Single Objective Network Flow (SONF) optimization is one of the most widespread techniques for modeling real systems. Network flow models have been used for many diverse applications. Easier understanding of the models by decision makers and the impressive computational performance of network-based algorithms have been major factors contributing to the popularity of network models. Although models such as these with a single objective are well justified and have been successful in many situations, they do not accurately represent situations where there are multiple and often conflicting criteria for measuring the quality of feasible alternatives. In such situations, a Multiple Objective Network Flow (MONF) model would be more appropriate. The intent of this research is to investigate the nature and characteristics of MONF problems and to propose algorithms for finding “good” solutions. With this research, an approach to multiple objective network flow problems is proposed. A characterization of solutions is investigated and a network-based algorithm is developed. Software is developed to solve for all efficient extreme point solutions. A second set of software is developed for aiding a decision maker in searching through the nondominated set for the most preferred solution. By nature MONF problems are accompanied by a large set of potential solutions. Investigation of how the size of the solution set varies with problem parameters is described. Also, a method of solving the MONF on parallel processors is explored. In this way, research and knowledge of an important class of problems will be extended.

INDEX WORDS: Multicriteria Decision Making, Multiple Objective Linear Programming, Network Flows, Parallel Algorithms.
AN INVESTIGATION OF MULTIPLE OBJECTIVE NETWORK FLOWS

By

CRAIG ALLAN PIERCY

B.S., Tennessee Technological University, 1985

M.B.A., The University of Georgia, 1991

A Dissertation Submitted to the Graduate Faculty of The University of Georgia in Partial

Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2001
© 2001

Craig Allan Piercy

All Rights Reserved
DEDICATION

To Estelle, my wonderful wife, best friend and most important supporter. To Nicolas, my son, who provided major motivation for completing this work. And, to my parents, Ed and Yvonne, who have provided constant support.
ACKNOWLEDGEMENTS

There are many people I would like to thank for their support and encouragement. Each and every one of them has made a vital contribution toward the completion of my dissertation. First and foremost, I wish to thank my major professor Dr. Ralph E. Steuer for his guidance, encouragement, collaboration, and patience. He is a first-rate scholar and there is no better example for me to follow as I carry on my research. I thank my committee members, Dr. Jay Aronson, Dr. Roscoe Davis, Dr. Pat McKeown, and Dr. Andy Seila for their support and invaluable suggestions and reviews of my work. I look forward to working with them further in days to come. I give special thanks to Pat McKeown for his friendship and for providing many opportunities for financial support and employment while I completed my work. I want to thank my special friends and comrades Chris Lutz, Seonmin Kim, Bob Brown, Francisco Ruiz, Mark Huber, and Terry Sutton. Each has provided companionship, good times, and frequent words of encouragement. I give thanks to my friends and colleagues of the Towson University Department of Computer and Information Sciences: Chao Lu, Blair Taylor, Chuck Dierbach, Darush Davani, Jim Clements, Joan Lang, and the late Jim Smith. In addition to the opportunity to work and add several years of teaching experience to my resume, they provided as a group a consummate example of dedication and professionalism in college instruction. I must also thank my family who has provided a lifetime of support. Special thanks to my parents, Ed and Yvonne Piercy, for their love and steadfast support; Clark Piercy, my brother and close friend; and Alex, Claire and Dimitri Theotikos for
welcoming and accepting me into their family. Finally, thanks to my wife, the love of my life, my constant companion and my best friend, Estelle. She provided much help in reviewing and proof reading my work, but I wish to thank her most for her patience and unwavering support. Although at times I probably took her for granted, without her expert management of all the other areas of our life, I would not have been able to complete this work. I look forward to many more years of collaboration and cohabitation with her.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS........................................................................................................... v

CHAPTER

1 INTRODUCTION TO MULTIPLE OBJECTIVE NETWORK FLOWS....... 1
2 MULTIPLE OBJECTIVE LINEAR PROGRAMMING: BASIC THEORY. 22
3 MINIMUM COST NETWORK FLOWS: BASIC THEORY ......................... 52
4 A SEARCH OF THE MONF LITERATURE ................................................ 76
5 SOLUTION CHARACTERISTICS FOR MONF PROBLEMS .................... 91
6 SIZE AND PROPORTIONS OF THE MONF EFFICIENT SET ............... 118
7 FINDING ALL EFFICIENT EXTREME POINTS OF MONF ................. 137
8 AN INTERACTIVE SOLUTION METHOD FOR MONF ......................... 193
9 ANALYSIS OF NUMBER OF EFFICIENT EXTREME POINTS VERSUS
   PROBLEM SIZE ....................................................................................... 221
10 CONE DECOMPOSITION FOR THE SOLUTION OF EFFICIENT
   EXTREME POINTS IN PARALLEL ........................................................ 253
11 CONCLUSIONS AND FUTURE RESEARCH ......................................... 270

BIBLIOGRAPHY .............................................................................................................. 283

APPENDICES

A MONFSOLV USER GUIDE...................................................................... 308
B MONFSOLV FORTRAN CODE .............................................................. 317
CHAPTER 1
INTRODUCTION TO MULTIPLE OBJECTIVE NETWORK FLOWS

1.1 Introduction

An important tool for analyzing and solving complex problems is modeling. A model can be defined as a simplified representation of reality. Look around every day and you can find many important decisions that are being supported by the use of models. Architects and engineers provide scaled-down replicas of buildings to aid in planning a site. Meteorologists present computer simulation models of the weather forecast to help us plan our day. Pollsters prepare statistical models to describe public opinion and predict election outcomes.

The benefits of using models in decision-making are well known. They can aid in learning and understanding of a problem by reducing the problem to its key elements. Most risks derived from experimentation with alternatives are transferred from the real systems to the model and are reduced. The costs and difficulties of manipulating the model are usually much less than those related to manipulation of the real system. Also, models can provide efficiencies through the compression of time. Long periods of real time may be investigated in a moment's analysis of a model.

The primary disadvantage of modeling is derived from the fact that the model is an abstraction of reality. There is usually no guarantee that characteristics observed with the model will hold true in the real world. Often, analysts attempt to overcome this
problem by reducing the level of abstraction in the model. This typically adds greater complexity to the model itself leading to more difficult model manipulation and longer solution times. The use of modeling typically requires a trade-off between the level of abstraction in the model design and the time and effort required to work with the model.

The subject of this dissertation is a specific type of mathematical optimization model called the Multiple Objective Network Flow (MONF) model. The general MONF model illustrates the trade-off between model simplification and solution complexity. The MONF models are based on well-known Minimum Cost Network Flow (MCNF) models that are typically modeled using a single objective. Network flow models are characterized by a mathematical structure that may be exploited with very efficient solution algorithms. The introduction of multiple objectives to the model is an effort to reduce the level of abstraction of the model from the real world system. The purpose of this research is to investigate the effect the addition of multiple objectives has on the solution of the network flow models.

In this chapter, a foundation is provided for the understanding of the MONF model and the organization and goals of the research are discussed.

1.2 Linear Programming

Perhaps the best-known mathematical modeling technique for optimization problems is linear programming (LP). LP models deal with the optimal allocation of limited resources among competing activities. This situation occurs in a vast variety of applications. Examples include: selecting the optimal product mix to manufacture; bond portfolio construction in finance; assigning personnel to job activities in the most
productive way; and shipping goods between various locations with the least cost. There are many texts which discuss LP and solution methods. Among these are Bazzarra and Jarvis (1977), Charnes and Cooper (1961), Gass (1985), and Hillier and Lieberman (1990).

1.2.1 Linear Program Models

Each LP model is made up of a standard set of components. First, there is a set of decision variables whose values are unknown. Typically, each decision variable represents the level of some activity. There is also an objective function which is a linear equation relating the decision variables to an overall measure of performance $z$. The goal is typically to either maximize or minimize the value of $z$. In addition, there are usually a number of constraints on how the resources may be allocated. These constraints are also modeled in the form of linear equations. To solve the problem, we must search through the set of all possible combinations of decision variable values until we find the combination that provides the best possible value for the performance measure $z$.

Example 1.1 is provided to illustrate these concepts.

**Example 1.1 - A Product Mix Problem.** Nick, Inc., a manufacturing company has two large products that it can produce. Product 1 provides revenue of $1000 per unit sold while Product 2 provides revenue of $6000 per unit sold. The two factories that are owned by the company are each dedicated to one of the products. Both factories have a capacity for producing 8000 units per week. There are three primary materials used to create the two products. For Material 1, Product 1 requires 3000 pounds and Product 2 requires 1000 pounds. Both products require 1000 pounds each of Material 2. For Material 3, Product 1 requires 1000 pounds and Product 2 requires 3000 pounds. However, there is a limited supply of each material. For Materials 1, 2, and 3 there are available 26, 12, and 26 thousand pounds, respectively. The managers of Nick, Inc. would like to determine the optimal amount of each product to create per week in order to maximize revenue.
An LP model may be formulated in a variety of ways. The objective function may be formulated with either a minimization or a maximization objective and the constraints may be formed using either \( \geq \), \( \leq \), or \( = \). In order to model Example 1.1 as a LP, we first identify the appropriate decision variables. Since we need to determine the level for each product, we will use variables \( x_1 \) and \( x_2 \) to represent these amounts. The objective is to optimize the total revenue per week so the variable \( z \) will represent the value of this measure of performance. Using these variables, the LP model may be formulated as:

\[
\begin{align*}
\text{max} : & \quad x_1 + 6x_2 = z \\
\text{s.t.:} & \quad x_1 \leq 8 \\
& \quad x_2 \leq 8 \\
& \quad 3x_1 + x_2 \leq 26 \\
& \quad x_1 + x_2 \leq 12 \\
& \quad x_1 + 3x_2 \leq 26 \\
& \quad x_1 \geq 0, \quad x_2 \geq 0
\end{align*}
\]

The first line of this formulation represents the objective to maximize the total profit, which is set equal to a linear expression using the decision variables. The coefficients of this expression are known as objective function coefficients, \( c_j \)'s. In addition, this model incorporates seven constraints on the decision variables. The first five constraints are of a type known as structural constraints. Structural constraints are typically determined by restrictions on one or more resources. The first two constraints correspond to the two plant capacities. Constraints three, four, and five correspond to the limitations on the raw materials. The coefficients in the structural constraints are known as constraint coefficients, \( a_{ij} \). The capacity levels located on the right of the equality or
inequality signs for the structural constraints are called right hand side (RHS) values.

Finally, many LPs as this model illustrates will also include nonnegativity constraints when negative values would not make sense for the problem.

There are several assumptions that must be made about a problem before it may be formulated as a LP model. Each of these is illustrated in the example. First, it is assumed that the overall measure of performance $z$ as well as the usage of each resource $i$ are all directly proportional to the level of each activity $k$ conducted by itself. This implies that there is no extra startup cost associated with beginning the activity and the proportionality holds over the entire range of levels of the activity. Second, it is assumed that there are no interactions between activities. If there were, the model would need cross-product terms to represent the interaction destroying the assumed linearity of the model. Third, it is assumed that activity levels can be divided into any fractional levels - in other words, the resulting decision variable values need not be integer. Finally, the assumption is made that all parameters, the coefficients and RHS values, are known with certainty. Since these assumptions may not always hold in the "real-world," the model should be considered as a simplification of the actual problem.

1.2.2 Solving Linear Programs

While only practical for problems with a small number of decision variables, it is sometimes helpful to view a graphical representation of the model. In Figure 1.1, the graph corresponds to the LP model of Example 1.1.

The shaded region on the graph is known as the feasible region and is often denoted by $S$. The feasible region together with its boundary includes all values of the
decision variables that are feasible given the constraints of the model. The feasible region is found by simply charting the straight line defined by each equality or inequality constraint including the nonnegativity constraints. Each line divides the entire space into two regions called \textit{half-spaces}. For equality constraints, only points on the line corresponding to the constraint will be feasible. For inequality constraints, solutions in one of the half-spaces defined by the line are feasible. This includes the nonnegativity constraints. The region defined by the lines and overlapping half-spaces is the feasible region.

![Graphical Representation of Example 1.1 Model](image)

Figure 1.1 - Graphical Representation of Example 1.1 Model

In Figure 1.1 the feasible region is bounded by linear constraint lines that intersect. The points of intersection, labeled $x^1$ through $x^7$, are known as \textit{extreme point} solutions.

Here the extreme points are $x^1 = (0, 8)$, $x^2 = (2, 8)$, $x^3 = (5, 7)$, $x^4 = (7, 5)$, $x^5 = (8, 2)$, $x^6 = (8, 0)$, $x^7 = (0, 0)$.
Two extreme point solutions that are directly connected by one linear constraint are said to be adjacent and the line segment connecting them is known as an edge. The vector $c$ drawn on the chart is determined by the coefficients of the objective function. It is notable in that it points in the direction that improves, in this case increases, the objective value $z$.

There are three key properties of extreme point solutions that form the basis of most LP solution methods (Bazaraa and Jarvis, 1977). First, if an optimal solution exists then there must be at least one extreme point that is optimal. Second, there are only a finite number of extreme point feasible solutions because $S$ is polyhedral. Third, if an extreme point solution $x'$ has no adjacent extreme point solutions that are better in terms of $z$, then the solution $x'$ is optimal. Together these properties imply that we can restrict our search for an optimal solution to only those feasible solutions that are extreme points. Note that this does not mean that there need be only one optimal solution or that a non-extreme point cannot be optimal. For example, it is possible that two adjacent extreme points and all solutions that lie between them on the boundary of the feasible region are optimal.

