SOLUTIONS OF SOME SINGLE FACILITY LOCATION PROBLEMS

A THESIS

SUBMITTED FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY (MATHEMATICS)

TO THE

UNIVERSITY OF NORTH BENGAL

1997

BY

Pali Das

DURAMARI C. K. HIGH SCHOOL

PURBA DURAMARI

JALPAIGURI, WEST BENGAL

INDIA

ST - VERF

ST - VERF

STOCKTAKING - [illegible]

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Prasanta Kumar Chaudhuri. From the time he introduced me to Operations Research he has been a constant source of inspiration and encouragement. My development as a mathematician is due, in large part, to his teaching and guidance. I consider it an honour and privilege to have had the opportunity to work with him these past few years.

I am indebted to Dr. Nisithranjan Chakrabarti, A. C. College, Jalpaiguri, for his helpful comments and suggestions.

I am pleased to acknowledge the hospitality and computer support of the Mathematics Department, North Bengal University, throughout the course of my work.

It is with great pleasure that I dedicate this thesis to the people for whom I feel the deepest gratitude-my parents.

Pali Das.
Pali Das

Table of Contents

CHAPTER 1

Introduction

1.1 Overview

This dissertation is concerned with the optimization and complexity of a certain class of facility location problems. We investigate ways in which plane analytical coordinate geometry and spherical trigonometry can be used as tools in the solutions of various facility location problems involving geodesic and rectilinear norms.

Owing to its potential application in many practical situations, these problems have been treated extensively in the literature. The distance functions most commonly encountered are euclidean, rectilinear and geodesic distances (see, for example [2], [4], [6], [14], [23], [26], [28], [39], [53], [57] and [58]).

Facility location problems have attracted the attention of the ancient Greeks. Fermat [11] in early seventeenth century formulated a version of the distance location problem as a geometrical problem which may be enunciated as follows:

Let there be given three points in a plane. To find a fourth point such that the sum of the distances from it to the three points is a minimum.

This problem was solved by Torricelli [25] in 1640. A German economist named Alfred Weber [56] generalised this problem by introducing weights in the analysis. The Weber problem consists in locating a warehouse so as to render the total weighted distance between it and the demand points a minimum. Such problems are referred to as the minisum problem in the literature and Harold Kuhn [33] must be credited with trying a purely mathematical solution procedure to solve the problem. Very often instead of minimising the total distance travelled the maximum distance is required to be minimised. Being naturally called the minimax criterion, this is most suitable for locating an emergency facility where maximum delay in rendering a service is more important than average or total delay (see, e.g., [26] and [39]). There is still another criterion which involves maximising the minimum distance. Known as the maximin criterion, this is applicable in case there is an obnoxious or undesirable facility (see, e.g., [6], [12], [19] and [41]), such as a nuclear plant or for that matter any polluting source, and the facility is to be so located that it is as far away as possible from the points it actually serves.

The study and development of methodologies to determine the location of new facilities

is carried out in such a manner that the potential users of the facilities are benefited most. The investigation is conducted by constructing suitable models involving one or more new facilities, and solving them.

The above criteria may be used to solve a single facility or a multifacility location problem (see, e.g., [2], [38], [42] and [49]). The former seeks to locate a facility amongst several demand points whereas the latter concerns locating any given number of variable points representing facilities with respect to any given number of fixed points representing potential users. The m-centre problem which is a generalisation of the 1-centre problem forms the most important class of problems in location analysis and has been extensively studied (see, e.g., [21], [25], [32], [39] and [46]).

Lee [34], and Shamos and Hoey [52] presented algorithms to construct the Voronoi diagram and proved that the smallest circle (in the L_p-metric) enclosing the set could be solved very efficiently.

Spherical location problems typically involve finding the optimal location of a service facility among a number of demand points situated on the surface of a sphere (see, e.g., [1], [2], [14]-[19], [31], [35]-[37], [39], [45], [50] and [58]). Quite a good number of military, civil and commercial logistics and location problems being concerned with globally distributed demand points (see, e.g., [35], [45] and [58]), the planar distance approximation becomes irrelevant and the choice naturally falls on a metric that spans over a sphere.

This research considers the minimax spherical problem that finds the location of a service facility for which the largest distance to a demand point is minimized. The minimax spherical problem differs from its planar counterpart (see, e.g., [19], [45] and [60]) in that its objective function is nonconvex and nondifferentiable. In a special case where all demand points lie on a hemisphere, geometrical algorithms (see, e.g., [5], [9], [23], [26], [29], [40], [43], [44] and [52]) for the two-dimensional minimax problem using the euclidean norm can be applied. In addition, Sarkar and Chaudhuri [50] present an efficient geometrical algorithm based on geodesic distance and Litwhiler's method [35] stereographically projects spherical surfaces bounded by planes on to plane circles so that techniques for two and three-dimensional spaces can be used. Given a set of demand points on a sphere, algorithms in [48] and [55] determine whether the points lie on a hemisphere.

When the demand points do not lie on a hemisphere, Drezner and Wesolowsky [19] proposed a steepest descent technique to solve the minimax problem. Recently, Patel [45] proposed another algorithm based on a factored secant update technique. Both algorithms only produce local optimal solutions. In theory, one can obtain a global optimal solution by generating

all local solutions. However, the approach is difficult, if not impossible, to implement in practice.

As regards the scope of application of the minimax criterion to the weighted rectilinear distance location problem (see, e.g., [20] and [25]) we might consider locating a new facility, say a polyclinic or a fire station in a large metropolitan area where the objective is to minimize rectilinear travel distance of a potential user plus a nonnegative constant, the weight being any positive number quantifying the nature of interaction between the facility and the category of user. On receiving a call for any emergency service, the person at the reception requires some time to pass the information to the appropriate location where the service is to be rendered from. Thus there is a time lag from the instant a demand for being serviced is requisitioned to the instant the particular facility is made available. This explains why a response parameter has been considered in the ongoing analysis.

The time complexity of various types of location problem may be found (see, e.g., [40] and [53]). A method-oriented selective survey of representative problems in location research occurs extensively in the works of [4], [13] and [28].

1.2. Chapter Summaries

In chapter 2 we have presented three different algorithms for spherical minimax location problems.

The algorithm presented in section 2.2 of this chapter solves a global single facility minimax location problem exactly in polynomial time. As far as we know there exists no algorithm which solves a global minimax location problem exactly in polynomial time. The algorithm we have developed is of complexity $O(n^3)$. It solves global as well as hemispherical minimax location problem. It also determines whether all demand points lie on a hemisphere or not. The procedure presented here is based on an enumeration technique and determines global optimal solutions in a finite number of steps. When a local minimum is obtained, one uses this information to obtain the next better solution(s). Thus the possibility of occurrence of inferior solution(s) in subsequent iterations can totally be eliminated. The Pascal Code of the problem has been developed. Using this code we have solved the problem given in [45]. From the solution of the problem it is clear that choosing the minimum from all local minima may sometimes fail. Because it is difficult to generate all local minima although theoretically it is sound. In fact the result given in [45] is not correct.

The algorithm given in section 3.2 of chapter 2 solves a hemispherical minimax location

problem, involving a geodesic norm, exactly in polynomial time. The time complexity of this algorithm is $O(n^2)$. No geometrical algorithm for solving the spherical problem when the demand points are assumed to lie on a hemisphere exists, except the one which relies on primal feasibility [50]. Although the present approach is based on dual feasibility [23] adapted from the planar case [29], the two differ significantly in their implementations. Both select initially a pair of points. But the former proceeds in such a way that if a demand point happens to be within a spherical disc (see [58]) at any iteration it continues to remain in a larger spherical disc at subsequent iterations until optimality is reached. The complexity of the algorithm discussed in [50] is $O(n^2)$, but from a computational point of view it takes much more time to solve a problem than the present method. We have developed the Pascal codes of the present algorithm and the algorithm given in [50] to make a comparison of the running time of the algorithms.

In section 4.2, chapter 2, we have developed an alternative algorithm, based on geometry, for solving a hemispherical minimax location problem. The method yields exact solution having a time complexity $O(n^2)$. It has been shown that the solution of the minimax problem when the norm under consideration is geodesic is equivalent to solving a maximization problem using the euclidean norm. We have proved that a hemispherical minimax problem reduces to finding a small circle of maximum radius on the surface of the sphere which contains either two demand points at the ends of a diameter or three demand points forming an acute triangle such that all demand points lie on one side of the plane of the small circle and the centre of the sphere on the other side. The basic difference between this algorithm and the algorithm given in section 3.2 of this chapter is that while the former depends on the maximization of the euclidean distance the latter uses the properties of spherical triangles. Pascal code of the problem has been developed to compare the performance of the present and the existing algorithms. The algorithm presented in this section is significantly faster than all existing algorithms. From a computational point of view the algorithm given in section 3.2 takes much more time than the present algorithm, despite the former being $O(n^2)$. This is so because the algorithm presented in section 3.2 depends on computation of the trigonometric functions whereas the present algorithm uses only euclidean distances.

In Chapter 3 we have included solutions of three minimax location problems involving a rectilinear norm.

The purpose of sections two and three of this chapter are to deal with the effect of a response parameter (see, e.g., [20] and [26]) when it is added to the weighted rectilinear distance

measured from a new facility point to each of n existing demand points so that the maximum value of the cost function attains its minimum value. The methods of solution are based on the notion of equipolygon, i.e., the locus of points whose generalized weighted rectilinear distances from two given points are equal.

The algorithm developed in section 2.2, of Chapter 3, depends on the concept of dual feasibility. Elzinga and Hearn [23] used this technique to solve an equiweighted euclidean minimax location problem. Hearn and Vijoy [29] extended this approach to the weighted case. The linear programming model (see, e.g., [25], [42] and [59]) of this problem is also possible. The dual simplex method is most appropriate in the present context. To solve this problem in a computer, the present algorithm requires approximately one fifth less computer memory storage than is necessary for the dual simplex method. Consequently the present algorithm can solve problems, approximately five times larger, which can be solved by the dual simplex method. When both methods can solve a problem, the present method requires less computer CPU time. T-transformation (see, Francis and White [25]) and the present algorithm can solve problems of the same size. But from a computational point of view T-transform algorithm is not very efficient.

In section 3.2 of Chapter 3, we have developed an alternative algorithm for the weighted rectilinear minimax location problem. This algorithm uses the concept of primal feasibility. Chakraborti and Chaudhuri [9] applied this idea to solve the equiweighted euclidean minimax location problem. Hearn and Vijoy [29] extended this principle to the weighted case. Although the algorithms given in 3.2 and 2.2 have the same complexity the primal feasible algorithm developed in section 3.2 requires less computer CPU time than the dual feasible algorithm given in section 2.2 . This is so because the former algorithm depends on a unidirectional search whereas the latter on a bidirectional search.

In section 4 we have solved the minimax location problem for an arbitrary shaped constrained region (see, e.g., [3], [5], [7], [8], [22], [27] and [51]) using the rectilinear norm. The method of solution is based on the concept of dominating sides. For a point P in the x-y plane a side of the rectangle ABCD, whose sides are inclined at an angle of 45° and 135° with the positive direction of the x-axis, is said to be dominating if the rectilinear distance of P from any point on this side is greater than that from any point on the remaining sides. In the constrained case, we find the minimum corresponding to each dominating side and the minimum among these will give the global minimum.

CHAPTER 2

Spherical Minimax Location Problems

1.1 Problem Formulation

Without loss of generality assume that the sphere has a unit radius, since the arc length is directly proportional to the radius of the sphere. Let (ϕ_i, θ_i) denote the location on the surface of a unit sphere of the i-th demand point where ϕ_i and θ_i represent its latitude and longitude, respectively, and S denote the set of all demand points. The geodesic distance, α_i, between a facility location (ϕ, θ) and the i-th demand point is given by (see, e.g., [45] and [54]).

$$\cos\alpha_i = \sin\phi\sin\phi_i + \cos\phi\cos\phi_i\cos(\theta - \theta_i) \tag{1}$$

Thus the problem on the surface of the sphere can be stated as

$$\min_{(\phi,\theta)} \max_i \{\alpha_i\} \tag{2}$$

Patel [45] has shown that minimizing the maximum of the spherical arc distances between the facility point and the demand points on a sphere is equivalent to minimizing the maximum of the corresponding Euclidean distances. We use this concept to obtain the optimum solution of the spherical and hemispherical minimax location problems.

Given the latitude and longitude of a demand point one can use the following coordinate transformations to obtain the corresponding cartesian coordinates of the point:

$$x = \cos\phi\cos\theta,\ y = \cos\phi\sin\theta,\ z = \sin\phi, \tag{T}$$

where $0 < \theta < \pi$ for east longitude and $-\pi < \theta < 0$ for west longitude, and $0 < \phi < \pi/2$ for north latitude and $-\pi/2 < \phi < 0$ for south latitude.

After obtaining the optimum solution of the problem one can apply the inverse of the transformation (T) to get the latitude and longitude of the facility point. It follows immediately from the above discussion that the inverse of the transformation T is given by:

(i) The latitude $\phi^\circ = \frac{180^\circ}{\pi}\arctan\left|\frac{z}{\sqrt{x^2+y^2}}\right|$ in magnitude . The latitude is north or south according as $z > 0$ or < 0. If $z = 0$ then the point is situated on the equator.

Assume $x \neq 0$. Let $\theta^\circ = \frac{180^\circ}{\pi}\arctan\left|\frac{y}{x}\right|$

If $x > 0$ and $y \geq 0$ then longitude is θ° east.

If $x > 0$ and $y < 0$ then longitude is θ° west.

If $x < 0$ and $y \geq 0$ then longitude is $90^\circ + \theta^\circ$ east.

If $x < 0$ and $y < 0$ then longitude is $90^\circ + \theta^\circ$ west.

If $x = 0$ and $y > 0$ then longitude is 90° east, if $x = 0$ and $y < 0$ then longitude is 90° west.

Prior to establishing the main results we are going to introduce some preliminary concepts on spherical trigonometry.

