i2] then uu := x[i2]
    else uu := x[i1]
  end
end; {end of zone 4 & 6}
if (m=0) then  {zone 2 & 8}
begin
 if (l=-2) then  {zone-2}
  begin
   product(k,x1,y1,x1,y2);
   if k<0 then uu := x1
   else
    if w[i1] > w[i2] then vv := y[i2]
    else vv := y[i1]
  end
```

```
  else   {zone-8}
   begin
    product(k,x2,y1,x2,y2);
    if k<0 then uu := x2
    else
     if w[i1] > w[i2] then vv := y[i2]
     else vv := y[i1]
   end
 end; {end of zone 2 & 8}
 if ((m=1) or (m= -1)) then
  begin
   if (l= -2) then uu:= x1
   else
    if (l=2) then uu:=x2;
    product(k,uu,y2,uu,y1);
    if k<0 then
     if (m1=0) then m1:=-m else m2:=-m
    else
     if (m1=0) then m1:=m else m2:=m
  end;
 if (( l=1) or (l= -1)) then
  begin
   if (m=2) then vv:=y2
   else
    if (m=-2) then vv:= y1;
    product( k,x1,vv,x2,vv);
    if k<0 then
     if (l1=0) then l1:=-l else l2:=-l
    else
     if (l1=0) then l1:=l else l2:=l
   end;
if uu = maxint then xx( uu,vv) else yy(vv,uu);
```

```
 u2:=uu; v2:=vv
end;{end of finduv}
procedure test(a1,b1,a2,b2:real);
 var xm,ym,p1,p2,p3,p4:real;
 begin
  xm:= (a1+a2)/2;  ym:= (b1+b2)/2;
  findlm(l2,m2,x[i],y[i],xm,ym);
  p1:=w[i1]*(l1*(x[i1]-u1)+m1*(y[i1]-v1))-g[i1];
  p2:=w[i]*(l2*(x[i]-u1)+m2*(y[i]-v1))-g[i];
  p4:=(w[i1]*(l1*(u2-u1)+m1*(v2-v1))-w[i]*(l2*(u2-u1)+m2*(v2-v1)));
  p3:=(p1-p2)/p4;
  if (p3>0) then
   begin
    p:=p3; i3:=i
   end
 end;{ end of the procedure test}
procedure convex;  {this procedure obtains the point of intersection of two equipolygons}
 var xt,yt,r,q:real;
begin
 p:=maxint;i3:=maxint;
 findlm(l1,m1,x[i1],y[i1],u1,v1);
 for i:= 1 to n do
  begin
   if((i<>i1) and (i<>i2)) then
    begin
     differ(f,i1,i,u2,v2);
     if f<0 then
      begin
       if u2=u1 then r:=maxint else r:= (x[i] - u1)/(u2-u1);
       if v2=v1 then q:=maxint else q:= (y[i]-v1)/(v2-v1);
       xt:=q*(u2-u1)+u1; yt:=r*(v2-v1)+v1;
       if(( r>=0) and (r<=1)) then
```

```
begin
 if ((q>=0)and(q<=1)) then
  begin
   if r>q then
    begin
     differ(f,i1,i,xt,y[i]);
     if f<0 then test(u1,v1,xt,y[i])
     else
      begin
       differ(f,i1,i,x[i],yt);
       if f<0 then test(xt,y[i],x[i],yt)
       else
        begin
         differ(f,i1,i,u2,v2);
         if f<0 then test(x[i],yt,u2,v2)
        end
      end
    end  { r>q }
  else {q>r}
   begin
    differ(f,i1,i,x[i],yt);
    if f<0 then test(u1,v1,x[i],yt)
    else
     begin
      differ(f,i1,i,xt,y[i]);
      if f<0 then test(x[i],yt,xt,y[i])
      else
       begin
        differ(f,i1,i,u2,v2);
        if f<0 then test(xt,y[i],u2,v2)
       end
     end
```

```
    end  {end of else q>r}
   end {q lies between 0 and 1}
  else {q does not lie in 0 and 1}
   begin
    differ(f,i1,i,x[i],yt);
    if f<0 then test(u1,v1,x[i],yt)
    else
     begin
      differ(f,i1,i,u2,v2);
      if f<0 then test(x[i],yt,u2,v2)
     end
   end {q does not lies between 0 and 1}
 end {r lies between 0 and 1}
else {r does not lies between 0 and 1}
 begin
  if ((q>=0)and(q<=1)) then
   begin
    differ(f,i1,i,xt,y[i]);
     if f<0 then test(u1,v1,xt,y[i])
     else
      begin
       differ(f,i1,i,u2,v2);
       if f<0 then test(xt,y[i],u2,v2)
      end
   end
  else
   begin
    differ(f,i1,i,u2,v2);
    if f<0 then test(u1,v1,u2,v2)
   end
 end; {q and r both  does not lie}
u2:=u1*(1-p)+u2*p; v2:= v1*(1-p)+v2*p;
```

```
      end {f<0}
     end  {i<>i1 and i<>i1}
   end ;{for loop}
    u1:=u2;v1:=v2
  end; {end of the procedure convex}
 procedure initial(var  ymin: real);