The most popular method for solving LPs, the simplex method, takes advantage of the three properties of extreme points. By utilizing mathematical operations on the system of equations in the model, the simplex method in a sense "hops" from one extreme point to another until an optimal solution is found. At each extreme point, a test is carried out to determine whether or not moving to an adjacent extreme point would improve the objective function. If so, then the method "hops" to an adjacent point with an improved $z$ that provides the most local improvement. If no improvement is found, the method halts
with the current extreme point deemed optimal. This is displayed in Figure 1.2 finding extreme point $x^3$ to be the optimal solution with $z = 58$.

While the basic LP has been successful for many practical applications, there are many types of problems that use the model for their basic structure but require a departure of some kind. These may include the addition of one or more additional objectives, restricting the values of the decision variables to integers, or introducing non-linearity. In the next section the first of these departures, the inclusion of multiple objectives, is introduced.

![Figure 1.2 - Graphical Representation of Simplex Method](image)

**Figure 1.2 - Graphical Representation of Simplex Method**

### 1.3 Multiple Objective Linear Programming

For many, if not most, real life problems it is not always easy to define a single objective for a problem. For example, when investing one hopes to maximize the overall
return while minimizing several types of risk. In assigning staff, objectives may include maximizing productivity and minimizing overtime while maintaining a balanced number of tasks per staff member. When transporting goods between several locations, the decision maker may wish to minimize the overall cost while minimizing the shipping time to key locations.

One area of research related to LP is concerned with decreasing the level of model abstraction by introducing multiple objectives. This moves the modeling and solution of problems into the realm of Multiple Objective Linear Programming (MOLP). With MOLP we are concerned with aiding the decision maker (DM) in choosing the best alternative within a feasible solution set in the face of several, often conflicting objectives. As might be expected, introducing complexity to the model in the form of additional objectives introduces complexity into any solution procedure employed. This is primarily due to the fact that the number of alternatives that are candidates for the solution to the problem may increase as additional objectives are added to the model. This is illustrated with Example 1.2.

**Example 1.2 - A Product Mix Problem.** The marketing managers of Nick, Inc. have decided that they have an opportunity to capture more market share for each of their products depending on the amount that they can produce and bring to market. Their best opportunity is with Product 1 that they estimate can capture 0.7% in additional market share with each unit produced. The forecast is for only 0.1% in additional market share for each unit of Product 2. With their long term interests in mind, they would like to maximize the market for their product in addition to maximizing their weekly revenue.

The MOLP model for Example 1.2 can be formulated just like the LP model from the previous example with the addition of the new objective. With this in mind the new model is formulated as:
Instead of a single measure of performance $z$, we now have two separate measures of performance, $z_1$ and $z_2$. The coefficients for the new objective have been scaled to integer values. The graph of this model is shown in Figure 1.3. The vectors $c_1$ and $c_2$ on the graph are derived from the objective function coefficients of the two objectives, respectively.

\[
\begin{align*}
\text{max: } & \quad x_1 + 6x_2 = z_1 \\
\text{max: } & \quad 7x_1 + x_2 = z_2 \\
\text{s.t.: } & \quad x_1 \leq 8 \\
& \quad x_2 \leq 8 \\
& \quad 3x_1 + x_2 \leq 26 \\
& \quad x_1 + 3x_2 \leq 26 \\
& \quad x_1 \geq 0, \quad x_2 \geq 0
\end{align*}
\]

\[c^1 = (1, 5)\]
\[c^2 = (7, 1)\]
\[x^1 = (0, 8)\]
\[x^2 = (2, 8)\]
\[x^3 = (5, 7)\]
\[x^4 = (7, 5)\]
\[x^5 = (8, 2)\]
\[x^6 = (8, 0)\]
\[x^7 = (0, 0)\]
An inspection of the model and the graph of Figure 1.3 reveals that the optimal solution of the original objective function $z_1, x^2 = (2, 8)$, does not provide the optimal value for the new objective function $z_2$. For the alternative solution $x^2$, the objective function values are $z_1 = 58$ and $z_2 = 22$, respectively. The alternative that provides the maximum value for $z_2$ is solution $x^5 = (8, 2)$. For solution $x^5$, the objective values are $z_1 = 20$ and $z_2 = 66$. This is illustrated in Figure 1.4.

![Figure 1.4 - Candidate Solutions for Example 1.2](image)

This situation illustrates a characteristic that is common for most MOLP models. Here, the DM must choose between an alternative which provides more revenue and an alternative which provides greater improvement to market share. In other words, a trade-off is necessary. Instead of finding one alternative that provides a single optimal objective value, we have at least two candidate alternatives for the decision maker to choose as the
"best" alternative. For the example problem, it turns out that the corner point solutions $x^2$, $x^3$, $x^4$, and $x^5$ as well all solutions lying on the boundary edges between any two of these corner points are candidates for the DM's "best" solution.

Notice that the only difference between the multiple and single objective formulations for the examples is the total number of objective functions. Note also that while it may be desirable to minimize some objectives, a minimization function can always be placed in a maximization form by simply multiplying its left-hand-side (LHS) by -1 and keeping in mind that the new objective function value is the negative of the original objective function value. This transformation process also works in reverse.

The transition of formulation from a single objective program to a multiple objective program is straightforward; simply include a new objective function for each optimization criteria of interest. Despite this simple addition, a surprising amount of additional complexity is encountered when searching for solutions to a multiple objective linear program. This is primarily due to the fact that in most cases the objectives conflict to some extent so that no single solution will simultaneously optimize all of the objectives. Therefore, we usually need to evaluate the trade-offs in objective function values at different decision variable alternatives.

In MOLP, we can think of the alternative solutions in terms of two different but related solution spaces (Steuer, 1989). The first, known as decision space, is composed of all vectors in $\mathbb{R}^n$ where $n$ is the number of decision variables in the model. The feasible region $S$ in decision space is shown in Figures 1.1 through 1.4. The second, known as criterion space, is composed of all vectors in $\mathbb{R}^k$ where $k$ corresponds to the number of objectives in the model. Criterion space is denoted $Z$. The image of $S \in \mathbb{R}^n$ in $\mathbb{R}^k$ is $Z$. 
Viewing the feasible region in criterion space can provide a better view of the trade-offs involved in a MOLP. The set Z for Example 1.2 is graphed in Figure 1.5 below. Plugging the values of $x_1$ and $x_2$ at each extreme point into the objective function equations to calculate the corresponding $z_1$ and $z_2$ values created the chart of Figure 1.5. Thus the point for $x^1 = (0, 8)$ in decision space is mapped to $z^1 = (48, 8)$ in criterion space. In the same manner, decision space points, $x^2$ through $x^6$, were mapped to criterion space points $z^2$ through $z^6$, respectively. Once all of the extreme points were mapped, the boundary was determined by simply connecting the dots. It can be seen here that as long as the constraints are linear, corner point solutions in decision space will map to corner point solutions in criterion space. What is not evident in this example is that because of the linearity of the problem two or more corner points in decision space may have the same corner point as their image in criterion space.

![Figure 1.5 - Feasible Region in Criterion Space](image)

- $z^1 = (48, 8)$
- $z^2 = (50, 22)$
- $z^3 = (47, 42)$
- $z^4 = (37, 54)$
- $z^5 = (20, 58)$
- $z^6 = (8, 56)$
- $z^7 = (0, 0)$
Notice in Figure 1.5 that it is fairly easy to pick out the points that provide the maximum values for each criterion. Since point \( z^2 = (50, 22) \) lies the farthest to the right on the graph it is seen to provide the highest value for \( z_1 \). Similarly, the highest point on the graph, \( z^5 = (20, 58) \), provides the highest value for \( z_2 \). Here are at least two candidates for the final solution value but the choice must be made between them. Are there other candidates? Consider point \( z^3 = (47, 42) \), when compared to point \( z^2 \), this point has a higher value for criterion \( z_2 \) but also lower value for criterion \( z_1 \). As we move from point \( z^2 \) to point \( z^3 \) we see improvement in one criterion accompanied by degradation in the other. When point \( z^3 \) is compared to point \( z^5 \), similar results occur. Therefore, point \( z^3 \) would also be a candidate for the final solution. In fact, the same can be said about all of the points on the boundary of the feasible region between point \( z^2 \) and point \( z^5 \).

A different story occurs when we compare points \( z^2 = (50, 22) \) and \( z^1 = (48, 8) \). Clearly, point \( z^2 \) has higher values for both criteria. In MOLP theory, we would say that \( z^1 \) is dominated by \( z^2 \). The points on the boundary of the feasible region between \( z^2 \) and \( z^5 \) are in fact nondominated. This is because for each of these points there does not exist another point in the feasible region that is better for at least one criterion and equal or better for the other criteria. For all of the points that are dominated, we can find at least one point that is at least as good for all criteria and better for at least one criteria. In criterion space, we can divide the set of feasible points into those that are dominated and those that are nondominated.
Recall how the graph of Figure 1.5 was produced, each point in the feasible region of decision space was mapped onto a point in criterion space. The points in decision space that map onto a nondominated point are said to be efficient. Those that do not map onto a nondominated point are called inefficient. These concepts will be defined formally in Chapter 2. For now it is important to note that when searching through the feasible region in criterion space for a DM’s most preferred solution, we will be interested primarily in nondominated points and their corresponding extreme points in decision space.

Most solution methods for MOLP search among the efficient set $E$ or the nondominated set $N$ for the decision makers most preferred solution. The efficient (nondominated) set can be completely characterized if one can find all efficient (nondominated) extreme points. Therefore, many solution methods begin with the enumeration of all efficient extreme points. The main difficulty of enumerating the efficient set is that its size increases dramatically as the number of objectives, variables or constraints is increased. Usually, sampling of one sort or another is required to locate a DM’s most preferred or nearly most preferred solution. In the next section, another key type of model is discussed that is very closely related to linear programming. Additional information concerning MOLP may be found in Zeleny (1982), Steuer (1989), and Vincke (1992).

1.4 Minimum Cost Network Flow Models

Another area of research related to LP deals with an important sub-class of models known as Minimum Cost Network Flow (MCNF) models. MCNF optimization is one of
the most widespread techniques for modeling real systems. A major reason for this is the
diversity of problems that can be modeled and solved under a network framework.

Network models have been used for such diverse applications as production and
distribution planning, project scheduling, employee job assignment, and facility layout.
The easier understanding of the models by decision makers and the impressive
computational performance of network-based algorithms have also been major factors
contributing to the popularity of network models. Informative texts in this area include
Kennington, and Helgason (1980); Ruhe (1991); and Ahuja, Magnanti, and Orlin (1993).

A MCNF problem may be modeled on a network consisting of nodes and arcs. A
node can represent a starting point (source) for some commodity, an ending point (sink)
for the commodity, or an intermediate point (transshipment). An arc can represent a
directed conduit between two nodes through which the commodity may travel. With each
arc, there is usually a value or cost that is assigned to each unit of commodity that flows
through the arc and possibly a capacity on the amount of flow. The objective of the
MCNF problem is to find a routing for the commodity to flow from the sources to sinks
with the least total cost.

**Example 1.3 - Routing Packets on a Radio Network.** A packet radio network consists
of a collection of terminals distributed over a wide geographical area and the radio links
that connect them. Information is transmitted between terminals in the form of discrete
blocks of data called packets. The communication medium is a radio channel that is
shared by many contending users. Propagation path loss can occur with the signal that
depends on the distance between the transmitter and the receiver. Assume that a message
that can be broken into 5 packets needs to be sent from terminal A to terminal D over the
radio network. The possible routes that the packets may take are shown in Figure 1.6.
Associated with the connections between any two computers in the network are the
distance and a capacity on the number of packets. The distances and capacities for the
connections are denoted as an ordered pair next to the connection. The objective is to
move the packets from terminal A to terminal D over the shortest routes possible assuming that each channel has a capacity of 3 packets.

Figure 1.6 illustrates a common method of modeling network flow problems like that of Example 1.3 on a graph. Each terminal in the network is represented as a node and directed arcs represent the connections between them. By inspection, one can see that the route from A to B to D has a distance of 3 units. This route will be referred to as Route 1. Route 2, A to D, has a total distance of 6 units while Route 3, A to C to D, has a distance of 4 units. The routing that provides the shortest total distance for all packets is to send 3 packets through Route 1 and the remaining 2 packets through Route 2. This results in a total transmission time of 17 units.

Figure 1.6 - MCNF Model for Packet Routing Example

One interesting aspect of MCNF problems is that in addition to the graphical model, they may also be modeled using a linear programming formulation. To do this, first associate a decision variable with each arc in the network representing the amount of flow on the arc. An objective function can then be formulated using the decision variables and using the arc costs as the objective function coefficients. A constraint is then
formulated for each node in the network using the rule that the amount of flow coming into a node must equal the amount of flow going out of the node. This is accomplished by setting the sum of the decision variables for in-coming arcs equal to the sum of the variables for the out-going arcs. These constraints are known as node-balance constraints. For source and sink nodes, the associated supply is usually handled as in-coming flow while demand is handled as out-going flow. Typically, these constraints are manipulated so that all decision variables are on the left hand side of the equality sign and all constants are on the right hand side. Finally, additional constraints are required to reflect the capacities on the arc and the nonnegativity of the flows. In this manner, the following LP formulation provides a mathematical model for the MCNF of Example 1.3.

\[
\begin{align*}
\min: & \quad 2x_{AB} + x_{AC} + 6x_{AD} + x_{BD} + 3x_{CD} = z \\
\text{s.t.:} & \quad x_{AB} + x_{AC} + x_{AD} + x_{BD} = 5 \\
& \quad -x_{AB} + x_{AC} + x_{AD} = 0 \\
& \quad -x_{AC} + x_{BD} = 0 \\
& \quad -x_{AD} - x_{BD} - x_{CD} = -5 \\
& \quad 0 \leq x_{AB} \leq 3 \\
& \quad 0 \leq x_{AC} \leq 3 \\
& \quad 0 \leq x_{AD} \leq 3 \\
& \quad 0 \leq x_{BD} \leq 3 \\
& \quad 0 \leq x_{CD} \leq 3
\end{align*}
\]

By setting \( x_{AB} = x_{BD} = 3, x_{AC} = x_{CD} = 2 \) and \( x_{AD} = 0 \), all constraints are satisfied with an objective function value of \( z = 17 \).