1.2 Basic Spherical Trigonometry

Basic definitions and results from [54] are restated here for convenience. Formal definitions of the concepts especially connected with the theoretical foundations of the method are also given.

Definition 1.2.1. Every plane section of a sphere is a circle. The largest circle which can be drawn on the surface of a sphere is a circle passing through the centre of the sphere. Such a circle is called a *great circle*. All other circles on the surface of the sphere are called *small circles*.

The above definition leads to the following proposition:

Proposition 1.2.1. *Through any two points on the surface of a sphere, which are not diametrically opposite, one and only one great circle can be drawn.*

Definition 1.2.2. The *poles* of a great circle are the extremities of a diameter of the sphere that is perpendicular to the plane of the great circle. This diameter is also known as the *axis* of the great circle.

Note that the two poles for great circles are equidistant from its plane and the centre of the sphere. The poles and axes of small circles are similarly defined. However, since the plane of a small circle does not contain the centre of the sphere, its two poles are at a different distance from the plane of the small circle, one is nearer and the other is more distant. For convenience, we refer to them as the nearer and distant poles of a small circle.

Definition 1.2.3. Let A and B be two points, not diametrically opposite, on the surface of a sphere. Then, there is a unique great circle that passes through the two points. Moreover, A and B divide the great circle into two arcs. The length of the shorter arc is the *distance* between A and B or the *length of arc joining* A *and* B.

Since the sphere has a unit radius, the length of arc AB is simply the angle (measured in, e.g., radians) between two rays emanating from the centre of the sphere, O, one passing through A and the other through B.

Definition 1.2.4. The surface area of a sphere that is bounded by arc segments of three great

circles is called a *spherical triangle.*

If the spherical triangle has zero area, i.e., the three arcs intersect at a single point, then it is a degenerate spherical triangle, a case not considered in this research. Assume as in Article 22 in [54] that each side of a spherical triangle is less than π. This assumption yields the following proposition:

Proposition 1.2.2. *The sum of any two sides of a spherical triangle is greater than the third.*

1.3 Preliminary Concepts

The following notation will be used throughout the remainder of this chapter.

$A\hat{B}C \equiv$ the *spherical angle* subtended from a point B (on the surface of a sphere) by the (shorter) arc AC.

$\Delta ABC \equiv$ the plane triangle with vertices at points A, B, and C. Denote the angles of ΔABC as $<A, <B, <C$ or, $<BAC, <ABC, <ACB$.

As in [54], the spherical angle ABC is measured as the angle between two lines tangential at point B to the two great circles, one passing through A & B and the other through B & C.

The following definitions are necessary for the development of the algorithm.

Definition 1.3.1. Given three distinct points, A, B, and C, on the surface of a sphere, Π(A, B, C) denotes the unique plane passing through the three points and bisecting the sphere.

Definition 1.3.2. $\Gamma 3$(A, B, C) denotes the circle traced by the plane Π(A, B, C) cutting through the sphere.

Generally, $\Gamma 3$(A, B, C) is expected to be a small circle. However, it is possible that $\Gamma 3$(A, B, C) is a great circle.

Definition 1.3.3. Let A and B be two distinct points on the surface of a sphere that are not diametrically opposite. Denote the mid point of the (shorter) arc AB as the point P. Then, $\Gamma 2$(A,B) represents the small circle that goes through points A and B and has its nearer pole located at point P.

Definition 1.3.4. The length of the great circle arc from any point on the circumference of a small circle to its nearer pole is called the spherical radius of the small circle.

Definition 1.3.5. $N\Gamma 2$(A, B) and $N\Gamma 3$(A, B, C) denote the surface area of a sphere that contains the nearer pole of and is bounded by $\Gamma 2$(A, B) and $\Gamma 3$(A, B, C), respectively.

Definition 1.3.6. $R\Gamma 2$(A, B) and $R\Gamma 3$(A, B, C) denote the surface area of a sphere that contains the distant pole of and is bounded by $\Gamma 2$(A, B) and $\Gamma 3$(A, B, C), respectively.

Global Minimax Location Problem

2.1 Lemmas Related to the Algorithm

Before establishing the validity of the algorithm, presented in section 2.2, of the problem mentioned in section 1.1, we prove several lemmas.

Lemma 2.1.1. Let P be the nearer pole of $\Gamma 3(A, B, C)$, where ΔABC is an acute triangle. Let $Q(\neq P)$ be any point on $N\Gamma 3(A, B, C)$ and within the spherical triangle ABC. Then, the spherical radius of $\Gamma 3(A, B, C)$ is greater than minimum {QA, QB, QC}, where QA, QB, QC denote the geodesic distances between Q and A, Q and B, and Q and C, respectively.

Proof. In Figure 1, O denotes the centre of the circle $\Gamma 3(A, B, C)$. Then A_1, B_1, and C_1 are the points on the circumference of the circle that are diametrically opposite of A, B, and C, respectively. Since ΔABC is acute, points B and C cannot lie on the same side of the line joining A and A_1. The same is true for points C and A and the line joining B and B_1, and points A and B and the line joining C and C_1. Let D be any point of the circumference of $\Gamma 3(A, B, C)$, then obviously

$$\text{minimum } \{ \angle AOD, \angle BOD, \angle COD \} < \frac{\pi}{2}.$$

Extend the arc from P passing through Q to meet the circumference of $\Gamma 3(A, B, C)$ at point D. Without any loss of generality, assume

$$\angle AOD < \frac{\pi}{2}, \text{ i.e., } Q\hat{P}A < \frac{\pi}{2} \text{ and } \angle AOD \leq \angle BOD.$$

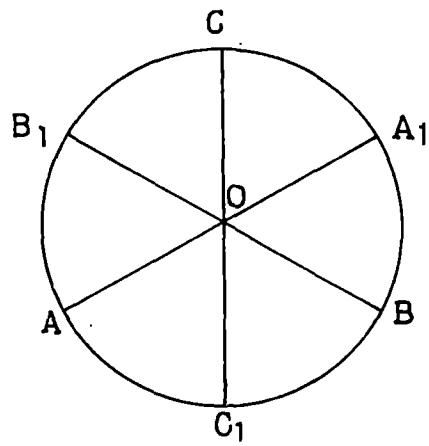


Figure - 1.

If Q lies on arc PA, then the proof is complete. When Q does not lie on arc PA, let M be the midpoint of the shorter arc segment between points A and B on the circumference of $\Gamma 3(A, B, C)$ (see Figure 2). Construct two great circle arcs, one joining points P and M and the other joining points A and Q. Extend arc AQ to meet arc PM at point T. By construction, the lengths of arcs BM and AM are the same. Since P is the nearer pole of $\Gamma 3(A, B, C)$, arcs PA and PB are of the same length also. Thus, the spherical triangles AMP and BMP are congruent and $A\hat{M}P = \frac{\pi}{2}$. From Article 42 in [54], we have

$$\cos(PA) = \cos(PM)\cos(AM) \tag{3}$$

$$\cos(TA) = \cos(TM)\cos(AM) \tag{4}$$

Now using the result PM > TM, we get from (3) and (4),

$$PA > TA \geq QA$$

Since PA is the spherical radius of $\Gamma 3(A, B, C)$, the proof is complete. □

Lemma 2.1.2. Let P_1 be the distant pole of $\Gamma 3(A, B, C)$ where ΔABC is an acute triangle. Let Q_1 be a point on $R\Gamma 3(A, B, C)$ and $Q_1 \neq P_1$. If Q_1 is sufficiently close to P_1 then

$$\text{maximum } \{AQ_1, BQ_1, CQ_1\} > AP_1$$

Proof. Let P be the nearer pole of $\Gamma 3(A, B, C)$. Let Q be the diametrically opposite point to Q_1. Obviously, Q is on $N\Gamma 3(A, B, C)$ and $P \neq Q$. Since Q_1 is sufficiently close to P_1 and P is in the spherical triangle ABC, Q must be in the spherical triangle as well. Assume that

$$QA = \text{minimum } \{QA, QB, QC\}.$$

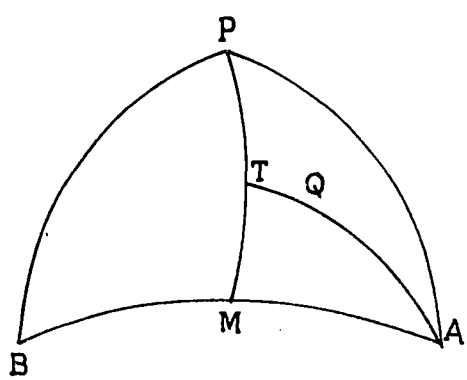


Figure - 2.

Then from Lemma 2.1.1 it follows that QA < PA. Construct two great circle arcs, one joining A to P_1 and the other joining A to Q_1. Since P and Q are diametrically opposite of P_1 and Q_1, respectively, we have

$$AP + AP_1 = \pi = AQ + AQ_1$$

Now $AQ < AP \Rightarrow AP_1 < AQ_1$. This proves lemma 2.1.2 . □

Lemma 2.1.3. Let A, B, and C be three points on a unit sphere with $<C> \frac{\pi}{2}$. Let P and P_1 be respectively the nearer and distant poles of Γ3(A, B, C). Then there exist points, Q say, close to P_1, such that arc CQ < arc AQ < arc AP_1.

Proof. To establish lemma 2.1.3 we make use of the following properties of a spherical triangle (see Articles 35 and 37 in [54]).

(a) The angles at the base of an isosceles spherical triangle are equal.

(b) If one angle of a spherical triangle is greater than another, the side opposite the greater angle is greater than the side opposite the smaller angle.

In figure 3, M denotes the mid point of the great circle arc AB. Since P_1 is the distant pole of Γ3(A, B, C), we have

$$\text{arc } AP_1 = \text{arc } BP_1 = \text{arc } CP_1 \tag{5}$$

Hence spherical triangles AMP_1 and BMP_1 are congruent (see Article 34 in [54]) and

$$B\hat{M}P_1 = \frac{\pi}{2} \tag{6}$$

Again $<C> \frac{\pi}{2}$ implies that C lies on the arc of the small circle AB. Without loss of generality, assume B and C to lie on the same side of the great circle arc PMP_1. Take a point Q on the great circle arc P_1M. Construct two great circle arcs, one joining Q to B and the other joining Q to C.

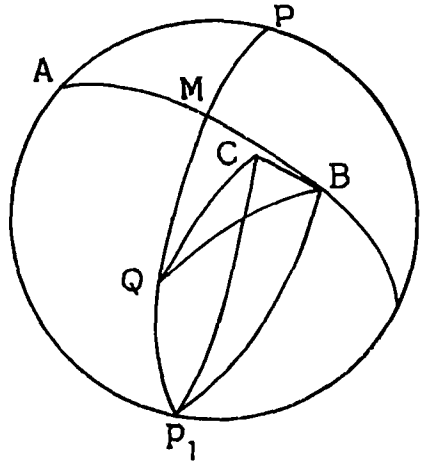


Figure - 3

Since arc BP_1 = arc CP_1 it follows from property (a) that

$$C\hat{B}P_1 = B\hat{P}C_1 \tag{7}$$

Since Q lies on arc MP_1 (see figure 3), we obtain

$$C\hat{B}Q < C\hat{B}P_1 \tag{8}$$

and $$B\hat{C}Q > B\hat{C}P_1 \tag{9}$$

It follows from property (b) and results (7) through (9),

$$\text{arc } BQ > \text{arc } CQ \tag{10}$$

Using (7), we get from the spherical triangles BMQ and BMP_1, (see Article 42 in [54])

$$\cos(BQ) = \cos(BM)\cos(MQ) \tag{11}$$

and $$\cos(BP_1) = \cos(BM)\cos(MP_1) \tag{12}$$

Since arc MQ < arc MP_1, we obtain from (11) and (12)

$$\text{arc } BQ < \text{arc } BP_1 \tag{13}$$

Again since spherical triangles AMQ and BMQ are congruent (see Article 34 in [54]), we get

$$\text{arc } AQ = \text{arc } BQ \tag{14}$$

Combining (14), (13), (10), and (5) we get the required result. □

It is to be noted that if $<C> \frac{\pi}{2}$ then from Lemma 2.1.3 it follows that the distant pole of $\Gamma 3(A, B, C)$ cannot be a solution of the spherical minimax location problem. Moreover, Lemma 2 of [50] implies that for $<C> \frac{\pi}{2}$, the nearer pole of $\Gamma 3(A, B, C)$ cannot yield the optimum solution of the hemispherical minimax location problem.

If one of the stated conditions in Theorem 2.1.1 below is true then we obtain the optimum solution of the hemispherical minimax location problem. The proof of Theorem 2.1.1 follows from lemmas 4 and 2 of [50].

Theorem 2.1.1. (i) If ΔABC is an acute triangle and $N\Gamma 3(A, B, C)$ contains all demand points then the nearer pole of $\Gamma 3(A, B, C)$ is the unique facility point.

(ii) If $N\Gamma 2(A, B)$ contains all demand points, then nearer pole of $\Gamma 2(A, B)$ is the required facility. □

We now state and prove a theorem regarding optimal solution of the global minimax problem.

Theorem 2.1.2. If there exists a triplet (A, B, C) of demand points such that

(i) ΔABC is acute,

(ii) The centre of the sphere and all demand points lie on the same side of $\Pi(A, B, C)$, and

(iii) (A, B, C) generates the plane closest to the centre of the sphere, then the distant pole of $\Gamma 3$(A, B, C) is the required facility point.

Proof. Lemma 2.1.3 implies that the triplet of points forming an obtuse triangle cannot yield an optimal solution. Furthermore, it follows from lemma 2.1.2 that the distant pole of the small circle defined by a triplet satisfying (i) and (ii) is a local minimum and (iii) implies optimality. □

It follows from lemma 2.1.3 that for a global minimax problem we have to consider only acute triangles, and there are at most n(n-1)(n-2)/6 acute triangles. Consequently, from Theorem 2.1.2 after a finite number of arithmetic operations we obtain the solution of the spherical minimax location problem. Our algorithm for global optimization is based on theorems 2.1.1 and 2.1.2. We are now going to present our algorithm.

