A Squared-Euclidean Distance Location-Allocation Problem

by

Cihan H. Tuncbilek

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Industrial Engineering and Operations Research

APPROVED:

[illegible]

Dr. Hanif D. Sherali, Chairman

[illegible]

Dr. C. Patrick Koelling

[illegible]

Dr. Anthony A. Trani

June, 1990

Blacksburg, Virginia

c.2

LD
5655
V855
1990
T874
c.2

A Squared-Euclidean Distance Location-Allocation Problem

by

Cihan H. Tuncbilek

Dr. Hanif D. Sherali, Chairman

Industrial Engineering and Operations Research

(ABSTRACT)

This thesis is concerned with the analysis of a squared-Euclidean distance location-allocation problem with balanced transportation constraints, where the costs are directly proportional to distances and the amount shipped. The problem is shown to be equivalent to maximizing a convex, quadratic function subject to transportation constraints. A branch and bound algorithm is developed that utilizes a specialized, tight, linear programming representation to compute strong upper bounds. These bounds are shown to substantially dominate several other upper bounds that are derived using standard techniques, to an extent which significantly increases the size of problems solvable within a reasonable effort. The special structure of the transportation constraints is used to derive a partitioning scheme, and this structure is further exploited via suitable logical tests which tighten the bounds implied on the transportation flows by the branching restrictions. The transportation structure is also used to generate additional cut-set inequalities based on a cycle prevention method which preserves a forest graph for any partial solution. Results of the computational experiments, and a discussion of possible extensions are also presented.

Acknowledgements

I would like to thank my advisor, Dr. Hanif D. Sherali, for his patience and input at each stage of this work. Without his support, this thesis would not have been completed. I would also like to thank Dr. C. Patrick Koelling and Dr. Antonio A. Trani for serving on the thesis committee and for their helpful suggestions.

I would finally like to acknowledge the National Science Foundation for their support under Grant No. ECS-8807090.

Table of Contents

List of Tables

Chapter I

Introduction

Location-allocation problems are concerned with determining the location and allocation of a number of supply centers, given the demand requirements and locations of a set of customers, so as to minimize the total location and transportation costs.

In this research we will consider a specific location-allocation problem : Given the discrete location points of N customers on a continuous plane, and their associated demands, the problem seeks to determine the location of M supply centers with known capacities so as to satisfy the demand requirements of the customers at minimal total cost. More specifically, the decisions are where to locate the M supply centers, and how much shipment to send from each supply center to each customer, so that the total cost, which is assumed to be directly proportional to the amount shipped and the squared-Euclidean distance over which this shipment occurs, is minimized. We will also assume a balanced situation in which total supply is equal to the total demand. Mathematically, this problem can be formulated as follows.

$$\begin{aligned}
\text{P}: \ \text{minimize} \ & \sum_{i=1}^{M}\sum_{j=1}^{N} w_{\text{ij}}\left[(x_i - a_j)^2 + (y_i - b_j)^2\right] \\
\text{subject to} \ & \sum_{j=1}^{N} w_{\text{ij}} = s_i \qquad i = 1,\ldots,M \\
& \sum_{i=1}^{M} w_{\text{ij}} = d_j \qquad j = 1,\ldots,N \\
& w_{\text{ij}} \geq 0 \qquad i = 1,\ldots,M, \quad j = 1,\ldots,N
\end{aligned} \tag{1}$$

where

M: number of supply points,

N: number of demand points,

(a_j, b_j): location of demand point j,

s_i: supply of source point i,

d_j: demand of destination point j,

and where the decision variables are:

(x_i, y_i): location of supply point i,

w_{ij}: amount shipped from supply point i to demand point j.

The above problem is a squared Euclidean distance location-allocation problem in that for a fixed set of allocations $w \equiv (w_{\text{ij}},\ i = 1,\ldots, M,\ j = 1,\ldots, N,)$ the problem reduces to a pure location problem having costs proportional to the square of Euclidean distance separating the supply and the demand points (see Francis and White, 1974), while for a fixed set of locations (x_i, y_i), $i = 1,\ldots, M$, the problem reduces to a simple transportation/allocation problem. However, the combined location-allocation problem is a difficult nonconvex programming problem, which is NP-hard even with all the demand points located along a straight line as follows from Sherali and Nordai (1988).

Note that if we fix $w = w^*$ in Problem P, the unconstrained minimum of the strictly convex objective function is readily obtained at the solution

$$x_i^* = \frac{\sum_{j=1}^{N} w_{ij}^* a_j}{\sum_{j=1}^{N} w_{ij}^*}, \text{ and } y_i^* = \frac{\sum_{j=1}^{N} w_{ij}^* b_j}{\sum_{j=1}^{N} w_{ij}^*}, \qquad i = 1,\ldots, M. \tag{2}$$

Using (1) and (2), we can project Problem P onto the space of the w -variables by substituting

$$x_i = \frac{\sum_{j=1}^{N} w_{ij} a_j}{s_i}, \text{ and } y_i = \frac{\sum_{j=1}^{N} w_{ij} b_j}{s_i}, \qquad i = 1,\ldots, M, \tag{3}$$

to obtain the following equivalent problem.

$$\text{P1: minimize } \sum_{i=1}^{M}\sum_{j=1}^{N} \frac{w_{ij}}{s_i^2}\left[\left(\sum_{k=1}^{N} w_{ik}a_k - s_i a_j\right)^2 + \left(\sum_{k=1}^{N} w_{ik}b_k - s_i b_j\right)^2\right]$$
$$\text{subject to } w \in W$$

where $w \equiv [w_{1,1}, w_{1,2},\ldots, w_{M,N}]^t$, and W is the transportation constraint set defined in (1). Simplifying the objective function of P1, we can rewrite it as

$$\sum_{i=1}^{M}\sum_{j=1}^{N} \frac{w_{ij}}{s_i^2}\left[\left(\sum_{k=1}^{N} w_{ik}a_k\right)^2 + a_j^2 s_i^2 - (2a_j s_i)\sum_{k=1}^{N} w_{ik}a_k + \left(\sum_{k=1}^{N} w_{ik}b_k\right)^2 + b_j^2 s_i^2 - (2b_j s_i)\sum_{k=1}^{N} w_{ik}b_k\right]$$

$$= \sum_{i=1}^{M} \frac{\left(\sum_{k=1}^{N} w_{ik} a_k\right)^2}{s_i^2} \sum_{j=1}^{N} w_{\mathrm{ij}} + \sum_{i=1}^{M} \frac{\left(\sum_{k=1}^{N} w_{ik} b_k\right)^2}{s_i^2} \sum_{j=1}^{N} w_{\mathrm{ij}} + \sum_{j=1}^{N} a_j^2 \sum_{i=1}^{M} w_{\mathrm{ij}} + \sum_{j=1}^{N} b_j^2 \sum_{i=1}^{M} w_{\mathrm{ij}}$$
$$- 2 \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{w_{\mathrm{ij}} a_j}{s_i} \left(\sum_{k=1}^{N} w_{ik} a_k\right) - 2 \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{w_{\mathrm{ij}} b_j}{s_i} \left(\sum_{k=1}^{N} w_{ik} b_k\right)$$

$$= \sum_{i=1}^{M} \frac{1}{s_i} \left[\left(\sum_{k=1}^{N} w_{ik} a_k\right)^2 + \left(\sum_{k=1}^{N} w_{ik} b_k\right)^2 \right] + \sum_{j=1}^{N} (a_j^2 + b_j^2) d_j$$
$$- 2 \sum_{i=1}^{M} \frac{1}{s_i} \left[\left(\sum_{k=1}^{N} w_{ik} a_k\right)^2 + \left(\sum_{k=1}^{N} w_{ik} b_k\right)^2 \right]$$

Let $w_i \equiv [w_{i1}, \ldots, w_{iN}]^t \ \forall\ i = 1, \ldots, M$, $a \equiv [a_1, \ldots, a_N]^t$, and $b \equiv [b_1, \ldots, b_N]^t$. Consequently, Problem P1 is equivalent to the following convex maximization problem.

$$\begin{aligned} \text{P2}: \ \ &\text{maximize} \ \ z = \sum_{i=1}^{M} \frac{1}{s_i} \left[(w_i^t a)^2 + (w_i^t b)^2 \right] \equiv \sum_{i=1}^{M} \frac{1}{s_i} (w_i^t T w_i) \equiv w^t G\, w \\ &\text{subject to} \ \ w \in W \end{aligned} \tag{4}$$

where

$$(w_i^t a)^2 + (w_i^t b)^2 = w_i^t a\, a^t w_i + w_i^t b\, b^t w_i = w_i^t (a\, a^t + b\, b^t)\, w_i \equiv w_i^t T w_i \quad \forall\ i = 1, \ldots, M. \tag{5}$$

The aim of this research effort is to develop a solution procedure for Problem P2. The proposed procedure is a branch and bound algorithm which exploits the special structure of the transportation constraints, and employs certain novel constructs in the partitioning scheme and in deriving tight upper bounds.

This study is organized as follows. In Chapter 2, we present a literature review on concave minimization problems, and on location-allocation problems. Chapter 3 then develops various upper bounding schemes based on the objective function of Problem P2 for use in the branch and bound algorithm. Additionally, a special upper bounding scheme is derived by constructing a particular tight linear programming representation for Problem P2. All the proposed upper bounding schemes, but one, are incorporated in a branch and bound framework in Chapter 4, and computational experience is given in Chapter 5 along with a discussion of several implemented modifications. Chapter 6 presents conclusions and suggestions for further research.

Chapter II

Literature Review

Due to the nature of our problem, the literature review discussion is grouped under two topics: concave minimization problems and location-allocation problems.

2.1 Concave Minimization Problems

A substantial amount of research has been accomplished on concave minimization problems over the last two decades. Our discussion on this subject will be approach-oriented. Extensive bibliographic surveys can be found in Pardalos and Rosen (1986), and - in a more extended form - in Pardalos and Rosen (1987). Consider the following concave minimization problem:

$$\begin{array}{lll} \text{P3}: & \text{minimize} & f(x) \\ & \text{subject to} & x \in D \subseteq R^n \end{array} \tag{6}$$

where $f(x)$ is a concave function defined on R^n, and D is a nonempty, compact polyhedral set. Problem P3 may have many local optimal solutions. However, it is well known that the global optimum is attained at an extreme point of D. Hence, one approach for this problem is based on enumerating the extreme points of D. The objective of the enumerative methods by ranking the extreme points is to find the global minimum, hopefully before having totally enumerated the extreme points of D. Cabot and Francis (1970) apply Murty's (1968) extreme point ranking approach by using a linear underestimating function to solve P3 when $f(\cdot)$ is quadratic. Later, Cabot (1974) also used Tuy cuts to cut off local optimal solutions in this type of a scheme. Taha (1973) utilizes Glover's (1973) cutting plane method to rank the extreme points.

The computational efficiency of algorithms of this type is very adversely affected by degeneracy. McKeown (1978) gives a computational survey of the extreme point ranking algorithms for certain types of problems, one of which is concerned with minimizing a concave, quadratic function subject to transportation constraints. In the randomly generated test problems the objective function is taken as the sum of a linear and a nonlinear component. He shows that as the nonlinearity of the objective function increases, the efficiency of the algorithms decreases.

Partitioning of the feasible region and the use of outer approximations have become two major approaches for solving P3 and its extensions. Before describing some of the algorithms based on these approaches, let us briefly refer to two bounding methods.

Bounding method $B1$ is a direct result of the extreme point optimality property of the problem P3. Let v_i, $i = 1,\ldots, m$, be the vertices of a bounded polyhedron $S \subseteq R^n$, $S \supseteq D$. Since any $x \in D$ can be expressed as a convex combination of v_i, $i = 1,\ldots, m$, we obtain

$$\min_{i=1,\ldots,m} f(v_i) \leq f(x), \quad \forall x \in D. \tag{7}$$

A second bounding method $B2$ is based on the convex envelope concept. Let $L(x)$ be the convex envelope of $f(x)$ over S . Then $L(x)$ is the tightest convex underestimator of $f(x)$ over S . Moreover, $\min\{L(x) \,:\, x \in S\} = \min\{f(x) \,:\, x \in S\}$. $L(x)$ can be expressed as

$$L(x) = \min\left\{ \sum_{i=1}^{m} \alpha_i f(v_i) \,:\, \sum_{i=1}^{m} \alpha_i v_i = x,\ \sum_{i=1}^{m} \alpha_i = 1,\ \alpha_i \geq 0,\ i = 1,\ldots, m \right\}. \tag{8}$$

Note that $L(v_i) = f(v_i)$, $i = 1,\dots, m$. If S is a simplex, then $m = n + 1$, and $L(x) = a^t x + b$, $a \in R^n$, $b \in R^1$, is a linear function, where (a, b) is the unique solution of the following $n + 1$ linear equations,

$$a^t v_i + b = f(v_i), \quad i = 1,\dots, n+1. \tag{9}$$

For related theorems and proofs see Pardalos and Rosen (1987).

In partitioning approaches the feasible region is usually contained in a region with a simple structure, e.g. a cone, a simplex or a rectangular region, which can be used conveniently in partitioning and bounding procedures. Such a method was first used by Tuy (1964) in a cone splitting algorithm to solve problem P3. Here, an initial cone $M_0 \supset D$ is constructed by extending the rays from a local minimum corner point $x_0 \in D$ through n neighboring vertices of x_0 (nondegeneracy is assumed). On each ray $i \in \{1,\dots, n\}$ the most remote point y_i from x_0 is located such that $f(x) \geq f(\hat{x})$ for all x on the line from x_0 to y_i, where $\hat{x}$ is the current best solution. At the initial step, $\hat{x}$ is taken as x_0 . If $f(x) \geq f(\hat{x})$ for all points along the ray i, then y_i is picked at a sufficiently large distance from x_0. Let C_0 be the cutting plane defined by the points y_i, $i = 1,\dots, n$. If C_0 does not trim off D, the algorithm stops. Otherwise, the most remote vertex $x_q \in D$ from C_0, that is on the opposite side of C_0 from x_0, is found by solving an auxiliary linear program. M_0 is then split into less than or equal to n subcones by extending rays from x_0 through x_q and through $n - 1$ points from $\{y_i,\dots, y_n\}$ such that x_0, x_q, and the $n - 1$ points from $\{y_i,\dots, y_n\}$ are affinely independent. If the cutting plane that is generated for a subcone M_k does not trim off $D \cap M_k$, then M_k is eliminated. Cone splitting continues for the remaining subcones until there is no unexplored subcone left.

Zwart (1973) shows that the point x_q related to a subcone M_k may not be contained in M_k, which results in cycling. To prevent cycling, Zwart (1974) modifies Tuy's algorithm by introducing additional constraints for each auxiliary problem. The idea of his algorithm is to try and cover D with simplices on which $f(x) \geq f(x^*)$, where x^* is the global minimum solution of P3. Each simplex is formed by truncating a cone by a Tuy cut. Also, the apex of the subcones changes as a better local minimum solution is found. In both algorithms, the bounding method $B1$ has been implicitly used. Krynski (1979) gives another convergent Tuy-type algorithm, where the auxiliary problem is no longer an ordinary linear program. He also proposes two methods to handle the case where x_0 is degenerate.

Thoai and Tuy (1980) propose a branch and bound cone splitting algorithm. Let C_k be a Tuy cut generated for the subcone M_k, and let T_0 be the hyperplane passing through the n neighboring vertices of the apex of the initial cone M_0. If C_k trims off $D \cap M_k$ then C_k is moved farther away from the apex of M_k, until it no longer cuts $D \cap M_k$. Now, the bounding method $B1$ can be used to give a lower bound $\mu(M_k)$. If $\mu(M_k) \geq f(\hat{x})$, then the subcone M_k is eliminated. Otherwise, M_k is split by bisecting the longest edge of the $(n-1)$-simplex spanned by the points where the edges of M_k meet T_0. This algorithm is extended by Tuy, Thieu and Thai (1985) to cover the case where D is a compact, convex set.

Horst (1976a) considers the general nonconvex programming problem, where $f(x)$ is continuous, and D is a compact set. He gives a theoretical branch and bound algorithm, which is based on a partitioning of the feasible region by using general compact partitions. When this algorithm is applied to concave programming with a compact, convex feasible region, simplex partitions and the bounding method $B2$ are used (see also Horst (1976b)). At any stage in this process, a lower bound is calculated over $S \cap D$, for each simplex partition S such that $S \cap D \neq \emptyset$. Rules for the deletion of the partition set S such that $S \cap D = \emptyset$ are provided by Horst (1988). Suppose, $\mu_k(S_k)$ is the smallest lower bound attained at $x_k \in S_k \cap D$. If $\mu_k(x_k) = f(x_k)$, the algorithm stops. Otherwise, S_k is partitioned into two subsimplices. Let $[v_p, v_q]$, v_p, $v_q \in V_k$, be the longest edge of S_k, where V_k is the set of vertices of S_k, and let x_l be the midpoint of $[v_p, v_q]$. Sets of the vertices of the two subsimplices are given by $(V_k - \{v_p\}) \cup \{x_l\}$, and $(V_k - \{v_q\}) \cup \{x_l\}$. If $\{x = (x_1, \ldots, x_n) \in R^n : x_i \geq 0 \ , i = 1, \ldots, n\} \subseteq D$ then the initial simplex that contains D can be determined by the points 0 (the origin), and $z_0 e_i$, $i = 1, \ldots, n$, where $z_0 = \max\{\sum_{i=1}^{n} x_i : x \in D\}$, and where e_i is the i^{th} unit coordinate vector. Later, Benson (1982) slightly generalized Horst's algorithm, and gave a prototype branch and bound algorithm. Notice that in Horst's algorithm, the upper bound $f(x_k)$ for the current partition is not necessarily the best objective value obtained thus far. Also, a simplex partition that has been shown not to contain an optimal solution is not eliminated. Horst (1986) includes these ideas in the previous prototype branch and bound algorithm. To guarantee convergence, at least after finitely many steps, the partition which has the smallest lower bound value is chosen as the next set to partition. Finiteness can also be satisfied by other strategies, such as choosing the oldest partition, or the one that gives the largest value for a measure related to the size of the partitions.

Using some of the ideas from Horst's algorithm, Benson (1985) presents an algorithm for solving the problem P3. One major difference is that the convex envelope of $f(x)$ over a simplex partition S is minimized over D instead of over $S \cap D$. The advantage of this change is that all the subproblems have the same constraint set. On the other hand, none of the simplex partitions can be eliminated from further consideration, since an optimum solution for P3 may lie in any of them.

Rosen (1983) uses rectangular partition sets in a concave quadratic programming algorithm. A rectangular region $\overline{R} \supseteq D$ is constructed by moving $2n$ hyperplanes as far as possible on D from the global maximum solution along selected orthogonal directions. The special structure of $\overline{R}$ and the objective function permits the calculation of a lower bound without explicitly finding the vertices of $\overline{R}$. By using surfaces parallel to those of $\overline{R}$, another rectangular region $\hat{R}$ is constructed such that $\hat{R}$ is inscribed with a maximum volume in the ellipsoid contour of $f(x)$ at $f(x) = f(\hat{x})$, where $\hat{x}$ is the current best solution. Since $f(x) \geq f(\hat{x})$, $\forall x \in \hat{R}$, $\hat{R} \cap D$ can be eliminated. Now, the remaining feasible polytopes, each of which is cut from D by a surface of $\hat{R}$, can be handled separately. Using a similar approach, Kalantari and Rosen (1987) give another algorithm. Here, the construction of $\overline{R}$ is accomplished by solving the $2n$ linear programs, $\max\limits_{x \in D} u_i Qx$, and $\min\limits_{x \in D} u_i Qx$, $i = 1,\ldots, n$, where Q is the Hessian of $f(x)$, and u_i, $i = 1,\ldots, n$ are Q-conjugate directions. The convex envelope of $f(x)$ over $\overline{R}$ is linear, and a lower bound is calculated by minimizing it over D. To partition the feasible region, a hyperplane parallel to a surface of $\overline{R}$ bisects $\hat{R}$, where $\hat{R}$ is defined as above, into two parallelpipeds. For a general problem where some y variables appear only linearly in the objective function in addition to $f(x)$ and where the feasible region is defined on the (x, y)-space, the feasible region is projected on the x-space.