Since the MCNF may be formulated as an LP, traditional LP solution methods such as the simplex method may be used to find a solution for the model. Looking closer at the LP formulation one can see some interesting features that turn out to be common
for MCNF models. First, the objective function seeks to minimize the performance measure $z$ as might be expected from the use of the term "min" in describing the problem category. Second, the coefficients of the node-balance constraints are all of only three possible values, 0, 1, and -1. As it turns out, this results in a special mathematical structure that along with specialized data structures may be exploited to develop solution methods that are many times more efficient in finding a solution than the traditional LP methods. In addition, it turns out that an LP with this structure will always have a solution which is integer valued (Kennington, and Helgason, 1980). Both of these aspects of the MCNF will be important as additional criteria are added to the model.

As previously mentioned, there are many diverse applications of the MCNF model. Important special cases of the MCNF include the Assignment problem, the Shortest Path problem, the Max-Flow problem, and the Transportation problem. The diversity of applications, the efficiency of available algorithms, and the ease of understanding with the graphical model have all served to increase the popularity of the MCNF model. Despite this, relatively little research has been reported concerning MCNF formulations incorporating multiple objectives. To see how multiple objectives might arise in practical problems consider the following example.

**Example 1.4 - Routing Packets on a Radio Network Part 2.** Propagation path loss between connections in a packet radio network can also depend on the terrain between the terminals. Assume that associated with each of the arcs of packet radio network of example 1.3 there is a terrain constant. The terrain constant provides a value similar to the resistance used in electrical networks. A higher terrain constant for a channel implies that there is a greater likelihood of propagation loss on transmissions. Assume now that in addition to minimizing the total distance, a second objective is to minimize the total terrain value. The network with the additional objective is shown in Figure 1.7.
In Figure 1.7, note that the route that provides the best overall terrain value, the direct one from terminal A to terminal D, is also the route with the longest distance. Because the two objectives conflict in this way, the trade-off in total distance and total terrain value will need to be evaluated when selecting the routing strategy. The evaluation of trade-offs in the face of multiple, conflicting objectives is the realm of MOLP theory so it seems natural that the two research disciplines of MOLP and MCNF should converge. The main focus of this dissertation will be to investigate the nature of this convergence.

![Figure 1.7 - Packet Routing Model with Two Objectives](image)

**1.5 Organization of Dissertation**

The chapters to follow deal with various aspects of the Multiple Objective Network Flow (MONF) problem and related issues of MOLP. Computer codes are described which were developed to perform computational research and provide empirical evidence of the nature of MONF solutions. The work is theoretical in nature as opposed to application based.
The dissertation is organized as follows. A brief overview of multiple objective decision aid theory is provided in chapter 2 followed by a brief overview of network theory in chapter 3. Then, in Chapter 4, a comprehensive literature review is provided covering the majority of works to date in the area of multiple objective network flows. Next, the characteristics of the MONF problem are explored in chapters 5 and 6 along with observations, theory and empirical results. A solver for all efficient corner point solutions is described in chapter 7 and a comparison between the MONF solver and the MOLP solver ADBASE is provided.

In chapter 8 an interactive solver for finding a decision-makers most preferred solution for the MONF problem is introduced. The solver presented is an extension of the work of Ralph E. Steuer, Jeff Kennington, and Iqbal Ali (1990). This solver will be able to provide solutions from the efficient set that are not necessarily extreme point solutions. In chapter 9, an investigation is described of how the size of the efficient set for MOLPs (of which MONF is a special case) relates to problem size parameters. A parallel solution strategy for MOLP and MONF is described in chapter 10 along with experimental results. Finally, chapter 11 concludes with a summary of results. With this research, several heretofore-unanswered questions are addressed for an interesting class of models.
CHAPTER 2
MULTIPLE OBJECTIVE LINEAR PROGRAMMING: BASIC THEORY

In this chapter, we give an overview of the theory of multiple objective linear programming (MOLP) that will be relevant to the research described. Basic mathematical background is provided. A basic knowledge of set theory and linear algebra is assumed. A formal definition of MOLP will be presented along with the standard notation. This is followed by a discussion of MOLP problem characteristics. We provide a survey of interactive approaches to MOLP. We close with a survey of the MOLP literature.

2.1 Mathematical Background

In this section, supporting mathematical background for MOLP theory is provided. The concepts and notation follows that of Steuer (1989).

2.1.1 - Bounded and Unbounded Sets

Several concepts have a bearing on the topology of the MOLP feasible region. The set $B_\delta = \{ x \in \mathbb{R}^n \mid (x_1 - \bar{x}_1)^2 + (x_2 - \bar{x}_2)^2 + \cdots + (x_n - \bar{x}_n)^2 < \delta^2 \}$ with radius $\delta > 0$ is an $n$-dimensional open hypersphere centered at $\bar{x} \in \mathbb{R}^n$. The set $B_\delta$ can be described in various ways depending on the dimension, $n$. In $\mathbb{R}$ it is an open interval. For $\mathbb{R}^2$ it is a disk that does not contain its edges. For $\mathbb{R}^3$ it is a sphere that does not contain its surface.
A point \( x' \in S \subset R^n \) is an *interior point* of \( S \) if and only if \( x' \) belongs to an \( n \)-dimensional open hypersphere centered at \( x' \in R^n \) such that \( B_x \subset S \). A *boundary point* of \( S \subset R^n \) is a point \( x' \in R^n \) such that every \( n \)-dimensional open hypersphere centered at \( x' \) contains points in \( S \) and points not in \( S \). A set is *open* if all of its members are interior points. A set is *closed* if it contains all of its boundary points. The complement of an open set is a closed set, and vice versa. A set \( S \subset R^n \) is *bounded* if and only if there exists an \( n \)-dimensional open hypersphere that contains \( S \). A set that is not bounded is *unbounded*.

### 2.1.2 Convex Sets and Extreme Points

A set \( C \subset R^n \) is *convex* if, for any two points \( x', x'' \in C \), all points of the form \( x(\lambda) = \lambda x' + (1 - \lambda)x'' \), where \( 0 \leq \lambda \leq 1 \), are in \( C \). In other words, the line segment between any two points in the set must lie in the set for it to be convex. A vector \( x \) is a *convex combination* of points \( x_1, \ldots, x_N \) if \( x = \sum_{i=1}^{N} \lambda_i x_i \), where \( \lambda_i \geq 0 \), all \( i \) and \( \sum_{i=1}^{N} \lambda_i = 1 \). In terms of convex combinations, a set \( C \) is convex when for any two points \( x', x'' \in C \) all points of the convex combination of \( x' \) and \( x'' \) are in \( C \).

The set \( H = \{ x \in R^n | a x = \beta \} \) where \( a \in R^n \), \( a \neq 0 \), and \( \beta \in R \) is a *hyperplane*. The set \( \overline{H} = \{ x \in R^n | a x \leq \beta \} \) is a *closed half-space*. The hyperplane \( H \) associated with the half-space \( \overline{H} \) is referred to as the *bounding hyperplane* for that half-space. A convex *polyhedron* is a set formed by the intersection of a finite number of closed half-spaces and hyperplanes. If it is non-empty and bounded, it is called a *convex polytope*, or simply a
polytope. Hyperplanes and closed half-spaces are convex and the intersection of convex sets is convex. Hence, a convex polyhedron as defined above is convex.

An extreme point of a convex set $C$ is a point $x \in C$ that cannot be expressed as a convex combination of (two) other points in $C$. Recall that if there is an optimal solution for an LP, then there exists at least one optimal extreme point. Extreme points are also significant in that any point in a convex set that is not extreme may be defined as a convex combination of two or more extreme points. Because of this, extreme points are often utilized to characterize the feasible region $S$ or subsets of $S$.

2.1.3 Cones

A set $V \subseteq \mathbb{R}^n$ is a cone if, for any point $v' \in V$ and any non-negative scalar $\alpha$, the point $\alpha v'$ is in $V$. The set $V = \{v \in \mathbb{R}^n \mid v = A\alpha, \alpha \geq 0, \alpha \in \mathbb{R}^m\}$ is a convex cone generated by the columns of $A \in \mathbb{R}^{m \times n}$. The number of linearly independent columns of $A$ gives the dimensionality of the cone $V$.

Two special cones that are related to the cone $V$ are the nonnegative polar of $V$ and the semi-positive polar of $V$. Let $V \subseteq \mathbb{R}^n$ be a cone. The nonnegative polar of $V$ is the convex cone $V^\geq = \{y \in \mathbb{R}^n \mid yv \geq 0 \ \forall \ v \in V\}$. In other words, all of the vectors in $V^\geq$ make an angle with each of the vectors in $V$ that is less than or equal to 90°. Let $V \subseteq \mathbb{R}^n$ be a cone generated by $\{v^1, \ldots, v^k\}$. The semi-positive polar of $V$ is the convex cone $V^\geq = \{y \in \mathbb{R}^n \mid yv^i \geq 0 \ \text{for all} \ i \ \text{and} \ yv^i > 0 \ \text{for at least one} \ i\} \cup \{0 \in \mathbb{R}^n\}$. Every vector in $V^\geq$ must have a positive vector product with at least one of the $v^i$. 
2.1.4 Facets

The following definitions are useful in describing the geometry of the feasible region and its subsets. A *supporting hyperplane* of a convex set \( C \) is a hyperplane \( H \) such that \( H \cap C \neq \emptyset \) and \( C \subseteq \overline{H} \), one of the two closed half-spaces associated with \( H \). Let \( P \) be a convex polyhedron and \( H \) any supporting hyperplane of \( P \). The intersection \( F = P \cap H \) defines a *face* of \( P \).

*Vertices, edges and facets* are faces of a \( d \)-dimensional convex polyhedron of dimension zero, one, and \( d-1 \), respectively. In general, a facet with dimension \( f \) is said to be an \( f \)-*facet* of \( S \). Vertices of a convex set are extreme points of a convex polyhedron \( P \) and we shall henceforth use the term extreme point. Edges are either line segments that connect adjacent extreme points or are semi-infinite lines emanating from an extreme point.

2.1.5 Norms and Metrics

For most multiple objective programs, work is carried out in vector space \( \mathbb{R}^n \).

When \( n \) is the number of decision variables we will call this decision space and when \( n \) is the number of objectives it will be criterion space. When working in a vector space it is necessary to be able to assign distances between different points in the space. Two families of mathematical constructs are available for this, norms and metrics.

A *norm* \( \| \cdot \| \) on \( \mathbb{R}^n \) is a mapping that assigns a scalar \( \| x \| \) to every \( x \in \mathbb{R}^n \) and has the following properties: (1) \( \| x \| > 0 \) for all \( x \in \mathbb{R}^n \); (2) \( \| cx \| = |c| \| x \| \) for every \( c \in \mathbb{R} \) and every \( x \in \mathbb{R}^n \); (3) \( \| x \| = 0 \) if and only if \( x = 0 \); (4) \( \| x + y \| \leq \| x \| + \| y \| \) for all \( x, y \in \mathbb{R}^n \). A
norm is used as a measure of the length of a vector in $\mathbb{R}^n$. An important family of norms is the family of $L_p$-norms. The $L_p$-norm of a vector $\mathbf{x} \in \mathbb{R}^n$ is given by

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \quad p \in \{1, 2, 3, \ldots\} \cup \{\infty\}.$$  

When $p = 1$, the $L_1$-norm is the sum of the absolute values of the components. The Euclidean norm, the $L_2$-norm with $p = 2$, is the square root of the sum of squared vector components. The Tchebycheff norm, when $p = \infty$, converges to the absolute value of the maximum component.

A metric $\|\cdot\|$ on $\mathbb{R}^n$ is a distance function that assigns to each pair of vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ a scalar $\|\mathbf{x} - \mathbf{y}\|$ if and only if the function has the following properties: (1) $\|\mathbf{x} - \mathbf{y}\| \geq 0$ and $\|\mathbf{x} - \mathbf{x}\| = 0$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$; (2) $\|\mathbf{x} - \mathbf{y}\| = \|\mathbf{y} - \mathbf{x}\|$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$; (3) $\|\mathbf{x} - \mathbf{y}\| \leq \|\mathbf{x} - \mathbf{z}\| + \|\mathbf{z} - \mathbf{y}\|$ for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n$; (4) If $\mathbf{x} \neq \mathbf{y}$ then $\|\mathbf{x} - \mathbf{y}\| \geq 0$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$.

The family of $L_p$-metrics is given by

$$\|\mathbf{x} - \mathbf{y}\|_p = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p} \quad p \in \{1, 2, 3, \ldots\} \cup \{\infty\}.$$  

An $L_p$-norm is equivalent to an $L_p$-metric with $\mathbf{y} = \mathbf{0}$.

When it is desired to allow some components to have more influence on the metric value than others. For this, a weighted metric may be more appropriate. The family of weighted $L_p$-metrics is given by

$$\|\mathbf{x} - \mathbf{y}\|_p^\lambda = \left(\sum_{i=1}^{n} \lambda_i |x_i - y_i|^p\right)^{1/p} \quad p \in \{1, 2, 3, \ldots\} \cup \{\infty\}.$$  

The family of weighted $L_p$-metrics will include the family of $L_p$-metrics when $\lambda = (1, \ldots, 1)$. Norms and metrics will be used in various methods for solving multiple objective
problems. In particular, the weighted Tchebycheff metric, with \( p = \infty \), has received much attention.