2.2 Algorithm

A few remarks are in order before discussing the algorithm:

Denote the set of demand points by $S = \{A_i : i = 1, 2, ..., n\}$. For convenience, from now on we will refer to the existing demand points as points. Throughout the algorithm the set {A, B, C} is the most recent triplet of points defining Π(A, B, C). The parameter d_1 denotes the most recent Euclidean distance between Π(A, B, C) and the centre of the sphere (d_1 being simply the distance between the centres of the circle and the sphere) when ΔABC is an acute triangle, and d refers to the corresponding quantity in the previous iteration. Take the initial value of d equal to the radius of the unit sphere. Let (X, Y) denote the initial pair of demand points. The set S1 will contain the triplet of demand points corresponding to the local minimum. Initially S1 = ⌀. If a local minimum obtained at a particular iteration is better than the one calculated with the current contents of S1, then we update S1. The set S2 contains the pair of demand points (X, Y). During the course of a search as soon as a triplet of points forming a non-acute triangle is encountered, we update S2 with the ends of the largest side of the triangle provided the Euclidean distance between the new pair of points is greater than that between the points stored in S2.

If i, j, and k are three positive integers such that $i < j < k$, and n(>2) is an integer. Then to update the triplet (i, j, k) we mean one of the following:

(i) If $k < n$ then

$i \leftarrow i, j \leftarrow j, k \leftarrow k + 1$

(ii) If $k = n$ and $j < n - 1$ then

$i \leftarrow i, j \leftarrow j + 1, k \leftarrow j + 1$

(iii) If $k = n, j = n - 1$ and $i < n - 2$ then

$i \leftarrow i + 1, j \leftarrow i + 1, k \leftarrow j + 1$

Global Minimax Algorithm

Initial Step. $i \leftarrow 1, j \leftarrow 2, k \leftarrow 3, d \leftarrow 1, S1 \leftarrow \varnothing, A \leftarrow A_i, B \leftarrow A_j, C \leftarrow A_k, X \leftarrow A, Y \leftarrow B,$ $S2 = \{ X, Y \}$, ρ = Euclidean distance between X and Y. Go to Step 1.

Step 1. Find d_1. If ΔABC is acute and $d_1 \leq d$ then go to Step 2. Otherwise, go to Step 4.

Step 2. If all points of $S - \{A, B, C\}$ lie on $N\Gamma 3(A, B, C)$ then the nearer pole of $\Gamma 3(A, B, C)$ is the required facility point of a hemispherical location problem; stop.

Else go to Step 3.

Step 3. If all points of $S - \{A, B, C\}$ lie on $R\Gamma 3(A, B, C)$ then the distant pole of $\Gamma 3(A, B, C)$ is a local minimum; $d \leftarrow d_1$, update S1. Go to Step 5.

Step 4. If ΔABC is not an acute triangle then find d_2, the length of the largest side of ΔABC. If $d_2 > \rho$ then $\rho \leftarrow d_2$ and update S2. Go to Step 5.

Step 5. If $i = n - 2, j = n - 1$, and $k = n$ then go to Step 6.

Else update the triplet (i, j, k). $A \leftarrow A_i, B \leftarrow A_j, C \leftarrow A_k$ and repeat Step 1.

Step 6. If $d_2 = 2$, the diameter of the sphere, then if there exists a great circle passing through a pair (X, Y) of S2 and containing a point of $S - \{X, Y\}$ such that all other points of S lie on one side of this great circle then the poles of the great circle are the required facility points; stop.

If $d_2 < 2$, and $N\Gamma 2(X, Y)$ contains all points then the nearer pole of $\Gamma 2(X, Y)$ is the facility point of the hemispherical location problem; stop.

Else, the distant pole of the triplet in S1 is a facility point.

Remarks

1. If we consider all possible triplets of points taken from the set S of n points and we verify whether the remaining n-3 points not considered in a particular triplet lie on the same side of the plane passing through this triplet then the algorithm is $O(n^4)$.

2. It is to be noted that the plane of $\Gamma 2(A, B)$ divides the sphere into two disjoint surfaces. For any point $C \in R\Gamma 2(A, B)$, $\angle ACB$ is acute. On the other hand if $D \in N\Gamma 2(A, B)$ then $\angle ADB$ is obtuse. Hence for two given points A and B, the triplet (A, B, C) yields an optimum solution

provided C $\in$ RΓ2(A, B), ΔABC is acute and all points of the set S - {A, B, C} lie on one side of Π(A, B, C). Now RΓ2(A, B) may contain at most n - 2 points. Consequently, there are no more than n - 2 planes through A, B and one of the points of RΓ2(A, B) at a time. Among these planes only two planes contain points of RΓ2(A, B) on one side. These two planes have the characteristic property that they have the maximum and minimum inclinations with the plane of Γ2(A, B). Now O(n) arithmetic operations will be needed to determine these maximum and minimum inclinations, and O(n) additional arithmetic operations are required to know whether the remaining points lie on one side of the plane and if so whether the distance, d_1, of the plane from the centre of the sphere is less than d, the previous distance. Therefore for a given pair (A, B), O(n) arithmetic operations will be needed to get an optimum provided such a solution exists. Since A and B are any two points of S, for different A and B we can construct n(n-1)/2 different Γ2(A, B) each requiring O(n) arithmetic operations for testing optimality. Taking the above facts into account, the algorithm turns out to be $O(n^3)$ complex.

3. The algorithm presented here determines all global minimax facility points and also finds the hemispherical minimax location point. It also determines whether all points lie on a hemisphere or not.

2.3 Numerical Results

In this section we are going to obtain the solutions of two minimax location problems. These problems are taken from Patel's paper [45]. Problem 1 finds the solution of the global minimax problem and problem 2 obtains the solution of the hemispherical minimax problem.

Problem 1. The Cartesian coordinates of 14 points are given in Table 1.

The coordinates of the optimal point are

(0.967, -0.230, -0.108)

The Euclidean distance between the facility point and the farthest demand point is 1.6745 and the corresponding geodesic distance is 1.985. The number of local optima encountered during the execution of the algorithm is 5. The distant pole of the small circle, defined by the 5-*th* and 13-*th* and 14-*th* points, is the required minimax point.

It is to be noted that Patel [45] predicted the coordinates of the optimal facility as (0.223, 0.077, 0.972), and the Euclidean distance between the farthest demand point and the optimal facility as 1.701. But this result is not correct. Such inaccuracies are usual, because generating

all local solutions may, at times, be very difficult to implement in practice.

Table 1: Coordinates of 14 points on a unit sphere

Points	x	y	z
1	0.5105	0.2209	0.8310
2	0.8949	-0.1433	0.4226
3	0.7235	0.6794	-0.1219
4	0.6895	-0.6895	0.2215
5	-0.1822	0.9832	0.0000
6	0.0854	-0.8869	0.4540
7	0.3422	-0.9250	-0.1650
8	0.0441	0.8422	-0.5373
9	0.4330	-0.7500	-0.5000
10	0.2500	-0.4330	0.8660
11	0.1830	0.6830	0.7071
12	0.0872	0.0000	0.9962
13	-0.6209	-0.7399	-0.2588
14	-0.2113	0.4532	0.8660

Problem 2. Table 2 below shows the latitudes and longitudes as also the corresponding cartesian coordinates of these 15 points situated in the northern hemisphere.

The solution of this problem is the nearer pole of $\Gamma 2(A, B)$, where A and B denote the points Paris and Manila, respectively. The cartesian coordinates of the facility point are (0.1191, 0.6435, 0.7561) and the corresponding latitude and longitude are 49.12°N and 79.51°E, respectively

To solve Problems 1 and 2 we have developed the PASCAL Code (Borland's Turbo Pascal Version 6) of the above problems and used a DX2 66 MHz PC AT to implement the programs. The CPU times to get the optimal solutions for both the problems is 0.06 sec. Sarkar and Chaudhuri [50] solved Problem 2 using the same Computer and Compiler and the CPU time

involved was 0.16 sec. Patel [45] solved both the problems in about 5 secs. of CPU time. Patel [45] derived the result in a Convex 220 main frame computer.

Table 2. Latitudes and Longitudes, and corresponding cartesian coordinates of 15 points.

City/Point	ϕ	θ	x	y	z
London, England	51.5N	0.4E	0.6225	0.0043	0.7826
Paris, France	48.9N	2.3E	0.6568	0.0264	0.7536
Zurich, Switzerland	47.5N	8.5E	0.6694	0.1000	0.7361
Rome, Italy	41.9N	12.5E	0.7267	0.1611	0.6678
Copenhagan, Denmark	55.7N	12.6E	0.5500	0.1229	0.8261
Berlin, Germany	52.5N	13.4E	0.5922	0.1411	0.7934
Stockholm, Sweden	59.3N	18.9E	0.4830	0.1654	0.8600
Athens, Greece	38.0N	23.7E	0.7216	0.3167	0.6157
Ankara, Turkey	39.9N	32.8E	0.6449	0.4156	0.6415
Telaviv, Israel	32.1N	34.8E	0.6956	0.4835	0.5314
Moscow, Russia	55.7N	37.7E	0.4459	0.3446	0.8261
Teheran. Iran	35.4N	51.4E	0.5058	0.6370	0.5793
Bombay, India	18.9N	72.8E	0.2798	0.9038	0.3239
Manila, Philipines	14.6N	121.0E	-0.4984	0.8295	0.2521
Tokyo, Japan	35.6N	139.7E	-0.6201	0.5260	0.5820

Computational Results

We have considered 5 sets containing 10, 20, 50, 80 and 100 data points distributed at random over a unit sphere. Each of the above set was randomly generated a hundred times. Table 3 shows the results of computation.

Table: 3.

No. of points	Maximum No. of local minimum	Average CPU time in secs.
10	5	0.02
20	8	0.22
50	8	4.11
80	7	17.58
100	8	34.95

2.4 Conclusion

We have shown in this paper that solving the global minimax location problem with the geodesic norm when the points are spread over a sphere is to find the plane, closest to the centre of the sphere, containing three points forming an acute triangle. The process of optimization consists in finding a sequence of planes, containing acute triangles, of gradually diminishing distances from the centre of the sphere, provided such acute triangles exist.

At a particular iteration, if a triplet of points (A, B, C) forming an acute triangle is such that NΓ3(A, B, C) contains all points then the nearer pole of Γ3(A, B, C) is the required unique optimum location point; but if NΓ2(A, B) contains all points then the corresponding location point is the nearer pole of Γ2(A, B). On the contrary, if there exists neither NΓ3(A, B, C) nor NΓ2(A, B) having properties mentioned above then the distant pole of Γ3(A, B, C) is a required facility point provided Π(A, B, C) is closest to the centre of the sphere and ΔABC is acute.

In developing our algorithm we have made use of the distance between the centre of the sphere and the centre of the circle Γ3(A, B, C). The equation of the plane, containing three points A, B, and C, is

$$ax + by + cz + p = 0, \text{ where } a^2 + b^2 + c^2 = 1,$$

(a, b, c, p) being determined by the coordinates of A, B, and C. If Π(A, B, C) is closest to the centre of the sphere, then for a spherical minimax location problem, (a, b, c) is the facility point provided $p > 0$; for $p < 0$, (-a, -b, -c) is the facility point; when $p = 0$ either pole of the great circle containing A, B, C will represent the facility point.

For a hemispherical minimax location problem the coordinates of the facility point are (a, b, c) when $p < 0$ and (-a, -b, -c) when $p > 0$.

If the nearer pole of Γ2(A, B) is the facility point of a hemispherical minimax location problem then the coordinates of the facility point are (aλ, bλ, cλ) where (a, b, c) are the coordinates of the midpoint of the line segment AB and