  var xmax:real;
   begin
    maximum(xmax, ymin);
    distance1(xmax, ymin);
    i2:=0; u1:= xmax; v1:=ymin; u2:=x[i1];v2:=ymin;
    convex;
    if i3=maxint then
     begin
      u1:= x[i1]; u2:= x[i1]; v2:=y[i1];
      convex;
      v1:=v2;
     end
    else u1:=u2;
        i2:= i3;
  end;{ end of the procedure initial}
 procedure selection; {this procedure selects the two points for the next iteration}
  begin
   findlm(l1,m1,x[i1],y[i1],u1,v1);
   findlm(l2,m2,x[i2],y[i2],u1,v1);
   if (l1=l2) and (m1=m2) then
    if (w[i1]<w[i2]) then
     i2:=i3 else i1:=i3
   else
    begin
     findlm(l2,m2,x[i3],y[i3],u1,v1);
     if (l1=l2) and (m1=m2) then i1:=i3 else i2:=i3;
```

```
  end
 end;{ end of the procedure selection}
procedure rectangle(x1,x2,y1,y2,a,b:real);
 begin
  findlm(l1,m1,x1,y1,a,b);
  findlm(l2,m2,x2,y2,a,b);
  lmsum;
  if ((l=0) and (m=0)) then flag:= false;
 end; {end of the procedure rectangle}
procedure interchange(var c1,c2:integer); {this procedure swaps two numbers}
 var c:integer;
 begin
  c:=c1; c1:=c2; c2:=c
 end; { end of the procedure interchange}
 procedure stretch;{Procedure to find a line segment on which the optimal solutions lie}
 var l,m:integer;
    f1,f2,f3,f4:real;
 begin
  maxmin(x1,x2,x[i1],x[i2]);  maxmin(y1,y2,y[i1],y[i2]);
  findlm(l1,m1,x[i1],y[i1],u1,v1);  findlm(l2,m2,x[i2],y[i2],u1,v1);
  l:=l1+l2;m:=m1+m2;u2:=maxint;
  differ(f1,i1,i2,x1,y1); differ(f2,i1,i2,x2,y1);
  differ(f3,i1,i2,x2,y2); differ(f4,i1,i2,x1,y2);
  if (l=0) and (m=0) then
   begin
    if (f1*f2<0) then
     begin
      v2:=y1; xx(u2,v2);
      differ(f,i1,i3,u2,v2);
       if f>0 then convex
        else u2 :=maxint
     end;
```

```
if (f2*f3<0) and (u2=maxint) then
 begin
  u2:=x2; yy(v2,u2);
  differ (f,i1,i3,u2,v2);
  if f>0 then convex
  else u2 := maxint
 end;
if (f3*f4<0) and (u2=maxint) then
 begin
  v2:=y2; xx(u2,v2);
  differ(f,i1,i3,u2,v2);
  if f>0 then convex
  else u2:= maxint
 end;
if u2=maxint then
 begin
  u2:=x1; yy(v2,u2)
 end
end  {(h,k) inside the rectangle}
else
begin
 if (m=-1) then
  begin
   if (f2*f3<0) then
    u2:=x2
   else
    if (f3*f4<0) then v2:=y2 else u2:=x1
  end
 else
  if (l=1) then
   begin
    if (f1*f2<0) then v2:=y1
```

```
      else
       if (f3*f4<0) then v2:=y2 else if (f1*f4<0) then u2:=x1
      end
     else
      if (m=1) then
       begin
       if (f1*f2<0) then v2:=y1
       else
        if (f2*f3<0) then u2:=x2 else u2:=x1
      end
     else
      begin
       if (f1*f2<0) then v2:=y1
       else
        if (f2*f3<0) then u2:=x2 else v2:=y2
      end;
     if u2=maxint then yy(v2,u2) else xx(u2,v2);
     convex
    end
end;  {end of the procedure stretch}
begin  {main action block}
 clrscr;
 assign(infile,'file.dat');
 reset(infile);
 for i := 1 to n do
  readln(infile,x[i], y[i], w[i], g[i]);
 gettime(hh,mm,ss,hs);
 writeln(hh, ':',mm,':',ss,'.',hs);
 i3:=0;ic:=0;i1:=1;i2:=2;
 initial (v1);
 flag:=true;
 rectangle(x[i1],x[i2],y[i1],y[i2],u1,v1);
```

```
if flag=false then i3:=maxint;
while flag do
 begin
 if (i3=maxint) then flag:= false
 else
 begin
 finduv;
 convex;
 if (i3<>maxint) then
 begin
 rectangle(x[i1],x[i3],y[i1],y[i3],u1,v1);
 if flag=false then interchange (i2,i3)
 else
 begin
 rectangle (x[i2],x[i3],y[i2],y[i3],u1,v1);
 if flag=false then interchange(i1,i3)
 else
 selection
 end
 end
 end;
 ic := ic+1;
 end; {end of while}
writeln('no. of iterations = ',ic);
write('the stretch extends from (',u1:5:2,',',v1:5:2,') ');
stretch;
writeln('to (',u1:5:2,',',v1:5:2,')');