2.2 Definition of MOLP

A problem that is an MOLP may be modeled using the form

\[
\begin{align*}
\min: & \quad c^i x = z_i \\
\vdots & \\
\min: & \quad c^k x = z_k \\
\text{s.t.:} & \quad x \in S
\end{align*}
\]

\( S \) is called the feasible region and is defined as

\[
S = \{ x \in R^n \mid Ax = b, \, x \geq 0, \, b \in R^m \}
\]

In this general formulation, there are \( k \) objectives. Each objective may be identified by a criterion value \( z_i \) where \( i = 1, \ldots, k \). There may be a mixture of minimization and maximization objectives in the typical problem but it is standard practice to perform a conversion so that all objectives have a common optimization direction. Since we will be dealing with MCNF formulations, the minimization optimization direction will be standard for this research. Each objective is formed as a linear equation using a set of structural variables, denoted by \( x \), and a vector of objective function or criterion coefficients (also known as gradients), denoted by \( c^i \) where \( i = 1, \ldots, k \).

Letting \( R \) represent the set of real numbers, \( x^i \in R^n \) denotes a point in decision space where \( x^i = [x^i_1, \ldots, x^i_n]^T \). Here, \( n \) is the number of decision variables and \( T \) represents the standard vector transpose operator. The feasible region \( S \) is then the portion of decision space lying in the positive orthant \( (x \geq 0) \) and defined by the linear constraints \( Ax = b \). Here, \( A \), the constraint coefficient matrix, is an \( n \times m \) matrix formed
by the \( n \) coefficients of each of the \( m \) constraints. Vector \( \mathbf{b} \), the *RHS vector*, is formed by the RHS values of the \( m \) constraints.

The criterion coefficient vector, \( \mathbf{c}^i = [c_{i1}^i, \ldots, c_{i_n}^i] \), can be used to form the \( i \)th row of a \( k \times n \) matrix \( \mathbf{C} \) known as the criterion coefficient matrix. Using matrix \( \mathbf{C} \) and a vector \( \mathbf{z} \) of the \( k \) criterion values, we may write our general MOLP model in matrix form as

\[
\min \{ \mathbf{C} \mathbf{x} = \mathbf{z} \mid \mathbf{x} \in \mathcal{S} \}.
\]

**Example 2.1** - Consider the following MOLP:

\[
\begin{align*}
\text{min: } & 2x_1 - 2x_2 = z_1 \\
\text{min: } & x_1 + 2x_2 = z_2 \\
\text{s.t.: } & 3x_1 + x_2 \geq 14 \\
& x_1 + x_2 \geq 8 \\
& x_1 + 4x_2 \geq 14 \\
& 7x_1 + 8x_2 \leq 78 \\
& x_i \geq 0 \text{ for all } i
\end{align*}
\]

In this formulation, inequality constraints define the feasible region. To bring the formulation into our standard form, variables may be added to each inequality in the form of slack (for less than constraints) and surplus variables (for greater than constraints), \( s_i \), where \( i = 1, \ldots, m \). The objective function coefficients for nonstructural variables are set to 0. The new formulation is:
\min: \quad 2x_1 - 2x_2 = z_1 \\
\min: \quad x_1 + 2x_2 = z_1 \\
\text{s.t.:} \quad \begin{align*}
3x_1 + x_2 - s_1 &= 14 \\
x_1 + x_2 - s_2 &= 8 \\
x_1 + 4x_2 - s_3 &= 14 \\
7x_1 + 8x_2 + s_4 &= 78 \\
x_i, s_i \geq 0 \text{ for all } i
\end{align*}

From this formulation, the following matrices and vectors are formed:

\[
C = \begin{bmatrix}
2 & -2 & 0 & 0 & 0 & 0 \\
1 & 2 & 0 & 0 & 0 & 0
\end{bmatrix}, \quad z = \begin{bmatrix}
z_1 \\
\end{bmatrix},
\]

\[
A = \begin{bmatrix}
3 & 1 & -1 & 0 & 0 & 0 \\
1 & 1 & 0 & -1 & 0 & 0 \\
1 & 4 & 0 & 0 & -1 & 0 \\
7 & 8 & 0 & 0 & 0 & 1
\end{bmatrix}, \quad b = \begin{bmatrix}
14 \\
8 \\
14 \\
78
\end{bmatrix},
\]

\[
x = \begin{bmatrix}
x_1 \\
x_2 \\
s_1 \\
s_2 \\
s_3 \\
s_4
\end{bmatrix}.
\]

A point in decision space that simultaneously minimizes all of the objectives is rare and the goal is to help the DM find the point that is most satisfactory given the tradeoffs in objectives values.

**2.3 Basic Concepts of Multiple Objective Programming**

Since there is generally no single solution that is better than all other solutions for all criteria of a MOLP, the search methods appropriate for a LP are no longer sufficient.
Instead, methods and theory have evolved to aid a DM in selecting from a set of possible solutions the one that is most preferred. In this section, we discuss the basic concepts related to this aid including: modeling of DMs preferences, characteristics of problem solutions, the notions of dominance and efficiency, and the use of weights in multiple objective problems.

### 2.3.1 Utility Function

When searching for solutions to a multiple criteria decision problem, the existence of a utility function is often assumed. A utility function, \( U: \mathbb{R}^k \rightarrow \mathbb{R} \), maps the criterion vectors onto the real line such that higher values of \( U \) correspond to more preferred solutions. It is assumed that the DM attempts to maximize the function in order to obtain the solution of greatest utility. While this assumption is intuitive it is not very practical because it is usually very difficult to obtain a mathematical representation of the utility function \( U \). In the context of the utility function we can speak of a DM’s preferred solution as optimal even though each individual criterion may not be optimized.

In the absence of a well-defined utility function, many methods assume certain characteristics for this function and then use implicit information obtained from the DM to search through the set of potential solutions for a DM’s most preferred action.

### 2.3.2 Feasible, Nondominated and Efficient Solutions

The set of feasible solutions of a MOLP problem in decision space, \( S = \{ \mathbf{x} \in \mathbb{R}^n \mid \mathbf{A} \mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0 \} \) is a convex polyhedron, since it is the intersection of the half-spaces defined by the inequalities \( \mathbf{a}_i \mathbf{x} \leq b_i, \ldots, \mathbf{a}_m \mathbf{x} \leq b_m \) and \( \mathbf{x} \geq 0 \) where \( \mathbf{a}_i \) is the \( i \)-th
row of $A$, the constraint coefficient matrix. Typically, the feasible region is a closed set but it may be unbounded.

A solution, $\bar{x} \in \mathbb{R}^n$, in decision space is feasible if and only if $\bar{x} \in S$. If the objectives are modeled using linear functions, we may also consider potential solutions in criterion space by mapping $S$, the feasible region in decision space, to $Z$, the feasible region in criterion space using

$$Z = \left\{ z \in \mathbb{R}^k \left| z = Cx, \ x \in S \right. \right\}.$$  

A solution in criterion space, $\bar{z} \in \mathbb{R}^k$, is feasible if and only if $\bar{z} \in Z$.

In criterion space, we can divide the set of feasible points into those that are dominated and those that are nondominated. If $z^1 \in \mathbb{R}^k$ and $z^2 \in \mathbb{R}^k$ are two criterion vectors corresponding to minimization objectives, then $z^1$ dominates $z^2$ if and only if $z^1 \leq z^2$ and $z^1 \neq z^2$. A criterion vector $\bar{z}$ is nondominated if there is not another feasible criterion vector that dominates it.

When searching through the feasible region in criterion space for a DM’s most preferred solution, we will be interested only in nondominated points. This is because for each dominated point there will exist some point that is at least as good for all criteria and better for at least one criteria. The set of all nondominated criterion vectors is denoted by $N \subset \mathbb{R}^k$.

In decision space, we can divide feasible points into those points that are efficient and those points that are inefficient. A point $\bar{x} \in S$ is efficient if and only if there does not exist another $x \in S$ such that $Cx \leq C\bar{x}$ and $C\bar{x} \neq Cx$. Otherwise, the point $\bar{x}$ is called inefficient. In other words, a point is efficient in decision space if and only if when
mapped into criterion space it is nondominated. A search of nondominated points in criterion space corresponds to some extent to a search of efficient points in decision space. The set of all efficient points in decision space is denoted as $E \subset R^n$. The set of all efficient points in decision space that are extreme is denoted as $E_x \subset R^n$.

![Figure 2.1 - Feasible Region in Decision Space](image)

$$x^1 = (3, 5)$$
$$x^2 = (2, 8)$$
$$x^3 = (10, 1)$$
$$x^4 = (6, 2)$$
$$c^1 = (2, -2)$$
$$c^2 = (1, 2)$$

Figure 2.1 - Feasible Region in Decision Space

The feasible regions in both decision and criterion spaces for the MOLP of Example 2.1 is shown in Figures 2.1 and 2.2. The set of efficient solutions includes all points on the boundary from $x^2$ to $x^1$ to $x^4$. The mapping of efficient points in decision space to nondominated points in criterion space is not necessarily one-to-one. Figure 2.2 illustrates that the feasible region in criterion space is not limited to the strictly positive orthant.
Two special points in criterion space are often used in MOLP solution methods, the ideal point and the nadir point. The **ideal point** $z^{\text{min}}$ in $\mathbb{R}^k$ for minimization criteria is the point whose coordinates are $(z_1^{\text{min}}, \ldots, z_k^{\text{min}})$ where $z_j^{\text{min}} = \min_{x} \{ c^i / x | x \in S \}$ for $j = 1, \ldots, k$. This is the point at which all objectives would be simultaneously minimized. Usually, the ideal point is infeasible but in many methods potential solutions are evaluated based on how close they are to the ideal.

The **nadir point**, $z^{\text{max}}$, is defined as the point $(z_1^{\text{max}}, \ldots, z_k^{\text{max}})$ in criterion space such that each $z_j^{\text{max}}$ equals the maximum value for objective $j$ over the nondominated set.

While the ideal point is fairly easy to find for most multiple objective models, the nadir point is more difficult to determine. The nadir point can be used in like manner to the ideal point by evaluating potential solutions based on how distant the point is from the nadir. The ideal point and the nadir point are often used together to specify the range for

Figure 2.2 - Feasible Region in Criterion Space

$z^1 = (-4, 13)$
$z^2 = (-12, 18)$
$z^3 = (18, 12)$
$z^4 = (8, 10)$
each objective over the nondominated set. Figure 2.3 illustrates the ideal, $z^{\text{min}}$, and nadir, $z^{\text{max}}$, points for the example.

\[
\begin{align*}
z^1 &= ( -4, 13 ) \\
z^2 &= ( -12, 18 ) \\
z^3 &= ( 18, 12 ) \\
z^4 &= ( 8, 10 ) \\
z^{\text{min}} &= ( -12, 10 ) \\
z^{\text{max}} &= ( 18, 18 )
\end{align*}
\]

Figure 2.3 - The Ideal and Nadir Points

The Tchebycheff metric can be combined with a utopian vector $z^{**}$ as a tool in solving the MOLP. The vector $z^{**}$ is determined by

\[
z^{**}_i = z^{\text{min}}_i - \varepsilon_i = \min f_i(x) \mid x \in S - \varepsilon_i \quad \text{for } i = 1, \ldots, k
\]

and $\varepsilon_i \geq 0$. The distance between any $z \in \mathbb{R}^k$ and $z^{**}$ can then be found using a weighted Tchebycheff metric of the form

\[
\|z^{**} - z\|_\infty^\rho = \max_{i=1,\ldots,k} \lambda_i |z^{**}_i - z_i|.
\]

With $\rho$ sufficiently small, we may also define a member of the family of augmented weighted Tchebycheff metrics as

\[
\|z^{**} - z\|_\infty = \|z^{**} - z\|_\infty^\rho + \rho \sum_{i=1}^k |z^{**}_i - z_i|.
\]
2.3.4 Payoff Matrix

Payoff matrices are often utilized to provide the values for the ideal point and an estimate of values for the nadir point. A payoff matrix is a $k \times k$ table of values with rows that result from individually maximizing each objective. Special measures are usually taken when calculating the payoff matrix so that the criterion rows will be nondominated when facing the possibility of alternative optima. The components of the ideal point are found along the main diagonal of the matrix. Taking the maximum column value for each objective is often used in making estimates of the components of the nadir point. Note that the nadir point determined by the payoff matrix is only an estimate of the true nadir point. Experimentation has shown that for most multiple objective programs, the nadir value will be underestimated for one or more of the objectives (Steuer (1989) and Isermann (1977)). In addition, the amount the nadir point is underestimated is typically more severe as the problem size increases.

Example 2.2 - Consider the following MOLP:

\[
\begin{align*}
\text{min:} & \quad 2x_1 - 2x_2 = z_1 \\
\text{min:} & \quad x_1 + 2x_2 = z_2 \\
\text{min:} & \quad x_2 = z_3 \\
\text{s.t.:} & \quad 3x_1 + x_2 \geq 14 \\
& \quad x_1 + x_2 \geq 8 \\
& \quad x_1 + 4x_2 \geq 14 \\
& \quad 7x_1 + 8x_2 \leq 78 \\
& \quad x_i \geq 0 \text{ for all } i 
\end{align*}
\]

By minimizing each of the three objectives one at a time we obtain the payoff matrix of Figure 2.4. From this payoff matrix, we would set $z^{\text{min}} = (-12, 10, 1)$ and estimate $z^{\text{max}} =$
(18, 18, 8). Incidentally, we are able to obtain the correct nadir point from the payoff table for this problem because the problem size is small.