$$\lambda = \frac{1}{\sqrt{a^2 + b^2 + c^2}}$$

2.5 Pascal Program of the Global Minimax Algorithm

In this section we develop the Pascal program corresponding to the global minimaxlocation algorithm given in section 2.2.

```
program spherical_location(input,output,infile);
{This program obtains exact solution, in finite number of steps, of the global minimax
location problem by using selective enumeration technique. In case of a global minimax
problem, it obtains all optimum solution when solution is not unique. This program also
determines the unique optimum solution of the hemispherical minimax problem.}
 uses crt,dos;
 type list=array[1..100] of real;
 var
  infile:text;
  x,y,z:list; {these three vectors denote the coordinates of the demand points }
  i,j,k,i1,i2,i3,j1,j2,j3,n,n1,n2,k1,k2,p1,p2:integer;
  a,b,c,d1,d2,e1,e2,e3,u,v,v1,dd:real;
  flag,aflag,count:boolean;
  hh,mm,ss,hs:word;
 {p1, p2 represent indices of the ends of the largest side of a non acute triangle. In a course of
a search as soon as a larger side is encountered, one has to update the values of p1 and p2. dd
is the length of the corresponding side. i1, i2, i3, denote the indices of vertices of an acute
triangle. We update these values when we get a better solution.
 u = the square of the diameter of the circle containing three points}
 procedure plane(p,q,r:integer);
{this procedure determines the equation to the plane through three demand points}
  begin        {constants a,b and c are the direction ratios of the normal to the plane}
   a:=(y[q]-y[p])*(z[r]-z[p])-(y[r]-y[p])*(z[q]-z[p]);
   b:=(z[q]-z[p])*(x[r]-x[p])-(z[r]-z[p])*(x[q]-x[p]);
   c:=(x[q]-x[p])*(y[r]-y[p])-(x[r]-x[p])*(y[q]-y[p]);
   d1:=a*x[p]+b*y[p]+c*z[p]
  end;{end of the procedure plane}
 procedure distance(var dist:real;r,s:integer);
{this procedure finds the square of the Euclidean distance between two points}
  begin
   dist:=sqr(x[r]-x[s])+sqr(y[r]-y[s])+sqr(z[r]-z[s])
  end;{end of the procedure distance}
```

```
procedure interchange(var e11,e22:real; var ii,jj:integer);
  var kk:integer;
{this procedure obtains the maximum of two numbers and the indices corresponding to these
numbers}
 begin
    if e11<e22 then
      begin
        v:=e11;kk:=ii;e11:=e22;e22:=v;ii:=jj;jj:=kk
      end
  end;{end of the procedure interchange}
 procedure  acute(s1,s2,s3:real); {this procedure tests whether three points form an acute
triangle}  begin
    j1:=i;j2:=j;j3:=k;aflag:=true;
{ when the value of the boolean variable aflag = false then we have to update the triplet (i, j,
k)}
   interchange(s1,s2,j1,j3);
    interchange(s1,s3,j2,j3);
    if s1>=s2+s3 then{this condition states that the triangle is not acute, consequently aflag =
false}
      begin
        aflag:=false;
        if s1>dd then
        { this condition implies that the largest side of the non-acute triangle is greater than the
previous value of the corresponding quantity, so one has to update the values of p1, p2 and
dd}
         begin
           p1:=j1;p2:=j2;dd:=s1;
         end
      end {end of s1>=s2+s3}
    else          {this condition states the triangle having vertices i, j, k is an acute triangle}
      begin
```

```
        v1:=(s1)/(1-sqr(s2+s3-s1)/(4*s2*s3));   {v1= square of the diameter of the small
circle}
        if v1<=u then aflag:=false
        {this condition implies that the radius of the circle through points i, j, and k is not
greater than the corresponding previous quantity  and one need updating the values of  i, j, k}
      end                                {v1>u implies better solution and aflag=true}
  end;{end of the procedure  acute}
 procedure test;
 {tests whether the points are on the same or opposite sides of the origin with respect to the
plane through points i, j, k}
   var l:integer;
   begin
    {k1>0 implies that there exists at least one point such that the centre of  the sphere and this
point lie on the opposite sides of the plane defined by the triplet (i, j, k)}
     k1:=0;k2:=0;l:=1;flag:=true;plane(i,j,k);
    {k2>0 implies that there exists at least one point such that the centre of  the sphere and this
point lie on the same side of the plane defined by the triplet (i, j, k)}
     while flag and (l<=n) do
      begin
       if (l<>i) and (l<>j) and (l<>k) then
        begin
         d2:=a*(x[l]-x[i])+b*(y[l]-y[i])+c*(z[l]-z[i]);
         if d1*d2>0 then k1:=k1+1 else k2:=k2+1
        end;          {end of test that points lie on the same side of the plane }
      {d1*d2>0 implies centre of the sphere and the point l lie on the opposite sides of the
plane defined by the triplet (i, j, k)}
       if (k1>0) and (k2>0) then flag:=false;
       l:=l+1;  {flag=false implies points do not lie on the same side of the plane defined by the
triplet (i, j, k)}
      end;{end of while}
     if flag=true then    {flag=true implies that the points lie on the same side of the plane
defined by the triplet (i, j, k)}
```

```
      begin
        u:=v1;i1:=i;i2:=j;i3:=k          {k1=0 and flag=true determines local optimum solution}
      end    {k2=0 or k1=n-3 implies optimal solution of the hemispherical problem defined by
three points}
  end;{end of procedure test}
 procedure optpoint;  {this procedure determines Cartesian coordinates of the optimum
point}
  begin
    plane(i1,i2,i3);
    d2:=1/sqrt(a*a+b*b+c*c);
    if d1>0 then
      begin
        a:=-a;b:=-b;c:=-c
      end;
    a:=a*d2;b:=b*d2;c:=c*d2
  end;{end of the procedure optpoint}
 procedure two_point;      {this procedure finds the coordinates of the facility point of the
hemispherical location problem when it is determined by two demand points}
  begin
   a:=(x[p1]+x[p2])/2;b:=(y[p1]+y[p2])/2;c:=(z[p1]+z[p2])/2;
   flag:=true;i:=1;
   writeln('p1, p2 ',p1,' ',p2);
   while (i<=n) and flag do
    begin
     if (i<>p1) and (i<>p2) then
       if  (x[i]-a)*a+(y[i]-b)*b+(z[i]-c)*c<0 then flag:=false;
       i:=i+1
    end
  end;{end of the procedure two_point}
 procedure latitude;     {this procedure finds latitude of the facility point}
  begin
    if c<0 then i:=-1;              {i=-1 implies south latitude}
```

```
    v:=d1*arctan(abs(c/sqrt(a*a+b*b)));
  end; {end of the procedure latitude}
procedure longitude;          {this procedure finds the longitude of the facility point}
  begin
   if a<>0 then v1:=d1*arctan(abs(b/a));
   if (a>0) and (b<0) then j:=-1;      {j=-1 implies longitude is west}
   if (a<0) and (b<0) then
     begin
      j:=-1;v1:=90+v1
     end;
   if (a<0) and (b>0) then v1:=90+v1;
  end; {end of the procedure longitude}
begin   {main action block}
  clrscr;
  assign(infile,'file1.hem');          {file1.hem contains coordinates of the demand points}
  reset(infile);
  writeln('supply the number, n, of demand points');
  readln(n);
  for i:=1 to n do
    readln(infile,x[i],y[i],z[i]);
  gettime(hh,mm,ss,hs);
  writeln(hh,':',mm,':',ss,'.',hs); {dd denotes the largest side of a non-acute triangle and p1, p2
represent the indices of the end point of  this side}
  u:=0;i1:=0;i2:=0;i3:=0;i:=1;n1:=n-2;n2:=n-1;count:=true;dd:=0;   {u gives the information
about the radius of the circle through the vertices, i, j and k of the acute angled triangle}
  while (i<= n1) and count do
{boolean variable count is used to determine whether there is an  acute angled triangle, having
vertices i, j, k , which yields the solution of the hemispherical minimax location problem}
    begin
     j:=i+1;
     while (j<=n2) and count do
       begin
```

```
        distance(e1,i,j);
        k:=j+1;
        while (k<=n) and count do
          begin
            distance(e2,j,k);
            distance(e3,k,i);
            acute(e1,e2,e3);
            if aflag=true then test;  {the value of the boolean variable aflag = true implies that
an acute triangle may have better solution}
            k:=k+1;
{the condition k1=n-3 implies optimum solution of a hemispherical minimax location problem}
            if k1=n-3 then count:=false
          end;{end of k loop}
        j:=j+1
      end;{end of loop j}
    i:=i+1;
  end; {end of i while}
  if count=false then  {count =false implies a triplet of the point (i, j, k) determines the
optimum solution of a hemispherical  minimax location problem}
    begin
      writeln('a hemispherical minimax problem');
      optpoint;
      a:=-a;b:=-b;c:=-c;
      writeln('Cartesian coordinates of the facility points are:');
      writeln(a,',',b,',',c)
    end
  else   {count=true implies either solution of a hemispherical minimax problem obtained by
two points or global problem}
    begin
      two_point;
      if flag=true  then
{flag = true implies solution a hemispherical problem generated by  two demand points}
```

```
      begin
        writeln('two point hemispherical problem');
        dd:=1/sqrt(a*a+b*b+c*c);
        a:=a*dd;b:=b*dd;c:=c*dd;
        writeln('Cartesian coordinates of the facility points are:');
        writeln('(',a:4:2,', ',b:4:2,', ',c:4:2,')')
      end
    else                    {flag=false yields solution of a global problem}
      begin
        writeln('a global minimax problem');
        optpoint;           {this procedure finds the coordinates of the optimum point}
        writeln('Cartesian coordinates of the facility points are:');
        writeln(a:4:2,',',b:4:2,',',c:4:2)
      end
  end;
d1:=45/arctan(1);
latitude;
longitude;
if i=-1 then {this condition implies that the latitude is south}
   writeln('latitude is ',v:4:2,'S')
else          {this condition implies that the latitude is north}
   writeln('latitude is ',v:4:2,'N');
 if j=-1 then {this condition implies that the longitude is west}
    writeln('longitude is ',v1:4:2,'W')
 else         {this condition implies that the longitude is east}
    writeln('longitude is ',v1:4:2,'E');
gettime(hh,mm,ss,hs);
writeln(hh,':',mm,':',ss,'.',hs);
close(infile)
end.{end of action block}
```

Solution of a Single Facility Location Problem on a Hemisphere using the Geodesic Norm

3.1 Lemmas and Theorems Related to the Algorithm

In this section we will present some basic tools which will be used to develop the algorithm, in section 3.2, of the problem given in section 1.1. The following theorems will be needed to develop our theory, the proofs of which are given in [54].

Theorem 3.1.1. *Any two sides of a spherical triangle are together greater than the third side.*

Theorem 3.1.2. *If one side of a spherical triangle is greater than another, the angle opposite the greater side is greater than the angle opposite the smaller side.*

Identical and symmetrical equality of triangles (ISET) [54]:

Two spherical triangles on the same sphere are either congruent or symmetrically equal, and have all their corresponding elements equal, when two sides and the included angle of one are equal to the corresponding quantities of the other.

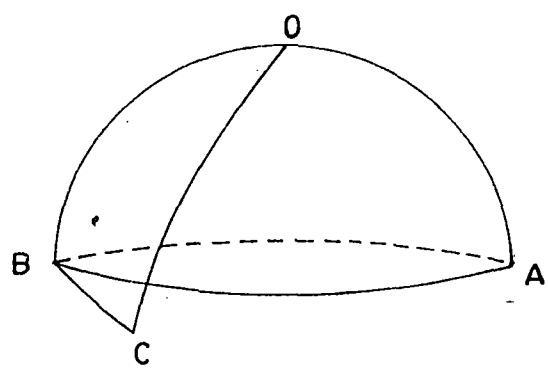
Lemma 3.1.1. *If* $C \in \Sigma\Gamma 2(A, B)$, *then*

$$A\hat{C}B \geq A\hat{B}C + B\hat{A}C \qquad (1)$$

and conversely (1) *implies* $C \in \Sigma\Gamma 2(A, B)$.

The equality sign occurs when $C \in \Gamma 2(A, B)$.

Proof. The result of the first part is given in [50]. Let us prove the converse, i.e., if (1) holds, then we are to prove $C \in \Sigma\Gamma 2(A, B)$. Assume that O is the nearer pole of $\Gamma 2(A, B)$ (see Fig.1). If possible, let $C \notin \Sigma\Gamma 2(A, B)$. Construct great circle arc joining C to B.



FIG·1

$C \notin \Sigma\Gamma 2(A, B) \Rightarrow OC > OB = OA.$

Consider the spherical triangles OCB and OAC. From Theorem 3.1.2 we have

$$O\hat{B}C > O\hat{C}B \text{ and } O\hat{A}C > O\hat{C}A$$

Adding the above inequalities we get,

$$A\hat{B}C + B\hat{A}C > A\hat{C}B$$

which contradicts (1), completing the proof of the converse. □

Lemma 3.1.2. *For a given* $\Gamma 3(A, B, C)$ *if* $A\hat{C}B > A\hat{B}C + B\hat{A}C$ *then* $\Sigma\Gamma 2(A, B)$ *contains* C *and the SR of* $\Sigma\Gamma 2(A, B) <$ *SR of* $\Sigma\Gamma 3(A, B, C)$.

Lemma 3.1.3. *Let* O *be the nearer pole of* $\Gamma 3(A, B, C)$. *Further, if*

$$\hat{A} < \hat{B} + \hat{C},\ \hat{B} < \hat{C} + \hat{A} \text{ and } \hat{C} < \hat{A} + \hat{B}$$

then there exists no $\Sigma\Gamma 3(P, Q, R)$ *containing* A, B, C *and having SR of* $\Gamma 3(P, Q, R) <$ *SR of* $\Gamma 3(A, B, C)$.

Proofs of lemmas 3.1.2 and 3.1.3 are given in [50].

Lemma 3.1.4. *Let* L *be the mid point of* AB. *If* C *and* B *lie on the same side of the plane of the great circle* γ *through* L *and orthogonal to* AB *then* AC > BC. *Conversely,* AC > BC *implies* B *and* C *lie on the same side of the plane through* γ.

Proof. Join A and C to intersect γ at D and also join B to C, and B to D (see Fig.2). ISET property implies AD = BD.

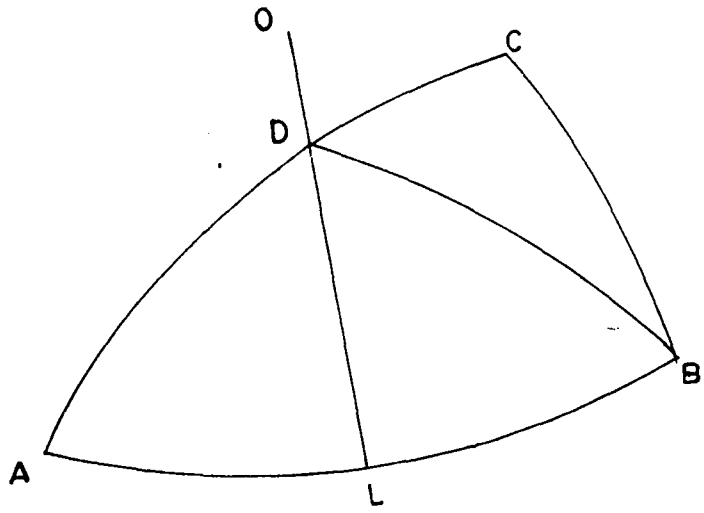


FIG. 2

It follows from Theorem 3.1.1,

$$BD + CD > BC$$

$$\Rightarrow AC > BC.$$

To prove the converse, let us assume that A and C lie on the same side of γ. It follows from the result mentioned above

$$BC > AC,$$

a contradiction. □

Lemma 3.1.5. *Let* D *be a point on* AC *produced. Assume that* AC *and* AD *are both less than* π*;* B *is a point not on the great circle* AC*, and* $B \in \Sigma\Gamma 3(A, C, D)$*. Suppose that the triplets* A, B, C *and* A, B, D *satisfy the conditions of Lemma* 3.1.3*, then*

$$SR \text{ of } \Gamma 3(A, B, C) < SR \text{ of } \Gamma 3(A, B, D).$$

Proof. Let us draw great circles γ_1 and γ_2 through the mid points of AC and AD, respectively, and perpendicular to AD. If γ_1 and γ_2 intersect at P, then $PC = PD = \pi/2$. Consequently, the nearer pole, O, of $\Gamma 3(A, B, C)$ and A lie on the same side of γ_2. It follows from Lemma 3.1.4, $OD > OC$, i.e., $D \notin \Gamma 3(A, B, C)$. Now applying Lemma 3.1.3 we get the result of Lemma 3.1.5.