gettime(hh,mm,ss,hs);
writeln( hh, ':',mm,':',ss,'.',hs);
close(infile);
end. { end of action block}
```

Minimax Location For An Arbitrary Shaped Constrained Region Using The Rectilinear Norm

4.1. Introduction.

The town of Coach Behar in the northern part of the state of West Bengal has been plagued by housing problems as a result of influx of people in search of living. The population, which was a meagre 34,000 according to the 1941 census report, rose to a little over 70,000 in 1991. The total municipal area covers 3.2 square miles. The economy of the region depends largely on timber and tobacco. The district is strategically important owing to its having a common international boundary with Bangladesh. North Bengal, and the whole of North East for that matter, receive heavy rainfall from May to October making the area perpetually in danger of flooding. During floods, the only communications link between the waterlogged region and the rest of the country lies through Cooch Behar. Recent floods have seen Alipurduar and the North Eastern states cut off from the rest of India, when flood operations had to be routed through Cooch Behar, emphasizing once again the crucial importance of the town owing to its strategic location. Coupled with this there is also economic backwardness of the region which calls for adopting strict security measures. In view of these the Government has over the years stepped up developmental activities. But scarcity of rented dwelling houses hinders these efforts despite substantial spending by the Government on this objective. Although it is an old town, it is well-planned, a fact borneout by nicely laid out road network extending east-west or north-south. The present research was motivated by an actual problem that seeks to meet the growing demand for accommodation. As there is no vacant space left within the municipal limits this problem can be effectively tackled by building a housing complex outside the periphery of the town. Our problem thus reduces to a minimax location problem under the rectilinear norm which consists in minimizing the maximum rectilinear distance of the set of demand points from the complex situated outside a given irregular shaped region.

Although we have devised a solution for acute housing problems in Cooch Behar, this model is applicable to other problems such as locating a hospital for infectious diseases or an explosive factory so as to maintain a safe distance from existing location points, thereby reducing the hazardous effects accompanying such establishments to a minimum.

This article appeared in Asia-Pacific Journal of Operation Research Vol 11, 207-216, 1994.

4.2. Problem Formulation and Basic Concepts

Assume that the set G is defined by

$$G = \{g_i \mid i = 1, 2, 3, ..., n\}$$

where $g_i = (a_i, b_i)$ are the existing demand points in R^2. Also assume that the new facility point is to be located at $T = (x, y)$ in such a way that the maximum rectilinear distance between T and the set G is a minimum, subject to the restriction that T is constrained to lie outside or on the boundary of a region $\Sigma \subset R^2$ of arbitrary shape. The rectilinear distance between T and any point g_i is given by

$$d(T, g_i) = |x - a_i| + |y - b_i|$$

The problem to be considered here is the following:

$$\underset{T \notin \Sigma}{\text{Min}} \quad \underset{1 \le i \le n}{\text{Max}} \quad d(T, g_i)$$

where T may lie on $\partial\Sigma$, the boundary of Σ.