<table>
<thead>
<tr>
<th></th>
<th>$Z_1$</th>
<th>$Z_2$</th>
<th>$Z_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z^1$</td>
<td>-12</td>
<td>18</td>
<td>8</td>
</tr>
<tr>
<td>$z^2$</td>
<td>8</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>$z^3$</td>
<td>18</td>
<td>12</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2.4 - Payoff Table

2.3.5 Weak and Strong Efficiency

As noted, a characterization of the efficient set is important to research concerning multiple objective problems. Many authors, including Charnes, Cooper, and Evans (1972) and Yu (1974), have developed characterizations and methods for generating the efficient set. In addition to determining characteristics of efficient points, other notions of efficiency such as weak efficiency have been developed.

A point $x^* \in S$ is weak efficient if and only if there is not another $x \in S$ such that $Cx < Cx^*$. Efficient points that are defined as before are then called strong efficient. In all cases, the set of (strong) efficient points, $E$, is a subset of the set of weak efficient points, $E^w$, and in most cases $E = E^w$.

2.3.6 Domination Sets

A method involving cones may be used to detect efficiency on a graph. This method uses a domination set to test for the efficiency at a given point on the graph of the feasible region $S$. Assuming an MOLP, let $\bar{x} \in S$ and let $C^\leq$ be a semi-positive polar cone that is generated by the minimization gradients of the $k$ objective functions where
\[ C^\geq = \{ y \in \mathbb{R}^n \mid -Cy \geq 0, -Cy \neq 0 \} \cup \{ 0 \in \mathbb{R}^n \} . \]

Then the domination set \( D_x \) associated with the point \( x \in S \) is given by the set addition of \( \{ x \} \) and \( C^\geq \), \( D_x = x \oplus C^\geq \). The domination set \( D_x \) contains \( x \) and all points in \( \mathbb{R}^n \) whose criterion vectors dominate \( x \). Steuer (1989) shows that if \( D_x \cap S = \{ x \} \), then the point is efficient. Also, the larger the domination set the greater the chance that a point will be inefficient since there is a greater likelihood of including other points of the feasible region. Figure 2.5 illustrates the use of domination sets on the graph of the feasible region in decision space for the MOLP of Example 2.1. The domination sets here show that the boundary points between \( x^2 \) and \( x^4 \) are efficient while the point \( x^3 \) is inefficient because its domination set contains other points besides \( x^3 \).

\[
\begin{align*}
x^1 &= (3, 5) \\
x^2 &= (2, 8) \\
x^3 &= (10, 1) \\
x^4 &= (6, 2) \\
e^1 &= (2, -2) \\
e^2 &= (1, 2)
\end{align*}
\]

Figure 2.5 - Domination Sets in Decision Space

The concept of domination set may also be applied in criterion space. The primary difference is that when the MOLP is graphed in criterion space the criterion gradients correspond to the axes. Assuming an MOLP, let \( \mathbf{z} \in \mathbb{Z} \), let \( \mathbf{I}_k \) be a \( k \times k \) identity matrix and
let $C_z^\geq$ be a semi-positive polar cone that is generated by the minimization gradients of the $k$ objective functions where

$$C_z^\geq = \{ y \in R^k | -1, y \geq 0, -I, y \neq 0 \} \cup \{ 0 \in R^k \}.$$ 

Then the domination set $D_z$ associated with the point $\bar{z} \in Z$ is given by the set addition of $\{ \bar{z} \}$ and $C_z^\geq$ or $D_z = \bar{z} \oplus C_z^\geq$. The domination set $D_z$ in criterion space contains $\bar{z}$ and all points in $R^k$ whose corresponding criterion vectors dominate $\bar{z}$. Figure 2.6 shows the domination set concept used in criterion space for the MOLP of Example 2.1.

$$z^1 = (-4, 13)$$

$$z^2 = (-12, 18)$$

$$z^3 = (18, 12)$$

$$z^4 = (8, 10)$$

Figure 2.6 - Domination Sets in Criterion Space

### 2.3.7 The Criterion Cone

For the MOLP formulated as

$$\min: \quad c^1 x = z_1$$

$$\vdots$$

$$\min: \quad c^k x = z_k$$

$$s.t.: \quad x \in S$$

The criterion cone is the convex cone generated by the $k$ minimization objective function gradients $\{-c^1, \ldots, -c^k\}$. The criterion cone is important because it can indicate the size of
the efficient set. In addition, the relative interior of the criterion cone provides an alternative to the use of domination sets for graphically detecting efficiency.

In most cases, the efficient set $E$ for an MOLP is a strict subset of the feasible region $S$ unless the null vector condition holds. The null vector condition holds if there exists a strictly positive linear combination of the objective function gradients that produces the null vector. In other words, the null vector condition is in effect if there exists an $\alpha \in R^k$ with $\alpha_i > 0$ for all $i$ such that $\sum_{i=1}^k \alpha_i c_i = 0 \in R^n$. When the null vector condition is in effect, the efficient set $E$ is equal to the feasible region $S$.

The criterion cone is also useful as an aid in detecting efficiency on a graph. Let $V$ be a cone generated by $\{v_1,\ldots,v_k\}$, the relative interior of $V$ consists of all strictly positive linear combinations of the $v_i$. The relative interior of $V$ is denoted $\text{rel } V$. The relative boundary of $V$ is the difference in $V$ and its relative interior. Steuer (1989) shows that by minimizing all vectors that point into the criterion cone we can generate the efficient set.

In most cases, when the null vector condition does not hold, the efficient set tends to get larger as the size of the criterion cone increases.

**Example 2.3** - Consider the following maximization MOLP which is graphically illustrated in Figure 2.7:
max : \( x_1 + 6x_2 = z_1 \)
max : \( 5x_1 + 5x_2 = z_2 \)
max : \( 7x_1 + x_2 = z_3 \)
s.t. : \( x_1 \leq 8 \)
\( x_2 \leq 8 \)
\( x_1 + x_2 \leq 12 \)
\( x_1 + 3x_2 \leq 26 \)
\( 3x_1 + x_2 \leq 26 \)
\( x_i \geq 0 \) for all \( i \)

In Figure 2.7, the vectors \( c^1, c^2 \) and \( c^3 \) represent the gradients defined by the three objective functions and the bounded area represents the feasible region defined by the constraints. The points \( x^1 \) through \( x^6 \) represent the extreme points on the boundary of the feasible region. Consider this MOLP with only the first objective function \( z_1 \). In this case, the MOLP becomes a LP with \( x^2 = (2, 8) \) providing the optimal value of \( z_1 = 50 \). When
the second objective function \( z_2 \) is added to the model the efficient set grows from a single point to include all points on the boundary of the feasible region between \( x^2 \) and \( x^4 \). As the third objective is added to the model, the efficient set grows to include all points on the boundary of the feasible region between \( x^2 \) and \( x^5 \).

2.3.8 Efficient Facets

The notion of facet along with several definitions can be useful in describing the shape of the efficient set. Let \( F \) be an \( f \)-facet of the feasible region \( S \). Then \( F \) is an efficient facet if and only if all points \( \overline{F} \in F \) are efficient. \( F \) is a maximal facet if and only if there does not exist a \( g \)-facet \( G \) of \( S \) such that \( GF \subset \) and \( f < g \).

Let \( F \) be an efficient \( f \)-facet of the feasible region \( S \). Then \( F \) is a maximally efficient facet if and only if there does not exist an efficient \( g \)-facet \( G \) of \( S \) such that \( F \subset G \) and \( f < g \). When the null vector condition does not hold, the union of all maximally efficient facets gives the efficient set.

2.3.9 Theorems Concerning Efficient Points

Several important theorems have been presented in the literature concerning the characterization of efficient points (Zeleny (1982), Steuer (1989), and Vinke (1992)). For the theorems presented here, it is helpful to define to sets of weighting vectors. Let \( \Lambda \) represent a set of strictly positive weighting vectors such that

\[
\Lambda = \left\{ \lambda \in \mathbb{R}^k \mid \lambda_i > 0, \sum_{i=1}^{k} \lambda_i = 1 \right\}.
\]

Let \( \overline{\Delta} \) represent a set of strictly positive weighting vectors such that
Theorem 2.1 Assume an MOLP. Let $S$ be convex and $\bar{x} \in S$ minimize the weighted sums LP \[ \min \left\{ \bar{x} C \bar{x} \mid \bar{x} \in S \right\} \] where $\bar{x} \in \Lambda$. Then $\bar{x}$ is an efficient point.

**Proof** (by contradiction): Let $\bar{x}$ be inefficient. Then there exists some $\bar{x}^1 \in S$ such that $C\bar{x} \leq C\bar{x}^1$ and $C\bar{x} \neq C\bar{x}^1$. Since $\bar{x} \in \Lambda$ is strictly positive, $\bar{x} C \bar{x} \leq \bar{x} C \bar{x}^1$. This contradicts the assumption that $\bar{x} \in S$ minimizes the LP, $\bar{x}$ must be efficient.

Theorem 2.1 shows that the optimization of a positive linear combination of the criteria always yields an efficient point. The converse of this theorem can only be true under certain convexity conditions. The linear program of Theorem 2.1 is known as the weighted-sums LP. This theorem is the basis for an approach to solving MOLP called the point estimate weighted sums approach. With this approach, weights are assigned to each of the objectives. These weights are often assigned in a manner to reflect the relative importance of each criterion to the DM. Then, the weighted objectives are summed to form a composite, single objective so that the problem can be solved with normal linear programming techniques. Difficulties arise, however, when trying to determine the appropriate weighting vector. In addition, when the feasible region is not convex, possibly resulting from non-linearity or restriction to integer solutions, there may be some efficient solutions that are not reachable through the use of weights in this way.

The weighted sums vector from Theorem 2.1 can be thought of as directional vector anchored at the origin and pointing in a direction of improving efficiency
corresponding to the particular weight vector used. The following theorems show that the search for efficient points may be carried out from a different point of reference than the origin, namely the utopian criterion vector \( \mathbf{z}^{**} \).

**Theorem 2.2** Assume an MOLP. If \( \mathbf{x} \in \mathcal{S} \) (\( \mathcal{S} \) not necessarily concave) minimizes the augmented weighted Tchebycheff distance

\[
\max_j \lambda_j \left( z_j^{**} - c_j x \right) - \sum_{j=1}^k \rho_j c_j x
\]

where \( \lambda_j > 0, \ z_j^{**} < z_j^{*} \), the arbitrarily small \( \rho_j > 0 \) and \( j=1,\ldots,k \), then \( \mathbf{x} \) is efficient.

Background and theory leading to a proof of Theorem 2.2 is provided in Steuer (1989). Theorem 2.2 provides another approach for characterizing efficient solutions. This theorem uses an augmented weighted Tchebycheff metric to locate efficient points relative to an ideal criterion vector. Theorem 2.2 states that for a given set of weights, the point in \( \mathcal{S} \) that is closest to the ideal point when measured by the augmented weighted Tchebycheff metric is efficient. The augmentation is necessary to separate efficient points from weakly efficient points that may be derived from using a weighted Tchebycheff metric sans augmentation.

Both theorems are important in that they demonstrate that the determination of efficient points of the feasible region may be obtained by solving parametric optimization problems. Procedures discussed in section 2.4, will utilize these results to develop interactive algorithms.
2.3.10 Characterizing the Nondominated Set

When the constraints and objective functions are all linear and variables are continuous, the feasible region in criterion space is convex. In this case, if there exists a nondominated point, then there exists at least one nondominated extreme point. The nondominated set can be completely characterized by finding all nondominated extreme points and nondominated unbounded edges, if any. All other non-extreme nondominated points can be portrayed as convex combinations of the nondominated extreme points and unbounded edges. If the feasible region is a bounded, convex set, then all nondominated points can be characterized as either nondominated extreme points or convex combinations of the nondominated extreme points. In addition all nondominated points will be boundary points. For this case, any of the three theorems listed above are helpful in characterizing the nondominated set. For MCNF problems, this will be the case only as long as non-integer values are allowed for arc flows.

When the feasible region is not defined by linear functions or continuous variables, the nondominated set is not as easily characterized. If non-linearity exists, nondominated points will not necessarily be convex combinations of nondominated extreme points.

2.3.11 Supported and Unsupported Points

Let \( Z^\# = \text{Convex Hull of } \{N\} \cup \{z \mid z \in \mathbb{R}^k, z \geq z^* \text{ for some } z^* \in Z\} \) and let \( z \in N \).

Then if \( z \) is on the boundary of \( Z^\# \), \( z \) is a supported nondominated criterion vector. Otherwise, \( z \) is an unsupported nondominated criterion vector. Unsupported nondominated criterion vectors will be dominated by some convex combination of other nondominated criterion vectors. Nondominated extreme points are always supported.
Unsupported criterion vectors can only occur when the feasible region is made up of discrete integer values or when it is defined by nonlinear constraints. The possibility of unsupported points needs to be addressed when considering MCNF problems with integer values.

**Example 2.4** - Consider the following MOLP:

\[
\begin{align*}
\text{min:} & \quad 4x_1 + 4x_2 + x_3 + 4x_4 + 2x_5 = z_1 \\
\text{min:} & \quad 2x_1 - x_2 + 2x_3 - 4x_4 + 2x_5 = z_2 \\
\text{s.t.:} & \quad x_1 + x_2 + x_3 = 3 \\
& \quad -x_1 + x_4 = 0 \\
& \quad -x_2 + x_5 = 0 \\
& \quad -x_3 - x_4 - x_5 = -3
\end{align*}
\]

\[x_i \geq 0 \text{ and integer for all } i\]

The feasible region in criterion space for this MOLP is shown in Figure 2.8. The convex hull \(Z^\#\) is represented by the shaded area on the graph. Here the points \(z^1=(22, -3)\), \(z^2=(15, 1)\), and \(z^3=(8, 5)\).
3), \( z^2 = (15, 1) \), and \( z^3 = (8, 5) \) are unsupported nondominated solutions for the MOLP as shown using domination sets. These points are not on the boundary of the convex hull \( Z'^n \).