In the outer approximation approach, a region containing D is successively refined to have a tighter approximation with the motivation of solving P3 before the containing region reduces to D. Falk and Hoffman (1976) take the initial containing region as a simplex. This simplex is constructed by truncating the n rays that extend from a local minimum corner point of D by a hyperplane, if possible by a nonbinding constraint of D. At some iteration k, let S_k be the containing polytope with vertices $v_1, \ldots, v_m$. Using the bounding method $B2$ the lower bound $\sum_{i \in A_k} \alpha_i f(v_i)$, where $A_k = \{i : \alpha_i > 0, \ i = 1, \ldots, n\}$, is obtained. If $v_i \in D$, $\forall i \in A_k$, then the algorithm stops. Otherwise, the vertex $v_p \notin D$ that contributes most to the lower bound is determined. Activating a constraint that is violated at v_p tightens the containing region. After finding the newly generated vertices, the algorithm continues with the bounding operation. Later, Falk and Hoffman (1986) give another algorithm, in which the redundant constraints do not affect the computation. If the feasible region D is defined as $D = \{x : Ax \leq b\}$, then D is a face of the $n+1$ dimensional polytope $D' = \{x, y : Ax + y \leq b\}$. Let (x_0, y_0) be the solution to the problem $\max\{y : (y, x) \in D'\}$ (nondegeneracy and uniqueness of the optimum are assumed). The points where the rays extended from (x_0, y_0) through its n + 1 neighboring vertices intersect the plane $y = 0$ give the vertices of the initial containing region. Suppose that the minimum of $f(x)$ over the vertices of the containing region is attained at v_p. If $v_p \in D$ then problem P3 is solved. Otherwise, the most promising neighboring vertex (y_p, x_p) of (y_0, x_0) is found. Rays emanating from (y_p, x_p) are extended through some of its unencountered neighboring vertices that have a smaller y-coordinate value than y_p, until they pierce through the plane $y = 0$. Then, these intersection points are used to refine the containing region.

Horst, Thoai, and de Vries (1988) give a method for determining the vertices of the refined containing region, as it is computationally the most expensive part of the outer approximation algorithms. Outer approximation approaches have also been used by Carillo (1977), Hoffman (1981), and by Thieu, Tam, and Ban (1983), where the latter two papers consider the case where D is a general compact, convex set.

Next, we will describe three other approaches for various types of concave minimization problems. In the first, Konno (1976b) shows that for quadratic functions $f(x)$, problem P3 is equivalent to an associated bilinear program. By using this result he develops an iterative procedure to generate deeper cuts than Tuy cuts. He also presents a cutting plane algorithm, which uses the results given by Konno (1976a) for bilinear programming. The idea of the algorithm is to find local optimal solutions, and cut off suitable regions that contain them until there is no feasible region left.

The second type of approach is due to Mukhamediev (1982) who presents an inner approximation algorithm. A sequence of polyhedral sets are constructed such that each set is a subset of the next one, and the current best solution gives the global minimum over the current polyhedral set. Let Z_k be such a set, that is defined by some inequality constraints. The set Z_k is relaxed by adding a tolerance factor δ to the right hand sides of the constraints that define Z_k. Let Z_k^δ be the relaxed polyhedral set. If $D \subseteq Z_k^\delta$ and the lower bound calculated over Z_k^δ is less than some ε away from the current best solution, $\varepsilon \geq 0$, the algorithm stops. Otherwise, either $D \subseteq Z_k^\delta$, but the lower bound associated with Z_k^δ is more than ε away from the current best solution, or $D \not\subseteq Z_k^\delta$. In the first case, Z_k^δ is tightened by reducing the tolerance factor, and a new lower bound is calculated over the tightened Z_k^δ. In the second case, there exists a point $x' \in D - Z_k^\delta$, and $\mathrm{conv}(Z_k \bigcup \{x'\})$ determines the polyhedral set $Z_{k+1} \supseteq Z_k$. Then, the relaxed set $Z_{k+1}^\delta \supseteq Z_{k+1}$ is constructed, and the procedure continues by finding a lower bound over Z_{k+1}^δ as above.

A concave function of a single variable can be approximated by a piecewise linear function using a 0-1 mixed integer program. Piecewise linear functions can be generated through an interpolation of the values of the original function evaluated at adjacent grid points. If x is not a grid point, then only two adjacent grid points can be used to represent x. Hence, a third approach is to use general mixed integer programming techniques as an alternative solution method for nonconvex programs with linear constraints and separable objective functions. But to take advantage of the structure of these problems, Beale and Tomlin (1970) suggest to use special ordered sets. A Type $S1$ special ordered set is a set of variables in which only one variable can be nonzero. A Type $S2$ special ordered set is an ordered set of variables in which only two adjacent variables can be nonzero. Beale and Forrest (1976) extend the concept of type $S2$ sets, and use an automatic interpolation procedure to generate grid points as the algorithm proceeds, instead of fixing the grid points initially.

2.2 Location-Allocation Problems

A location-allocation problem can be stated as follows: Given the location or distribution of a set of customers (destinations) and their associated demands for a given product or service, determine the optimum location of a number of service facilities (sources) and the allocation of products or services from facilities to customers, so as to minimize total location and transportation costs. In the vast amount of literature, heuristic and exact solution procedures have been proposed for different cases of location-allocation problems and their extensions. According to the restrictions on the potential sites at which supply centers can be located, location-allocation problems can be divided into three groups: continuous location-allocation problems, discrete location-allocation problems, and location-allocation problems on networks.

2.2.1 Continuous Location-Allocation Problems: In continuous location-allocation problems, sources can be located at any point in a continuous plane, while the location of the destinations are fixed at some discrete points on the plane. Let $d(X_i, P_j)$ represent the distance between the locations of source i and destination j. Then the continuous location-allocation problem for M sources and N destinations can be stated as follows:

$$\begin{aligned}
\text{LAP}: \quad & \text{minimize} \quad \sum_{i=1}^{M}\sum_{j=1}^{N} c_{ij} w_{ij} d(X_i, P_j) \\
& \text{subject to} \quad \sum_{j=1}^{N} w_{ij} \leq s_i \qquad i = 1,\ldots, M \\
& \qquad\qquad\quad \sum_{i=1}^{M} w_{ij} = d_j \qquad j = 1,\ldots, N \\
& \qquad\qquad\quad w_{ij} \geq 0 \qquad i = 1,\ldots, M; \quad j = 1,\ldots, N
\end{aligned}$$

where the decision variables are:

w_{ij}: amount shipped from source i to destination j,

$X_i = (x_i, y_i)$: location of source i, and where the data is symbolized by:

s_i: capacity of source i,

d_j: demand of destination j,

c_{ij}: cost of shipping one unit from source i to destination j through a unit distance,

$P_j = (a_j, b_j)$: location of destination j.

Notice that given the locations of the sources, LAP reduces to an ordinary transportation problem, and given the transportation flows, it becomes a pure location problem.

Cooper (1963) first formulated the uncapacitated version of LAP with Euclidean distances. For this problem, $s_i = +\infty$, $i = 1,\ldots, M$ and $d(X_i, P_j) = \sqrt{(x_i - a_j)^2 + (y_i - b_j)^2}$. Since there is no restriction on the capacity of the sources, each customer is served by a single source. Given the assignment of destinations to each source, optimum locations are found by using the first order optimality conditions. Such an enumerative exact solution procedure considers all different combinations of destination-to-source assignments. This method has been shown to be computationally impractical. Based on the ideas introduced in his previous paper, Cooper [1964, 1967] gives several heuristic solution procedures, which have the basic idea of alternately solving the location and the allocation subproblems. Mainly two approaches are used. In the first one, source locations are restricted to a subset of destination sites. In the second one, a subset of destinations are assigned to a single source, and then for each source, an exact location method is used to find an optimum single source location. Elion, et al. (1971) computationally improves one of Cooper's heuristic procedures. Kuenne and Soland (1972) give an alternative heuristic procedure and a branch and bound type of exact solution procedure. At some iteration of the branch and bound algorithm, assigning a subset of destinations to the sources gives a partial solution. At the branching step a 'free' destination is included in the partial solution. Lower bound calculations are based on an application of the triangle inequality. Love and Juel (1982) consider the uncapacitated location-allocation problem with l_p distances. This problem has been shown to be equivalent to a concave minimization problem. They also give several heuristic solution procedures, which differ from each other in the manner in which they perturb a given local optimum solution. A local optimum solution is defined as one for which the current locations are optimal when the allocations are fixed, and vice versa. In the perturbation step, current allocations are changed to get a better local optimum solution. Chen (1984) shows that Cooper's exact solution procedure is related to a steepest descent approach, and thereby proposes a modified step length to improve the iterative process.

For the capacitated case, Cooper (1972) gives an enumerative type of exact algorithm, and two heuristic solution procedures. Recognizing that an optimum solution occurs at an extreme point of the feasible region, all basic feasible transportation flows are enumerated, and for each of them, the optimum location is determined. The first heuristic method is based on Cooper's previous work for the uncapacitated case. Some of the destination sites are taken as the initial location of the sources. With the given locations a transportation problem is solved to find the optimum allocation. Then new optimal locations of the sources are determined with respect to these allocations. This procedure continues until there is no further improvement. A second heuristic is more complicated. First, ignoring the capacity constraints, the corresponding uncapacitated problem is solved by using one of the heuristics. If some capacity constraints are violated, the sources with a surplus capacity are utilized to serve the destinations with unsatisfied demand. During this operation, appropriate source and destination points are selected by considering the Euclidean distance between the interacting points, so as to keep the contribution of the allocation changes to the objective function small. Once the new allocations are determined, the location and allocation optimization procedures are applied alternatingly as above. Another heuristic method is given by Murtagh and Niwattisyawong (1982), using the commercial nonlinear programming package MINOS along with a reasonable starting solution. Avriel (1980) develops a geometric programming approach for this problem. by exploiting the special structure of the location-allocation subproblems, and Selim (1979) proposes a biconvex programming cutting plane technique to solve this problem exactly.

In the case of the rectilinear norm, $d(X_i, P_j) = |x_i - a_j| + |y_i - b_j|$, problem LAP becomes a nonconvex bilinear programming problem (see Vaish (1974)). Love and Morris (1975) have developed an exact solution procedure for the uncapacitated version of this problem. The procedure consists of using a set reduction algorithm to reduce the possible solution set, and the problem is shown to be equivalent to the p-median problem on a weighted connected graph - which will be explained in Section 2.2.3. For the capacitated problem, Sherali and Shetty (1977) have developed a convergent polar cutting plane algorithm, while Shetty and Sherali (1980) improve this algorithm by using deeper polar cutting planes, based on negative edge extensions, and they provide a more efficient computational implementation. This procedure has actually been developed for a more general problem which permits interactions between new facilities and involves multiple commodities. Except for general concepts, however, none of the foregoing methods for the various continuous location-allocation problems on a plane are directly suitable or applicable to our problem.

2.2.2 Discrete Location-Allocation Problems: In discrete location-allocation problems, which are also known as fixed-charge location problems, the location of supply centers are restricted to a fixed, finite set of potential sites. The 'fixed charge' is the cost associated with opening a supply center at a given location. Problems of this type and their extentions have received a considerable amount of attention in the literature. Since we are not directly concerned with this class of problems, we refer the reader to the reviews given by Aikens (1985), and Hansen, et al. (1987), for further information.

2.2.3 Location-Allocation Problems on Networks: In location-allocation problems on networks the location of the supply centers are restricted to be on a network. Let $G(N,A)$ be an n-vertex undirected network with weights (demands) h_i attached to its vertices $V_i \in N$ and weights (distances) w_j associated with its arcs. Let $X_p = \{x_1, \ldots, x_p\}$ represent the decision variables, namely, the locations of the p facilities which are to be located on G, and let $d(V_i, X_p) = \min\{d(V_i, x_1), \ldots, d(V_i, x_p)\}$ measure the shortest path distances in G .

The objective of the (absolute) p-median problem is to locate p facilities on the network G so as to minimize the sum of the weighted distances to each vertex from its nearest facility, that is, to

$$\underset{X_p \text{ on } G}{\text{minimize}} \quad f(X_p) = \sum_{i=1}^{n} h_i \, d(V_i, X_p). \tag{10}$$

Note that when $p \geq 2$, this problem may be viewed as a location-allocation problem since the location of facilities determines the allocation of its services to meet the nodal demands.

The objective of the (absolute) p-center problem is to locate p facilities on G so as to minimize the maximum weighted distance between a vertex and its nearest facility. That is, to

$$\underset{X_p \text{ on } G}{\text{minimize}} \; f(X_p) = \max_{1 \leq i \leq n} h_i \, d(V_i, X_p). \tag{11}$$

Due to the nature of the objective functions in (10) and (11), the former problem is also referred to as a minisum network location problem, while the latter is called a minimax network location problem. This subject has been addressed in detail by Minieka (1980), Halpern and Maimon (1980), Kariv and Hakimi (1979a, b), Handler and Mirchandani (1979), Hansen, et al. (1987). For a more extensive tutorial on location problems on networks, the reader is referred to Tansel, et al. (1983a, b).

Chapter III

Upper Bounding Schemes

This section presents various upper bounding schemes for problem P2. These schemes can be divided into two groups; upper bounds derived from the objective function of P2 and linearization based upper bounds.

3.1 Objective Function Based Upper Bounds

The special structure of the objective function of Problem P2 yields several appealing upper bounding functions. Four of them are developed below, and designated as $z_{UBt}(w)$, for $t = 1, 2, 3, 4$. By maximizing these functions over $w \in W$ or over any restrictions thereof within a branch and bound algorithm, we can derive various upper bounds on the problem.

3.1.1. Upper Bounding Function $z_{UB1}(\cdot)$: Let $T = aa^t + bb^t$, as defined in (5). Then the elements of T are given by $T_{ii} = (a_i^2 + b_i^2)$ and $T_{ij} = (a_i a_j + b_i b_j) \leq \sqrt{a_i^2 + b_i^2}\ \sqrt{a_j^2 + b_j^2} = T'_{ij}$, and so, $0 \leq T \leq gg^t$, where $g = \left[\sqrt{a_1^2 + b_1^2}, \ldots, \sqrt{a_N^2 + b_N^2}\,\right]^t$. Hence,

$$w_i^t(aa^t + bb^t)w_i = (w_i^t gg^t w_i) - \sum_{j=1}^{N} \sum_{\substack{k=1 \\ k\neq j}}^{N} (T'_{jk} - T_{jk}) w_{ij} w_{ik}$$
$$= (w_i^t g)^2 - \sum_{j=1}^{N} w_{ij} \left[\sum_{\substack{k=1 \\ k\neq j}}^{N} (T'_{jk} - T_{jk}) w_{ik} \right] \tag{12}$$

Let $\Delta_i = \max\{w_i^t g \ : \ w \in W\}$, $i = 1, \ldots, M$, and define $\lambda_{ij} = \min\ \{\sum_{\substack{k=1 \\ k\neq j}}^{N} (T'_{jk} - T_{jk}) w_{ik} \ : \ w \in W\} \ \geq 0$, $i = 1, \ldots, M$, $j = 1, \ldots, N$. This gives from (4) and (12), that the objective function of Problem P2 is bounded by

$$z \leq z_{UB1}(w) = \sum_{i=1}^{M} \frac{\Delta_i}{s_i}(w_i^t g) - \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{\lambda_{ij}}{s_i} w_{ij}. \tag{13}$$

3.1.2. Upper Bounding Function $z_{UB2}(\cdot)$*:* Define $\theta_i = \max\{w_i^t a \ : \ w \in W\}$, and $\phi_i = \max\{w_i^t b \ : \ w \in W\}$, $i = 1, \ldots, M$. Hence, quite evidently, from (4),

$$z \leq z_{UB2}(w) = \sum_{i=1}^{M} \frac{1}{s_i} \left[\theta_i (w_i^t a) + \phi_i (w_i^t b) \right] . \tag{14}$$

3.1.3. Upper Bounding Function $z_{UB3}(\cdot)$*:* As before, let $T = aa^t + bb^t$, and denote by T_j the jth row of T. Define $\gamma_{ij} = \max\{T_j w_i \ : \ w \in W\}$, $j = 1, \ldots, N$, for each $i = 1, \ldots, M$. Then we obtain from (4) and (5) that

$$z \leq z_{UB3}(w) = \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{\gamma_{ij}}{s_i} w_{ij}. \tag{15}$$

3.1.4. Upper Bounding Function $z_{UB4}(\cdot)$: Let T, g, T', and λ_{ij} be defined as in the derivation of z_{UB1}. Then, we obtain

$$
\begin{aligned}
z &\le \sum_{l=1}^{M} \frac{1}{s_l} (w_l^t g)^2 - \sum_{i=1}^{M}\sum_{l=1}^{N} \frac{\lambda_{ij}}{s_i} w_{ij} \\
&= \left(\sum_{l=1}^{M} \frac{w_l^t g}{\sqrt{s_l}} \right)^2 - \sum_{k=1}^{M} \sum_{\substack{l=1 \\ k\neq l}}^{M} \frac{1}{\sqrt{s_k s_l}} (w_k^t g)(w_l^t g) - \sum_{l=1}^{M}\sum_{j=1}^{N} \frac{\lambda_{ij}}{s_l} w_{ij}.
\end{aligned}
\tag{16}
$$

Define

$$
\Gamma = \max \left\{ \sum_{l=1}^{M} \frac{(w_l^t g)}{\sqrt{s_l}} \; : \; w \in W \right\}, \tag{17}
$$

and let

$$
\gamma_k = \underset{w \in W}{\text{minimum}} \left[\sum_{\substack{l=1 \\ l \neq k}}^{M} \frac{w_l^t g}{\sqrt{s_l}} \right] \quad \text{for } k = 1, \ldots, M. \tag{18}
$$

Then, from (16), (17), and (18) we get the upper bounding function,

$$z \le z_{UB4}(w) = \Gamma \sum_{i=1}^{M} \frac{w_i^t g}{\sqrt{s_i}} - \sum_{k=1}^{M} \frac{\gamma_k}{\sqrt{s_k}} (w_k^t g) - \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{\lambda_{ij}}{s_i} w_{ij}$$
$$= \sum_{i=1}^{M} \frac{(\Gamma - \gamma_i)}{\sqrt{s_i}} (w_i^t g) - \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{\lambda_{ij}}{s_i} w_{ij}. \tag{19}$$

Note that in Section 3.1.1, Δ_i , $i = 1,..., M$, λ_{ij}, $i = 1,..., M, j = 1,..., N$; in Section 3.1.2, θ_i, ϕ_i, $i = 1,..., M$; and in Section 3.1.3, γ_{ij}, $i = 1,..., M, j = 1,..., N$ can be computed by solving bounded knapsack problems.