Before working out the algorithm of this problem let us construct a rectangle (see Elzinga and Hearn [23], Francis [24]) $A_1A_2A_3A_4$ enclosing G by drawing lines $f_i(x, y) = 0$, i = 1, 2, 3, 4, where

$$f_1(x, y) = x + y - \min_i\{a_i + b_i\}$$
$$f_2(x, y) = x + y - \max_i\{a_i + b_i\}$$
$$f_3(x, y) = -x + y - \min_i\{-a_i + b_i\}$$
$$f_4(x, y) = -x + y - \max_i\{-a_i + b_i\}$$

The vertices $A_i = (x_i, y_i)$, i = 1, 2, 3, 4 are obtained by solving the following pairs of equations:

$$f_1 = 0 = f_3;\ f_1 = 0 = f_4;\ f_2 = 0 = f_4;\ f_2 = 0 = f_3.$$

Let the perpendicular bisector of the longer sides of the rectangle $A_1A_2A_3A_4$ terminated by the horizontal and vertical lines drawn through the extremities of the sides be denoted by KQ, K having a greater ordinate than Q (see Figure 1).

Before proceeding, let us introduce the concept of a dominating side. For a point P in the xy-plane, a side of $A_1A_2A_3A_4$ is said to be *dominating* if the rectilinear distance of P from any point is greater than that of any other point on the remaining sides. With reference to Figure 1, let P belong to the cone having vertex at Q and A_2Q, A_3Q as extreme directions. For the point P, A_2A_3 will clearly be the dominating side.Let the regions corresponding to the dominating sides

be denoted by L_i, i = 1, 2, 3, 4, where

$L_1 = \{(x, y): x > x_1, y > y_2$ and bounded below by KQ$\}$

$L_2 = \{(x, y): x \le x_1, y > y_4\}$,

$L_3 = \{(x, y): x \le x_3, y \le y_4$ and bounded above by KQ$\}$,

$L_4 = \{(x, y): x > x_3, y \le y_2\}$,

when $f_1 = 0$ represents the smaller side, and

$L_1 = \{(x, y): x \ge x_1, y \ge y_2\}$,

$L_2 = \{(x, y): x \le x_1, y \ge y_4$ and bounded below by KQ$\}$,

$L_3 = \{(x, y): x \le x_3, y \le y_4\}$,

$L_4 = \{(x, y): x \ge x_3, y \le y_2$ and bounded above by KQ$\}$,

when $f_1 = 0$ denotes the longer side.

We next define $R_i = L_i \cap \partial\Sigma$, i = 1, 2, 3, 4. Assume that each R_i consists of piecewise smooth curves $g_{ij}(x, y) = 0$, j = 1, 2,..., k(i). Calculate

$$m_i = \operatorname{Min}_{(x,y) \in R_i} |f_i(x, y)|; \text{ then}$$

$$\text{minimum } \{m_i : i = 1, 2, 3, 4\}$$

will give the required solution.

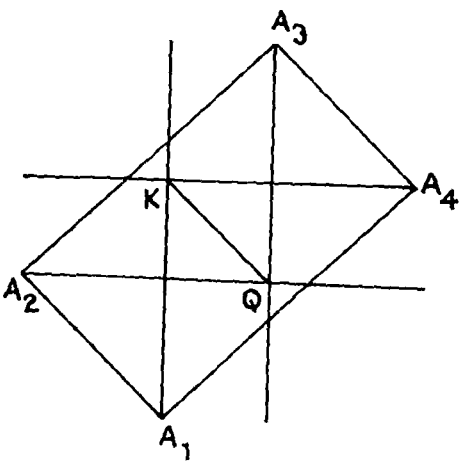

Figure - 1

For two given points $B_1 = (u_1, v_1)$ and $B_2 = (u_2, v_2)$, let us define

$\alpha_1 = \min\{u_1, u_2\}$,

$\alpha_2 = \max\{u_1, u_2\}$,

$\beta_1 = \min\{v_1, v_2\}$,

$\beta_2 = \max\{v_1, v_2\}$.

We next define the following regions:

$S_1 = \{(x, y) | x \le \alpha_1, y \ge \beta_2\}$,

$S_2 = \{(x, y) | x \ge \alpha_2, y \le \beta_1\}$,

$S_3 = \{(x, y) | x \le \alpha_1, y \le \beta_1\}$,

$S_4 = \{(x, y) | x \ge \alpha_2, y \ge \beta_2\}$.

The proof of the following lemma is obvious.