When unsupported nondominated criterion vectors exist, they will not be found using the result of Theorem 2.1. They can however be found using the Tchebycheff metric of Theorem 2.2.

### 2.3.12 The Use of Weights in Multiple Objective Programming

Weights have been used in various ways in multiple objective solution procedures. One way that weights have been used is as a measure of the relative importance of the various objectives. This idea is further sustained by the weighted-sums LP of Theorem 2.1. While this is valid and straightforward in many instances, the indiscriminate use of weights in this way has several drawbacks. First, it is usually difficult to obtain good estimates of weights from the DM. Second, it can be shown that small changes in weights can often lead to drastic changes in the indicated solutions. Finally, in the cases of nonlinear and integer problems, a subset of the nondominated set, namely unsupported points, will be left out of the possible solutions investigated. Despite these drawbacks, many procedures have incorporated weights in this manner.

Another common use for weights in multiple objective procedures utilizes the family of weighted \( L_p \)-metrics. For many procedures, the weighting vector is used to define a metric that will probe the nondominated set based on a point’s distance from a utopian point. The weights themselves usually have no meaning to the DM or the procedure itself other than to define a probe’s direction from the ideal point. The
Weighted Tchebycheff metric has become one of the most popular metrics of this sort used in multiple objective procedures.

2.4 Survey of Interactive Methods

The most successful approaches to finding the DM’s most preferred solution of multiple objective problems have been the interactive methods. These methods seek to combine the fundamental theory of multiple criteria decision making with information gathered from a dialog with the DM. In general, most interactive methods work as follows. After a model is developed for the problem at hand, a first computation step is conducted to provide the DM with a set of initial solutions to consider. The DM then provides preference information by answering various questions about the potential solutions under consideration. This information is incorporated into the next computation step in order to provide a next generation of potential solutions. The process is repeated until either the DM is satisfied or a stopping rule comes into effect.

Important interactive MOLP methods include: STEM (Benayoun et al. (1971)), the method of Geoffrion, Dyer and Feinberg (1972), the method of Zionts and Wallenius (1976), the Interactive Weighted Sums approach (Steuer and Schuler (1978)), the Visual Interactive approach (Korhonen and Laakso (1986)), and the Interactive Weighted Tchebycheff method (Steuer and Choo (1983)). These methods employ many of the basic ideas discussed earlier.

2.4.1 The Interactive Weighted Tchebycheff Approach

With the Interactive Weighted Tchebycheff (IWT) method of Steuer and Choo (1983), potential solutions are presented using weights to sample from a progressively
smaller weight space. One difference from earlier procedures is that the weights are used in a different way. Instead of weights to representing the relative importance of criterion vectors, with IWT weighting vectors define different weighted Tchebycheff metrics.

Several advantages of the IWT are derived from its use of the weighted Tchebycheff metric to probe the nondominated set. It is not limited to extreme point solutions. In addition, unsupported nondominated criterion vectors may also be computed allowing the approach to be useful for discrete alternative and nonlinear problems. While the number of calculations required per iteration is high, the calculations are fairly uncomplicated and can be carried out using conventional single objective solvers. The information that is solicited from the DM is also fairly natural and of a qualitative nature.

2.5 The Multiple Objective Programming Literature

One reference on multiple objective linear programming is *Multiple Criteria Optimization: Theory, Computation and Application* by Steuer (1989). This volume introduces and builds on the basic concepts of MOLP such as: utility functions, dominance, efficiency, and characteristics of efficient solutions. The author describes the vector maximum problem and its relationship to the determination of the efficient set. The book describes most of the main methods of solving MOLP that existed at the time of publication, including interactive methods. In particular, several sections and chapters are devoted to the method of Steuer and Choo, which utilizes the Tchebycheff weighted distance to locate nondominated solutions of a MOLP.

Several classic texts that were consulted during the course of this research include Keeney and Raiffa (1976), Zeleny (1982), and Yu (1985). Concepts introduced in these
volumes include the use of utility functions, ideal points and the characterization of the
efficient set. Vinke (1992) provides a more recent text that provides a brief overview of
MDA and the major related theory and methods. Vincke’s text provides a contrast
between the “American” schools of multiattribute utility theory and MOLP and the
“French” school of MDA preference modeling.

A frequently studied aspect of MOLP concerns the characterization of the efficient
solution set. Studies have explored such questions as: how to determine if a solution is
efficient; how to enumerate all efficient corner points; what metrics are useful in
determining the “nearness” of efficient points to various reference points; and how can we
reduce the efficient set to a manageable level. Authors including Yu and Zeleny (1975),
Gal (1977), Isermann (1977), and Ecker and Kouada (1978) have worked on algorithms
for computing all efficient extreme points of a MOLP. Evans and Steuer (1973), Philip
(1972), and Ecker, Hegner, and Kouada (1980) provide further examples of this research.

One class of MOLP solution procedures is made up of methods called *interactive
procedures*. Interactive procedures generally consist of alternating computation steps and
dialogue with the decision maker. The first computation step provides a first solution or
solution set for the DM’s consideration. The DM then reacts by giving information about
his preferences related to the current set of solutions. This information is then used in
subsequent computations to narrow the search. Interactive MOLP methods include:
STEM (Benayoun *et al.* (1971)), the method of Geoffrion, Dyer and Feinberg (1972), the
method of Zionts and Wallenius (1976), the Interval Criterion Weights approach (Steuer
(1977)), the Interactive Weighted Sums approach (Steuer and Schuler (1978)), the
Reference Point method (Wierzbicki (1980) and (1982)), the Interactive Weighted
Tchebycheff method (Steuer and Choo (1983)), and the Visual Interactive approach (Korhonen and Laakso (1986)). A paper by Steuer and Gardiner (1990) is helpful in gaining an overall insight into the current status and directions of multiple objective programming.

The ADBASE Multiple Objective Programming Package by Steuer (1998) was used to carry out computation and empirical testing in as well as the software developed specifically for the research described in this dissertation. In addition to finding all efficient extreme points and unbounded edges for MOLP, ADBASE includes a MOLP random problem generator. Features of the generator are described in Steuer (1994). In those instances where insight into 3-objective MOLPs is desired, the TRIMAP software has been utilized (Climaco and Antunes (1989)).

2.6 Conclusion

The discussion of sophisticated interactive methods illustrates how many of the basic ideas of multiple objective decision aid have been creatively incorporated into decision support methods. Several commonalities can be seen when examining these methods. First, most successful techniques for multiple objective decision aid have been designed around a dialog with the DM. It is generally accepted that DM interaction is central and crucial to the understanding and solution of multiple objective problems. Because of this much thought has been carried out about the nature of the discussion with the DM. The methods illustrate a variety of questions that must be answered by the DM. To date, there is still much continuing research and debate about what types of questions
are appropriate and of most aid to the DM. However, there is agreement that the questioning must be manageable by and supportive of the DM’s decision making process.

A second aspect of these methods relates to the degree of complex calculations required. Many of the methods were designed to keep the amount of computation to a minimum or to at least keep the bulk of the computation process in the background. In most cases, it is clear that an attempt has been made to develop methods that could be carried out with existing single objective solvers with only minor modifications.

In most procedures, the main stopping rule is determined by the DM’s satisfaction with the current solution or their unwillingness to continue. In many decision situations, convergence to a satisfactory solution may be more important than mathematical convergence. The speed of mathematical convergence will be important if the DM becomes overburdened by the process and decides to stop before a satisfactory solution is obtained. Finally, a trend with these methods is the degree to which the method aids the DM in learning more about a particular problem while the solution search takes place.

The theory and methods discussed here are just a small portion derived from the relatively short but rich history of research into solving multiple objective decision problems. The foundation is provided for further research in the area and the application of these ideas to various optimization problems. In particular, much of the research presented later will apply these ideas to multiple criteria network flow problems.
CHAPTER 3

MINIMUM COST NETWORK FLOWS: BASIC THEORY

3.1 Introduction

One of the most frequently encountered optimization problems in practice is the minimum cost network flow (MCNF) problem and its variants. Problems of this type include shortest path, assignment, location, transportation, transshipment, maximum flow and minimum cost problems.

In addition to their wide applicability, several features of network optimization problems have contributed to their popularity. The formulation of network problems is straightforward and in many cases can be presented graphically providing an easy to understand format for the DM. Their formulation allows for the use of several important data structures. Due to their special structure, solution algorithms have been developed which are hundreds of times faster than traditional LP approaches. In addition, research into improving network optimization techniques has led to insights and approaches for improving the solutions of non-network optimization problems (Evans and Minieka (1992)). Finally, they arise frequently as important sub-problems of other, more complex problems.

In this chapter, an overview of network optimization is presented. First, basic concepts of minimum cost network flow models are discussed. Second, an overview of the main algorithmic strategies for network models is provided. Third, there is a review
of MCNF literature. Finally, the chapter concludes with a summary of how these concepts will be useful for the research described in this dissertation.

3.2 Basic Concepts of Minimum Cost Network Flows

In this section, an overview of the basic concepts related to MCNF problems is provided. This provides a basis for understanding the strategies that follow as well as the research described in subsequent chapters.

3.2.1 Terms and Definitions

A directed graph, $G = (N, A)$, is defined to consist of a set of nodes, $N$, and a set of directed arcs, $A$. The number of nodes and arcs, denoted by $n$ and $m$ respectively, are assumed to be positive and finite. An arc $(i, j)$ is an ordered pair connecting nodes $i$ and $j$. Arc $(i, j)$, which is incident to $i$ and $j$, is outgoing from node $i$ and incoming to node $j$. Node $i$ is called the start node and node $j$ is called the end node of arc $(i, j)$. The degree of a node $i$ is equal to the number of arcs that are incident to it. The indegree of a node is the number of incoming arcs to the node and the outdegree is the number of outgoing arcs away from the node. The sum of indegrees of all nodes in a network equals the sum of the outdegrees of all nodes and both are equal to the number of arcs in the network. A graph is bipartite if its nodes can be divided into two sets, $X$ and $Y$, so that all arcs have their start nodes in $X$ and their end nodes in $Y$.

In a directed graph, a sequence of $k$ nodes $(n_1, \ldots, n_k)$ and a corresponding sequence of $k-1$ arcs such that the $i$th arc in the sequence is either forward $(n_i, n_{i+1})$, or backward $(n_{i+1}, n_i)$ is called a path, $P$. Node $n_1$ is called the start node and node $n_k$ is called the end node of $P$. If all arcs in $P$ are oriented in the same direction then $P$ is a directed
path. If \( n_i \) and \( n_k \) are the same, then \( P \) is a cycle. A path is simple if none of the nodes are repeated within the path. A cycle is called simple if no nodes except the start and end nodes are repeated. A cycle is a directed cycle if it can be considered a directed path with the same start and end node. In this research, we will allow only one arc between any two nodes in the forward direction and only one arc between any two nodes in the reverse direction. This condition is not limiting, since any network with multiple arcs in the same direction between two nodes may be transformed into a network without multiple arcs. Nevertheless, for a complete specification of a path or a cycle, the arc list must be specified along with the nodes.

Figure 3.1 - A Directed Network

A directed network is shown in Figure 3.1. Several of the concepts described above are illustrated here. For example, the indegree of node 6 is three while its outdegree is one. One path, \( P^1 \), from node 1 to node 7 consists of nodes \( N_{p^1} = \{1, 2, 3, 6, 7\} \) and arcs \( A_{p^1} = \{(1, 2), (3, 2), (3, 6), (6, 7)\} \). A directed path, \( P^2 \), from node 1 to node 7 consists of nodes \( N_{p^2} = \{1, 3, 4, 5, 7\} \) and arcs \( A_{p^2} = \{(1, 3), (3, 4), (4, 5), (5, 7)\} \).
While there are several cycles present in this network, none of them are directed cycles. The cycle, \( C^1 \), consisting of nodes \( N_{c^1} = \{1, 2, 3\} \) and arcs \( A_{p^1} = \{(1, 2), (3, 2), (3, 1)\} \) is an example of a cycle.

If for each pair of nodes \( i \) and \( j \) there is a path starting at \( i \) and ending at \( j \), then the graph is connected. If for each pair of nodes \( i \) and \( j \) there is a forward path starting at \( i \) and ending at \( j \), then the graph is strongly connected.

A cut is a partition of the node set \( N \) into two subsets, \( K \) and \( \overline{K} = N - K \). For each cut, a set of arcs will be defined consisting of all arcs that have one endpoint in \( K \) and the other endpoint in \( \overline{K} \). We can then refer to the set of arcs as the cut, \([K, \overline{K}]\). We define an \( s-t \) cut as a cut with the property that node \( s \) is in set \( K \) of the cut and node \( t \) is in set \( \overline{K} \) of the cut. One possible 1-7 cut on the network of Figure 3.1 is illustrated in Figure 3.2. This cut consists of arcs \((2, 5)\), \((3, 6)\), \((4, 5)\), and \((4, 6)\). Here, \( K = \{1, 2, 3, 4\} \) and \( \overline{K} = \{5, 6, 7\} \).

![Figure 3.2 - A Cut](image)

A tree is a connected graph that contains no cycles. A collection of trees is known as a forest. A connected subgraph of a tree is known as a subtree. Trees are an integral
part of many network flow algorithms. A tree that has \( n \) nodes will have \( n-1 \) arcs. Nodes in a tree with degree = 1 are known as leaf nodes and every tree with \( n > 1 \) will have at least two leaf nodes. A unique path will connect every two nodes, \( s \) and \( t \), in a tree.

Designating a specially selected node of the tree as the root creates a rooted tree. If a tree created from a graph \( G = (N, A) \) contains all nodes \( N \) of the graph, then the tree is a spanning tree. The arcs of \( A \) that are included in a spanning tree are referred to as tree arcs and those that are not included are referred to as nontree arcs. Given a spanning tree \( T \) for graph \( G = (N, A) \) the addition of any nontree arc to \( T \) will create exactly one cycle. We refer to such a cycle as a fundamental cycle of \( G \) with respect to \( T \). Given a spanning tree \( T \) for graph \( G = (N, A) \) the subtraction of any tree arc from \( T \) will create a cut made up of two sub-trees. We refer to such a cut as a fundamental cut of \( G \) with respect to \( T \).