In order to compare the performance of these upper bounding functions, we evaluated bounds implied by them for some randomly generated test problems. The results presented in Table 1 also include heuristic solution values obtained by applying Zoutendijk's (1960) improving feasible direction algorithm to the solution $w \in W$ obtained via the upper bounding schemes. The results do not favor $z_{UB4}(\cdot)$ in terms of both the resulting upper bound and the computational time. The other three upper bounding functions yield competitive results, with, however, a somewhat discernable ranking in decreasing merit order according to $z_{UB1}(\cdot)$, $z_{UB2}(\cdot)$, and $z_{UB3}(\cdot)$. Hence, these three bounding schemes are further evaluated in the sequel.

3.2 Linearization Based Upper Bounds

The quadratic terms which appear in the objective function (4) of Problem P2 can be linearized by defining a new variable for each such term. In order to get an equivalent problem, these definitions should be appropriately reflected in the constraint set. However, an exact representation will simply transfer the nonlinearity to the constraint set. Since our interest lies only in getting an upper bound, a relaxation scheme can be applied by generating linear constraints, implied by the exact nonlinear representation, which involve the interacting variables in the definitions.

Table 1. Comparison of upper bounding schemes.

		UB1			UB2			UB3			UB4			LY_{Ω_1}		
M	N	UB	z_{Heur}	t	UB	z_{Heur}	t	UB	z_{Heur}	t	UB	z_{Heur}	t	UB	z_{Heur}	t
3	5	23380.9	19579.5	0.00	22779.9	19579.5	0.00	22432.5	19579.5	0.00	29087.9	18171.6	0.00	20573.6	19170.1	0.83
3	20	66346.2	51642.2	0.01	68460.0	50449.4	0.00	60946.9	51203.2	0.01	87339.3	51642.2	0.02	52197.2	48981.5	67.37
4	20	94262.2	68445.1	0.02	98481.3	66296.0	0.01	87853.8	67849.1	0.02	143831.7	68445.1	0.05	69730.4	67701.2	230.52
5	12	43679.9	32570.7	0.01	44606.9	32570.0	0.01	40566.0	32568.5	0.01	62962.1	32550.6	0.03	33103.1	32670.2	27.44
5	16	65227.3	44530.1	0.01	62294.2	44247.2	0.00	60674.9	44247.3	0.01	99039.9	44530.1	0.04	45233.9	44354.4	43.01
5	20	73755.8	52568.3	0.01	74537.7	52393.7	0.01	68386.1	52568.3	0.02	110540.8	52568.3	0.06	53363.7	52037.9	152.68
8	10	26138.6	20111.7	0.01	29299.7	20023.4	0.00	24980.1	19723.7	0.01	49833.8	19860.5	0.07			
10	15	65970.0	47575.9	0.01	66978.8	47541.2	0.01	61614.9	47680.8	0.02	153675.2	46702.3	0.15			
15	20	88728.9	66447.8	0.04	94103.6	66210.7	0.03	84818.9	66320.8	0.04	253153.7	65941.2	0.47			
15	25	115032.4	83307.4	0.06	117633.0	82977.8	0.03	110128.0	82980.8	0.05	250067.6	83022.7	0.59			
20	30	122086.2	82745.7	0.11	128161.7	82775.6	0.05	114495.3	82304.6	0.10	434841.8	82692.1	1.35			

Legend: UB = upper bound value, z_{Heur} = corresponding P2 objective function value after applying an improving feasible direction algorithm of Section 4.4 to the solution which gives the upper bound, t = cpu seconds for computing the upper bound.

The tightness of the representation afforded by the new constraints will largely determine the quality of the bounds derived. It should be noted, however, that more refined constraints promise a possibly tighter relaxation, but result in a larger problem.

With this motivation, let us add certain implied constraints to write Problem P2 equivalently as follows.

$$
\begin{array}{lllr}
\text{LY}': \text{ maximize} & \displaystyle\sum_{i=1}^{M}\sum_{k=1}^{N}\sum_{l=k}^{N} c_{ikl}w_{ik}w_{il} & & (20a)\\
\text{subject to} & (u_{ik}-w_{ik})(w_{il}-l_{il}) \geq 0 & \forall\, i=1,\dots,M,\quad 1\leq k\leq l\leq N & (20b)\\
& (u_{il}-w_{il})(w_{ik}-l_{ik}) \geq 0 & \forall\, i=1,\dots,M,\quad 1\leq k< l\leq N & (20c)\\
& w_{il}\displaystyle\sum_{k=1}^{N} w_{ik} = s_i w_{il} & \forall\, i=1,\dots,M,\quad l=1,\dots,N & (20d)\\
& (u_{ik}-w_{ik})(u_{il}-w_{il}) \geq 0 & \forall\, i=1,\dots,M,\quad 1\leq k\leq l\leq N & (20e)\\
& (w_{ik}-l_{ik})(w_{il}-l_{il}) \geq 0 & \forall\, i=1,\dots,M,\quad 1\leq k\leq l\leq N & (20f)\\
& w\in W & & (20g)
\end{array}
$$

where from (4), and (5), we have,

$$
c_{ikl} \equiv \begin{cases} (a_k^2+b_k^2)/s_i & \forall\, i=1,\dots,M,\quad k=1,\dots,N,\quad l=k\\ 2(a_ka_l+b_kb_l)/s_i & \forall\, i=1,\dots,M,\quad 1\leq k<l\leq N \end{cases} \tag{21}
$$

and where $l_{ik}\leq w_{ik}\leq u_{ik}$, $i=1,\dots,M$, $k=1,\dots,N$, are valid lower and upper bounds on the transportation variables. If there are no explicitly specified upper bounds available on w_{ik}, then we can adopt $l_{ik}=0$, and $u_{ik}=\min\{s_i, d_k\}$ $\forall\, i=1,\dots,M$, $k=1,\dots,N$.

Let us now substitute

$$
y_{ikl}=w_{ik}w_{il} \qquad \forall\, i=1,\dots,M,\quad 1\leq k\leq l\leq N, \tag{22}
$$

in Problem LY′ in order to linearize it into the following linear program LY. Notice that the additional nonlinear, implied constraints in Problem LY′ play the role of relating w and y variables in LY via the transformation (22).

$$
\begin{array}{llll}
\text{LY: maximize} & c^t y \equiv \sum_{i=1}^{M}\sum_{k=1}^{N}\sum_{l=k}^{N} c_{ikl}y_{ikl} & & (23a) \\
\text{subject to} & l_{il}w_{ik} + u_{ik}w_{il} - y_{ikl} \ge (u_{ik}l_{il}) & \forall\ i = 1,\dots,M,\quad 1 \le k \le l \le N & (23b) \\
& u_{il}w_{ik} + l_{ik}w_{il} - y_{ikl} \ge (u_{il}l_{ik}) & \forall\ i = 1,\dots,M,\quad 1 \le k < l \le N & (23c) \\
& y_{ill} + \sum_{k=1}^{l-1} y_{ikl} + \sum_{k=l+1}^{N} y_{ilk} = s_i w_{il} & \forall\ i = 1,\dots,M,\quad l = 1,\dots,N & (23d) \\
& (u_{ik}w_{il} + u_{il}w_{ik}) - y_{ikl} \le u_{ik}u_{il} & \forall\ i = 1,\dots,M,\quad 1 \le k \le l \le N & (23e) \\
& (l_{ik}w_{il} + l_{il}w_{ik}) - y_{ikl} \le l_{ik}l_{il} & \forall\ i = 1,\dots,M,\quad 1 \le k \le l \le N & (23f) \\
& w \in W & & (23g)
\end{array}
$$

Note that Problem LY is a relaxation of Problem P2 in that for each feasible solution w to P2, there corresponds a feasible solution (w, y) to LY with the same objective value, where y is defined as in (22). However, since (22) does not necessarily hold for any feasible solution (w, y) to LY, the converse is not necessarily true. Hence, the solution to LY gives an upper bound on the value of P2.

Problem LY has $MN(N+1)/2$ linearization variables y in addition to the MN original variables w of Problem P2, and it has $2MN(N+1) + M + N$ constraints (other than the nonnegativity constraints). The number of constraints in LY can be reduced if some of the constraints are shown to be ineffective. Towards this end, let us define the following sets, which identify the constraint sets (23e) and (23d) of LY.

$$
\begin{array}{l}
\Omega_1 = \{(w,y)\ :\ (u_{ik}w_{il} + u_{il}w_{ik}) - y_{ikl} \le u_{ik}u_{il}\ \ \forall\ i = 1,\dots,M,\ \ 1 \le k \le l \le N\} \\
\Omega_2 = \{(w,y)\ :\ y_{ill} + \sum_{k=1}^{l-1} y_{ikl} + \sum_{k=l+1}^{N} y_{ilk} = s_i w_{il}\ \ \forall\ i = 1,\dots,M,\ \ l = 1,\dots,N\}
\end{array}
\qquad (24)
$$

Let LY_{Ω_1} and LY_{Ω_2} be the relaxed versions of problem LY when the restrictions $(w,y)\in\Omega_1$ and $(w,y)\in\Omega_2$ are respectively dropped from the constraint set of LY. For various sizes of randomly generated problems, P2, the problems LY, LY_{Ω_1}, and LY_{Ω_2} were solved by using the MINOS mathematical programming package (Murtagh, and Saunders, 1987). The results in Table 2 indicated a very small difference (typically less than 0.1%) between the values of LY and LY_{Ω_1}, thereby implying that $(w,y) \in \Omega_1$ is not very effective in tightening the relaxation, while it adds $MN(N+1)/2$ constraints to the size of the problem. This can be explained by noting that the constraints in Ω_1 impose a lower bound on the y variables, while LY is a maximization problem with nonnegative objective function coefficients. More technically, the constraints in Ω_1, generate a convex under-estimating function, whereas a concave envelope is more relevant in this context. Hence, unless some y variables are forced positive by these constraints, which prevents other more attractive y variables from taking on larger values, these constraints do not have a significant effect. On the other hand, Problem LY_{Ω_2} yielded undesirably large values (often 2 to 3 times that of LY), thereby showing that the presence of $(w,y) \in \Omega_2$ is crucial for a tight relaxation.

Notice that the useful set of constraints Ω_2 has been derived via the transportation supply capacity constraints, while avoiding products between variables w_{ik} and w_{jl} for $i \neq j$, since such products do not appear in the objective function of Problem P2. For this same reason, we avoided using the transportation demand constraints in a fashion similar to (23d). However, by permitting all possible variable products, we could have expanded the constraints in (23) to include the following additional constraints, where we have assumed for simplicity that all lower bounds l_{ik} are zeros, for $i = 1,\dots, M$, $k = 1,\dots, N$.

$$\begin{aligned}
&(u_{ik} - w_{ik})w_{jl} \geq 0 \qquad \forall\ 1 \leq i < j \leq M,\ l = 1,\dots, N,\ k = 1,\dots, N,\\
&(u_{jl} - w_{jl})w_{ik} \geq 0 \qquad \forall\ 1 \leq i < j \leq M,\ l = 1,\dots, N,\ k = 1,\dots, N,\\
&(u_{ik} - w_{ik})(u_{jl} - w_{jl}) \geq 0 \qquad \forall\ 1 \leq i < j \leq M,\ l = 1,\dots, N,\ k = 1,\dots, N,\\
&w_{ik} \sum_{l=1}^{N} w_{jl} = s_j w_{ik} \qquad \forall\ i = 1,\dots, M,\ i \neq j,\ k = 1,\dots, N, \text{ for each } j = 1,\dots, M,\\
&w_{ik} \sum_{j=1}^{M} w_{jl} = d_l w_{ik} \qquad \forall\ i = 1,\dots, M,\ k = 1,\dots, N, \text{ for each } l = 1,\dots, N,\\
&w_{ik} w_{jl} \geq 0 \qquad \forall\ 1 \leq i < j \leq M,\ l = 1,\dots, N,\ k = 1,\dots, N.
\end{aligned} \tag{25}$$

Linearity for (25) is attained by substituting

$$y_{ikjl} = w_{ik} w_{jl}, \qquad \forall\ 1 \leq i < j \leq M,\ l = 1,\dots, N,\ k = 1,\dots, N, \tag{26}$$

Table 2. Comparison of Problems LY, LY_{Ω_1} and LY_{Ω_2}.

		LY		LY_{Ω_1}		LY_{Ω_2}	
M	N	z^*	t	z^*	t	z^*	t
3	5	9230.5	1.59	9269.8	1.32	13719.0	1.37
3	5	1938.3	1.72	1938.3	1.35	2554.7	1.32
3	5	20465.5	1.51	20573.6	1.25	37603.2	1.30
3	10	22973.8	8.29	22973.8	4.89	54078.4	3.03
3	10	54434.4	6.46	54495.1	4.50	127712.2	2.97
4	8	8337.3	4.76	8350.0	3.50	21801.4	2.66
4	8	19252.0	4.33	19263.3	2.93	51027.9	2.64
5	12	33101.6	43.53	33103.1	23.17	104942.7	16.90
5	12	15608.2	25.70	15614.2	15.25	54367.9	9.21

Legend: z^* = solution of linearized problem,

t = cpu seconds to set up and to solve linearized problem.

and incorporating the resulting linearized version of (25) into (23), a tighter linear relaxation problem is obtained, and is given as follows.

$$
\begin{array}{lll}
\text{BLY: maximize} & c^t y \equiv \sum_{i=1}^{M}\sum_{k=1}^{N}\sum_{l=k}^{N} c_{ikl}y_{ikl} & \\
\text{subject to} & y_{ikl} \le (u_{ik}w_{il}) & \forall\ i=1,\dots,M,\ \ 1\le k\le l\le N \\
& y_{ikl} \le (u_{il}w_{ik}) & \forall\ i=1,\dots,M,\ \ 1\le k< l\le N \\
& y_{ill} + \sum_{k=1}^{l-1} y_{ikl} + \sum_{k=l+1}^{N} y_{ilk} = s_i w_{il} & \forall\ i=1,\dots,M,\ \ l=1,\dots,N \\
& (u_{ik}w_{il} + u_{il}w_{ik}) - y_{ikl} \le u_{ik}u_{il} & \forall\ i=1,\dots,M,\ \ 1\le k\le l\le N \\
& y_{ikjl} \le u_{ik}w_{jl} & \forall\ 1\le i<j\le M,\ l=1,\dots,N,\ k=1,\dots,N \\
& y_{ikjl} \le u_{jl}w_{ik} & \forall\ 1\le i<j\le M,\ l=1,\dots,N,\ k=1,\dots,N \\
& (u_{ik}w_{jl} + u_{jl}w_{ik}) - y_{ikjl} \le u_{jl}u_{ik} & \forall\ 1\le i<j\le M,\ l=1,\dots,N,\ k=1,\dots,N \\
& \sum_{l=1}^{N} y_{ikjl} = s_j w_{ik}, & \forall\ 1\le i<j\le M,\ \ k=1,\dots,N \\
& \sum_{l=1}^{N} y_{jlik} = s_j w_{ik}, & \forall\ 1\le j<i\le M,\ \ k=1,\dots,N \\
& y_{ikl} + \sum_{j=1}^{i-1} y_{jlik} + \sum_{j=i+1}^{M} y_{ikjl} = d_l w_{ik}, & \forall\ i=1,\dots,M,\ \ 1\le k\le l\le N \\
& y_{ilk} + \sum_{j=1}^{i-1} y_{jlik} + \sum_{j=i+1}^{M} y_{ikjl} = d_l w_{ik}, & \forall\ i=1,\dots,M,\ \ 1\le l< k\le N \\
& y_{ikl} \ge 0 & \forall\ i=1,\dots,M,\ \ 1\le k\le l\le N \\
& y_{ikjl} \ge 0 & \forall\ 1\le i<j\le M,\ l=1,\dots,N,\ k=1,\dots,N \\
& w \in W &
\end{array}
\tag{27}
$$

Problem BLY, in contrast with problem LY, has $MN^2(M-1)/2$ more variables, and $MN[3N(M-1)/2 + M + N - 1]$ more constraints (other than the nonnegativity restrictions). To compare the relative performance of BLY and LY, randomly generated problems were solved using the MINOS package. The results, as reported in Table 3, show that BLY brings some improvement in the upper bounds . However, this is not proportionate to the large increase in the computational effort. Hence, there is little likelihood that BLY will contribute advantageously in an enumeration type of algorithm, and we decided to use LY_{Ω_1} as one principal linear program for obtaining upper bounds. Another related linear program fashioned around solving a relaxation of LY, and then appending violated constraints along with other valid inequalities is described in Section 4.2, and is recommended for larger sized problems.

Table 1 presents some comparative results for LY_{Ω_1} versus the upper bounding schemes of the previous section. The upper bound via LY_{Ω_1} is of a significantly better quality (typically within 1-2% of optimality), although the heuristic solution obtained from the other bounding schemes is often better. Note that LY_{Ω_1} is not solved for large sized problems because the computational effort for solving LY_{Ω_1} is substantially more than that for obtaining the other upper bounds. However, as will be seen in the sequel, the strength of the bounds derived via LY_{Ω_1} far outweighs this additional effort, and provides for a substantially superior branch and bound algorithm in comparison with the other bounding schemes.

Table 3. Comparison of Problem LY and Problem BLY.

M	N	z_{LY}	t_{LY}	z_{BLY}	t_{BLY}
2	3	874.5	1.03	874.5	1.17
2	3	7470.8	1.03	7470.8	1.13
3	5	1938.3	2.56	1922.0	4.26
3	5	9230.5	2.48	9150.7	4.45
4	6	3926.5	3.64	3921.1	34.19
4	6	8077.3	3.31	8054.7	47.11
4	8	19252.0	6.10	19201.3	154.91

Legend: z_{LY} = solution of Problem LY,

t_{LY} = cpu seconds to set up and solve Problem LY,

z_{BLY} = solution of Problem BLY,

t_{BLY} = cpu seconds to set up and to solve Problem BLY.

Chapter IV

A Branch and Bound Enumeration Algorithm

Recall that Problem P2 is a convex maximization problem for which there exists an extreme point optimal solution. In this chapter, we present a branch and bound algorithm for Problem P2 which uses this fact in partitioning the problem based on the dichotomy that a variable is either positive or zero at an extreme point optimum, with the arcs corresponding to the positive flows forming a forest subgraph of the bipartite transportation graph. (This graph is a tree in the absence of degeneracy; see Bazaraa et al., 1990). Hence, a binary branch and bound tree is developed, where each node represents a partial solution for which the arcs can be divided into the three disjoint sets as $J^+ = \{(i, k) \ : \ w_{ik} > 0\}$, $J^0 = \{(i, k) \ : \ w_{ik} = 0\}$, and $J^F = \{(i, k) \ : (i, k) \notin J^+ \bigcup J^0\}$. We will refer to the variable w_{ik} as *positive fixed* if $(i, k) \in J^+$, as *zero fixed* if $(i, k) \in J^0$, and as a *free variable* if $(i, k) \in J^F$. To guarantee that the positive fixed variables can be made part of the set of basic variables corresponding to a basic feasible solution of W, the forest constructed by the arcs in J^+ will be kept cycle free at all stages of the algorithm. Also, if any arc $(i, k) \in J^F$ has the potential of forming a cycle in the current forest, the variable w_{ik} will be fixed to zero, and (i, k) will be moved from the set J^F to the set J^0. Hence, whenever all the variables are fixed, i.e. $J^F = \emptyset$, the resulting solution will correspond to a vertex of W.