□

Lemma 3.1.6. *Let* A *and* B *be two given points;* P_1 *and* P_2 *are situated on the same side of the plane of the great circle* AB*. Assume that the spherical angles of the triangles* ABP_1 *and* ABP_2 *satisfy the conditions of lemma 3.1.3. Then* $\theta_1 > \theta_2$ *implies that*

$$SR \text{ of } \Gamma 3(A, B, P_1) > SR \text{ of } \Gamma 3(A, B, P_2)$$

and P_2 *is contained in* $\Sigma\Gamma 3(A, B, P_1)$.

Proof. Let L be the mid point of AB (see Fig.3), and O, O′ be the nearer poles of $\Gamma 3(A, B, P_1)$

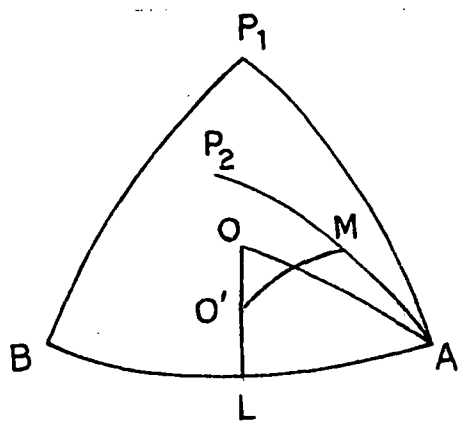


Fig. 3.

and $\Gamma 3(A, B, P_2)$ respectively. Clearly O, O′ lie on the great circle through L and orthogonal to AB. If we apply the two sides, the included angle, and another angle formula (see article 48, [54]) to the spherical triangle ALO, we have

$$\cos A L \cos L\hat{A}O = \sin A L \cot A O$$

$$\therefore \tan AO = \frac{\tan\left(\frac{AB}{2}\right)}{\cos\theta_1} \tag{2}$$

since $L\hat{A}O = \theta_1$ (see article 122, [54])

Similarly, from spherical triangle ABP_2, we get

$$\tan AO' = \frac{\tan\left(\frac{AB}{2}\right)}{\cos\theta_2} \tag{3}$$

$\theta_1 > \theta_2$, and equations (2) and (3) imply that

$$AO > AO' . \tag{4}$$

From (4) we get

$$SR \text{ of } \Gamma 3(A, B, P_1) > SR \text{ of } \Gamma 3(A, B, P_2)$$

Now $OA > O'A$, $OL < \pi/2$, and OL orthogonal to AB imply $OL > OL'$. Therefore, O and P_2 lie on the same side of MO′ where M is the mid point of AP_2. Hence from Lemma 3.1.4 one can conclude that $OP_2 < AO$. This proves the last part of the Lemma 3.1.6. □

Theorem 3.1.3. *Assume that* O *is the nearer pole of* $\Gamma 3(A, B, C)$ *and angles of the spherical triangle* ABC *satisfy the conditions of Lemma* 3.1.3. *Let* P *be a point on the sphere and* P *not in* $\Sigma\Gamma 3(A, B, C)$ *such that* P *and* C *lie on the same side of the plane of the great circle* AO *and let* AP *be greater than* BP *and* CP, *and* $AP < \pi$.

If $A\hat{B}P < A\hat{P}B + B\hat{A}P$ *then*

$C \in \Sigma\Gamma 3(A, B, P)$ *and SR of* $\Gamma 3(A, B, P) > SR$ *of* $\Gamma 3(A, B, C)$

Else

$C \in \Sigma\Gamma 2(A, P)$ *and SR of* $\Gamma 2(A, P) > SR$ *of* $\Gamma 3(A, B, C)$

Proof. Let L and M be the mid points of AB and AC respectively. We have to consider the following two cases:

Case I $A\hat{B}P < A\hat{P}B + B\hat{A}P$

Let O′ be the nearer pole of $\Gamma 3(A, B, P)$ (see Fig. 4).

From Theorem 3.1.2, $BP < AP \Rightarrow B\hat{A}P < A\hat{B}P \Rightarrow B\hat{A}P < A\hat{B}P + A\hat{P}B$

Again P is outside $\Sigma\Gamma3(A, B, C)$ implies $OB = OA < OP$. From these relations and Theorem 3.1.2 we get

$$O\hat{P}B < O\hat{B}P \text{ and } O\hat{P}A < O\hat{A}P$$

From the above inequalities we obtain

$$A\hat{P}B < O\hat{B}P + O\hat{A}P < A\hat{B}P + B\hat{A}P$$

Consequently, angles of the spherical triangle ABP satisfy the conditions of Lemma 3.1.3. From Lemma 3.1.5, we get

$$SR \text{ of } \Gamma3(A, P, B) > SR \text{ of } \Gamma3(A, Q, B) = SR \text{ of } \Gamma3(A, B, C),$$

where Q is on AP such that $OQ = OA$. Hence O' lies on the extended portion of LO. Consequently, C and O' lie on the same side of the great circle through M orthogonal to AC. Lemma 3.1.4, therefore, implies that

$$AO' > CO', \text{ i.e., } C \in \Sigma\Gamma3(A, P, B).$$

Case II $A\hat{B}P \geq A\hat{P}B + B\hat{A}P$.

It follows from Lemma 3.1.1 that $B \in \Sigma\Gamma2(A, P)$. Consequently the nearer pole O' of $\Gamma2(A, P)$ is the mid point of AP (see Fig.5). Since $O'B < O'A$, from Lemma 3.1.4 we conclude that O' and B lie on the same side of OL. Let E denote the point of intersection of AP and LO produced. Since O' and B lie on the same side of LO, $AO' > AE$. Again E and A being on opposite sides of OM, O' and C lie on the same side of OM. From Lemma 3.1.4 we have

$$CO' < AO', \text{ i.e., } C \in \Sigma\Gamma2(A, P).$$

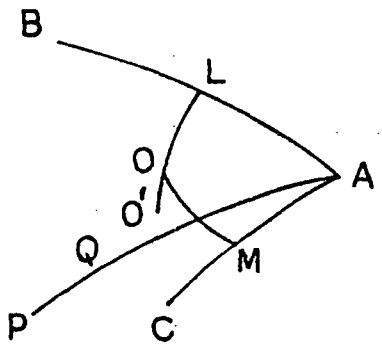


Fig. 4.

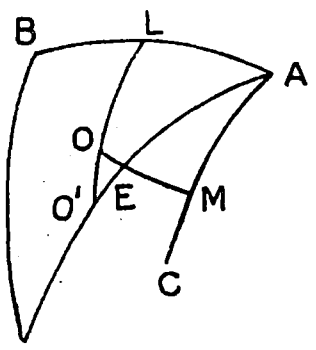


Fig. 5

Hence from Lemma 3.1.3 we obtain

SR of $\Gamma 3(A, B, C) <$ SR of $\Gamma 2(A, P)$.

3.2 Algorithm

We now proceed to describe the solution procedure of the problem. The set of demand points S is partitioned into two disjoint sets S_1 and S_2, where $S_1 = \varnothing$ and $S_2 = S$ initially. With each iteration S_1 absorbs at least one point which S_2 gives up. This continues until at last S_2 becomes void.

Algorithm Hemispherical Minimax Location Problem

Step 1. *Take any two points of S. Denote these points by* A, B.

Let $S_1 = S \cap \Sigma\Gamma 2(A, B)$ *and* $S_2 = S - S_1$ *and go to Step* 2.

Step 2. *If* $S_2 = \varnothing$, *stop; the nearer pole of* $\Gamma 2(A, B)$ *is the required facility point*

Else choose some point $C \in S_2$, *say, at a maximum distance (geodesic) from the nearer pole of* $\Gamma 2(A, B)$. *If* $AC < BC$ *then swap* A *with* B, B *with* C *and go to Step*3.

Step 3. *Let*

$\theta_k = max \{\theta_i : P_i \in S_1 - (A, B, C)$; P_i *and* C *lie on the same side of the plane of the great circle* AB$\}$

If $A\hat{P}_kB > A\hat{B}P_k + B\hat{A}P_k$ *then* $B \leftarrow P_k$, $S_1 = S \cap \Sigma\Gamma 2(A, B)$ *and go to Step* 2

Else $C \leftarrow P_k$, $S_1 = S \cap \Sigma\Gamma 3(A, B, C)$, $S_2 = S - S_1$ *and go to Step* 4.

Step 4. *If* $S_2 = \varnothing$, *stop. The nearer pole of* $\Sigma\Gamma 3(A, B, C)$ *is the required facility point.*

Otherwise choose $D \in S_2$, *and label the point among* A, B, C *that is farthest from* D *as* A. *Rename the other points* B *and* C. *Denote the nearer pole of* $\Gamma 3(A, B, C)$ *by* O.

If B *and* D *lie on the same side of* OA *then swap* B *with* C; $C \leftarrow D$ *and swap* B *with* C. *Repeat Step* 3.

Remarks

1. From Lemma 3.1.6 and Theorem 3.1.3 it follows that at each step of the algorithm, the spherical radius of the small circle which bounds the sphere containing the nearer pole, strictly increases. The algorithm needs the pole and the spherical radius of $\Gamma 2(A, B)$ or $\Gamma 3(A, B, C)$ at each iteration. Since $\Sigma\Gamma 2(A, B)$ and $\Sigma\Gamma 3(A, B, C)$ are defined uniquely, the same set of points cannot occur in any two iterations. Also the number of demand points is finite, and therefore the algorithm is finite.

2. Lemma 3.1.6 and Theorem 3.1.3 imply that if a demand point P belongs to S_1 at a particular iteration then $P \notin S_2$ in subsequent iterations. Consequently in no case the algorithm requires more than (n - 2) iterations. If there are k points to be considered in a particular iteration then the algorithm needs to compare at most k geodesic distances. Each distance can be calculated by using "three sides and an angle" formula (see article 41, [54]). Hence in the worst case we have to consider $O(n^2)$ distances to obtain the exact solution.

3. If conditions of Lemma 3 hold in a plane triangle then we have the following:

$$A < B + C \Rightarrow A < \pi/2.$$

The condition max(B + C - A) in Lemma 6 implies A being minimum. That is in step 3 we search for a minimum acute angle.

3.3 Numerical Example

Consider the following example given in Table 2, section 2.3. The reason for choosing this problem arises from the fact that all demand points are situated in the northern hemisphere. We have appended a comparative estimate of running time of algorithms [45] and [50] with that of the present one in Table 1. It would not be out of place to mention that the computer CPU time to run our program depends on the initial choice of the pair of demand points. The time .06 mentioned in Table 1 is the maximum computer CPU time corresponding to the initial choice of Athens and Stockholm.

Table 1: Comparison of execution times for different algorithms

Algorithm	Execution Time	Computer Type
Patel	Less than 5 sec.	Convex C220 main frame
Sarkar-Chaudhuri	0.16 sec.	PC AT 486 DX2 66 MHz
Present Algorithm	0.06 sec.	PC AT 486 DX2 66 MHz

For making a comparative estimate of the present algorithm and that of Sarkar-Chaudhuri [50] we have developed the PASCAL codes of both with Turbo Pascal (Borland) compiler 6.0.