A spanning tree for the network of Figure 3.1 is shown in Figure 3.3.

![Figure 3.3 - A Spanning Tree](image)

Flows in a network are modeled using a flow vector \( \mathbf{x} = \{x_{ij} \mid (i, j) \in A\} \) with dimension equal to the number of arcs, \( m \). The flow vector can be considered the vector of decision variables for the network. Here, the components of the flow vector
correspond to the arcs in $G$ with $x_{ij}$ representing the amount of flow of some commodity on arc $(i, j)$. Unless otherwise noted, we will assume that $x_{ij}$ is nonnegative and possibly subject to an upper bound. We can also associate a divergence vector with the nodes of $G$. The divergence vector $y$ has components defined by

$$y_i = \sum_{j : i, j \in A} x_{ij} - \sum_{j : j, i \in A} x_{ji} \quad \forall i \in N.$$ 

In words, the divergence of node $i$ is the total flow leaving node $i$ less the total flow arriving to node $i$. The divergence vector has dimension $n$ equal to the number of nodes. For a given flow vector, if $y_i > 0$ for node $i$, then node $i$ is a source node for the flow vector $\mathbf{x}$ and if $y_i < 0$ for node $i$, then node $i$ is a sink node for the flow $\mathbf{x}$. If $y_i = 0$ for all nodes $i \in N$ for a given flow vector $\mathbf{x}$, then $\mathbf{x}$ is a circulation.

In some solution strategies it will be useful to decompose a flow vector into components such as simple paths and cycles. A path $P$ conforms to a flow vector $\mathbf{x}$ if $x_{ij} > 0$ for all forward arcs of $P$, $x_{ij} < 0$ for all backward arcs of $P$, and either $P$ is a cycle or the start and end nodes of $P$ are a source and sink, respectively. A simple path flow is a flow vector that occurs when a positive flow is sent along a simple path. In Bertsekas (1991) it is shown that a flow vector $\mathbf{x}$ can be decomposed into a sum of $t$ simple path flow vectors $\mathbf{x}^1, \ldots, \mathbf{x}^t$ that conform to $\mathbf{x}$ with $t \leq m + n$. If the flow vector $\mathbf{x}$ has integer components, then the simple path flow vectors may be chosen with integer components.

Many network flow algorithms utilize relative cost values on arcs called reduced costs for evaluating the next move in the solution procedure. The reduced costs are calculated based on values assigned to the nodes called node potentials or prices. These potentials are typically intermediate data that are computed for the nodes in the context of
the algorithm, often they represent dual variables. With respect to a given set of node potentials \( \mathbf{\pi} = (\pi_1, \ldots, \pi_n) \), the reduced cost of arc \((i, j)\) is \( c_{ij}^n = c_{ij} - \pi_i + \pi_j \). The reduced costs are often used instead of the actual arc costs during a pricing phase of a network flow algorithm to determine which arcs should be operated on next. A problem formulated and solved with reduced costs will result in the same final objective function value as one formulated and solved using actual arc costs (Ahuja, Magnanti, and Orlin (1993)).

Another common tool for network solution strategies is the residual network. A residual network is an alternative way of looking at the flow on a network. A residual network is formed to measure flow in terms of the change in flow possible for some given feasible solution. This is based on the idea that given a flow \( x^*_ij \) on arc \((i, j)\), an additional flow of \( u_{ij} - x^*_ij \) could be sent along the arc. At the same time, we could send a reverse flow \( x^*_ji \) from node \( j \) to node \( i \), in effect reversing the current flow over the arc. A residual network is created for a given flow vector \( \mathbf{x}^* \) by replacing each arc \((i, j)\) from the original network with two “new” arcs \((i, j)\) and \((j, i)\). For each new arc \((i, j)\), the cost is \( c_{ij} \) and the residual capacity is \( u_{ij} - x^*_ij \). For each new arc \((j, i)\), the cost is \(-c_{ij}\) and the residual capacity is \( x^*_ji \). Only arcs with positive residual capacity are included in the residual network which will be denoted \( G(\mathbf{x}^*) \). Feasible flows and costs in both the original and the residual networks are related (Evans and Minieka (1992)). This provides flexibility since we may work within whichever network is convenient and the results converted for the other.
A network and its residual network corresponding to a given flow vector is shown in Figure 3.4. In this figure, the flow vector is \( \mathbf{x} = (2, 3, 3, 1, 2) \). The arcs of the residual network have been labeled with \( c_{ij} \) and \( u_{ij} \) following the relationships described above. For example, new arc (1, 2) keeps cost \( c_{12} = 3 \) and takes a new capacity \( u_{12} = \text{"old"} u_{12} - x_{12} = 6 - 2 = 4 \). The new arc (2, 1) takes cost \( c_{21} = -3 \) and \( u_{21} = x_{12} = 2 \).

### 3.2.2 Minimum Cost Flow Problem Formulation

Of all network flow problems, the MCNF is the most fundamental. With this problem, we wish to ship some commodity from available supply nodes over a network to satisfy the demand at sink nodes and do this at the minimum cost. In the most general formulations, the amount of flow on any particular arc may be constrained by upper and lower bounds. For our purposes, we will assume a lower bound of zero for all arcs. This is not restrictive since any problem that has arcs with nonzero lower bounds may be converted easily into a problem with all arcs having zero lower bounds.
The MCNF problem is formulated as a directed graph $G = (N, A)$ that is defined by a set of $n$ nodes and a set of $m$ arcs. There are two quantities associated with each arc $(i, j)$, a cost $c_{ij}$ representing the cost per unit flow over $(i, j)$ and a capacity $u_{ij}$ representing the maximum amount of flow over $(i, j)$. A quantity $b_i$ representing the supply is associated with each of the $n$ nodes in the graph. If $b_i < 0$, the node $i$ is a sink, also called demand, node. If $b_i > 0$, the node $i$ is a source, also called supply, node. If $b_i = 0$, the node $i$ is known as a transshipment node. A flow vector represents the flow of the commodity on the network. The components of the flow vector correspond to the decision variables in the optimization model and may be represented as $x_{ij}$ for arc $(i, j)$.

The algebraic formulation of the minimum cost flow network is

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

subject to:

$$\sum_{(j \mid (i,j) \in A)} x_{ij} - \sum_{(j \mid (j,i) \in A)} x_{ji} = b_i$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \in A$$

For the model to be feasible, a further restriction of $\sum_{i=1}^{n} b_i = 0$ is usually imposed. In the model there are two groups of constraints. The first group sets the divergence of each node equal to the node supply. The second set represents the capacity constraints for each arc. For most cases, it is assumed that all model parameters are integer.

It is immediately evident that the algebraic formulation incorporates a special structure derived directly from its graphical formulation. In particular, all coefficients of the left-hand side (LHS) of the divergence constraints will be 1, 0, or -1. This special structure allows for specialized data structures and corresponding solution strategies that are as much as a hundred times faster than conventional linear programming solution
techniques (Evans and Minieka (1992)). In addition, the integrality assumption for the coefficients ensure integer arc flows for the optimal solution (Kennington and Helgason (1980)).

### 3.2.3 Variants of the Minimum Cost Flow Problem

Perhaps the simplest and most common of all network flow problems is the *shortest path problem*. For this problem, the goal is to find a path from a starting node $s$ to another node $t$ such that the path is the shortest among all possible paths between the two nodes. Each arc $(i, j)$ in the network will have associated with it a cost $c_{ij}$ representing the length of the arc. By setting the supply of the start node $b_s = 1$, the supply of the end node $b_t = -1$, and the supply of all other nodes to 0, the problem can be solved as a minimum cost flow problem resulting in a flow of unit 1 along the shortest path from $s$ to $t$. Variations of the shortest path problem exist which incorporate multiple start and multiple end nodes.

The *maximum flow problem* is used to model situations in which the arcs of a network are capacitated but have no associated costs. For this problem, the objective is to send the maximum amount of flow feasible from a start node $s$ over the network to an end node $t$. The maximum flow problem can be modeled as a minimum cost flow problem by first setting the supply $b_i = 0$ for all nodes $i$ and the cost $c_{ij} = 0$ for all arcs $(i, j)$. Then we introduce a new arc from node $t$ to node $s$ with $c_{ts} = -1$ and $u_{ts} = \infty$. The minimum cost flow solution will maximize the flow across the new arc $(t, s)$ in order to obtain the minimum cost and since all of the flow on the new arc must pass through the network from $s$ to $t$ the solutions will also maximize the flow between these two nodes.
The assignment problem is another common variation of the minimum cost flow problem. For this problem, we have two sets $N_1$ and $N_2$ of equal size. We wish to match each element of set $N_1$ with one element of set $N_2$. For each possible matching between the sets there is an associated cost. The object is to determine a complete matching of the two sets at the lowest possible cost. The assignment problem may be modeled as a minimum cost flow network by defining nodes for each element in the two sets. Arcs are then added between the nodes of each set representing the possible assignments. The cost $c_{ij}$ for each arc represents the cost of assigning node $i$ to node $j$. Let the supply $b_i = 1$ for all $i \in N_1$; the supply $b_i = -1$ for all $i \in N_2$; and $u_{ij} = 1$ for all arcs $(i, j)$.

The transportation problem is another variation of the MCNF problem that like the assignment problem partitions the node set into two sets, supply and demand nodes. In this case, the sets may be unequal in size and the amount of supply at each node varies. Nodes in the supply set have $b_i > 0$ while nodes in the demand set have $b_i < 0$. For the problem to be feasible, $\sum_{i \in N} b_i = 0$. Arcs in the network are directed from the supply nodes to the demand nodes and have associated with them a cost $c_{ij}$ and a capacity $u_{ij}$. The objective is to transfer all of the supply to the demand nodes at the least cost. The primary difference between the transportation problem and the general minimum cost flow problem is the presence of intermediate nodes with $b_i = 0$ in the minimum cost flow problem. Because of this similarity, the minimum cost flow problem is sometimes known as the transshipment problem.

Other related models have been invaluable in gaining insights into the minimum cost flow problem and solution strategies. The circulation problem attempts to find the
lowest cost flow around a network with no introduced supply (i.e. \( b_i = 0 \) for all \( i \in N \)).

The *minimum spanning tree problem* is concerned with finding a spanning tree within a network that has the lowest cost of all possible spanning trees. Spanning trees themselves play an important role in many MCNF solution procedures. *Matching problems* are useful when it is desired to match some objects in a set with the goal of optimizing some criteria. Among the matching problems that exist, the assignment and transportation problems are special cases. Research into matching problems has also provided interesting results that are applicable to the minimum cost flow problem.

### 3.2.4 Network Data Structures

Important to any study of network flow models and solution procedures are the practical aspects of how computers are used to support the solution process. This includes how the model data are represented in the computer along with the relative efficiency of the various algorithms. In this section, several of the more popular data structures used to represent network flow problem data are discussed along with their respective advantages and disadvantages. For each representation, the node supplies are usually stored as a vector of size \( n \). While data structures are intrinsic to any computer implementation, Kennington and Helgason (1980), Evans and Minieka (1992), and Ahuja, Magnanti and Orlin (1993) in particular provide insight into those useful with MCNF algorithms.

In the context of the mathematical formulation, a node-arc incidence matrix seems the most logical. The *node-arc incidence matrix* uses the matrix of divergence constraint coefficients as a representation of the nodes and arcs of the physical network. It is an
$n \times m$ matrix with a row corresponding to each node and a column corresponding to each arc. In each column corresponding to arc $(i, j)$ there are only two nonzero elements, a 1 in the row corresponding to the from node $i$ and a -1 in the row corresponding to the to node $j$. An example of a node-arc incidence matrix is shown in Figure 3.5. This form of network representation is important because it represents the constraint matrix and because it has some special theoretical properties. However, this data structure uses space inefficiently. Only $2m$ out of the $nm$ elements of the matrix are nonzero (Evans and Minieka (1992)).

![Figure 3.5 - A Network and Its Node-Arc Incidence Matrix](image)

If a network is dense, having a high arc to node ratio, then a simple and slightly more efficient representation scheme is the node-node adjacency matrix. The node-node adjacency matrix is an $n \times n$ matrix made up entirely of zeros and ones. Each row corresponds to a node of the network, as does each column. If an arc $(i, j)$ exists in the network, then a 1 is placed in the network in row $i$ and column $j$ signifying that nodes $i$ and $j$ are adjacent and that the arc starts at $i$ and ends at $j$. Only $m$ of the $n^2$ elements of the matrix will be nonzero, thus the matrix will only be space efficient for networks which are dense with arcs (Ahuja, Magnanti, and Orlin (1993)). Schemes that utilize this
structure typically will store the costs and capacities of the arcs in two additional $n \times n$ matrices. This makes searching for the cost or capacity very simple since the values for arc $(i, j)$ will reside in the $ij$th place of the matrix. In addition, we can identify the incoming arcs of a node by examining its corresponding column and the outgoing arcs of a node by examining its corresponding row. Again, this operation will be efficient for dense networks, i.e. $m$ several times as large as $n$, but not for networks with low arc density.

The node-node adjacency network for the above network is shown in Figure 3.6.

![Figure 3.6 - A Network and Its Node-Node Adjacency Matrix](image)

For sparse or low-density networks, an arc list is very space efficient. An *arc list* is simply a listing of the start and end nodes of a network. An example is shown in Figure 3.7. Only $2m$ storage spaces are required to store the network structure. $2m$ additional storage spaces may be used to store the corresponding arc costs and capacities.