An efficient implementation of the cycle prevention method depends largely on utilizing a suitable data structure. A cycle prevention scheme, which does not require an explicit representation of the current forest, is proposed in Appendix A.

We now describe the principal components of the branch and bound algorithm, and then present the overall procedure.

4.1 Logical Tests

As the algorithm progresses, we will be modifying the lower and upper bounds on the variables w. The special structure of the transportation constraints can be further exploited to give tighter implied lower and upper bounds on the variables. A "logical test" which accomplishes this can be executed as follows. Suppose that the bounds on the variables are initialized as $l_{ik} = 0$, $u_{ik} = \min\{s_i, d_k\}$, $i = 1,....., M,$ $k = 1,..., N$. Then, given any explicitly imposed bounds on the variables, let us rewrite the transportation constraints as shown below.

$$\begin{aligned}
&\sum_{k=1}^{N} w_{ik} \leq s_i, && i = 1,..., M, \\
&\sum_{k=1}^{N} w_{ik} \geq s_i, && i = 1,..., M, \\
&\sum_{i=1}^{M} w_{ik} \leq d_k, && k = 1,..., N, \\
&\sum_{i=1}^{M} w_{ik} \geq d_k, && k = 1,..., N, \\
&l_{ik} \leq w_{ik} \leq u_{ik}, && i = 1,..., M, \; k = 1,..., N.
\end{aligned} \tag{28}$$

Accordingly, we can compute the maximum slack values for each of the four sets of supply and demand constraints as

$$
\begin{aligned}
SL_i &= s_i - \sum_{k=1}^{N} l_{ik}, & i &= 1,\dots, M,\\
SU_i &= \sum_{k=1}^{N} u_{ik} - s_i, & i &= 1,\dots, M,\\
DL_k &= d_k - \sum_{i=1}^{M} l_{ik}, & k &= 1,\dots, N,\\
DU_k &= \sum_{i=1}^{M} u_{ik} - d_k, & k &= 1,\dots, N.
\end{aligned}
\tag{29}
$$

For a feasible set of lower and upper bounds (l, u) all the maximum slack values should be non-negative. A tighter (if not the same) bounds, say (l^{new}, u^{new}) , can then be found as follows.

PROPOSITION 1: The following set of lower and upper bounds are valid implications of (28).

$$
\begin{aligned}
l_{ik}^{new} &= \max\{l_{ik},\ u_{ik} - SU_i,\ u_{ik} - DU_k\},\\
u_{ik}^{new} &= \min\{u_{ik},\ SL_i + l_{ik},\ DL_k + l_{ik}\}, \qquad i = 1,\dots, M,\ \ k = 1,\dots, N.
\end{aligned}
\tag{30}
$$

Moreover, at any stage in a continuous sequential updating process, the order of computing either l_{ik}^{new} or u_{ik}^{new} for a particular variable w_{ik} does not change the outcome.

Proof: Clearly, $l_{ik}^{new} \geq l_{ik}$. Furthermore, note that if all the supply nodes except node i send flows to demand node k at their upper bounds then node i should send a flow of at least $d_k - \sum_{j=1, j\neq i}^{M} u_{jk}$ $(= u_{ik} - DU_k)$ to satisfy the demand requirement d_k. In a likewise manner, if a supply node i sends flows at their upper bounds to all the demand nodes except k, then it should send a flow of at least $s_i - \sum_{j=1, j\neq k}^{N} u_{ij}$ $(= u_{ik} - SU_i)$ to node k to exhaust its capacity. This justifies the expression for l_{ik}^{new}.

Next, consider u_{ik}^{new}. We have $u_{ik}^{new} \leq u_{ik}$, and similar to above, if all the supply nodes except i send flows at their lower bounds to a demand node k then node i should not send a flow of more than $d_k - \sum_{j=1, j\neq i}^{M} l_{jk}$ $(= DL_k + l_{ik})$ to node k , or else the total flow to node k will exceed the demand requirement d_k . Also, if a supply node i sends flows at their lower bounds to all the demand nodes except node k, then it should not send a flow of more than $s_i - \sum_{j=1, j\neq k}^{N} l_{ij}$ $(= SL_i + l_{ik})$ to node k, otherwise the total outgoing flow from the supply node i will exceed its supply capacity. This establishes (30).

Finally, noting that (30) is equivalent to

$$\begin{aligned} l_{ik}^{new} &= \max\left\{ l_{ik},\ s_i - \sum_{\substack{j=1\\ j\neq k}}^{N} u_{ij},\ d_k - \sum_{\substack{j=1\\ j\neq i}}^{M} u_{jk} \right\} \\ u_{ik}^{new} &= \min\left\{ u_{ik},\ s_i - \sum_{\substack{j=1\\ j\neq k}}^{N} l_{ij},\ d_k - \sum_{\substack{j=1\\ j\neq i}}^{M} l_{jk} \right\}, \end{aligned} \tag{31}$$

since the value of u_{ik} does not affect the computation of l_{ik}^{new}, and the value of l_{ik} does not affect the computation of u_{ik}^{new}, the order of updating these bounds is inconsequential, and this completes the proof. □

The lower and upper bounds can be updated according to (30), along with an updating of the maximum slack values in (29), in a circular, sequential fashion, using the most recent updated quantities at each stage of evaluating the pair $(l_{ik}^{new}, u_{ik}^{new})$ for any i, k. This can continue until there is no change in any circular pass, or at some stage a negative maximum slack value indicates infeasibility, whence the current node of the branch and bound tree can be fathomed. During the implementation of the branch bound algorithm, and even within the logical test itself, if any maximum slack value associated with a node changes, then all the arcs incident to that node are subject to this logical test. All such nodes will be stored in a list called NT, and are candidates for applying the logical test. Note that as indicated in Proposition 2 below, it is sufficient to check the nonnegativity of the maximum slack values for fathoming purposes, and there is no need for further feasibility checks on updated variable bounds.

PROPOSITION 2: Let SU_i^o, DU_k^o, SL_i^o, DL_k^o be the current nonnegative maximum slack values for source i and destination k at any stage of the continuous updating process, and let $l_{ik}^o \leq u_{ik}^o$ be the current bounds on w_{ik}. Then, after an update of the bounds via (30), if the maximum slack values are still nonnegative, we have $l_{ik}^{new} \leq u_{ik}^{new}$ holding as well.

Proof: By Proposition 1, assume without loss of generality that l_{ik}^{new} is updated first, and let SL_i^{new} and DL_k^{new} be the updated values of SL_i and DL_k, respectively, using l_{ik}^{new} in lieu of l_{ik}^o in (29). If $SL_i^{new} \geq 0$ and $DL_k^{new} \geq 0$, then $l_{ik}^{new} \leq SL_i^{new} + l_{ik}^{new}$, and $l_{ik}^{new} \leq DL_k^{new} + l_{ik}^{new}$, and since we also have $l_{ik}^{new} = \max\{l_{ik}^o,\ u_{ik}^o - SU_i^o,\ u_{ik}^o - DU_k^o\} \leq \max\{u_{ik}^o,\ u_{ik}^o - SU_i^o,\ u_{ik}^o - DU_k^o\} = u_{ik}^o$, we get by (30) that $l_{ik}^{new} \leq u_{ik}^{new}$, and this completes the proof. □

The following result shows that the foregoing logical tests will automatically detect the situation in which the arcs fixed at zero yield a unique completion to the supply and demand constraints, and will compute such a solution if it is not nonnegative, i.e., if it corresponds to a basic feasible solution.

PROPOSITION 3: Consider a forest subgraph of the transportation graph, and suppose that $l_{ik} = u_{ik} = 0$ for all arcs (i, k) not in the forest arcs, while l_{ik} and u_{ik} are some valid lower and upper bounds for the flows on the forest arcs. Then the above sequential procedure will either determine the unique basic feasible solution corresponding to this forest with the final $u_{ik} = l_{ik}$ coinciding with the flows for all the arcs (i, k), or will terminate with the conclusion that the given bound restrictions are infeasible for the problem.

Proof: Suppose that in the transportation graph all the arcs incident to demand node k except the arc (i, k) carry fixed flows. Define $f = \sum_{\substack{j=1 \\ j\neq i}}^{M} l_{jk} \ (= \sum_{\substack{j=1 \\ j\neq i}}^{M} u_{jk})$. Then by (31),

$$l_{ik}^{new} = \max\left\{ l_{ik},\ s_i - \sum_{\substack{j=1 \\ j\neq k}}^{N} u_{ij},\ d_k - f \right\}$$
$$u_{ik}^{new} = \min\left\{ u_{ik},\ s_i - \sum_{\substack{j=1 \\ j\neq k}}^{N} l_{ij},\ d_k - f \right\}. \tag{32}$$

Now, suppose that the given forest representation is feasible. Then, we must have $l_{ik}^{new} = d_k - f$ in (32), or else if $l_{ik}^{new} > d_k - f$, then we would have $DL_k \equiv d_k - f - l_{ik}^{new} < 0$ in (29) after the update. Similarly, we must have $u_{ik}^{new} = d_k - f$ in (32), since otherwise, $u_{ik}^{new} < d_k - f$ would imply in (29) that $DU_k \equiv f + u_{ik}^{new} - d_k < 0$. Hence, we have $l_{ik}^{new} = u_{ik}^{new} = d_k - f$, and moreover, the updated values of

DL_k and DU_k are both zeros. A similar result holds for a supply node i for which all the incident arcs except a single arc (i, k) carry fixed flows. Of course, for nodes having fixed flows on all incident arcs, (29) directly checks the corresponding supply and demand constraints for feasibility.

Now, since a forest must have an end node, by applying the above scheme to the arc incident at a chosen end node, we can fix its flow as above and eliminate the arc from the forest. Progressing repeatedly in this fashion, we either obtain a basic feasible solution with all maximum slacks zero in (29), or detect infeasibility en route, via a negative maximum slack value. This completes the proof. □

4.2 Bounding Step

The functional upper bounding schemes 1, 2 and 3 involve little computational effort, and are therefore attractive methods for deriving upper bounds in the branch and bound algorithm. However, as will be seen in the sequel, the main fathoming power comes from the linearization based upper bounding scheme. One way to apply this scheme is to solve Problem LY_{Ω_1}, as discussed in Section 3.2. However, as will be seen in the next chapter, for large size problems, solving LY_{Ω_1}becomes computationally expensive. At the expense of sacrificing the quality of the upper bound obtained, we will present a two step application of the linearization method to reduce the computational effort. In this application, by using only a subset of the constraints defining LY, a reduced size problem LY_1 is first solved. Then, the constraints which have been excluded in LY_1 are tested for feasibility, and the violated constraints are appended along with some other suitable inequalities to get the problem LY_2. The solution of this problem gives the desired upper bound.

To present this strategy, given the sets J^+, J^0 and J^F at some current node of the branch and bound tree, we let the constraint set of LY_1 be comprised of the following constraints from LY.

$$
\begin{aligned}
& l_{il}w_{ik} + u_{ik}w_{il} - y_{ikl} \geq u_{ik}l_{il} \quad \forall\ (i, k) \in J^{+}, \text{for each } (i, l) \in J^{+}, l > k \\
& (u_{ik} + l_{ik})w_{ik} - y_{ikk} \geq l_{ik}u_{ik} \quad \forall\ (i, k) \in J^{+} \\
& u_{il}w_{ik} + l_{ik}w_{il} - y_{ikl} \geq u_{il}l_{ik} \quad \forall\ (i, k) \in J^{+}, \text{for each } (i, l) \in J^{+}, l > k \\
& 2l_{ik}w_{ik} - y_{ikk} \leq l_{ik}^{2} \quad \forall\ (i, k) \in J^{+} \\
& l_{ik}w_{il} + l_{il}w_{ik} - y_{ikl} \leq l_{il}l_{ik} \quad \forall\ (i, k) \in J^{+}, \text{for each } (i, l) \in J^{+}, l > k
\end{aligned}
\tag{33a}
$$

along with the bounds

$$
\begin{aligned}
& 0 \leq y_{ikk} \leq u_{ik}w_{ik} \quad \forall\ (i, k) \in J^{F} \\
& 0 \leq y_{ikl} \leq u_{ik}u_{il} \quad \forall\ (i, k) \in J^{F}, \text{for each } (i, l) \in J^{F}, l > k \\
& l_{ik} \leq w_{ik} \leq u_{ik} \quad \forall\ (i, k) \in J^{+} \bigcup J^{F}
\end{aligned}
\tag{33b}
$$

in addition to the restriction

$$
w \in W \bigcap \Omega_2. \tag{33c}
$$

Note that, for all the arcs $(i, k) \in J^0$, we have $u_{ik} = l_{ik} = 0$, thereby implying that $w_{ik} = 0$, $y_{ilk} = 0$, for $l = 1,..., k$, and $y_{ikl} = 0$, for $l = k + 1,..., N$. Therefore, $\forall (i, k) \in J^0$, w_{ik} and the associated y variables are dropped from the bounding problem. Also, $\forall (i, k) \in J^+$ such that $u_{ik} = l_{ik}(> 0)$, we substitute the following in the bounding problem.

$$
\begin{aligned}
& w_{ik} = u_{ik}, \\
& y_{ikk} = u_{ik}u_{ik}, \\
& y_{ikl} = u_{ik}w_{il}, \quad \text{for } k < l \leq N, \text{ and} \\
& y_{ilk} = u_{ik}w_{il}, \quad 1 \leq l < k.
\end{aligned}
\tag{34}
$$

Consequently, the particular constraint of Ω_2 that has been generated by multiplying both sides of the supply capacity constraint of source i by w_{ik}, becomes redundant. Also, the constraints which contain y variables which are all fixed in value (positive or zero) become redundant and are hence dropped from the bounding problem.

Now, let $(\overline{w}, \overline{y})$ be the solution of LY_1 . Based on this solution, a larger problem LY_2 is generated by augmenting the constraints of LY_1 as follows. First, all the constraints in LY (including those in the set Ω_1) which were omitted from LY_1 and which are violated by the solution $(\overline{w}, \overline{y})$ are included in the constraint set of LY_2.

Second, a class of constraints based on the convexity of the relationship

$$y_{ikk} = w_{ik}^2 \quad \text{for } i = 1,..., M,\ k = 1,..., N, \tag{35}$$

is included in LY_2. By the definition of convexity,

$$y_{ikk} \geq \overline{w}_{ik}^2 + 2(w_{ik} - \overline{w}_{ik})\overline{w}_{ik} = 2\overline{w}_{ik}w_{ik} - \overline{w}_{ik}^2 \quad \text{for } i = 1,..., M,\ k = 1,..., N. \tag{36}$$

For an arc (i, k), if $\overline{y}_{ikk} < \overline{w}_{ik}^2$, then the constraint (36) is included in the constraint set of LY_2.

A third class of valid constraints are derived using the current forest represented by J^+. Let C_p and C_q be two components of the current forest such that the cardinality of the cut set $CC_{pq} = [\{(i, k) : i \in C_p, k \in C_q\} \bigcup \{(i, k) : i \in C_q, k \in C_p\}] \bigcap J^F$ is at least two. For all such component pairs, we can define the following constraint based on the requirement that the arcs with positive flows should not form a cycle, and so, at most one arc in CC_{pq} can carry a positive flow.

$$\sum_{(i,\,k) \in CC_{pq}} w_{ik} \leq u(CC_{pq}) \tag{37}$$

where $u(CC_{pq}) = \max\{u_{ik} : (i, k) \in CC_{pq}\}$. Again, only those members of (37) which are infeasible for $(\overline{w}, \overline{y})$ are added to the constraint set of LY_2. Note that we do not generate new linearization constraints from (37) by multiplying it with other nonnegative factors because it contains mixed (i, k) terms. (This is the same problem as was encountered earlier with respect to the demand requirement constraints). Notice that when $|CC_{pq}| = 1$, (37) represents a simple upper bound of the type $w_{ik} \leq u_{ik}$. Hence, we can expect (37) to be more useful when $|CC_{pq}|$ is large. In particular, when the number of forest components decreases, (37) may lead to the detection of infeasibility, given that there exists no basic feasible solution completion to the current partial solution. From our computational experience we have seen that even with small $|CC_{pq}|$ values, (37) lends some advantage to the linear program LY_2, without introducing much of an additional computational burden. If LY_{Ω_1} is used as the bounding problem, then this set of constraints is also included in the problem LY_{Ω_1}

4.3 Partitioning Scheme and Branching Variable Selection Rule

At the branching step, some free variable w_{pq} is chosen, as indicated below, to further partition the problem associated with the current node of the branch and bound tree into two subproblems. Along one branch, w_{pq} is "positive fixed," and along the other branch, it is "zero fixed." For the branch where $w_{pq} = 0$, we explicitly set $y_{pkq} = 0 \ \forall 1 \leq k \leq q$ and $y_{pqk} = 0 \ \forall q < k \leq N$. For the other branch where $w_{pq} > 0$, an initial lower bound of $l_{pq} = 1$ is used to tighten the related constraints in LY. Also, the introduction of this new link in the current forest connects two components of this forest, and so, the flows on all the links in the cut-set corresponding to these two components are set to zero in order to maintain a forest in any completion of the partial solution.

Since the branching process tightens the relaxed problem LY in the above fashion, we would like to choose a branching variable that gives a tighter relaxation than other alternatives. Let $(\overline{w}, \overline{y})$ be the solution obtained for the bounding linear program solved at the current branch and bound node. The magnitude of violation of equations (22) for the components of the solution $(\overline{w}, \overline{y})$, and the contribution of this violation to the upper bound, are two main factors that drive our choice for the branching variable. The following branching rule is prescribed, based on extensive computational experiments.

Branching Variable Selection Rule:

Step 1: Compute

$$\alpha_1 = \max_{(i,k)\in J^F} \left\{ \sum_{\substack{l>k \\ (i,l)\in J^F \cup J^+}} c_{ikl} \max\{0, (\bar{y}_{ikl} - \bar{w}_{ik}\bar{w}_{il})\} \right. \tag{38a}$$
$$\left. + \sum_{\substack{l<k \\ (i,l)\in J^F \cup J^+}} c_{ilk} \max\{0, (\bar{y}_{ilk} - \bar{w}_{ik}\bar{w}_{il})\} + c_{ikk} \max\{0, (\bar{y}_{ikk} - \bar{w}_{ik}\bar{w}_{ik})\} \right\}.$$

If $\alpha_1 = 0$, go to Step 2. Otherwise, if $\alpha_1 > 0$, and $(\underline{i}, \underline{k})$ uniquely evaluates (38a) , then select $(p, q) = (\underline{i}, \underline{k})$ as the branching variable indices. If $\alpha_1 > 0$, but there is more than one arc that gives the maximum value α_1, then let $T_{IE1} \subseteq J^F$ be the set of arcs that tie in (38a), $|T_{IE1}| > 1$, and compute

$$\alpha_2 = \max_{(i,k)\in T_{IE1}} \left\{ \sum_{\substack{l>k \\ (i,l)\in J^F \cup J^+}} \max\{0, (\bar{y}_{ikl} - \bar{w}_{ik}\bar{w}_{il})\} \right. \tag{38b}$$
$$\left. + \sum_{\substack{l<k \\ (i,l)\in J^F \cup J^+}} \max\{0, (\bar{y}_{ilk} - \bar{w}_{ik}\bar{w}_{il})\} + \max\{0, (\bar{y}_{ikk} - \bar{w}_{ik}\bar{w}_{ik})\} \right\}.$$

If the value α_2 is given uniquely by the arc $(\underline{i}, \underline{k})$, then select $(p, q) = (\underline{i}, \underline{k})$. If there is still a tie, then letting $T_{IE2} \subseteq T_{IE1}$ be the set of arcs that tie in (38b), select that branching variable which corresponds to

$$(p, q) = \operatorname{argmax}\{\bar{w}_{ik} : (i, k) \in T_{IE2}\}, \tag{38c}$$

with persisting ties broken arbitrarily.