3.4 Pascal Code of the Hemispherical Minimax Problem

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

```
program dualfeasible(input,output,infile);
```

```
{This program uses the concept of dual feasibility to determine optimum solution of the
minimax location problem when the demand points are situated on a hemisphere}
uses crt,dos;
type position=array[1..1000] of real;
var
 infile:text;
 x,y:position;
{these two vectors contain the latitudes and longitudes of the demand points on a unit sphere}
 i,j,k,n,i1,i2,i3,i4:integer;
 u,v,w,z,u1,u2,u3,v1,v2,v3,w1,z1,s,s1,s2,s3,r:real;
 flag:boolean;
 hh,mm,ss,hs:word;
procedure cosine(var a:real;i,j:integer);
{this procedure obtains the cosine of a side of a spherical triangle when other two sides and
the opposite angle are known}
 begin
   a:=sin(y[i])*sin(y[j])+cos(y[i])*cos(y[j])*cos(x[i]-x[j])
 end; {end of the procedure cosine}
procedure distance(var uu:real;ii,jj:integer);  {this procedure finds cot(side/2)}
 begin
    cosine(uu,ii,jj);
    uu:=sqrt((1+uu)/(1-uu));
 end;{end of the procedure distance}
procedure distance1(var uu:real;ii,jj:integer);   {this procedure determines tan(side/2)}
  begin
     cosine(uu,ii,jj);
     uu:=sqrt((1-uu)/(1+uu));
  end;{end of the procedure distance1}
procedure spangle(var v11,v22,v33:real;u11,u22,u33:real);    {when three sides of a spherical
triangle are known this procedure determines the spherical angles}
  begin
     s1:=sin(u22+u33-u11);
```

```
    s2:=sin(u33+u11-u22);
    s3:=sin(u11+u22-u33);
    s:=sin(u11+u22+u33);
    v11:=2*arctan(sqrt(s2*s3/(s*s1)));
    v22:=2*arctan(sqrt(s3*s1/(s*s2)));
    v33:=2*arctan(sqrt(s1*s2/(s*s3)));
  end;{end of the procedure spangle}
procedure midpoint;
{this procedure finds latitude and longitude of the mid point of a side of a spherical triangle}
  begin
    distance(u,i1,i2);
    v:=cos(y[i1])*sin(y[i2])/cos(y[i2])-sin(y[i1])*cos(x[i1]-x[i2]);
    v:=v/abs(sin(x[i1]-x[i2]));
    w:=cos(y[i1])*sqrt(1+v*v)*u-sin(y[i1])*v;
    z:=(v+sin(y[i1])*w)/(sqrt(1+w*w)*cos(y[i1]));
    if w>0 then w:=arctan(1/w)
    else w:=4*arctan(1)-arctan(-1/w);
    if x[i2]>x[i1] then w:=x[i1]+w
    else w:=x[i1]-w;
    z:=arctan(z);
  end;{end of the procedure mid point}
procedure angle(var v1,v2,v3:real;ii,jj,kk:integer);
 {this procedure uses latitude and longitude to obtain sides of the spherical triangle then with
the help of the procedure spangle it finds spherical angles}
  begin
    distance1(u1,jj,kk);u1:=arctan(u1);
    distance1(u2,kk,ii);u2:=arctan(u2);
    distance1(u3,ii,jj);u3:=arctan(u3);
    spangle(v1,v2,v3,u1,u2,u3)
  end;{end of procedure angle}
procedure interchange(var ii,jj:integer);
  begin
```

```
    j:=ii;
    ii:=jj;
    jj:=j
  end;{end of the procedure interchange}
 procedure centre;
 {this procedure determines the latitude and longitude of the nearer pole of the small circle}
 var
  v11,v22,v33:real;
  begin
    midpoint;
    angle(v11,v22,v33,i1,i2,i3);
    u1:=(sin(y[i1])*sqrt(1+u*u)-u*sin(z))/cos(z);
    v11:=(v11+v22-v33)/2;
    v1:=sqrt(1+u*u)*cos(v11)/sin(v11);
    v2:=(cos(z)*v1-sin(z)*sqrt(1-u1*u1))/abs(u1);
    z1:=(sin(z)*v2+sqrt(1-u1*u1)/abs(u1))/(sqrt(1+v2*v2)*cos(z));
    z1:=arctan(z1);{z1 denotes the latitude of the nearer pole}
    if v2<0 then
      v2:=4*arctan(1)-arctan(-1/v2)
    else
      v2:=arctan(1/v2);
    if u1<0 then w1:=w-v2
    else w1:=w+v2;    {w1 represents longitude of the nearer pole}
    u:=u*cos(v11);
    u:=u/sqrt(1+u*u);
  end;{end of procedure centre}
 procedure step1;
 { this procedure finds if there is any optimum solution of the problem given by two points}
  begin
    midpoint; {gives w, z,  i.e., longitude and latitude}
    cosine(u,i1,i2);
    u:=sqrt((1+u)/2);
```

```
 i3:=0;
 for i:=1 to n do
  begin
   if (i<>i1) and (i<>i2) then
    begin
      u1:=sin(z)*sin(y[i])+cos(z)*cos(y[i])*cos(w-x[i]);
      if u1<u then      {u1<u implies point i is outside}
       begin
         u:=u1;
         i3:=i
       end
    end;
  end {end of for}  {aflage=true represents optimal solution}
end;{end of the procedure step1}
procedure step2; {this procedure examines whether the solution of the problem is obtained by
two or three points}
 begin
  cosine(u,i1,i3);
  cosine(v,i2,i3);
   if u>v then interchange(i1,i2);
   angle(v1,v2,v3,i1,i2,i3);
   if  v2>v1+v3 then   {v2>v1+v3 represents two point problem}
    begin
     i2:=i3;i3:=0
    end
 end;{end of the procedure step2}
procedure step3;{this procedure finds whether there is a point outside the spherical disc}
 begin
 i4:=0;
 if y[i1]>y[i3] then interchange(i1,i3);
 if y[i2]>y[i3] then interchange(i2,i3);
 if x[i1]>x[i2] then interchange(i1,i2);
```

```
centre;
for i:=1 to n  do
 begin
  if (i<>i1) and (i<>i2) and (i<>i3) then
   begin
    u3:=sin(z1)*sin(y[i])+cos(z1)*cos(y[i])*cos(w1-x[i]);
    if u3<u then {u3<u implies that the point is outside the spherical disc}
      begin
        u:=u3;
        i4:=i
      end
   end;
 end; {end of for}
if i4<>0 then    {i4=0 implies optimal solution}
 begin
   cosine(u1,i1,i4);
   cosine(u2,i2,i4);
   cosine(u3,i3,i4);
   if u1>u2 then interchange(i1,i2);
   if u1>u3 then interchange(i1,i3);
   if u2>u3 then interchange(i2,i3);
   angle(u1,u2,u3,i1,i2,i4);
   angle(v1,v2,v3,i1,i3,i4);
   s1:=u3+u1-u2;
   s2:=v3+v1-v2;
   if (s1<0) and (s2<0) then
{from this condition we get optimum solution of the problem by two demand points}
     begin
       i2:=i4;i3:=0
     end
   else
```

```
{if this condition is true then we get the optimum solution of the problem from three demand
points}
      begin
       if s1<0 then i2:=i4
       else
         if s2<0 then i3:=i4
         else
           begin
             if s1>s2 then i2:=i4
             else i3:=i4
           end
      end
   end
 end;{end of step 3}
begin{main action block}
 clrscr;
 assign(infile,'file.pkc'); {this file contains latitude and longitude of the demand points}
 reset(infile);
 writeln('supply the value of n, i.e., no. points');
 readln(n);
{from the infile the vectors x[i], y[i] read the latitudes and longitudes of the demand points
expressed in degrees}
 for i:=1 to n do
    readln(infile,x[i],y[i]); {from the following transformation law one gets the latitudes and
longitudes in terms of radians}
 r:=arctan(1)/45;
 for i:= 1 to n do
  begin
    x[i]:=x[i]*r; y[i]:=y[i]*r
  end;
 gettime(hh,mm,ss,hs);
 writeln(hh,':',mm,':',ss,'.',hs);
```

```
flag:=true;i1:=7;i2:=8;k:=0;i3:=0;r:=1/r;
while flag do
 begin
  if i3=0 then
   begin
    step1;
    if i3=0 then
     begin
       flag:=false;
       writeln('the lat. and long. are ',z*r,' and ',w*r)
     end
   end;
 if i3<>0 then
  begin
   step2;
   if i3<>0 then
    begin
      step3;
      if i4=0 then
      begin
        flag:=false;
        writeln('the lat. and long. are ',z1*r,' and ',w1*r)
      end
    end
  end;
 k:=k+1;
end;{end of while}
writeln('no. of iteration= ',k);
gettime(hh,mm,ss,hs);
writeln(hh,':',mm,':',ss,'.',hs);
close(infile)
end.
```

Polynomial Time Algorithm for a Hemispherical Minimax Location Problem

4.1 Background

In this section we are going to develop an efficient algorithm to solve the problem (2), for the hemispherical minimax location problem, mentioned in section 1.1. We assume that all demand points lie on a hemisphere. We first mention the strategy of the Sarkar-Chaudhuri Algorithm [22]. Let I = {1, 2,...,n}. Then Sarkar-Chaudhuri algorithm may be described as follows:

Initial Step. Choose any point P on the surface of the hemisphere which contains all the demand points and let A_k, $k \in I$, be the farthest demand point from P. Denote this farthest point by A. Let $I \leftarrow I - \{k\}$. Connect A with P by a great circle arc. Let P_1 be a point on the arc PA such that

$$\text{arc } P_1A = \text{arc } P_1A_i ,$$

where $i \in I$ and arc PP_1 is minimal. Let the demand point satisfying the above be denoted by B and $I \leftarrow I - \{i\}$. Go to Step 1.

Step 1. If all the demand points lie on $\Sigma\Gamma 2(A, B)$ then stop; P*, the nearer pole of $\Gamma 2(A, B)$ is the required facility point. Else $P \leftarrow P_1$, and go to Step 2.

Step 2. Join P and the middle point, D, of AB by the arc of a great circle. Find a point P_1 on arc PD such that arc P_1A = arc P_1A_k, $k \in I$ and arc P_1P is minimum. Denote this point by C.

If $\hat{A} < \hat{B} + \hat{C}$, $\hat{B} < \hat{C} + \hat{A}$ and $\hat{C} < \hat{A} + \hat{B}$

then stop; $P^* = P_1$ is the required facility point.

Else go to Step 3.

Step 3. If $A_k\hat{A}_iA_j > A_i\hat{A}_jA_k + A_j\hat{A}_kA_i$ where $A_i, A_j, A_k \in \{A, B, C\}$ and i, j, k are all different then drop A_i. Denote the point A_j and A_k by A and B respectively. Let $P \leftarrow P_1$, $I \leftarrow I - \{i\}$ and repeat Step 1.

The optimality criteria of the hemispherical minimax location problem have been given in Steps 1 and 2. Let us see the geometrical significance of these two.

Assume that P* is the midpoint of the arc AB and if all demand points lie on $\Sigma\Gamma 2(A, B)$ then Step 1 states that P* is the required facility point. For optimality, arc AP* = arc BP*. The result follows from Lemma 4.1.1.

Lemma 4.1.1. *Let* $P \in \Sigma\Gamma 2(A, B)$. *Then either arc* AP *or arc* BP *is greater than or equal to arc* AP*.

Proof. Join AP, BP, and AB by great circle arcs (see Figure 1). From Proposition 1.2.2, we have from the spherical triangle ABP

$$\text{arc AP} + \text{arc BP} \geq \text{arc AB} = 2 \text{ arc P*A}$$

The above inequality implies either arc AP or arc BP is greater than or equal to arc AP*. □

Step 2 states that if all demand points lie on $\Sigma\Gamma 3(A, B, C)$ then P*, the nearer pole of $\Gamma 3(A, B, C)$, is the required facility point provided

$$\hat{A} < \hat{B} + \hat{C}, \hat{B} < \hat{C} + \hat{A} \text{ and } \hat{C} < \hat{A} + \hat{B}$$

The following lemma explains the significance of the stopping criteria given in Step 2 of the Sarkar-Chaudhuri algorithm [22].

Lemma 4.1.2. *In a spherical triangle* ABC

$$\hat{A} < \hat{B} + \hat{C},\ \hat{B} < \hat{C} + \hat{A} \text{ and } \hat{C} < \hat{A} + \hat{B}$$

imply Δ ABC *is an acute triangle.*

Proof. Let P be the nearer pole of $\Gamma 3(A, B, C)$ and let O be the centre of $\Gamma 3(A, B, C)$. Assume that the great circle arcs AP, BP and CP intersect $\Gamma 3$ (A, B, C) at A_1, B_1 and C_1 respectively. It follows from Lemma 3 and Corollary 2 (see, [22]) that B and C lie on opposite sides of AA_1, C and A lie on opposite sides of BB_1, and A and B lie on opposite sides of CC_1 (see Figure 2). Therefore, ΔABC is an acute triangle. □

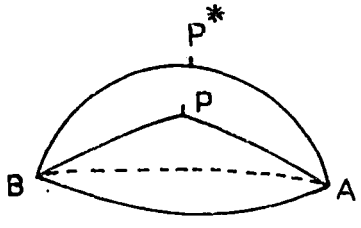


Figure-1

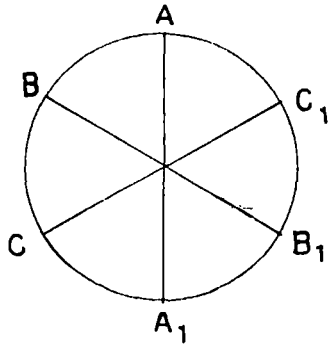


Figure - 2

It is clear form Lemmas 4.1.1 and 4.1.2 that the optimal solution of the hemispherical minimax location problem is the nearer pole P* of $\Gamma 2(A, B)$ [$\Gamma 3(A, B, C)$] provided $\Sigma\Gamma 2(A, B)$ [$\Sigma\Gamma 3(A, B, C)$] contains all demand points. Thus a hemispherical minimax problem reduces to finding a small circle of a maximum radius on the surface of the sphere which contains either two demand points at the ends of a diameter or three demand points forming an acute triangle such that all demand points lie on one side of the plane of the small circle and the centre of the sphere on the other side.

Our algorithm is based on the following lemma:

Lemma 4.1.3. *Let* A *and* B *be two points on the surface of the sphere* Σ *such that* A *and* B *are not the ends of a diameter of the sphere. Let* $Q \in \Sigma$ *such that* $Q \notin \Sigma\Gamma 2(A, B)$. *Then* $<AQB$ *is an acute angle.*

Proof. Construct the sphere S(A, B) with $\Gamma 2(A, B)$ as a great circle. Clearly all points of $\Sigma\Gamma 2(A, B) - \Gamma 2(A, B)$ lie within S(A, B) and all points of $\Sigma - \Sigma\Gamma 2(A, B)$ lie outside S(A, B). Now Q being a point outside S(A, B) it is obvious that $<AQB$ is less than a right angle. □

Corollary 4.1.1 *Let* A, B, *and* C *be three points on the surface of the sphere* Σ *such that* ΔABC *is an acute triangle. Further assume that* $Q \in \Sigma$ *and* $Q \notin \Sigma\Gamma 3(A, B, C)$ *where* $\Sigma\Gamma 3(A,B,C)$ *is a small circle. Then* $OQ > OA = OB$, *where* O *is the centre of* $\Gamma 3(A, B, C)$.

Proof. The result of the corollary follows immediately from the proof of the Lemma 4.1.3. □

Using Lemma 4.1.3 and the corollary 4.1.1 we have the following algorithm for the hemispherical minimax location problem given in section 4.2.

4.2 Algorithm Hemispherical Minimax Location

Let the set S, of demand points, lie on the surface of the sphere Σ. We assume that these demand points lie on a hemisphere. Then our algorithm may be described as follows:

Initial Step *Take any two demand points, say* A *and* B, *and go to Step* 1.

Step 1 *If all demand points lie on* $\Sigma\Gamma 2(A, B)$, *stop. The nearer pole,* P*, *of* $\Gamma 2(A, B)$ *is the required facility point.*

Else chose a demand point, C *say, not in* $\Sigma\Gamma 2(A, B)$ *such that* $<ACB$ *is a minimum. Go to Step* 2.