Lemma:

(a) If B_1B_2 *is a straight line segment inclined at an angle of* $45°$ *with the x-axis where*

$B_1 = (u_1, v_1)$ *and* $B_2 = (u_2, v_2)$,

then the rectilinear distance of $P \in S_1 \cup S_2$ *from* B_1B_2 *is a constant.*

If we draw a line segment $l \in S_1 \cup S_2$ *through* P *parallel to* B_1B_2 *then the rectilinear distance of any point to* l *from* B_1B_2 *is the same.*

(b) If, on the other hand, B_1B_2 *make an angle of* $135°$ *with the x-axis, then the above properties will hold for any point* P *lying in the region* $S_3 \cup S_4$.

As a consequence of the above lemma, the rectilinear distance of any point lying on A_3A_4 in Figure 1 from any point of A_2A_1 is a constant.

It may be noted that the optimum cannot occur at a point outside the constrained region. For, if we move from this point towards the boundary of the region in a direction perpendicular to the dominating side corresponding to this point, the objective value continuously diminishes. Thus, our problem is reduced to the search for optimum on the boundary of the constrained region. Now, the rectilinear distance function is continuous and positive everywhere. Therefore, by a well-known theorem due to Weierstrass, there exists a global minimum of the objective function. In the discussion that follows we outline a method to obtain this minimum.

4.3. Algorithm

Step 1. If $KQ \cap \Sigma^c$ then any point $\in KQ \cap \Sigma^c$ is a required solution.

Else $i \leftarrow 0$, $M \leftarrow \infty$, $S_0 = \varnothing$ and go to Step 2

Step 2. $i \leftarrow i + 1$.

If $i > 4$ then go to Step 3.

Else let

$m_i = \min\{|f_i(x, y)| : g_{ij}(x, y) = 0, j = 1, 2, ..., k(i)\}$;

$S = \{(x, y) : m_i = |f_i(x, y)|, g_{ij} = 0, \text{ and } j = 1, 2, ..., k(i)\}$;

If $m_i < M$ then $M \leftarrow m_i$, $S_o \leftarrow S$

Else if $m_i = M$ then $S \leftarrow S_o \cup S$

Repeat Step 2.

Step 3. Optimal distance is M and S_o is the set of optimal solution points.

Remarks.

1. If R_i is convex then it will be sufficient to calculate m_i for just two points which are respectively the points of intersection of each of $g_{i1} = 0$ and $g_{ik(i)} = 0$ with the boundary of L_i.
2. Step 1 of the algorithm discuses the unconstrained version of the present problem.
3. Steps 2 and 3 determine the solution in the constrained case.

4.4. Numerical Examples

To obtain the solution of the problem, either explicit analytic equations representing the boundary or the coordinates of the points lying on the boundary are given. In the latter case, an approximate equation of the boundary of the region can be obtained. The usual procedure is to approximate the given boundary curve by a polynomial. In practical problems, interpolating polynomials are not suitable for use as an approximation (see, Prenter [47]). For example, in order to obtain a good approximation to a function f(x) by an interpolating polynomial of degree n, it may be necessary to use a fairly large value of n. Unfortunately, polynomials of high degree often have a marked oscillatory behaviour which is undesirable in approximating functions, which should be reasonably smooth. Again, computational problems arise when the number of data points is large. For instance, given 100 data points $(x_1, y_1), (x_2, y_2), ..., (x_{100}, y_{100})$ it is difficult

2to find the 99th-degree polynomial P(x) such that $P(x_i) = y_i$, i = 1, 2,...100. Moreover, a high degree polynomial is not suitable for obtaining the solution to our problem. For this reason we approximate the boundary curve by the piecewise Lagrangian polynomial of *m*th degree, where m = 1, 2. This method is very effective and we can easily implement this concept to determine the minimum of the objective function subject to the constraints. In Problem 1, we use a piecewise linear function to approximate the boundary to obtain the optimal solution by applying our algorithm. In Problem 2 we apply a combination of linear and a quadratic approximations to obtain the optimal solution.

Problem 1. Let Σ be a bounded convex region defined by the following set of inequalities:

$$x - y \leq 30, \quad \text{(i)}$$
$$-2x - 3y \leq 90, \quad \text{(ii)}$$
$$x \leq -30, \quad \text{(iii)}$$
$$-x + 4y \leq 150, \quad \text{(iv)}$$
$$7x + 5y \leq 270, \quad \text{(v)}$$
$$11x - 5y \leq 270. \quad \text{(vi)}$$

Suppose that the set G comprises the following points:

$P_1 = (-9, 8)$, $P_2 = (-15, -8)$, $P_3 = (22, 5)$, $P_4 = (17, 20)$,
$P_5 = (10, 0)$, $P_6 = (3, 4)$, $P_7 = (-5, -9)$, $P_8 = (-16, -4)$,
$P_9 = (12, 4)$, $P_{10} = (-10, 17)$, $P_{11} = (1, 14)$, $P_{12} = (-7, 6)$,
$P_{13} = (-14, 3)$, $P_{14} = (12, 24)$, $P_{15} = (-1, 1)$, $P_{16} = (0, -13)$.