Step 2: Define $BR_1 = \{(i, k, l) : (i, k)\in J^F, (i, l)\in J^F,\ k \le l\}$, $BR_2 = \{(i, k, l) \colon\ (i, k)\in J^+,\ (i, l)\in J^F,\ k < l\}$, and $BR_3 = \{(i, k, l) : (i, k)\in J^F, (i, l)\in J^+,\ k < l\}$. Find $(\underline{i}, \underline{k}, \underline{l})$ according to

$$(\underline{i}, \underline{k}, \underline{l}) = \operatorname{argmax}\{c_{ikl}\bar{y}_{ikl} : (i, k, l) \in BR_1 \cup BR_2 \cup BR_3\}, \tag{39a}$$

with ties broken arbitrarily. If $c_{\underline{ikl}}\bar{y}_{\underline{ikl}} > 0$ then select the branching variable indices as

$$(p, q) = \begin{cases} \operatorname{argmax}\{\bar{w}_{\underline{ik}}, \bar{w}_{\underline{il}}\} & \text{if } (\underline{i}, \underline{k}, \underline{l}) \in BR_1 \\ (\underline{i}, \underline{l}) & \text{if } (\underline{i}, \underline{k}, \underline{l}) \in BR_2 \\ (\underline{i}, \underline{k}) & \text{if } (\underline{i}, \underline{k}, \underline{l}) \in BR_3 \end{cases} \tag{39b}$$

If $c_{\underline{ikl}}\bar{y}_{\underline{ikl}} = 0$, then proceed to Step 3.

Step 3: Select the branching variable indices (p, q) as

$$(p, q) = \operatorname{argmax}\{\bar{w}_{ik} : (i, k) \in J^F\}, \tag{40}$$

with ties broken arbitrarily.

4.4 Lower Bounds Via an Improving Feasible Direction Algorithm

An important issue in branch and bound algorithms is to obtain good quality incumbent solutions at the early stages of the process. Since all the bounding problems of Chapter 3 include the constraints $w \in W$, we obtain a feasible allocation scheme via any such problem. Denoting $\bar{w} \in W$ to be any such solution, we obtain $\bar{w}'G\bar{w}$ as a lower bound on the global optimum. Having the feasible solution $\bar{w}$, we derive an improved solution using Zoutendijk's (1960) feasible direction algorithm.

Define $\bar{w}_i = [\bar{w}_{i1}, \ldots, \bar{w}_{iN}]'$ and $d_i = [d_{i1}, \ldots, d_{iN}]'$ for $i = 1, \ldots, M$, and let $T = aa' + bb'$ as in (5). To find an improving feasible direction, we solve the following (transportation) problem, where the objective function represents the directional derivative of the objective function of Problem P2 at $\bar{w}$ in the direction d.

$$\text{IFD:}\quad \text{maximize} \quad \sum_{i=1}^{M} \frac{2}{s_i} \overline{w}_i^t T d_i$$

$$\begin{aligned} \text{subject to} \quad & \sum_{k=1}^{N} d_{ik} = 0 && \text{for } i = 1,\dots, M \\ & \sum_{i=1}^{M} d_{ik} = 0 && \text{for } k = 1,\dots, N \\ & L_{ik} \le d_{ik} \le 1 && \text{for } i = 1,\dots, M \quad k = 1,\dots, N \end{aligned}$$

where

$$L_{ik} = \begin{cases} 0 & \text{if } \overline{w}_{ik} = 0 \\ -1 & \text{if } \overline{w}_{ik} > 0, \quad i = 1,\dots, M, \ k = 1,\dots, N \end{cases}$$

Let $\overline{d}$ solve Problem IFD. If $\overline{d} = 0$, then the current solution $\overline{w}$ is a KKT (Karush-Kuhn-Tucker) point for P2, and we terminate this iterative process. Otherwise a new feasible point $\hat{w}$ is found by, taking

$$\hat{w}_{ik} = \overline{w}_{ik} + \theta_m \overline{d}_{ik} \qquad \text{for } i = 1,\dots, M, \ k = 1,\dots, N,$$

where the step length $\theta_m \ge 0$ is chosen as the maximum step length such that $\hat{w} \in W$, since the directional derivative along $\overline{d}$ is positive, and so, the convex objective function of problem P2 is increasing along the direction $\overline{d}$. Hence, we compute

$$\theta_m = \min\{\overline{w}_{ik}/(-\overline{d}_{ik}) \ : \ \overline{d}_{ik} < 0, \ i = 1,\dots, M, \ k = 1,\dots, N\}.$$

Consequently, $\hat{w}^t G \hat{w} > \overline{w}^t G \overline{w}$. By replacing $\overline{w}$ by $\hat{w}$, this method can be applied to the newly found point. We employ two stopping criteria for this iterative scheme. The first one places a limit of 75 on the maximum number of iterations. The second criterion is based on the magnitude of the improvement in the lower bound, that is, if $\hat{w}^t G \hat{w} - \overline{w}^t G \overline{w} \le \varepsilon$ then we stop. We prescribe

$\varepsilon = \min\{\alpha(\overline{w}^t G \overline{w}),\ 0.05\}$, where $\alpha \in (0, 1)$ is a predetermined optimality criterion for the branch and bound algorithm.

4.5 Other Features of the Algorithm

Search strategy: A depth first search strategy is used in the branch and bound algorithm. A partial solution list, PS, keeps track of the branch and bound tree using the framework due to Geoffrion (1967). Each entry (i, k) in PS has two attributes; it is either $+(i, k)$ or $-(i, k)$, and it may or may not be underlined. The sign of (i, k) shows whether w_{ik} is positive or zero fixed, that is, $+(i, k)$ indicates $l_{ik} > 0$, $(i, k) \in J^+$, and $-(i, k)$ indicates $l_{ik} = u_{ik} = 0$, $(i, k) \in J^0$. If $\mp(i, k)$ is underlined then this indicates that the descendents of its complementary branch have been fathomed. For example, $PS = \{(1,5),\ -(2,8),\ -\underline{(3,4)}, \underline{(2,7)}\}$ implies that paths $\{(1,5),\ -(2,8),\ (3,4)\}$ and $\{(1,5),\ -(2,8),\ -(3,4),\ -(2,7)\}$ of the branch and bound tree have been fathomed.

Optimality criterion: To avoid undue excessive computations involved in sifting through alternative optimal solutions or close to global optimal solutions, the fathoming criterion is adapted as follows,

$$UB \leq (1 + \alpha)z^*, \tag{41}$$

where $0 < \alpha < 1$, UB is a valid upper bound at the current branch and bound node, and z^* is the current best solution (incumbent) value. Hence, when the algorithm stops, we can claim that the global maximum is within $100 \times \alpha\%$ of the current best solution. Also, at some node of the branch and bound tree, an improvement in z^* may result in the fathoming of some of the ancestors of the current node via (41). To implement this, the upper bound UB_{pq} for each active node is stored along with the associated choice of a branching variable w_{pq}. Then, whenever a node is selected for branching upon backtracking, it is either fathomed based on its upper bound, or else it is partitioned using the corresponding branching variable.

4.6 Summary of the Algorithm

Step 0 : Initialization. Set $PS = \emptyset$, $J^+ = \emptyset$, $J^0 = \emptyset$, $J^F = \{(i, k) : i = 1,..., M, k = 1,..., N\}$. Set $l_{ik} = 0$, $u_{ik} = \min\{s_i, d_k\} \forall (i, k) \in J^F$. Calculate the maximum slack values using (29). Initialize the forest with all the $(M + N)$ nodes and with no arcs, and set the component count to $m + n$. Include all the nodes of the transportation graph in the set NT of nodes eligible for applying the logical test, and apply the logical tests as described in Step 2. Assuming feasibility of the transportation constraints, this logical test should end with $NT = \emptyset$. Go to Step 3.

Step 1 : Cycle Prevention. Let (p, q) be the arc last added to J^+, and let C_p and C_q be the two components in the current forest which include p and q respectively; note that $C_p \neq C_q$. Set $u_{ik} = 0$ $\forall (i, k) \in CC_{pq} - J^0$, where $CC_{pq} = \{(i, k) : i \in C_q, k \in C_p\} \bigcup \{(i, k) : i \in C_p, k \in C_q, (i, k) \neq (p, q)\}$, and update SU_i, and DU_k using (29). If any of the maximum slack values becomes negative, go to Step 5. Otherwise, decrease the component count by 1 after combining the components C_p and C_q. Update J^F, PS, NT, and J^0, as $J^F \leftarrow J^F - (CC_{pq} - J^0)$, $PS \leftarrow PS \bigcup \{ - \underline{(i, k)} : (i, k) \in CC_{pq} - J^0\}$, $NT \leftarrow NT \bigcup NT_{CC_{pq}}$ (where $NT_{CC_{pq}}$ is the set of nodes incident to the arcs in $CC_{pq} - J^0$), and $J^0 \leftarrow J^0 \bigcup CC_{pq}$. Proceed to Step 2.

Step 2 : Logical Tests. All the arcs incident to the nodes in NT are eligible for applying the logical tests of Section 4.1. Also, when testing an arc (p, q), where $p \in NT$ and $q \notin NT$, if any maximum slack value of node q changes then q is included in NT. The same argument applies for an arc (p, q) such that $q \in NT$ and $p \notin NT$.

Let (i, k) be the arc being tested. Replace l_{ik} by $\max\{l_{ik}, u_{ik} - DU_k, u_{ik} - SU_i\}$, and replace u_{ik} by $\min\{u_{ik}, DL_k + l_{ik}, SL_i + l_{ik}\}$. If l_{ik} has changed, update SL_i, and DL_k using (29). If u_{ik} has changed, update SU_i, and DU_k using (29). Now, consider the following situations:

(a) If any maximum slack value becomes negative, go to Step 5.

(b) If $l_{ik} > 0$, where $(i, k) \in J^F$, update J^+, J^F and PS, as $J^+ \leftarrow J^+ \bigcup \{(i, k)\}$, $J^F \leftarrow J^F - \{(i, k)\}$, and $PS \leftarrow PS \bigcup \{\underline{(i, k)}\}$. Return to Step 1.

(c) If $l_{ik} = u_{ik} = 0$, where $(i, k) \in J^F$, update J^0, J^F and PS, as $J^0 \leftarrow J^0 \bigcup \{(i, k)\}$, $J^F \leftarrow J^F - \{(i, k)\}$, and $PS \leftarrow PS \bigcup \{ - \underline{(i, k)}\}$. Continue the logical tests on the other arcs.

If none of the bounds change for all the arcs incident to a node, drop that node from NT. Continue until $NT = \emptyset$. At this stage, if $|PS|$ is less than the total number of variables, then proceed to Step 3. Otherwise, by Proposition 3, set $\bar{w}_{ik} = u_{ik}(= l_{ik})$ for all (i, k) in the graph. If $z^* < \bar{w}^t G \bar{w}$ then set $z^* = \bar{w}^t G \bar{w}$. Go to Step 5.

Step 3 : Bounding Step. Using the current lower and upper bounds (l, u) on the variables in addition to the transportation constraints, construct the linear functions $z_{UB1}(\cdot)$, $z_{UB2}(\cdot)$, and $z_{UB3}(\cdot)$, and maximize them individually over the set $\{w \in W, l \leq w \leq u\}$. Let UB be the lowest of the optimal objective function values of these three problems. Apply the improving feasible direction method of Section 4.4 to the optimal solution for each bounding problem. If the incumbent improves, update z^*. If $UB \leq (1 + \alpha)z^*$, go to Step 5. Otherwise, set up Problem LY_{Ω_1} along with the constraints (37) and solve it. Let UB denote its optimal objective value, and apply the improving feasible direction algorithm to the allocations obtained in the optimum to LY_{Ω_1}. If the incumbent improves update z^*. If $UB \leq (1 + \alpha)z^*$, then go to Step 5, and otherwise, proceed to Step 4. Alternatively, if the pair of linear programs LY_1-LY_2 is to be used in lieu of LY_{Ω_1} for computing bounds, then first set up Problem LY_1 and solve it. Let UB be its optimal objective function value. Apply the improving feasible direction algorithm to the allocations obtained via the optimum to LY_1. If the incumbent improves, update z^*. If $UB \leq (1 + \alpha)z^*$, go to Step 5. Otherwise, set up problem LY_2 and solve it. Let UB be its optimal objective function value. Apply the improving feasible direction algorithm to the optimum allocations for LY_2, and if the incumbent improves, update z^*. If $UB \leq (1 + \alpha)z^*$, go to Step 5. Otherwise, store the best upper bound for the current node, and proceed to Step 4.

Step 4 : Branching Step. Let $(\bar{w}, \bar{y})$ be the optimal solution obtained for the most recent bounding linear program. Select the branching variable w_{pq}, where $(p, q) \in J^F$, using the rule of Section 4.3. Store this choice at the current node and designate the available upper bound for this node as UB_{pq}. Update J^+, J^F and PS, as $J^+ \leftarrow J^+ \cup \{(p, q)\}$, $J^F \leftarrow J^F - \{(p, q)\}$, and $PS \leftarrow PS \cup \{(p, q)\}$. Accordingly, set $l_{pq} = 1$, and update SL_p, and SL_q using (29). Also, include nodes p and q in NT, and return to Step 1.

Step 5 : Fathoming Step. Let (p, q) be the rightmost non-underlined entry in PS such that $UB_{pq} > (1 + \alpha)z^*$. If such an entry does not exist then stop; the incumbent solution is within $100\alpha\%$ of optimality. Otherwise, set $l_{ik} = 0$ and $u_{ik} = \min\{s_i, d_k\}$ $\forall(i, k)\in PS^A_{pq} \cup J^F$, where PS^A_{pq} is the set of arcs included in PS after (p, q). Drop the corresponding arcs in $PS^A_{pq} \cup \{(p, q)\}$ from J^+ and J^0, and include the arcs in PS^A_{pq} to J^F. Set $l_{ik} = 1$ and $u_{ik} = \min\{s_i, d_k\}$ $\forall(i, k)\in PS^B_{pq} \cap J^+$, where $PS^B_{pq} = PS - [PS^A_{pq} \cup \{(p, q)\}]$. Also, noting that currently $\{ + (p, q)\} \in PS$ by the branching process, set $l_{pq} = u_{pq} = 0$, and update PS and J^0 as $PS \leftarrow PS^B_{pq} \cup \{ - \underline{(p, q)}\}$, and $J^0 \leftarrow J^0 \cup \{(p, q)\}$. Update the maximum slack values according to the changes in (l, u) using (29), and include the nodes for which a maximum slack value has changed in NT . Since at least one of the positive fixed variables has changed its status, the current forest is no longer valid. Let c_a be the total number of affected components in the current forest. If c_a is equal to the component count then rebuild the forest by using the current J^+. Otherwise rebuild only the components which are affected. Update the component count, and return to Step 2.

Chapter V

Computational Experience

In this chapter we present our computational experience with the proposed branch and bound algorithm. At the bounding step, we investigate the use of the upper bounds of Section 3.1 alone, versus their use in conjunction with the linear program LY_{Ω_1}, or with the two step approach associated with the sequential solutions of the linear programs LY_1 and LY_2 as described in Sections 4.2 and 4.6. Several algorithmic refinements and modifications which were tested are also referred to in our discussion. We first describe the problem generation method, and then present the computational results along with a discussion of the computational aspects of the algorithm.

5.1 Problem Generation

Data for the problems were randomly generated by using a uniformly distributed pseudo random number generator. Distinct sink locations, (a_k, b_k), for $k = 1,..., N$, were randomly generated with each component being an integer lying in the interval [0, 25]. After generating a valid set of sink locations , the point $(a_{\min}, b_{\min})$ was taken as the origin, where $a_{\min} = \min_{1 \le k \le N} a_k$, and $b_{\min} = \min_{1 \le k \le N} b_k$, and all the locations were adjusted accordingly. Integer customer demands were generated over the interval [1, 15]. To generate the capacity of the M supply centers such that the total supply is equal to the total demand, the following iterative procedure was used. Here, $\lfloor \alpha \rfloor^-$ denotes truncating $\alpha \in R$ to the greatest integer less than or equal to α.

Step 0. Set the counter $j = 0$. Generate initial integer supplies, $s_i^{(j)}$, for $i = 1,..., M$, over the interval $[1, \lfloor d_{\max} N/M \rfloor^-]$, where $d_{\max} = \max_{1 \le k \le N} d_k$. Define $t_s^{(j)} = \sum_{i=1}^{M} s_i^{(j)}$, and let $t_d = \sum_{k=1}^{N} d_k$. Proceed to Step 1.

Step 1. If $t_s^{(j)} = t_d$, then stop and use the current supplies. Otherwise, compute $r = t_d / t_s^{(j)}$, and set $s_i^{(j+1)} = \max\{1, \lfloor r s_i^{(j)} \rfloor^-\}$ $\forall$ $i = 1,..., M$. Increment the counter j by one, and proceed to Step 2.

Step 2. Compute $t_s^{(j)}$. If $t_s^{(j)} < t_d$, then noting that $t_d - t_s^{(j)} \le M$, set $s_i^{(j)} \leftarrow s_i^{(j)} + 1$, for each $i \in \{1,..., t_d - t_s^{(j)}\}$, and terminate, using these revised supply values. Otherwise, return to Step 1.

5.2 Computational Results

The proposed algorithm was coded in Fortran and was implemented on an IBM 3090 computer. To solve the linear programming bounding problems, the mathematical programming package MINOS was used as an imbedded subroutine. The transportation problems were solved using an improved specialized version of NETFLO, the network flow code developed by Kennington and Helgason (1980). For small sized problems, where $MN \le 45$, the optimality criterion α was taken as 0.001, whereas for larger sized problems it was taken as 0.01. At the beginning of each bounding step, if the cumulative execution time exceeded 370 cpu seconds, the execution of the algorithm was halted prematurely. The execution time for the input of the data for the problems and the output of the statistics are not included in the reported total execution time.

Table 4 presents the results for the case where problem LY_{Ω_1} was used as the bounding linear program. With our time restriction, the largest problem solved to completion has $(M, N) = (5, 16)$ for which Problem P2 has 80 variables and 21 constraints, and the initial linear program LY_{Ω_1} has 760 variables and 1381 constraints (other than nonnegativity restrictions). However, except for the problem $(M, N) = (4, 8)$, for which Problem P2 has 32 variables and 12 constraints, and the initial

Table 4. Results of the computational experiments with the algorithm using the bounding linear program LY_{Ω_1}.