Step 2 *If all demand points lie on* $\Sigma\Gamma 3(A, B, C)$ *and* ΔABC *is an acute triangle then stop. The nearer pole* P* *of* $\Gamma 3(A, B, C)$ *is the required facility point.*

Else go to Step 3.

Step 3 *If* ΔABC *is an obtuse triangle then drop the point with the obtuse angle, rename the remaining points* A *and* B, *and go to Step* 1

Else find a demand point, D *say, in* $\Sigma - \Sigma\Gamma 3(A, B, C)$ *such that the distance of* D *from the centre of* $\Gamma 3(A, B, C)$ *is maximum and go to Step* 4.

Step 4 *Find the maximum distance of* D *from* A, B *and* C. *Denote the point having maximum distance from* D *by* A *and rename the other two points as* B *and* C. *Find minimum* {<ABD, <ACD}.

If this minimum is greater than $\frac{\pi}{2}$ *then* B ← D *and go to Step* 1.

Else drop the point with the maximum angle. Denote the other point by B. *Let* C ← D *and go to Step* 2.

Remarks

1. The optimality conditions, of the present algorithm, are given in Steps 1 and 2. To obtain optimality conditions we may proceed as follows: Let

$f(x, y, z) = lx + my + nz - p = 0$, where

$l^2 + m^2 + n^2 = 1$ and $p > 0$,

be either the plane $\Sigma\Gamma 2(A, B)$ or the plane $\Sigma\Gamma 3(A, B, C)$. If for all demand points (x, y, z), $f(x, y, z) \geq 0$ then the coordinates of the facility point P* are (l, m, n).

2. The point D(x, y, z) mentioned in Step3 of the algorithm has the following characteristic properties;

(i) $f(x, y, z) < 0$

(ii) f(x, y, z) is a minimum.

4.3 Numerical Example

In this section we are going to compare the efficiency of the algorithm, which has been presented in section 4.2, with the existing algorithms. We now consider the following problem given in Table 2, section2.4, and develop the Turbo PASCAL (Borland) code of the algorithm. We have used a PC AT 486, DX2 66 MHz to compare the CPU time of the present method and the existing algorithms. The results are given in Table 1. The Cartesian coordinates of the facility point are (0.1191, 0.6435, 0.7561) and the optimum latitude and longitude of the facility point are 49.12° N and 79.51^0E respectively. The nearer pole of $\Gamma 2(A, B)$ is the required facility point

where A and B denote the demand points Paris and Manila respectively.

Table 1: CPU time for different algorithms

Algorithm	CPU Time	Computer
Patel	Less than 5 secs.	Convex C220 mainframe
Sarkar-Chaudhuri	0.16 sec.	PC AT DX2 66 Mhz
Method of section 2.2	0.06 sec.	PC AT DX2 66 Mhz
Present Algorithm	0.00 sec.	PC AT DX2 66 MHz

We have considered 5 sets containing 10, 20, 50, 80, and 100 data points distributed at random over a unit sphere. Each of the above set was randomly generated twenty five times. Table 2 shows the computation time (in seconds) of the present algorithm and that of the algorithm given in section 2.2. We have used Turbo PASCAL (Borland) Version 6 to run both the programs in a PC AT 486, DX2 66 MHz computer.

Table 2:

No. of Points	CPU time of the present algorithm	CPU time of the Algorithm given in Section 2.2
10	0.00	0.00
20	0.00	0.11
50	0.00	2.00
80	0.01	7.24
100	0.03	15.05

Table 3:

No. of points	CPU time of the present algorithm
500	0.11
1000	0.23
1500	0.36
2000	0.48
2500	0.61
3000	0.73

We next consider 500 to 3000 data points at an interval of 500. Table 3 above shows the corresponding CPU time (in seconds) to run the program in the same computer and using the same source code.

Conclusion.

The algorithm presented here is significantly faster than all the existing algorithms. From the computational point of view the Sarkar-Chaudhuri Algorithm [22] takes much more time than the present algorithm, despite the former being $O(n^2)$. This is so because the Sarker-Chauduri Algorithm involves inter alia computation of trigonometric functions whereas the present method uses only Euclidean distances.

In each step, the algorithm requires the equation of a plane through three demand points, forming an acute triangle, or through two demand points, which are the ends of the diameter of a small circle. Then we have to verify whether all other demand points lie on the same side of this plane. If this is true then we obtain the optimum solution of the problem which is the nearer pole of the small circle determined by the section of the sphere by this plane. Otherwise we have to determine a new set of points and the plane through it and we have to continue this process until we obtain the optimum solution. It is to be noted that the maximum number of arithmetic operations necessary to get the exact optimum solution is $O(n^2)$.

4.4 Pascal Code of the algorithm

In this section we implement the Pascal code of the algorithm developed in section 2.2.

```
program hemisphere(input,output,infile);
{ This program finds the optimum solution of a hemispherical minimax location problem }
 uses crt,dos;
 type
   list=array[1..3500] of real;
 var
   infile:text;
   x,y,z:list;  {these three vectors contain the Cartesian coordinates of the demand points
    situated on a unit sphere}
   i,j,i1,i2,i3,i4,n:integer;
   a,b,c,u,v,d:real;
```

```
flag:boolean;
    hh,mm,ss,hs:word;
 procedure distance(var d1:real;j1,j2:integer);
{This procedure obtains square of the distance between two demand points}
  begin
   d1:=sqr(x[j1]-x[j2])+sqr(y[j1]-y[j2])+sqr(z[j1]-z[j2])
  end; {end of distance}
 procedure swap(var d1,d2:real;var j1,j2:integer);
  {This procedure interchanges the memory location of two quantities and the corresponding
indices}
 var
    u1:real;
    k:integer;
  begin
     u1:=d1;d1:=d2;d2:=u1;
     k:=j1;j1:=j2;j2:=k
  end;{end of swap}
 procedure radius(var r:real;d1,d2,d3:real);
{This procedure finds the square of the diameter of the circle containing three demand points}
   var
    u1:real;
   begin
    u1:=4*d2*d3;
    r:=(d2+d3-d1);
    r:=u1*d1/(u1-r*r);
   end;{end of radius}
 procedure update2;
{This procedure determines which two demand points to be considered for the next iteration }
  var
    u1,u2,u3:real;
   begin
    distance(u1,i2,i3);
```

```
    distance(u2,i3,i1);
    distance(u3,i1,i2);
    if (u1<u2) then swap(u1,u2,i1,i2);{interchange the points i1,i2}
    u1:=u1-u2-u3;
    if u1>=0 then {drop the point i1}
      begin
       i1:=i2;i2:=i3;i3:=0
      end
  end;{end of update2}
procedure update3;
{This procedure obtains three demand points to be considered for the next iteration}
  var
    d1,d2,u1,u2,u3,v2,v3:real;
  begin
    distance(u1,i1,i4);
    distance(u2,i2,i4);
    distance(u3,i3,i4);
    if (u1<u2) then swap(u1,u2,i1,i2);
    if (u1<u3) then swap(u1,u3,i1,i3);
    distance(v2,i1,i2);
    distance(v3,i1,i3);
    d1:=u1-u2-v2;
    d2:=u1-u3-v3;
    if (d1>=0) and (d2>=0) then {drop points i2,i3}
      begin
        i2:=i4;i3:=0
      end
    else if (d1<0) and (d2>=0) then  {drop the point i3}
      begin
        i3:=i2;i2:=i4
      end
    else if (d1>=0) and (d2<0) then i2:=i4 {drop the point i2}
```

```
    else
     begin
      radius(d1,u1,u2,v2);
      radius(d2,u1,u3,v3);
      if (d1>d2) then i3:=i4 {drop the point i3}
      else i2:=i4    {drop the point i2}
     end;
    i4:=0
   end;{end of update 3}
  procedure plane;
{ This procedure obtains the direction ratios of the  normal to the plane containing three demand points}
   begin
    a:=(y[i2]-y[i1])*(z[i3]-z[i1])-(y[i3]-y[i1])*(z[i2]-z[i1]);
    b:=(z[i2]-z[i1])*(x[i3]-x[i1])-(z[i3]-z[i1])*(x[i2]-x[i1]);
    c:=(x[i2]-x[i1])*(y[i3]-y[i1])-(x[i3]-x[i1])*(y[i2]-y[i1]);
    d:=a*x[i1]+b*y[i1]+c*z[i1]
   end;{end of plane}
  procedure plane1;
{ This procedure obtains the direction ratios of the normal to the plane containing two demand points}
   begin
    a:=x[i1]+x[i2];b:=y[i1]+y[i2];c:=z[i1]+z[i2];
    d:=a*x[i1]+b*y[i1]+c*z[i1]
   end; {end of procedure plane1}
  procedure optimum;
{ This procedure determines Cartesian coordinates of  the required facility point}
   begin
    if i3=0 then
     begin
      a:=x[i1]+x[i2];b:=y[i1]+y[i2];c:=z[i1]+z[i2];
      u:=sqrt(a*a+b*b+c*c);
```

```
      a:=a/u;b:=b/u;c:=c/u;
      writeln('Solution is obtained by the demand points',i1,', ',i2)
     end
    else
     begin
      plane;
      u:=sqrt(a*a+b*b+c*c);
      if d>0 then
       begin
        a:=a/u;b:=b/u;c:=c/u
       end
      else
       begin
        a:=-a/u;b:=-b/u;c:=-c/u
       end;
      writeln('Solution is obtained by the demand points ',i1,' ',i2,' ',i3);
     end
  end;{end of optimum}
procedure latitude; {This procedure finds the latitude of the facility point}
  begin
   i:=1;
   if c<0 then i:=-1;
   u:=d*arctan(abs(c)/sqrt(a*a+b*b));
  end;{end of latitude}
procedure longitude; {This procedure determines the longitude of the facility point}
  begin
   j:=1;
   if (a<>0) then v:=d*arctan(abs(b/a));
   if (a>0) and (b<0) then j:=-1;
   if (a<0) and (b<0) then
    begin
     j:=-1;v:=90+v
```

```
    end;
   if (a<0) and (b>0) then v:=90+v
  end;{end of longitude}
procedure twopoint;
  var
   u1,u2:real;
  begin
   v:=0;u:=0;
   plane1;
   u2:=sqrt(a*a+b*b+c*c);
   u2:=1/u2;
   for i:=1 to n do
    if ((i<>i1) and (i<>i2)) then
     begin
      u1:=d-a*x[i]-b*y[i]-c*z[i];
      if (d*u1>0) then
       begin
        v:=abs(u1*u2);
        if v>u then
         begin
          u:=v;i3:=i
         end
       end;
     end; {end of loop}
   if i3=0 then flag:=false
   else  update2;
  end;{end of twopoint}
procedure threepoint;
 var
   u1,u2:real;
  begin
    v:=0;u:=0;
```

```
    plane;
    u2:=sqrt(a*a+b*b+c*c);
    for i:=1 to n do
     if ((i<>i1) and (i<>i2) and (i<>i3)) then
      begin
       u1:=d-a*x[i]-b*y[i]-c*z[i];
       if (d*u1>0) then
        begin
         v:=abs(u1/u2);
         if v>u then
          begin
           u:=v;i4:=i
          end
        end;
      end; {end of loop}
   if i4=0 then flag:=false
   else  update3;
  end;{end of threepoint}
begin {main action block}
  clrscr;
  assign(infile,'file1.hem');{file1.hem contains coordinates of the data points}
  reset(infile);
  writeln('supply the number of demand points');
  readln(n);
  for i:=1 to n do
    readln(infile,x[i],y[i],z[i]);
  gettime(hh,mm,ss,hs);
  writeln(hh,':',mm,':',ss,':',hs);
  i3:=0;i4:=0;i1:=1;i2:=2;
  flag:=true;
  while flag do
    begin
```

```
      if (i3=0) then twopoint;
      if ((flag=true) and (i3<>0)) then threepoint;
    end;
  optimum;
  writeln('The Cartesian coordinates of the optimum point are');
  writeln(' (',a:2:4,', ',b:2:4,', ',c:2:4,')');
  d:=45/arctan(1);
  latitude;
  longitude;
  write('The latitude and longitude of the facility point are');
  if i=-1 then write('(',u:4:2,' S, ')
  else write('(',u:4:2,' N, ');
  if j=-1 then writeln(v:4:2,' W)')
  else writeln(v:4:2,' E)');
  gettime(hh,mm,ss,hs);
  writeln(hh,':',mm,':',ss,'.',hs);
  v:=sqr(x[i1]-a)+sqr(y[i1]-b)+sqr(z[i1]-c);
  u:=sqr(x[i2]-a)+sqr(y[i2]-b)+sqr(z[i2]-c);
  if u>v then v:=u;
  u:=sqr(x[i3]-a)+sqr(y[i3]-b)+sqr(z[i3]-c);
  if u>v then v:=u;
  writeln(v);
  for i:= 1 to n do
   begin
     if ((i<>i1) and (i<>i2) and (i<>i3)) then
       begin
          u:=sqr(x[i]-a)+sqr(y[i]-b)+sqr(z[i]-c);
          if u>v then writeln('u, i= ',u,'  ',i);
       end
    end;
  close(infile)
end. {end of action block}
```

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 $EQ(P_i, P_j, P_k)$.