Here KQ is given by the line segment joining K= (-3, 10) and Q = (5, 2) and lies wholly within the region Σ. Starting from the point (20, -10), we move along the boundary of the active constraint given by the equation x - y = 30 and by Step 2 of our algorithm we obtain Table 1. The first, second and third columns of Tables 1 and 2 represent active constraint, minimum objective value and coordinates of the point at which minimum occurs, respectively.

Table 1.

(i)	57.00	Any point of the segment joining (20, -10), (5, -25)
(ii)	67.00	(0.00, -30.00)
(iii)	77.00	(-30.00, -10.00)
(iv)	56.75	(-3.00, 36.75)
(v)	65.00	(30.00, 12.00)
(vi)	50.45	(25.45, 2.00)

From Step 3 we conclude that the required facility point is (25.45, 2.00) and the corresponding objective value is 50.45.

Problem 2. Let Σ be a nonconvex region defined by the following set of inequalities:

$$y^2 \geq 4(x - 5y - 30) \quad \text{(i)}$$

$$2x + 3y \leq 60 \quad \text{(ii)}$$

$$-x + 5y \leq 100 \quad \text{(iii)}$$

$$-9x + y \leq 240 \quad \text{(iv)}$$

$$-x - 4y \leq 150 \quad \text{(v)}$$

$$19x - 24y \leq 650 \quad \text{(vi)}$$

Suppose the set G comprises the following points:

$P_1 = (-5, 11)$, $P_2 = (-10, -5)$, $P_3 = (8, -4)$, $P_4 = (5, 5)$,
$P_5 = (0, 0)$, $P_6 = (1, -5)$, $P_7 = (-10, -1)$, $P_8 = (-5, -7)$,
$P_9 = (-5, 0)$, $P_{10} = (-10, 4)$, $P_{11} = (3, 6)$, $P_{12} = (2, -9)$.

Here KQ is given by the line segment joining K = (-1.5, 0.5) and Q = (-3.0, -1.0), which lies wholly within the region Σ. Starting from the point (14, -16) we move along the boundary of the active constraint given by the equation 19x - 24y = 650 and by Step 2 of our algorithm we obtain Table 2:

Table 2.

(i)	30.00	(6.00, -8.00)
(ii)	35.00	(0.00, 20.00)
(iii)	33.20	(-1.50, 19.70)
(iv)	37.78	(-26.78, -1.00)
(v)	55.00	(-10.00, -35.00)
(vi)	42.46	(-3.00, -29.46)

From Step3 we conclude that the required facility point is (6.00, -8.00) and the corresponding objective value is 30.00.

4.5. Summary

The problem we have studied is:

Minimize f(x, y)

where (x, y) lies outside or on the boundary of a given region and

$$f(x, y) = \text{maximum } \{ |x - a_i| + |y - b_i| : i \in I \}.$$

To solve this problem we used the concept of the dominating side. We have enclosed all the demand points by the smallest rectangle whose sides are inclined at an angle of 45° or 135° with the positive direction of the x-axis. By drawing vertical and horizontal lines suitably through the angular points of the above rectangle, the whole space may be divided into four zones, the characteristic property of each of which is that there is a unique dominating side corresponding to every zone. Since the minimum objective value occurs on the boundary, our problem reduces to finding the minimum of the objective on the portion of the boundary lying in a particular region. Since the rectilinear distance function is positive everywhere and continuous, a minimum must exist and be attainable.

The algorithm is divided into two parts. The first deals with the unconstrained case while the second part deals with the constrained case. In the constrained case, we obtain the minimum corresponding to each dominating side; the minimum among all these will be the actual global minimum. If the boundary curve has a complex mathematical form, or if its mathematical specification is not known explicitly, we replace the boundary curve by a piecewise smooth polynomial of degree at most two.

Although the solution procedure here pertains to the case with equal weights, it gives us insight into an analogous problem when the weights associated with the demand points are unequal or where response time has to be considered.