Problem	M	N	v^0	v^1	n_g	n_f	n_f^{LY}	n_{opt}	n_{opt}^{α}	t	t_{LY}	n_{LY}	α
1	3	5	1.0508	1.0508	59	30	13	1	1	12.77	92.8	42	0.001
2	3	5	1.0130	1.1044	9	5	4	3	2	4.23	96.5	8	0.001
3	3	5	1.0085	1.0085	3	2	2	1	1	1.75	97.7	3	0.001
4	3	10	1.0154	1.0154	35	18	17	1	1	91.88	97.3	34	0.001
5	3	15	1.0100	1.0100	11	6	6	10	1	147.17	98.9	11	0.001
6	3	20	1.0073	1.0073	1	1	1	1	1	92.21	99.8	1	0.01
7	3	20	1.0106	1.0107	3	2	2	3	1	205.88	99.4	3	0.01
8	4	6	1.0106	1.0145	25	13	13	10	2	20.16	96.5	25	0.001
9	4	6	1.0014	1.0014	3	2	2	1	1	3.41	97.6	3	0.001
10	4	6	1.0126	1.0144	63	32	31	4	4	47.86	94.9	62	0.001
11	4	8	1.0196	1.0334	71	35	34	25	25	122.97	96.7	70	0.001
12	4	8	1.0098	1.0134	105	53	46	101	101	145.69	95.3	98	0.001
13	4	10	1.0097	1.0122	5	3	3	5	1	28.06	97.6	5	0.01
14	4	10	1.0114	1.0115	7	4	4	4	1	34.23	97.2	7	0.01
15	4	10	1.0114	1.0138	15	8	8	10	1	72.26	97.8	15	0.01
16	4	15	1.0126	1.0126	5	3	3	2	1	149.11	99.2	5	0.01
17	4	15	1.0072	1.0072	1	1	1	1	1	31.47	99.0	1	0.01
18	4	20	1.0070	1.0070	1	1	1	1	1	113.03	99.6	1	0.01
19	4	20	1.0046	1.0047	1	1	1	1	1	111.72	99.5	1	0.01
20	4	20	1.0099	1.0099	1	1	1	1	1	113.03	99.5	1	0.01

Table 4. continued

Problem	M	N	v^0	v^1	n_g	n_f	n_f^{LY}	n_{opt}	n_{opt}^{α}	t	t_{LY}	n_{LY}	α
21	4	24	1.0188	1.0188	2	0	0	1	1	370.00+	99.6	2	0.01
22	5	12	1.0090	1.0132	7	4	4	7	1	166.92	99.0	7	0.01
23	5	12	1.0073	1.0250	13	5	5	13	1	238.39	98.7	13	0.01
24	5	16	1.0071	1.0156	3	1	1	3	1	130.43	99.1	3	0.01
25	5	16	1.0093	1.0164	3	1	1	3	1	195.61	99.1	3	0.01
26	5	16	1.0096	1.0096	1	1	1	1	1	90.76	99.6	1	0.01
27	5	16	1.0264	1.0264	9	3	3	2	1	370.00+	99.3	8	0.01
28	5	16	1.0165	1.0259	4	0	0	2	1	370.00+	99.5	3	0.01
29	5	20	1.0117	1.0151	4	1	1	2	1	370.00+	99.3	3	0.01
30	6	14	1.0144	1.0144	4	0	0	1	1	370.00+	99.5	3	0.01

Legend: v^0 = initial upper bound / final incumbent, v^1 = initial upper bound / initial incumbent,

n_g = number of branch and bound nodes generated, n_f = number of nodes fathomed,

n_f^{LY} = number of nodes fathomed by LY_{Ω_1}, n_{opt} = number of nodes generated before the final incumbent z^* is found,

n_{opt}^{α} = number of nodes generated before a lower bound at most αz^* away from z^* is found,

t = total execution time in cpu seconds (370.00^+ indicates a premature termination after 370.00 cpu seconds),

t_{LY} = percentage of t spent for solving LY_{Ω_1}, n_{LY} = number of times Problem LY_{Ω_1} is solved,

α = optimality criterion as defined in Section 4.5.

linear program LY_{Ω_1} has 176 variables and 300 constraints (other than nonnegativity restrictions), a near-optimal solution which is at most αz^* away from z^*, where z^* is the final incumbent solution, was found at the early stages of the algorithm. Furthermore, the initial linearization based upper bound is usually within 1-1.5% of the optimal solution value, except for a few cases, like for problem 1 where this gap is 5.08%. Note that particularly for large size problems, fluctuations in computational time for the same sized problems, e.g. for problems 16 and 17, are due to the substantial amount of time required to solve even a few additional linear programs when an early fathoming does not occur.

Table 5 reports on the results for the case where the two step approach (LY_1 followed by LY_2) has been used for the linearization based upper bounding scheme. Here, a near-optimal solution has always been found at the early stages of the algorithm, thereby prompting that it can be terminated prematurely without much loss in solution quality. This facilitates a good heuristic scheme that can terminate after analyzing node zero. Also, compared to the results of Table 4, computational times are more uniform with respect to the number of nodes generated. For example, compare the results for the problems of size $(M, N) = (5, 16)$. However, note that the quality of the bounds produced by the pair of linear programs LY_1-LY_2 has deteriorated with respect to that produced by LY_{Ω_1}. As a result several more nodes need to be generated to verify optimality, and four problems solved using LY_{Ω_1} remain unsolved in this case (see Problems 6, 18, 19, 25). However, for the problems solved by both methods, the overall effort remains competitive.

From both Table 4 and Table 5, it is clear that the linearization based upper bound calculations are the main computational burden of the algorithm. On the other hand, the initial upper bound as well as the few number of nodes enumerated exhibits their substantial fathoming power. Our next set of trial runs shows that these linearization based upper bounds are crucial for the performance of the algorithm.

To test the compromise between solution effort and the fathoming power of the linearization based bounds, we also made a set of runs using only the objective function based functional upper bounds of Section 3.1 at Step 3 of the algorithm described in Section 4.6. Accordingly, the branching variable selection rule of Section 4.3 was modified by taking $\bar{w}$ as the solution of best

Table 5. Results of the computational experiments with the algorithm using the bounding linear programs LY_1 and LY_2.

Problem	M	N	v^0	v^1	n_g	n_f	n_f^{LY}	n_{opt}	n_{opt}^{α}	t	t_{LY}	n_{LY_1}	n_{LY_2}	α
1	3	5	1.0507	1.0507	59	30	12	1	1	15.69	92.5	41	29	0.001
2	3	5	1.0092	1.1003	3	2	2	2	2	2.32	95.7	3	2	0.001
3	3	5	1.0306	1.0306	5	3	2	3	1	2.93	96.2	4	4	0.001
4	3	10	1.0227	1.0227	51	26	26	1	1	94.76	94.0	51	45	0.001
5	3	15	1.0079	1.0079	19	10	10	14	1	137.98	97.4	19	18	0.001
6	3	20	1.0347	1.0347	14	5	5	1	1	370.00$^+$	98.2	13	13	0.01
7	3	20	1.0155	1.0162	9	5	5	5	1	227.20	97.3	9	9	0.01
8	4	6	1.0106	1.0107	49	25	24	2	1	34.06	93.2	48	36	0.001
9	4	6	1.0044	1.0079	3	2	2	2	2	3.89	94.0	3	3	0.001
10	4	6	1.0132	1.0149	75	38	38	3	3	49.13	91.9	75	60	0.001
11	4	8	1.0201	1.0339	89	45	45	50	3	101.84	92.3	89	74	0.001
12	4	8	1.0132	1.0132	43	22	22	5	1	63.73	93.0	43	40	0.001
13	4	10	1.0143	1.0168	19	10	10	12	1	56.33	93.4	19	19	0.01
14	4	10	1.0198	1.0198	29	15	15	3	1	67.67	91.7	29	29	0.01
15	4	10	1.0136	1.0175	24	12	12	12	1	54.88	92.5	24	23	0.01
16	4	15	1.0165	1.0165	21	11	11	8	1	286.55	96.8	21	21	0.01
17	4	15	1.0118	1.0118	3	2	2	1	1	26.77	94.0	3	3	0.01
18	4	20	1.0139	1.0139	20	7	7	1	1	370.00$^+$	93.9	19	19	0.01
19	4	20	1.0227	1.0227	15	5	5	12	1	370.00$^+$	96.1	14	14	0.01
20	4	20	1.0120	1.0120	3	2	2	1	1	87.74	96.0	3	3	0.01

Table 5. continued

Problem	M	N	v^0	v^1	n_g	n_f	n_f^{LY}	n_{opt}	n^{α}_{opt}	t	t_{LY}	n_{LY_1}	n_{LY_2}	α
21	4	24	1.0257	1.0284	9	2	2	2	1	370.00^+	96.5	8	8	0.01
22	5	12	1.0143	1.0173	11	6	6	4	1	84.99	94.8	11	11	0.01
23	5	12	1.0136	1.0136	9	5	5	3	1	68.96	95.3	9	9	0.01
24	5	16	1.0138	1.0227	12	6	61	7	1	154.90	94.0	12	12	0.01
25	5	16	1.0140	1.0174	39	16	16	29	1	370.00^+	91.2	38	38	0.01
26	5	16	1.0152	1.0159	13	7	7	8	1	292.50	96.9	13	13	0.01
27	5	16	1.0147	1.0269	25	11	11	5	5	370.00^+	95.8	24	24	0.01
28	5	16	1.0280	1.0307	23	9	9	3	1	370.00^+	94.8	22	22	0.01
29	5	20	1.0277	1.0292	13	3	3	10	1	370.00^+	95.1	12	12	0.01
30	6	14	1.0147	1.0176	24	11	11	10	1	370.00^+	95.3	23	23	0.01

Legend: v^0 = initial upper bound / final incumbent, v^1 = initial upper bound / initial incumbent,

n_g = number of branch and bound nodes generated, n_f = number of nodes fathomed,

n_f^{LY} = number of nodes fathomed by LY_1 or LY_2, n_{opt} = number of nodes generated before the final incumbent z^* is found,

n^{α}_{opt} = number of nodes generated before a lower bound at most αz^* away from z^* is found,

t = total execution time in cpu seconds (370.00^+ indicates a premature termination after 370.00 cpu seconds),

t_{LY} = percentage of t spent for solving LY_1 and LY_2, n_{LY_1} = number of times Problem LY_1 is solved,

n_{LY_2} = number of times Problem LY_2 is solved, α = optimality criterion as defined in Section 4.5.

Table 6. **Results of the computational experiments with the algorithm using only the functional upper bounds.**

Problem*	M	N	v^0	v^1	n_g	n_f	n_f^{UB}	n_{opt}	n_{opt}^{α}	t	t_{UB}	n_{UB}	α
2	3	5	1.0595	1.1551	58	30	15	21	21	0.41	97.6	43	0.001
8	4	6	1.2228	1.2284	3344	1673	830	8	8	56.51	97.7	2501	0.001
13	4	10	1.1734	1.1763	9127	4558	1694	1	0	370.00^+	99.9	6262	0.01

Legend:		
	v^0	= initial upper bound / final incumbent,
	v^1	= initial upper bound / initial incumbent,
	n_g	= number of branch and bound nodes generated,
	n_f	= number of nodes fathomed,
	n_f^{UB}	= number of nodes fathomed by functional upper bounds,
	n_{opt}	= number of nodes generated before the final incumbent z^* is found,
	n_{opt}^{α}	= number of nodes generated before a lower bound at most αz^* away from z^* is found,
	t	= total execution time in cpu seconds (370.00^+ indicates a premature termination after 370.00 cpu seconds),
	t_{UB}	= percentage of t spent for evaluation of functional upper bounds,
	n_{UB}	= number of times functional upper bounds are evaluated,
	α	= optimality criterion as defined in Section 4.5.

* Problems are numbered according to Table 4.

performing functional upper bound, and using this at Step 3 of this procedure. The results in Table 6 show that while these bounds can be computed relatively quickly, this does not compensate for their comparative lower quality. Hence, in an overall algorithmic scheme, these bounds are not competitive.

Since we cannot totally disregard the linearization based upper bounds based on the foregoing observations, the next strategy we attempted was to compute them only at some designated nodes of the branch and bound tree. In the first such approach, we attempted to derive a strongest surrogate Bender's constraint (see Rardin and Unger, 1976) at node zero via the initial linear program, and use this at subsequent nodes as a bounding function in the spirit of the functions of Section 3.1. Let us rewrite LY_{Ω_1}, by taking $l_{ik} = 0,\ \forall\ i = 1,\dots, M,\ k = 1,\dots, N$, in matrix form as follows.

$$\begin{aligned} \text{maximize} \quad & cy \\ \text{subject to} \quad & Ay + Bw \leq b \\ & Dy + Ew = f \\ & Gw = g \\ & w \geq 0,\ y \geq 0 \end{aligned} \tag{42}$$

where $W \equiv \{w : Gw = g,\ w \geq 0\}$. Let $\bar{\alpha}$ and $\bar{\beta}$ be the optimal Lagrange multipliers in (42) corresponding to the constraints $Ay + Bw \leq b$, and $Dy + Ew = f$ respectively. Then the following problem gives an upper bound on Problem P2.

$$\begin{aligned} \text{SSB}: \quad \text{maximize} \quad & \bar{\alpha}(b - Bw) + \bar{\beta}(f - Ew) \\ \text{subject to} \quad & Gw = g \\ & w \geq 0 \end{aligned} \tag{43}$$

In our computational experiments with this method, for any basic feasible solution of Problem SSB, nearly all the components of the projection of the objective function in the nonbasic space took a value of zero. That is, maximizing or minimizing the objective function of SSB over the set W gives the same value, which makes this method non-useful. This result also indicates that we cannot utilize a reduced cost based cut in the logical tests.

In a second approach, we tried to avoid solving bounding linearized problems which might not lead to fathoming. Here, the important issue is to predict such cases with a reasonable accuracy. If we fail to solve a linearized problem which would have led to fathoming at some node, then we may need to solve more than one, although smaller, linearized problems along the branches of this node. Otherwise, if we avoid solving an inconsequential linearized problem, then we will have conserved computational effort. Accordingly, the following criterion was used to determine whether to solve a linearized problem at any branch and bound node t. Namely, we solve the linear program at any node t provided

$$\frac{z_{UB}^{(t)}}{(1+\alpha)z^*} \frac{z_{LY}^{(1)}}{z_{UB}^{(1)}} \leq q_1, \tag{44}$$

where $z_{UB}^{(t)}$ is the smallest functional upper bound value at node t, $z_{UB}^{(1)}$ and $z_{LY}^{(1)}$ are respectively the smallest functional upper bound value and the value of the linearized problem at the initial node, and where z^* and α are defined as in (41). From our computational experiments, a recommended range for the value of q_1 emerged as 1 ∓ 0.05. At the nodes where a linear program is not solved, the branching decision is based on the solution of the best performing functional upper bound according to Step 3 of the branching variable selection rule of Section 4.3. After implementing this version of the algorithm on different problems by using various values for q_1, we found that although the predictive power of (44) was satisfactory, Step 3 of the branching variable selection rule led to relatively poor branching decisions, even when we put a limit on the number of consecutive omissions of linear program solutions. Hence, it appears that the choice of branching variables afforded by the bounding linear programs also plays a crucial role in determining the efficiency of the algorithm.

The third approach we attempted involved branching on two variables at a time. Here, the motivation is to base the branching decision on the solution of the linearized problem, while skipping the solution of nonpromising linear programs. The first branching variable w_{pq} is found as described in Section 4.3. Determination of a second branching variable is based on Step 1 of the branching variable selection rule. Let (s, t) be the arc which gives the second highest value, α'_1 , for the maximization problem (38a). If

$$\frac{\alpha_1 - \alpha'_1}{\alpha_1} \leq q_2 \quad \text{and} \quad \frac{z_{LY}}{(1+\alpha)z^*} > q_3, \tag{45}$$

where q_2 and q_3 are prescribed constants, α_1 is as defined in (38a), and where z_{LY} is the solution of the linearized problem, then w_{st} is selected as a candidate for the second branching variable. Note that the second criterion in (45) recommends the selection of two branching variables only when the linear programming bound is not likely to fathom after branching on w_{pq} alone.

Now, after w_{pq} is "zero fixed" or "positive fixed," if w_{st} still remains to be a free variable at the end of the logical tests and the cycle prevention operations, then it is used as the second branching variable before solving the bounding linear program. In our computational experiments, where bounding linear programs LY_1 and LY_2 were used, some improvement was obtained although this was not consistent. For example, by fixing $q_2 = 0.2$ and $q_3 = 1.0045$, the improvement for problem 8 of Table 5 with $n_{LY_1} = 36$, $n_{LY_2} = 24$, and $t = 24.96$ did not repeat for problem 11 of Table 5 with $n_{LY_1} = 105$, $n_{LY_2} = 82$, and $t = 121.53$, where n_{LY_1} , n_{LY_2}, and t are defined as in Table 5. Nevertheless, this approach is the most promising one among the others we attempted.

For problems of size larger than those of Tables 4 and 5, we used the linearized problems LY_1 and LY_2 along with the functional upper bounds at Step 0 of the branch and bound algorithm without any further branching, as a linear programming based heuristic. Also, due to the large size of the problems, the limit on the maximum number of iterations in the improving feasible direction algorithm was increased to 150. With this approach problems of size up to $(M, N) = (6, 35)$, with 210 variables and 41 constraints, were solved. The results in Table 7 show that the upper bound value given by the linearized problem LY_2 is within 2.38-4.52% of the incumbent solution found at node zero. More importantly, the quality of this heuristic solution does not appear to worsen as problem size increases. The linearized bounding problem LY_{Ω_1} has not been used in a similar approach due to its potentially greater computational effort. For example, for a problem with

$(M, N) = (4, 45)$, the linear program LY_2 has 4320 variables and 1069 constraints, but although having the same number of variables, the linear program LY_{Ω_1} would have 7260 more constraints.

To summarize, we have attempted to reduce the main computational burden of the algorithm which employs bounding linear programs. In our first approach, relying on only the functional upper bounds we disregarded the bounding linear programs. While the computational time for the functional upper bounds is much less than the computational time for solving linearized problems, this advantage does not yield a favorable trade-off because of their relatively poorer bound quality. So, our next approach involved omitting the solution of linearized problems at some nodes of the branch and bound tree. In this context, using a strongest surrogate Bender's constraint based method is not useful due to the nature of the linearized problem. Also, omitting linearized problems based on predicting those which might not lead to fathoming did not improve the performance of the algorithm. But, on the other hand it revealed an important characteristic of the algorithm, namely, that the algorithm is very sensitive to branching decisions, and moreover, the linearization based bounding scheme is crucial for good branching decisions. In the light of these observations, our last attempt involved branching on two variables at a time using the solution of the linearized problems. Although this method improved the performance of the algorithm, this improvement was not consistent. Nonetheless, this strategy might be a promising modification to implement. For large sized problems, by using the functional upper bounds and the linearized bounding problems LY_1 and LY_2 only once, without further branching, we solved problems of size up to $(M, N) = (6, 35)$, where in the worst case the magnitude of the gap between the incumbent and the upper bound value was 4.52% of the incumbent, for a problem of size $(M, N) = (4, 45)$. Moreover, there was no discernible deterioration in the quality of the heuristic solution obtained as problem size increases. Due to the size of the linear programs that need to be solved, the performance of this approach can be further enhanced by using an interior point linear program solver as opposed to the simplex method.

Table 7. Results on the performance of the linear programs LY_1-LY_2 at node zero.

M	N	v^1	t	n_v	n_{c_1}	n_{c_2}
4	30	1.0376	116.44	1980	274	626
4	40	1.0341	347.13	3440	364	1059
4	45	1.0452	434.19	4320	409	1069
5	36	1.0402	430.58	3510	396	695
6	30	1.0238	313.60	2970	396	695
6	35	1.0278	151.29	3990	461	689
6	35	1.0350	475.77	3990	461	725

Legend: v^1 = upper bound value / incumbent value,

t = total execution time in cpu seconds,

n_v = number of variables in linearized problems

n_{c_1} = number of constraints in LY_1,

n_{c_2} = number of constraints in LY_2,

(n_{c_1}, and n_{c_2} do not include nonnegativity restrictions).