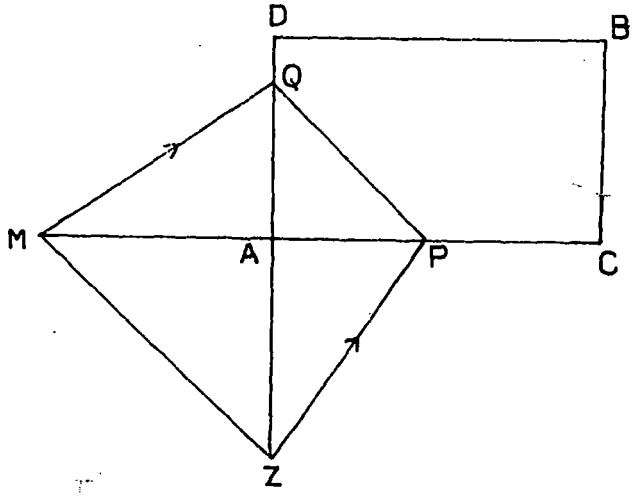


Figure - 1

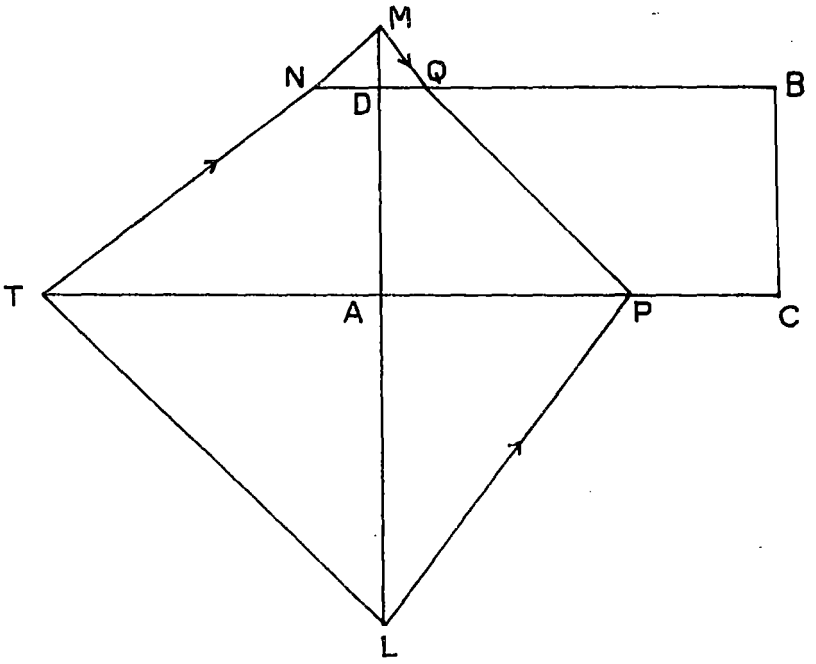


Figure – 2

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.

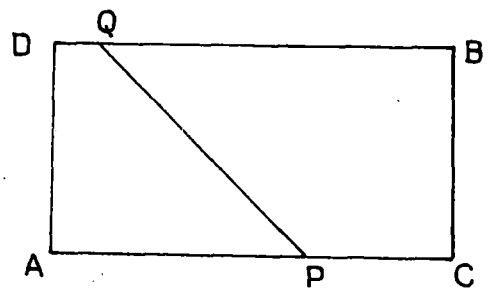


Figure - 1

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.

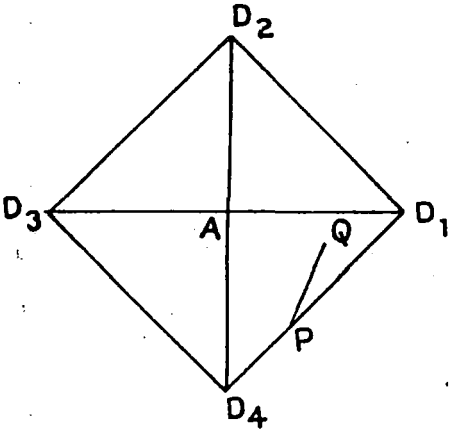


Figure - 1

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.

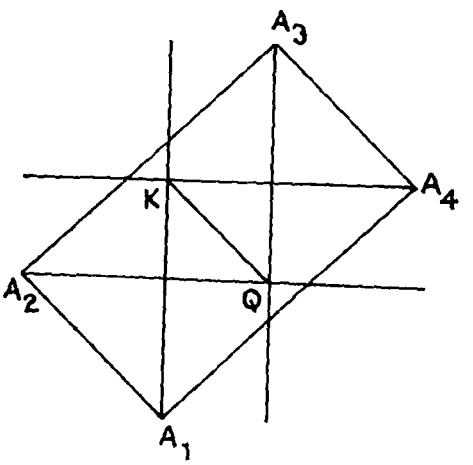


Figure - 1

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.

References

1. A. A. Aly, D. C. Kay and D. W. Litwhiler (1979), "Location Dominance on Spherical Surfaces", *Operations Research* **27**, pp. 972-981.
2. T. Aykin and A. J. G. Babu (1987), "Constrained Large-Region Multifacility Location Problems", *Journal of Operations Research Society* **38**, 241-252.
3. R. Batta, A. Ghosh and U. S. Palekar (1989), "Locating Facilities on the Manhattan Metric with Arbitrarily Shaped Barriers and Convex Forbidden Regions", *Transportation Science*, **23**, 26-36.
4. M. L. Brandeau and S. S. Chiu (1989), "An overview of Representative Problems in Location Research", *Management Science*, **35,** 645-674.
5. S. D. Brady and R. E. Rosenthal (1980), "Interactive Computer Graphical Solutions of Constrained Minimax Location problems", *A I I E. Tranctions*, **12,** 241-247.
6. N. R. Chakrabarti (1994), *Solution of certain locational problems arising in L_1 Norm, Ph.D Thesis,* North Bengal University.
7. N. R. Chakrabarty and P. K. Chaudhuri (1990) "Geometric solution of a constrained rectilinear distance minimax location problem", *Asia-Pacific Journal of Operational Research*, 7, 163-171.
8. N. R. Chakrabarty and P. K. Chaudhuri (1992), "Geometric solution to some planar constrained minimax problems involving the weighted rectilinear metric", *Asia-Pacific Journal of Operational Research,* **9**, 135 - 144.
9. R. K. Chakraborty and P. K. Chaudhuri (1981), "Note on Geometrical Solution for Some Minimax Location Problems," *Transportation Science*, **15,** 164-166.
10. R. Chen (1991), "An improved method for the solution of the problem of location on an inclined plane", *Recherche Operationnelle*, **25**. 45-53.
11. R. Courant and R. Robbins (1941), *What is Mathematics*? Oxford University Press, New York.
12. B. Dasarathy and L. J. White (1980), "A maxmin Location Problem", *Operations Research*, **28**, 1385-1401.
13. P. M. Dearing (1985), "Location problems", *Operations Research Letters*, **4,** 95 - 98.
14. U. R. Dhar and J. R. Rao (1982), "A Comparative Study of Three Norms for Facility Location Problems on Spherical Surface", *New Zealand Journal of Operational Research,*

8, 173-183.

15. U. R. Dhar and J. R. Rao (1982), "Domain Approximation Method for Solving Multifacility Location Problems on a Sphere", *Journal of Operations Research Society*, **33**, 639-645.
16. Z. Drezner (1981), "On Location Dominance on Spherical Surfaces", *Operations Research*, **29**, 1218-1219.
17. Z. Drezner (1983), "Constrained Location Problems in the Plane and on a Sphere," *IIE Transactions*, **15**, 300-304.
18. Z. Drezner and G. O. Wesolowsky (1979), "Facility Location on a Sphere," *The Journal of the Operational Research Society*, **29**, 997-1004.
19. Z. Drezner and G. O. Wesolowsky (1983), "Minimax and Maximin Facility Location Problems on a Sphere", *Naval Research Logistics Quarterly*, **30**, 305-312.
20. Z. Drezner and G. O. Wesolowsky (1980), "Single Facility L_p- distance Minimax Location, *SIAM Journal on Algebraic and Discrete Methods*, **1**, 315-321.
21. Z. Drezner (Eds.) (1995), *Facility Location a Survey of Applications and methods*, Springer-Verlag, New York.
22. D. Dutta and P. K. Chaudhuri (1989), "Geometrical Solution for a Constrained Minimax Location Problem", *Asia-Pacific Journal of Operational Research*, **6**, 148-157.
23. J. Elzinga and D. W. Hearn (1972), "Geometrical Solutions for Some Minimax Location Problems", *Transportation Science* **6**, 379- 394.
24. R. L. Francis (1971), "A Geometrical Solution Procedure for a Rectilinear Distance Minimax Location Problem", *AIIE Transactions*, **4**, 328-332.
25. R. L. Francis and J. A. White (1974), *Facility Layout and Location: An Analytic Approach*, Prentice Hall, Englewood Cliffs, N. J.
26. R. L. Francis, L. F. McGinnis and J. A. White (1992), *Facility Layout and Location: An Analytic Approach, Second Edition*, Prentice-Hall, Englewood, N. J.
27. P. Hansen, D. Peeters and J. F.Thisse (1981), "Constrained location and the Weber - Rawals Problem", *Annels of Discrete Mathematics*, **11**, 147 -166.
28. P. Hansen, D. Peeters and J. F. Thisse (1983), "Public Facility Location Models: A Selective Survey". In J. F. Thisse and H. G. Zoller (Eds). *Locational Analysis of Public Facilities*. North Holland.
29. D. W. Hearn and J. Vijay (1982), "Efficient Algorithms for the (Weighted) Minimum Circle Problem", *Operations Research*, **30**, 777-795.
30. N. S. Kambo (1991), *Mathematical Programming Techniques*, Affiliated East West Press Pvt. Ltd., New Delhi.

31 I. N. Katz and L.Cooper (1980), "Optimal Location on a Sphere", *Computers and Mathematics with Applications,* **6**, 175-196.

32. M. T. Ko, R. C. T. Lee and J. S. Chang (1990), "Rectilinear m-Centre Problem", *Naval Research Logistics*, **37**, 419-427.

33. H. W. Kuhn(1963) "Locational Problems and Mathematical Programming", *Separatum-colloquium on the Application of Mathematics to Economics*, 235-242.

34. D. T. Lee (1980), "Two-dimensional Vronoi diagrams in the L_p - metric", *Journal of the Association for Computing Machinery*, **27**, 604 - 618.

35. D. W. Litwhiler(1977), "Large Region Location Problem", *Ph.D.Thesis*, The University of Oklahoma, Norman, O.K.

36. D. W. Litwhiler and A. A. Aly (1979), "Large Region Location Problems", *Computers and Operations Research,* **6**, 1-12 .

37. D. W. Litwhiler and A. A. Aly (1981), "Minimax Facility Location Problems on the sphere", *I. E Research Report*, **81-12** (School of Industrial Engineering), University of Oklahoma, Norman, O.K.

38. R. F. Love and J. G. Morris (1975), "Solving Constrained Multifacility Location Problems involving L_p distances using convex programming", *Operations Research Technical Report,* Graduate School of Business, University of Wisconsin, Madison.

39. R. F. Love, L. G. Morris and G. O. Wesolowsky (1988), *Facilities Location: Models and Methods*, North-Holland, New York.

40. N. Megiddo (1983), "Linear-Time Algorithms for Linear Programming in R^3 and Related Problems", *SIAM J. Comput,* **12**, 759-776.

41. E. Melachrinoudis and T. P. Cullinane (1986), "Locating an undesirable facility with a minimax criterion", *European Journal of Operational Research*, **24**, 239 - 246.

42. J. G. Morris (1973), "A linear programming approach to the solution of constrained multi - facility minimax location problems where distances are rectangular", *Operational Research Quarterly*, **24,** 419-435.

43. K. P. K. Nair and R. Chandrasekaran (1971), "Optimal Location of a Single Service Centre of Certain Types", *Naval Research Logistics Quarterly*, **18**, 503-510.

44. B. J. Oommen (1987), "An Efficient Geometric Solution to the Minimum Spanning Circle Problem", *Operations Research,* **35**, 80-86.

45. M. H. Patel (1995), "Spherical Minimax Location Problem Using the Euclidean Norm: Formulation and Optimization", *Computational Optimization and Applications,* **4**, 79-90.

46. B. Pelegrin(1991), "The p-centre problem in R^n with weighted Tchebycheff norms",

Belgian Journal of Operations Research, **31**, 49-62.

47. P.M. Prenter (1975), *Splines and Variational Methods*, John Wiley and Sons.
48. M. H. Patel, D. L. Nettles and S. J. Deutsch (1993), "A Linear Programming-Based Method for Determining Whether or Not n Demand Points are on a Hemisphere", *Naval Research Logistics*, **40**, 543-552.
49. F. Rado (1988), "The Euclidean Multificility Location Problem", *Operation Research*, **36**, 485-492.
50. A. K. Sarkar and P. K. Chaudhuri (1996), "Solution of an Equiweighted Minimax Problem on a Hemisphere", *Computational Optimization and Applications*, **6**, 73-82.
51. C. H. Scott, B. A. Murtagh and E.Sirri (1985), "Solution of Constrained Minimax Location with Euclidean Distance via Conjugate Duality", *New Zealand Operational Research*, **13**, 61- 67.
52. M. I. Shamos and D. Hoey (1975), "Closest-point problems", *Sixteenth Annual IEEE symposium on Foundations of Computer Science*, 151-162.
53. A. Tamir (1992), "On the complexity of some classes of location problems", *Transportation Science*, **26**, 352-354
54. I. Todhunter and J. G. Leathem (1960), "Spherical Trigonometry," *Macmillian & Co. Ltd.*.
55. W. H. Tsai, M. S. Chern and T. M. Lin (1991), "An Algorithm for Determining Whether m given demand points are on a Hemisphere or Not", *Transportation Science*, **25**, 91-97.
56. A. Weber (1909), "Uber den Standort der Industrien", Translated as *Alfred Weber's Theory of the Location of Industries* by C. J. Friedrich, Chicago (1929).
57. R. E. Wendell, A. P. Hurter and R. J. Lowe (1977), "Efficient Points in Location Problems", *AIIE Transactions*, **9**, 238-246.
58. G. O. Wesolowsky (1982), "Location Problems on a Sphere", *Regional Science Urban Economics*, **12**, 495-508.
59. G. O. Wesolowsky (1972), "Rectilangular Distance Location under the Minimax Optimality Criterion", *Transportation Science*, **6**, 103-113.
60. G. Xue and C. Wang (1994), "The Euclidean Facilities Location Problem", In D. Z. Du and J. Sun (Eds.). *Advances in Optimization and Approximation*, Kluwer Academic, Boston, 313-331.