Chapter VI

Conclusions and Recommendations for Further Research

In this thesis, we have studied the squared-Euclidean distance location-allocation problem subject to balanced transportation constraints, which has been shown to be transformable to a quadratic convex maximization problem. The special structure of the objective function was exploited to derive various upper bounding linear functions. However, considerably tighter and more powerful upper bounds were obtained using a special linearization technique. Here, various nonlinear implied constraints were first generated using products of original constraints with specific polynomial factors or terms, and the resulting nonlinear terms were subsequently linearized by making variable substitutions in order to obtain a relaxed linear program. The effectiveness of different classes of linearized constraints showed some variations, and a bounding linear program (LY_{Ω_1}) which best compromised between solution quality and effort was selected based on computational experimentation.

Using the proposed upper bounding schemes, a branch and bound algorithm was developed by partitioning the problem according to the dichotomy that a variable is either zero or positive at an optimal (vertex) solution. To maintain a search among vertex solutions, the forest formed by the "positive fixed" variables was always kept cycle free by fixing to zero the variables which otherwise would form a cycle. For a given partial solution, the bounds on the variables were further tightened by logical tests, by exploiting the special structure of the transportation constraints. The solution of the relaxed linear program was used in the hierarchical branching variable selection rule, which was based on a combined measure of the amount by which the above mentioned variable substitutions are violated, and on their contribution to the upper bound value.

Using this algorithm, along with LY_{Ω_1}, we solved problems of size $(M, N) = (5, 16)$, having 80 variables and 21 constraints, and for which the initial bounding linear program has 760 variables and 1381 constraints, within a reasonable computational effort. To reduce the computational time for solving a single bounding linear problem for larger sized problems, another strategy was adopted which involved solving a reduced size linearized problem LY_1 by omitting some of the constraints of the original linearized problem, and subsequently solving an augmented linear program LY_2 by including the violated constraints. Although this strategy usually gave higher upper bounds, the performance of the algorithm was still competitive with the previous case. This strategy has also been used to obtain provably good quality incumbents for larger sized problems by solving the linearized problems just once at node zero. With this heuristic, we solved problems of size $(M, N) = (6, 35)$, which have 210 variables and 41 constraints, within a reasonable computational time.

Our computational experience with the algorithm showed that solving the linear bounding problems constitutes the main computational burden of the algorithm. Hence, the performance of the algorithm can be further enhanced using a more efficient interior point linear programming routine as opposed to the simplex method. However, if we totally disregard the bounding linear programs, and use only the functional upper bounds, then this was shown to be relatively ineffective as an exact approach. Nonetheless, although the upper bounds obtained via these functions are not very strong, they do recover good quality solutions when an improving feasible direction routine is applied to the resulting allocations. Hence, a premature termination of an algorithm using these bounds is recommended as a heuristic for problems larger than the ones that can be handled by the LY_1-LY_2 approach.

The improvement in the performance of the algorithm was also sought by omitting the solution of linearized problems at some nodes of the branch and bound tree, based on estimating when such a solution might not lead to fathoming. Since using a strongest surrogate Bender's constraint for computing bounds was shown to be non-useful at the nodes where the linearized problem was omitted, only the functional upper bounds were evaluated, and the branching decision was based on the solution of the tightest functional upper bound. However, this approach led to poor branching decisions, which also shows that the algorithm is very sensitive to branching decisions. Based on this observation, after solving a linearized problem, branching on two variables at a time was considered. This approach improved the performance of the algorithm, although this was not consistent.

Finally, in Appendix B, we show that a more general squared-Euclidean distance location-allocation problem with a weighted objective function, and possibly with unbalanced transportation constraints is also equivalent to a convex maximization problem. But in this case the objective function of the convex maximization problem is fractional, and hence, the algorithm in its proposed form cannot be applied directly. However, one can adopt a specialized strategy as proposed herein by generating linear bounding problems via an adaptation of Charnes and Cooper's (1962) transformation for linear fractional problems and the factor product followed by the linearization technique presented in Section 3.2. We leave this as a topic for further research. Also, another subject for further investigation is to generalize the solution concept to other discrete or continuous special classes of polynomial programming problems.

Appendix A

A Data Structure for the Cycle Prevention Procedure

The cycle prevention procedure is designed to identify the arcs which can form a cycle and related cut-sets in the forest that is formed by the arcs in J^+ at some stage of the branch and bound algorithm. In this section, to implement the cycle prevention procedure, we will propose a data structure which does not require an explicit representation of the forest. This data structure involves pointers and circular linked lists which can easily be implemented by using the array structure that is common to many high level computer programming languages.

Throughout this section we will assume that the nodes are numbered as $1,..., M, M+1, ..., M+N$, where the first M indices correspond to the supply nodes, and the last N indices correspond to the demand nodes. Let C be a component of the current forest. Component C is identified by one of its supply nodes. If component C does not contain any supply node then, due to the transportation structure, C consists of a single demand node, and this single demand node identifies C. Note that this is the only case where a demand node identifies a component. After this preliminary information, we can now describe the arrays that are used in the procedure.

Array COMP of size $(M+N)$ stores the nodes that identify the component of each node. Each component of the forest is represented by two circular linked lists: one is for the supply nodes, and the other is for the demand nodes. All such lists are stored in a single array of dimension $(M+N)$ called NEXT. If there is a single supply node j in component C, then j points to itself, that is NEXT(j) $=j$. Otherwise, for each supply node j in C, NEXT(j) points to another supply node in C, but no two supply nodes point to the same supply node. The same structure also applies for the demand nodes. Notice that a supply node does not point to a demand node and vice versa. Another array, LASTSUP, of dimension M, stores the supply nodes which finally point back to the corresponding component identifying supply node. That is, if j is the supply node that identifies the component C, and $j =$ NEXT(i), where $i \in \{1,\ldots,M\} \cap C$, then LASTSUP($j$) $= i$. There is no counterpart of array LASTSUP for demand nodes, while another array LDEM of dimension M provides access to the demand nodes of a component having the identifier of that component. That is, for a component C, which has supply node j as its identifier, LDEM(j) points to a demand node of component C, if there is one. Once a demand node of C is known, all the other demand nodes in C can be easily reached by using the array NEXT. Finally, two arrays, POPS of dimension M and POPD of dimension M, respectively give the number of supply nodes and the number of demand nodes in the components which have at least one supply node. We will deal with the components having single demand nodes in a different fashion, so there is no corresponding entry in POPS and POPD for such components. The foregoing arrays are initialized as follows.

$$\begin{aligned}
&\text{COMP}(j) = j && j = 1,\ldots,M+N,\\
&\text{NEXT}(j) = j && j = 1,\ldots,M+N,\\
&\text{LASTSUP}(j) = j && j = 1,\ldots,M,\\
&\text{POPS}(j) = 1 && j = 1,\ldots,M,\\
&\text{POPD}(j) = 0 && j = 1,\ldots,M.
\end{aligned} \tag{46}$$

Array LDEM does not need to be initialized due to the operations described next.

Now, using the above data structure, let us describe the cycle prevention procedure when a new link (i, k) is introduced to the current forest. Let C_1 be the component that contains supply node i, and let C_2 be the component that contains demand node k . Note that $C_1 \not\equiv C_2$. First, all the arcs between the components C_1 and C_2 except arc (i, k) will be enumerated in order to fix the flows on them to zero. (As far as this appendix is concerned, we will be interested in only the enumeration of such arcs, however assuming that previous applications of this routine have fixed the flows on other similar arcs to zero.) Next, components C_1 and C_2 will be linked to form a single component. According to the number of supply and demand nodes in components C_1 and C_2 one of the following four cases occurs.

1) Each of the components C_1 and C_2 contain at least one supply node and one demand node.

2) Component C_1 contains at least one supply node and one demand node, while component C_2 consists of the single demand node k.

3) Component C_2 contains at least one supply node and one demand node, while component C_1 consists of the single supply node i.

4) Component C_1 consists of the single supply node i , and component C_2 also consists of the single demand node k.

We can easily determine which case has occured by checking the following two conditions. If $i = \text{COMP}(i)$ and $\text{POPD}(i) = 0$, then component C_1 consists of the single supply node i. Otherwise, C_1 contains both supply and demand nodes. If $k = \text{COMP}(k)$, then component C_2 consists of the single demand node k. Otherwise, C_2 contains both supply and demand nodes.

Now, we will consider the first case. Let the components C_1 and C_2 be identified by the supply nodes s_1 and s_2 respectively, that is $s_1 = \text{COMP}(i)$, and $s_2 = \text{COMP}(k)$. First, all the arcs from the supply nodes of C_2 to the demand nodes of C_1 are enumerated. Starting from s_2, for each supply node in C_2, all the demand nodes in C_1 are identified, each time starting from d_1, where $d_1 = \text{LDEM}(s_1)$, and using the array NEXT in nested do-loops which are controlled by $\text{POPS}(s_2)$ and $\text{POPD}(s_1)$. Second, all the arcs from the supply nodes of C_1, except i , to the demand nodes of C_2 are enumerated. Starting from $\text{NEXT}(s_1)$, for each supply node in C_1 up to i, but not including i, all the demand nodes in C_2 are identified, each time starting from d_2 , where

$d_2 = \text{LDEM}(s_2)$, and using the array NEXT in nested do-loops which are controlled by $\text{POPS}(s_1) - 1$ and $\text{POPD}(s_2)$. Finally, for the supply node i, starting from $\text{NEXT}(k)$, all the demand nodes in C_2 except k are identified by using the array NEXT in a do-loop controlled by $\text{POPD}(s_2) - 1$.

Since all the flows on the arcs that can form a cycle are fixed to zero, we next combine the components C_1 and C_2 by merging C_2 into C_1. First, all the nodes in C_2 become the members of C_1 by setting

$$\begin{aligned} &\text{COMP}(j) \leftarrow s_1 \qquad \text{for all the nodes } j \in C_2, \\ &\text{POPS}(s_1) \leftarrow \text{POPS}(s_1) + \text{POPS}(s_2), \\ &\text{POPD}(s_1) \leftarrow \text{POPD}(s_1) + \text{POPD}(s_2). \end{aligned} \tag{47}$$

Second, the supply nodes in C_2 are merged into the supply nodes of C_1 by setting

$$\begin{aligned} &\text{NEXT}(\text{LASTSUP}(s_1)) \leftarrow s_2, \\ &\text{NEXT}(\text{LASTSUP}(s_2)) \leftarrow s_1, \\ &\text{LAST}(s_1) \leftarrow \text{LAST}(s_2). \end{aligned} \tag{48}$$

Finally, the procedure is completed by merging the demand nodes in C_2 into the demand nodes in C_1 as follows.

$$\begin{aligned} &l \leftarrow \text{NEXT}(\text{LDEM}(s_1)), \\ &\text{NEXT}(\text{LDEM}(s_1)) \leftarrow \text{NEXT}(\text{LDEM}(s_2)), \\ &\text{NEXT}(\text{LDEM}(s_2)) \leftarrow l. \end{aligned} \tag{49}$$

For the second and the third cases, operations for the enumeration of the arcs can easily be adapted from the first case. However, the merging operations show slight differences. For the second case, after updating the arrays COMP and POPD as $\text{COMP}(k) \leftarrow s_1$, and $\text{POPD}(s_1) \leftarrow \text{POPD}(s_1) + 1$, since the component C_2 does not contain any supply node, only the demand node k is merged into the demand nodes of C_1 by replacing $\text{LDEM}(s_2)$ by k in (49). For the third case, the single supply node i is merged into the supply nodes of C_2 by interchanging the roles of s_1 and

s_2 in (48), and by obvious updates in arrays COMP and POPS. For the fourth case, since only the arc (i, k) is involved in the cycle prevention operations, no flow is fixed to zero. Components C_1 and C_2 are combined as follows.

$$\begin{aligned} &\mathrm{COMP}(k) \leftarrow i \\ &\mathrm{POPD}(i) \leftarrow 1 \\ &\mathrm{LDEM}(i) \leftarrow k \end{aligned} \tag{50}$$

In the branch and bound algorithm, as a new variable is fixed to a positive value, the current forest is updated as described above. However, when a fathoming occurs, at least one link in the current forest is broken, and so the affected components need to be re-built. First, the affected components are broken down to single node components by using (46). Then, the valid links of the affected components are introduced to the new forest one at a time by using the above operations.

Appendix B

Weighted Objective Function

In this section, we will show that a squared-Euclidean distance problem with a weighted objective function is equivalent to a convex maximization problem. Here, we also permit the transportation constraints $w \in W$ to be unbalanced. Let c_{ij} be a strictly positive weight associated with arc (i, j), $\forall i = 1, \ldots, M, j = 1, \ldots, N$. After including these weights in the objective function of Problem P, a more generalized problem can be given as follows.

$$\begin{aligned} \text{PW}: \ \text{minimize} \ & \sum_{i=1}^{M} \sum_{j=1}^{N} c_{ij} w_{ij} \left[(x_i - a_j)^2 + (y_i - b_j)^2 \right] \\ \text{subject to} \ & w \in W \end{aligned} \tag{51}$$

As in Chapter 1, if we fix $w = w^1$, the unconstrained minimum of the strictly convex objective function is obtained at the solution

$$x_i^1 = \frac{\sum_{j=1}^{N} c_{ij} w_{ij}^1 a_j}{\sum_{j=1}^{N} c_{ij} w_{ij}^1}, \text{ and } y_i^1 = \frac{\sum_{j=1}^{N} c_{ij} w_{ij}^1 b_j}{\sum_{j=1}^{N} c_{ij} w_{ij}^1}, \qquad i = 1, \ldots, M. \tag{52}$$

To simplify the notation, let us define the vectors $w_i = [w_{i1}, \ldots, w_{iN}]^t$, and similarly, $\bar{c}_i = [c_{i1}a_1, \ldots, c_{iN}a_N]$, $\hat{c}_i = [c_{i1}b_1, \ldots, c_{iN}b_N]$, and $c_i = [c_{i1}, \ldots, c_{iN}]$, $\forall i = 1, \ldots, M$. After dropping the superscripts in (52), we can project Problem PW onto the w-space by substituting (52) in the objective function of Problem PW to obtain the equivalent problem

$$\text{PW1 : minimize } \sum_{i=1}^{M} \sum_{j=1}^{N} c_{ij} w_{ij} \left[\left(\frac{\bar{c}_i w_i}{c_i w_i} - a_j \right)^2 + \left(\frac{\hat{c}_i w_i}{c_i w_i} - b_j \right)^2 \right]$$
$$\text{subject to } w \in W$$

Simplifying the objective function of Problem PW1, we can rewrite it as

$$\sum_{i=1}^{M} \sum_{j=1}^{N} \frac{c_{ij} w_{ij}}{(c_i w_i)^2} \left[(\bar{c}_i w_i)^2 + a_j^2 (c_i w_i)^2 - 2a_j (c_i w_i)(\bar{c}_i w_i) + (\hat{c}_i w_i)^2 + b_j^2 (c_i w_i)^2 - 2b_j (c_i w_i)(\hat{c}_i w_i) \right]$$

$$= \sum_{i=1}^{M} \frac{(\bar{c}_i w_i)^2}{(c_i w_i)} + \sum_{i=1}^{M} \frac{(\hat{c}_i w_i)^2}{(c_i w_i)} + \sum_{i=1}^{M} \sum_{j=1}^{N} c_{ij} (a_j^2 + b_j^2) w_{ij} - 2 \sum_{i=1}^{M} \frac{(\bar{c}_i w_i)^2}{(c_i w_i)} - 2 \sum_{i=1}^{M} \frac{(\hat{c}_i w_i)^2}{(c_i w_i)}$$

$$= \sum_{i=1}^{M}\sum_{j=1}^{N} c_{ij}(a_j^2 + b_j^2)w_{ij} - \sum_{i=1}^{M} \frac{(\bar{c}_i w_i)^2 + (\hat{c}_i w_i)^2}{(c_i w_i)}$$

Consequently, Problem PW1 is equivalent to the following problem.

$$\text{PW2}: \ \text{maximize} \ \sum_{i=1}^{M} \frac{(\bar{c}_i w_i)^2 + (\hat{c}_i w_i)^2}{(c_i w_i)} - \sum_{i=1}^{M}\sum_{j=1}^{N} c_{ij}(a_j^2 + b_j^2)w_{ij} \qquad (53)$$
$$\text{subject to} \ \ w \in W$$

Since the objective function of Problem PW2 is fractional, we cannot apply the proposed branch and bound algorithm to Problem PW2 directly. However, as the following result indicates, the objective function of Problem PW2 is convex over the set W, for positive objective weights.

PROPOSITION 4: Let the objective weights c be such that $c_i w_i > 0 \ \forall i = 1,\dots, M, \ \forall w \in W$. (For example as with $c_{ij} > 0 \ \forall i = 1,\dots, M, j = 1,\dots, N$). Then, the objective function in (53) is convex over W.

PROOF: Since the linear component of the objective function in (53) does not affect the result, and the sum of the convex functions is also a convex function, it is sufficient to show that the following function is convex over W.

$$f(w_i) = \frac{(\bar{c}_i w_i)^2}{(c_i w_i)}, \qquad (54)$$

for some $i \in \{1,\dots, M\}$. The gradient of $f(w_i)$ is as follows.

$$\nabla f(w_i) = \frac{2(\bar{c}_i w_i)}{(c_i w_i)} \bar{c}_i^t - \frac{(\bar{c}_i w_i)^2}{(c_i w_i)^2} c_i^t \qquad (55)$$

Therefore, the Hessian matrix , $H(w_i)$, of $f(w_i)$ is given as follows.

$$\begin{aligned} H(w_i) &= \frac{2}{(c_i w_i)^3}\left[(c_i w_i)^2 \bar{c}_i^t \bar{c}_i - (\bar{c}_i w_i)(c_i w_i)[\bar{c}_i^t c_i + c_i^t \bar{c}_i] + (\bar{c}_i w_i)^2 c_i^t c_i\right] \\ &= \frac{2}{(c_i w_i)^3}[(c_i w_i)\bar{c}_i^t - (\bar{c}_i w_i) c_i^t][(c_i w_i)\bar{c}_i - (\bar{c}_i w_i) c_i] \\ &\equiv \frac{2}{(c_i w_i)^3} p p^t \end{aligned} \tag{56}$$

where $p = [(c_i w_i)\bar{c}_i^t - (\bar{c}_i w_i) c_i^t]$. Hence, $H(w_i)$ is positive semi-definite over W under the hypothesis of the proposition, and this completes the proof. □

Appendix C

Using MINOS 5.1 as a Subroutine to Solve Linear Programs within a Fortran Program

In this appendix, we share our experience on using MINOS 5.1 as a subroutine for solving linear programs within a Fortran program We use capital letters for CMS commands and Fortran statements, while user parameters are referred to in lowercase italic letters.

First of all, to link the appropriate minidisk, enter 'LDISK JOF' from CMS on the VM1 system. The Read Password for this minidisk can be obtained from the Management Science Department. If you did not make a note of the filemode letter assigned by ldisk, enter 'Q DISK' from CMS. Now, the files on the JOF minidisk are accessible as "read-only" files by using 'FL * * *filemode*' command from CMS. To make alterations in any of these files, you need to get a copy of it on your A disk. A very important document, "Using MINOS 5.1 at Virginia Tech", is found in the Minos Script file, and it can be printed by using the script command which has been given in the same file. A second document is "MINOS 5.1 User's Guide", which has been prepared by the creators of the MINOS code. Solving some of the examples given in both documents is helpful to get acquainted with MINOS.

Now, let us go through the files that should be prepared before using MINOS.

1) SPECS FILE: This file specifies certain parameters to be used in input, execution, and output operations of MINOS. Each parameter has been explained in detail in MINOS 5.1 User's Guide. Some of the unwanted, diagnostic or informative type outputs of MINOS can be suppressed (or abbreviated) by defining appropriate parameters here. However, before experimenting on different values for the parameters related to the performance of the software, we recommend trying default values first. For example, defining a large optimality tolerance to speed up the algorithmic process of MINOS may result in an undesirable early termination that occurs before reaching an optimal solution.

2) RUNMINOS EXEC: This exec file, which is written in REXX, includes CMS file definitions, and controls the execution of MINOS. Suggestions on the modification of this file will be referred to subsequently.

3) MINOS FILES FILE: This file is local to Virginia Tech, and is used for naming the files that are going to be defined in RUNMINOS EXEC. The use of this file has been well explained in the first document.

Next, we will describe how to use MINOS within a Fortran program with an emphasis on the related file operations. MINOS 5.1 has been designed to handle nonlinear programs, and some subroutines particular for this purpose are not used for linear programs. However, these routines should be defined in your Fortran program, and they are available in the MAINLP FORTRAN file on the JOF disk.

In the Fortran program, the workspace for MINOS is defined by

DOUBLE PRECISION *Z(iws)*

DATA NWCORE */iws/*

where *iws* should be large enough to accommodate your linear program. To see if the workspace has been sufficient, check the output file after the execution terminates. For further information, also see the relevant SPECS file parameter definitions in MINOS 5.1 User's Guide.

As a subroutine, MINOS is called by

CALL MINOS1 *(Z, NWCORE)*

Data for a linear program is passed to MINOS by a strictly formatted MPS file. Read the corresponding section in MINOS 5.1 User's Guide, and study the MPS file examples, before writing your own code. In the Fortran program, this file should be created before calling MINOS. For some sections of the MPS file, two entries can be written on the same line (record). To speed up the writing process, take advantage of this feature. Also, try to avoid writing entries with a zero value, since by default, they will be initialized as zero.

One approach to handle the MPS file is to define it as a direct access file, by making the following modification in RUNMINOS EXEC. This permits moving freely within the file.

'FILEDEF MPS   DISK' FN.5 FT.5 FM.5 '(RECFM F LRECL 61 DISP MOD DSORG DA XTENT *filelength*)'

where *filelength* defines the length of the MPS file, and it should be large enough to accommodate your linear program. In the Fortran program, MPS file should be opened appropriately as follows before starting to write on it.

```
          OPEN(UNIT=unmps, FILE='MPS', ACCESS='DIRECT',
     &  FORM='FORMATTED', STATUS='NEW', RECL=61)
```

where *unmps* is the file unit number. To avoid unnecessary complications, it is advisable to use a unit number that is not associated with any other file. Notice that the status of the file is taken as 'NEW', assuming that you do not have an old or dummy MPS file as defined above. Later, we will address this issue once again. Now, we are ready to write data on the MPS file referred with the unit number *unmps*. Due to the access mode, write statements have a slightly different form as follows.

```
          WRITE  (unmps,formatno, REC = nrec)x
```

This write statement writes some data x formatted according to *formatno*, on the line (record) *nrec* of the direct access file connected to unit *unmps*. Note that, as opposed to a regular write (or read) statement for sequential files, the record number *nrec* should be defined explicitly in the program. That is, even if you are entering (or reading) data sequentially, you need to increment *nrec* in your program, such as $nrec = nrec + 1$, before each write (or read) statement. And finally, before calling MINOS, the MPS file should be closed by

CLOSE(*unmps*)

After solving the problem, MINOS writes the results to the SOLUTION file with the file unit number defined as in the SPECS file. To read the outputs of interest, we will suggest two different file operations. The first one is to open the SOLUTION file as a direct access file, after defining a suitable file length in RUNMINOS EXEC as

'FILEDEF SOLUTION DISK' FN.13 FT.13 FM.13 '(RECFM F LRECL 111 XTENT *filelength*)'

Note that MINOS leaves open all the files it has read from and has written to, and so the SOLUTION file should be closed before opening it as a direct access file by

CLOSE(*unsolspecs*)

where *unsolspecs* is the unit number defined in the SPECS file for the SOLUTION file. Since, the SOLUTION file already exists, it should be re-opened with STATUS='OLD' as follows.

OPEN(UNIT=*unsol*, FILE='SOLUTION', ACCESS='DIRECT',

& FORM='FORMATTED', STATUS='OLD', RECL=111)

where *unsol* is the file unit number which can be different from *unsolspecs* . The structure and the formatting of the SOLUTION file has been explained in the MINOS 5.1 User's Guide. Before reading an output from the SOLUTION file, you should know at which line (record) it resides and the corresponding read format. (See the MINOS 5.1 User's Guide for this information). In a similar fashion to the MPS file, to read an output, say x , on the line *nrec*, use the following read statement.

READ (*unsol*,*formatno*, REC = *nrec*)*x*

The SOLUTION file should be closed before a subsequent call to MINOS is made by

CLOSE(*unsol*)

In the second method, the SOLUTION file remains as a sequential file, and so it should be rewound before starting to read an output, by

REWIND(*unsolspecs*)

To skip lines in order to reach the line on which a particular output, say x, lies, dummy read statements, e.g. READ (*unsolspecs*, *), each of which advances to the next record (line), can be used. Then, x is read by READ (*unsolspecs,formatno*)x. The SOLUTION file should be either rewound by REWIND(*unsolspecs*), or closed by CLOSE(*unsolspecs*) before a subsequent call to MINOS is made.

If MINOS is called more than once in the Fortran program, before opening the MPS file for an update, it should be closed by using

CLOSE(*unmpsspecs*)

where *unmpsspecs* is the unit number defined in the SPECS file for the MPS file. Then, the MPS file can be re-opened by replacing 'NEW' with 'OLD' in the open statement given above, and then the required modifications can be made on this file. As before, the MPS file should be closed before calling MINOS. Also, before calling MINOS, the SPECS file should be rewound by

REWIND(5)

Here, the fixed unit number 5 is used due to the definition of the SPECS file in RUNMINOS EXEC. Note that the SPECS file need not be rewound before calling MINOS for the first time, but it should be rewound each subsequent time.

To summarize, the following list of operations need to be performed.

0) Do not forget to include dummy subroutine definitions, although they are not used for linear programs.

1) Define NWCORE and dimension Z.

2) Open the MPS file as a direct access file with STATUS='NEW'.

3) Close the MPS file.

4) Call MINOS.

5) Close the SOLUTION file (or rewind it if the second method is used).

6) Open the SOLUTION file as a direct access file (ignore this if the second method is used).

7) Close the SOLUTION file.

8) Open the MPS file as a direct access file with STATUS='OLD'.

9) Close the MPS file.

10) Rewind the SPECS file.

11) Call MINOS.

12) Continue as from point 6 above.

Note that the above procedure may not be the best way for your purposes, but we hope that it will help you make your program work. After then, you may experiment with other ways which might serve your purposes better. Here we suggest, before activating MINOS in your program, to make sure that your MPS file is correct in terms of formatting, structure, problem representation, etc.

Now, having your Fortran program ready, use the FORTVS2 compiler to compile it. For execution, issue the following statement from CMS.

RUNMINOS L *programname*

If you still have problems with the file operations, you may seek help from User Services and/or read related sections from the IBM FORTVS2 manuals, but remember that User Services does not support the MINOS package, and so your best recourse is to study the two documents mentioned earlier.

Finally, we highly recommend using the Batch operation mode after you have been convinced that your program works. Besides being much cheaper, the Batch mode can save you from problems that frequently arise due to virtual memory and disk storage restrictions.

References

Aikens, C. H. (1985) : Facility Location Models for Distribution Planning, *European Journal of Operational Research*, **22**: 263-279.

Avriel, M. (1980) : A Geometric Programing Approach to the Solution of Locational Problems, *Journal of Regional Science*, **20**: 239-246.

Bazaraa, M. S., J. J. Jarvis, and H. D. Sherali (1990) : *Linear Programming and Network Flows*, John Wiley & Sons, New York.

Beale, E. M. L., and J. A. Tomlin (1970) : Special Facilities in a General Mathematical Programming System for Non-convex Problems Using Ordered Sets of Variables, pp. 447-454 in J. Lawrence (ed.), *Proceedings of the Fifth International Conference on Operational Research*, Tavistock Publications, London.

Beale, E. M. L., and J. J. H. Forrest (1976) : Global Optimization Using Special Ordered Sets, *Math. Progr.*, **10**: 52-69.

Benson H. P. (1982) : On the Convergence of Two Branch-and-Bound Algorithms for Nonconvex Programming Problems, *JOTA*, **36**: 129-134.

Benson H. P. (1985) : A Finite Algorithm for Concave Minimization Over a Polyhedron, *Naval Res. Logist. Quart.*, **32**: 165-177.

Cabot, A. V., and R. L. Francis (1970) : Solving Nonconvex Quadratic Minimization Problems by Ranking the Extreme Points, *Oper. Res.*, **18**: 82-86.

Cabot, A. V. (1974) : Variations on a Cutting Plane Method for Solving Concave Minimization Problems with Linear Constraints, *Naval Res. Logist. Quart.*, **21**: 265-274.

Carrillo, M. J. (1977) : A Relaxation Algorithm for the Minimization of a Quasiconcave Function on a Convex Polyhedron, *Math. Progr.*, **13**: 69-80.

Charnes, A., and W. W. Cooper (1962) : Programming with Linear Fractionals, *Naval Res. Logist. Quart.*, **9**: 181-186.

Chen, R. (1984) : Location Problems with Costs Being Sums of Powers of Euclidean Distances, *Comput. & Ops. Res.*, **11**: 285-294.

Cooper, L. (1963) : Location-Allocation Problems, *Oper. Res.*, **11**: 331-343.

Cooper, L. (1964) : Heuristic Methods for Location-Allocation Problems, *SIAM Review*, **6**: 37-53.

Cooper, L. (1967) : Solutions of Generalized Locational Equilibrium Models, *Journal of Regional Science*, **7**: 1-18.

Cooper, L. (1972) : The Transportation-Location Problem, *Oper. Res.*, **20**: 94-108.

Eilon, S., C. D. T. Watson-Gandy, and N. Christofides (1971) : *Distribution Management: Mathematical Modelling and Practical Analysis*, Griffin, London.

Falk, J. E., and K. L. Hoffman (1976) : A Successive Underestimation Method for Concave Minimization Problems, *Math. of Oper. Res.*, **1**: 251-259.

Falk, J. E., and K. L. Hoffman (1986) : Concave Minimization Via Collapsing Polytopes, *Oper. Res.*, **34**: 919-929.

Francis, R. L., and J. A. White (1974) : *Facility Layout and Location: An Analytical Approach*, Prentice-Hall, Englewood Cliffs, N.J..

Geoffrion, A. M. (1967) : Integer Programming by Implicit Enumeration and Balas' Method, *SIAM Review*, **7**: 178-190.

Glover, F. (1973) : Convexity Cuts and Cut Search, *Oper. Res.*, **21**: 123-134.

Halpern, J., and O. Maimom (1980) : Algorithms for the m-Center Problems: A Survey, *European Journal of Operational Research*, **10**: 90-99.

Handler, G. Y., and P. B. Mirchandani (1979) : *Location on Networks*, The MIT Press, Cambridge, Massachusetts.

Hansen, P., M. Labbé, D. Peeters, and J. Thisse (1987) : Facility Location Algorithm, *Fundamentals of Pure and Applied Economics*, Systems of Cities and Facility Location, **22**: 1-70.

Hoffman, K. L. (1981) : A Method for Globally Minimizing Concave Functions Over Convex Sets, *Math. Program.*, **20**: 22-32 .

Horst, R. (1976a) : An Algorithm for Nonconvex Programming Problems, *Math. Program.*, **10**: 312-321.

Horst, R. (1976b) : A New Branch-and-Bound Approach for Concave Minimization Problems, *Lecture Notes in Computer Sciences*, **41**: 330-337.

Horst, R. (1986) : A General Class of Branch-and-Bound Methods in Global Optimization with Some New Approaches for Concave Minimization, *JOTA*, **51**: 271-291.

Horst, R. (1988) : Deterministic Global Optimization with Partition Sets Whose Feasibility Is Not Known: Application to Concave Minimization , Reverse Convex Constraints, DC-Programming, and Lipschitzian Optimization, *JOTA*, **58**: 11-37.

Horst, R., N. V. Thoai, and J. de Vries (1988) : On Finding the New Vertices and Redundant Constraints in Cutting Plane Algorithms for Global Optimization, *Oper. Res. Letters*, **7**: 85-90.

Kalantari, B., and J. B. Rosen (1987) : An Algorithm for Global Minimization of Linearly Constrained Concave Quadratic Functions, *Math. of Oper. Res.*, **12**: 544-561.

Kariv, O., and S. L. Hakimi (1979a) : An Algcrithmic Approach to Network Location Problems. I : The p-Centers, *SIAM Journal of Applied Mathematics*, **37**: 513-538.

Kariv, O., and S. L. Hakimi (1979b) : An Algorithmic Approach to Network Location Problems. II : The p-Centers, *SIAM Journal of Applied Mathematics*, **37**: 539-560.

Kennington, J. L., and R. V. Helgason (1980) : *Algorithms for Network Programming* , Wiley, New York.

Konno, H. (1976a) A Cutting Plane Algorithm for Solving Bilinear Programs, *Math. Progr.*, **11**: 14-27.

Konno, H. (1976b) : Maximization of a Convex Quadratic Function Subject to Linear Constraints, *Math. Progr.*, **11**: 117-127.

Krynski, S. L. (1979) : Minimization of a Concave Function Under Linear Constraints (Modification of Tuy's Method), pp. 479-493 in *Survey of Mathematical Programming (Proceedings of the Ninth International Mathematical Programming Symposium, Budapest 1976)* , Vol. 1, North-Holland, Amsterdam.

Kuenne, R. K., and R. M. Soland (1972) : Exact and Approximate Solutions to the Multisource Weber Problem, *Math. Program.*, **3**: 193-209.

Love, R. F., and H. Juel (1982) : Properties and Solution Methods for Large Location-Allocation Problems, *Journal of Operational Research Society*, **33**: 443-452.

Love, R. F., and J. G. Morris (1975) : A Computational Procedure for the Exact Solution of Location Allocation Problems with Rectangular Distances, *Naval Res. Logist. Quart.*, **22**: 441-453.

McKeown, P. G. (1978) : Extreme Point Ranking Algorithms: A computational survey, pp. 216-222 in W. W. White (ed.), *Computers and Mathematical Programming*, National Bureau of Standards Special Publication 502, U. S. Government Printing Office, Washington, DC.

Minieka, E. (1980) : Conditional Centers and Medians of a Graph, *Networks*, **10**: 265-272.

Mukhamediev, B. M. (1982) : Approximate Methods of Solving Concave Programming Problems, *USSR Comput. Maths. Math. Phys.*, **22**: 238-245.

Murtagh, B. A., and S. R. Niwattisyawong (1982) : An Efficient Method for the Multi-Depot Location-Allocation Problem, *Journal of Operational Research Society*, **33**: 629-634.

Murtagh, B. A., and M. A. Saunders (1987) : MINOS 5.1 User's Guide, Technical Report SOL 83-20R, Stanford University.

Murty, K. (1968) : Solving the Fixed-Charge Problem by Ranking the Extreme Points, *Oper. Res.*, **16**: 268-279.

Pardalos, P. M., and J. B. Rosen (1986) : Methods For Global Concave Minimization: A Bibliographic Survey, *SIAM Review*, **28**: 367-379.

Pardalos, P. M., and J. B. Rosen (1987) : *Constrained Global Optimization: Algorithms and Applications*, Springer-Verlag, Berlin.

Rardin, R. L., and V. E. Unger (1976) : Surrogate Constraints and the Strength of Bounds Derived from 0-1 Bender's Partitioning Procedures, *Oper. Res.*, **24**: 1169-1175.

Rosen, J. B. (1983) : Global Minimization of a Linearly Constrained Concave Function by Partition of Feasible Domain, *Math. of Oper. Res.*, **8**: 215-230.

Selim, S. S. (1979) : Biconvex Programming and Deterministic and Stochastic Location Allocation Problems, Unpublished Ph.D Dissertation, Georgia Institute of Technology.

Sherali, A. D., and C. M. Shetty (1977) : The Rectilinear Distance Location-Allocation Problem, *AIIE Transactions*, **9**: 136-143.

Sherali, H. D., and F. Nordai (1988) : NP-Hard, Capacitated, Balanced p-Median Problems on a Chain Graph with a Continuum of Link Demands, *Math. of Oper. Res.*, **13**: 32-49.

Shetty, C. M., and H. D. Sherali (1980) : Rectilinear Distance Location-Allocation Problem: A Simplex Based Algorithm, *Lecture Notes in Economics and Mathematical Systems* , Extremal Methods and System Analysis, **174**: 442-464.

Taha, H. A. (1973) : Concave Minimization Over a Convex Polyhedron, *Naval Res. Logist. Quart.*, **20**: 533-548.

Tansel, B. C., R. L. Francis, and T. J. Lowe (1983a) : Location on Networks: A Survey, Part I: The p-Center and P-Median Problems, *Management Science*, **29**: 482-497.

Tansel, B. C., R. L. Francis, and T. J. Lowe (1983b) : Location on Networks: A Survey, Part II: Exploiting Tree Network Structure, *Management Science*, **29**: 498-511.

Thieu, H. V., B. T. Tam, and V. T. Ban (1983) : An Outer Approximation Method for Globally Minimizing a Concave Function Over a Compact Convex Set, *Acta Math. Vietnamica*, **8**: 21-40.

Thoai, N. V., and H. Tuy (1980) : Convergent Algorithms for Minimizing a Concave Function, *Math. of Oper. Res.*, **5**: 556-566.

Tuy, H. (1964) : Concave Programming Under Linear Constraints, *Soviet Math.*, **5**: 1437-1440.

Tuy, H., T. V. Thieu, and N. Q. Thai (1985) : A Conical Algorithm for Globally Minimizing a Concave Function Over a Closed Convex Set, *Math. of Oper. Res.*, **10**: 498-514.

Vaish, H. (1974) : Nonconvex Programming with Applications to Production and Location Problems, (unpublished) Ph.D. Dissertation, Georgia Institute of Technology.

Zoutendijk, G. (1960) : *Methods of Feasible Directions*, Elsevier, Amsterdam, and D. Van Nostrand, Princeton, N.J..

Zwart, P. B. (1973) : Nonlinear Programming: Counterexamples to Two Global Optimization Algorithms, *Oper. Res.*, **21**: 1260-1266.

Zwart, P. B. (1974) : Global Maximization of a Convex Function with Linear Inequality Constraints, *Oper. Res.*, **22**: 602-609.

Vita

Cihan Halit Tuncbilek was born in Istanbul, Turkey on December 12, 1963. He graduated from Bogaziçi University, Istanbul where he received a Bachelor of Science degree in Industrial Engineering in 1987. He attended Virginia Polytechnic Institute and State University where he received a Master of Science degree in Industrial Engineering and Operations Research in June 1990. Currently, he is studying for his doctorate in Industrial Engineering and Operations Research at Virginia Polytechnic Institute and State University.

[illegible]