![Figure 3.7 - A Network and Its Arc List](image)
By organizing the arc list in a specific fashion and using an additional array of pointers of length $n$, we can obtain a network representation that combines space efficiency with the ability to search the list for specific arcs efficiently (Evans and Minieka (1992)). This is known as a forward star representation. With a *forward star representation* the arc list is first put in a specific order. Namely, we start the list with all arcs emanating from node 1, followed by those emanating from node 2, and so on. The costs and capacities are stored in order with their corresponding arcs. Then an $n$-length array of pointers is created. At each position corresponding to node $i$, a pointer to the first position of the arc list that has an arc emanating from node $i$ is stored. If node $i$ has no outgoing arc, the $i$th pointer is set to the first position with an outgoing arc for node $i+1$. So, all arcs that emanate from node $i$ will be stored in the arc list starting at the position pointed to by pointer $i$ and ending one position before the position in the arc list pointed to by pointer $i+1$. An additional pointer is usually added to the end of the pointer list pointing to arc list position $m+1$ to maintain consistency. With the forward star representation, an efficient way is provided for obtaining all arcs outgoing from a particular node. If it is desired to obtain all arcs incoming to a particular node, ordering the arcs based on the incoming instead of outgoing arc may similarly create a reverse star representation. An example of the forward star is provided in Figure 3.8.

The forward and reverse star representations are among the most efficient network data structures in terms of space and search efficiency. They are, however, not the most efficient if it is important to be able to add or delete arcs frequently. Additions and deletions are accomplished efficiently through the computer data structure called a linked list. An *arc adjacency list* uses a form of linked list to represent a network by storing
together all arcs outgoing from each node. Here, a linked list is defined for each node.

Each list stores the arcs that emanate from the corresponding node. A marker, usually zero, is stored to signal when the end of each list is reached. An adjacency list for the sample network is shown in Figure 3.9. The adjacency list representation is space efficient and efficient for addition or deletion of arcs. Its main limitation is the difficulty involved in searching for a specific arc. Adjacency lists may be combined with star representations.

Figure 3.8 - A Network and a Forward Star Data Structure

Figure 3.9 - A Network and Its Adjacency List

3.2.5 Network Transformations
Transformations on optimization models are often used to place models in a more convenient form. These can be used to simplify the model structure, to illustrate the equivalence of various model types, and to place the model in a standard form related to a particular solution strategy. In this section, several common transformations that apply to network flow models are described. It is transformations such as these that have been used to establish the equivalence of such network problems as the shortest path problem, assignment problem, maximum flow problem and others to the MCNF problem. For this research, these transformations will be used to place network flow problems into the standard form for minimum cost flow problems described earlier. Kennington and Helgason (1980), Bertsekas (1991), Evans and Minieka (1992), and Ahuja, Magnanti, and Orlin (1993) all describe network transformations in detail. Bertsekas (1991) in particular demonstrates how several algorithms were first investigated for assignment problems. Then using the transformations described here extended the results to MCNF.

Recall that for a MCNF model to be feasible, a further constraint of $\sum_{i=1}^{n} b_i = 0$ is usually imposed. In general, this constraint is not prohibitive since a simple transformation can be used if $\sum_{i=1}^{n} b_i \neq 0$. This can be accomplished by adding either a dummy supply or a dummy demand node to the network depending on the sign of $\sum_{i=1}^{n} b_i$. If $\sum_{i=1}^{n} b_i < 0$, then there is more demand than supply. A dummy supply node $d$ can be added with $b_d = \sum_{i=1}^{n} b_i$ along with arcs $(d, i)$ with cost $c_{di} = 0$ added from the dummy node to each demand node. If $\sum_{i=1}^{n} b_i > 0$, then there is more supply than demand. A dummy demand
node $d$ can be added with $b_d = \sum_{i \in I} b_i$ along with arcs $(i, d)$ with cost $c_{id} = 0$ added from the each supply node to the dummy demand node. In this manner, the dummy node will absorb the excess supply or demand (Kennington and Helgason (1980)).

For our purposes, it is assumed that models are formulated without parallel, directed arcs. That is, for each pair of nodes, $i$ and $j$, there exists in the model only one arc $(i, j)$ directed from node $i$ and to node $j$. This assumption is made primarily to allow for the use of less complicated data structures to represent the model. If a network does contain parallel arcs, one may introduce a new node $k$ with supply $b_k = 0$ for each repeated arc. The arc itself will be replaced by two arcs $(i, k)$ and $(k, j)$, with costs $c_{ik} = c_{ij}$ and $c_{kj} = 0$, respectively. The flows of the original and transformed models will correspond as $x_{ij} = x_{ik} = x_{kj}$.

MCNF formulations may be simplified somewhat by removing nonzero lower bounds on arc flows (zero lower bounds is standard for our MCNF formulation) and removing arc capacities Bertsekas ((1991)). Let the flow on arc $(i, j)$ be constrained by upper and lower bounds, $l_{ij} \leq x_{ij} \leq u_{ij}$. The lower bound, $l_{ij}$, may be removed from the model formulation by replacing $x_{ij}$ with $x_{ij}' + l_{ij}$. The constraint on flow bounds then is transformed to $l_{ij} \leq x_{ij}' + l_{ij} \leq u_{ij} = 0 \leq x_{ij}' \leq u_{ij} - l_{ij}$. When this substitution is made in the divergence constraints, it leads to decreasing $b_i$ by $l_{ij}$ and an increasing $b_j$ by $l_{ij}$. The objective function is changed only by a constant value that can be ignored when solving the problem and used to adjust the minimum value after the problem is solved.
At times, it is desirable to find a network transformation with a special structure that may be exploited in designing efficient solution algorithms. The transformation of removing arc capacities is one such case from which we may obtain a bipartite uncapacitated network from a MCNF formulation. For each capacitated arc \((i, j)\), a new node is introduced to the network such that the capacity constraint on \((i, j)\) becomes the divergence constraint on the new node. Let the flow on arc \((i, j)\) be constrained by an upper bound, \(x_{ij} \leq u_{ij}\). This inequality constraint may be converted to an equality constraint by adding a slack variable \(s_{ij}\) to obtain \(x_{ij} + s_{ij} = u_{ij}\). This equality constraint can then be multiplied by -1 to obtain \(-x_{ij} - s_{ij} = -u_{ij}\) and then treated as a divergence constraint for a new node \(k\). To preserve the network formulation of the model, the new divergence constraint is then subtracted from the divergence constraint for node \(j\). This assures that each \(x_{ij}\) and \(s_{ij}\) appears in exactly two divergence constraints, negative in one and positive in the other. The corresponding network transformation involves replacing the original capacitated arc \((i, j)\) into two arcs and an additional node \(k\). The first arc \((i, k)\) will have a cost \(c_{ik} = c_{ij}\) and a flow \(x_{ik} = x_{ij}\) corresponding to the original flow of arc \((i, j)\). The second arc \((k, j)\) will have a cost \(c_{kj} = 0\) and a flow \(x_{kj} = u_{ij} - x_{ij}\). The supplies of the nodes will become \(b_i = b_i\), \(b_k = -u_{ij}\), and \(b_j = b_j + u_{ij}\), respectively. Since the costs of the slack variables are 0, there will be no change in the objective function for the transformed problem.

3.3 Algorithmic Strategies

Most solution procedures applied to the minimum cost network flow problem originate, at least in part, from three basic algorithmic ideas: primal cost improvement,
dual cost improvement, and auction. In this section, these algorithmic strategies and related theory are briefly described. Ideas from each of these will be used as bases for methods used in the research to be described.

### 3.3.1 Primal Cost Algorithms

With *primal cost improvement* strategies, algorithms begin with an initial feasible flow vector and then iteratively improve the objective function to its optimal value by constructing a sequence of feasible flows. This strategy derives its name from the fact that it works directly with the original problem formulation, often referred to as the *primal* problem. A related problem, called the *dual*, will be discussed in section 3.3.2.

A fact that is useful when generating the sequence of feasible flows is that the difference between any two successive flow vectors must be a circulation. In many cases, this circulation includes only a simple cycle. It can be shown that if a current flow vector is not optimal, then an improved flow vector may be obtained by pushing flow along a simple cycle with negative costs (Ahuja, Magnanti, and Orlin (1993)). The most successful algorithms of this type include version of the simplex method of linear programming that have been specialized for network flow problems.

### 3.3.2 Dual Cost Algorithms

The *dual cost improvement* strategy works mainly with a problem formulation called the *dual*, which is uniquely related to the primal formulation. Here, the variables, called *prices*, are sequenced through feasible values until the optimal dual cost is obtained. These prices often correspond to the node potentials calculated as intermediate values for the primal problem and used to calculate reduced costs.
Flows in the network are related to the dual prices through the *complementary slackness conditions* (Ahuja, Magnanti, and Orlin (1993)). The complementary slackness optimality condition states that a feasible flow vector \( x^* \) is an optimal solution of the primal minimum cost flow problem if and only if for some set of dual prices \( \pi \) (corresponding to primal node potentials), the reduced costs and flow values satisfy the following complementary slackness conditions for all arcs \((i, j)\) in the network:

\[
\begin{align*}
\text{If } c_{ij}^\pi > 0, \text{ then } x_{ij}^* &= 0. \\
\text{If } 0 < x_{ij}^* < u_{ij} \text{ then } c_{ij}^\pi &= 0. \\
\text{If } c_{ij}^\pi < 0, \text{ then } x_{ij}^* &= u_{ij}.
\end{align*}
\]

At each iteration, the major dual ascent algorithms choose a connected subset \( S \) of nodes and adjust the prices of these nodes by an equal amount while leaving the prices of all other nodes unchanged. Various techniques are used to choose the node set \( S \) so that improvements to the dual cost are made. Two important dual algorithms are the *primal-dual algorithm* and the *relaxation* algorithm (Bertsekas (1991)). The *primal-dual algorithm* chooses the node set \( S \) that will lead to maximum feasible improvement at each iteration. The *relaxation* algorithm chooses nodes sets that are usually easier to compute but do not necessarily lead to maximum feasible improvement at each iteration.

### 3.3.3 Auction Algorithms

A third strategy, known as *auction*, is quite different from the other two. Auction algorithms work by generating a sequence of prices in a manner that is similar to real-life auctions (Bertsekas (1991)). Any single iteration may not result in either primal or dual improvement, but in the end a primal optimal solution will be reached.
Auction algorithms use an \( \varepsilon \)-complementary slackness property to iteratively improve a dual solution that is only approximately optimal. For a feasible flow vector \( x^* \), a price vector \( p \), and a fixed positive scalar \( \varepsilon \), the \( \varepsilon \)-complementary slackness states that \( x^* \) satisfies \( \varepsilon \)-complementary slackness if and only if:

\[
\begin{align*}
p_i - p_j & \leq c_{ij} + \varepsilon \quad \forall (i, j) \in A \text{ with } x_{ij} < u_{ij}, \\
p_i - p_j & \geq c_{ij} + \varepsilon \quad \forall (i, j) \in A \text{ with } 0 < x_{ij}.
\end{align*}
\]

The optimality of a final solution of an auction algorithm depends on the choice of parameter \( \varepsilon \). It can be shown that if all problem data are integer, \( \varepsilon \) is chosen so that \( \varepsilon < 1/N \), where \( N \) equals the number of nodes in the network, and if a flow vector \( x \) is feasible and satisfies \( \varepsilon \)-complementary slackness with the price vector \( p \) then \( x \) is optimal. There is, however, a trade-off between the size of \( \varepsilon \) and the speed at which an optimal solution is obtained.

3.4 The MCNF Literature

The amount of available material and publications related to network flow models and their applications is vast. For this research, several of the more recent comprehensive texts were used to gain an overall appreciation of the field. In addition, several journal publications and software packages were reviewed for further background into specific areas of interest. These items are discussed here.

Considered a classic reference in the area of network flow models, Ford and Fulkerson (1962) provided one of the first broad overviews of network optimization models. Kennington and Helgason (1980) provided another major update covering many aspects of mathematical programs on networks. Their text set a standard for algorithmic
explication of network algorithms while incorporating many practical network applications. They also contributed a strategy for taking advantage of imbedded network structure to improve the solution efficiency for network problems with side constraints.

Bertsekas (1991) provides a thorough review of the algorithmic developments of the 1980s. In particular, this text focuses much attention on two approaches that are significantly different than earlier methods - the relaxation and auction methods. Both of these methods are challengers to earlier primal and dual cost methods in terms of practical efficiency and theoretical worst-case performance.

Both Ruhe (1991) and Evans and Minieka (1992) provide approachable explanations of various algorithmic aspects of flows in networks. The Evans and Minieka text, along with algorithms related to the study of networks, includes many practical considerations such as appropriate data structures and aspects related to computer implementation. The text is accompanied by NETSOLVE, a solver for single objective network flow problems. Ruhe also furnishes the reader with much practical information. A strong point is the comparison of many algorithms based on theoretical and empirical performance. Of particular interest to this research is a chapter concerning multicriteria flows.

A comprehensive text is presented by Ahuja, Magnanti and Orlin (1993). This text provides an integrative view that follows the history of theory, algorithms and applications related to network flows. The subject is presented in an approachable manner such that the reader can easily see the similarities and differences between the major solution approaches.
Several journal articles were also helpful in support of the research presented here. Two publications (Klingman, et al. (1974) and Arthur and Frendeway (1994)) discuss the random generation of single-objective network flow problems. The algorithms presented in these papers were used as a basis for creating a multiple-objective problem generator.

3.5 Conclusion

In this chapter, a brief overview of the fundamentals of network optimization, the basic concepts of minimum cost network flow models, and main algorithmic strategies for network models have been provided. This discussion along with the previous overview of MCDM theory provides a basis from which to launch a study into MONF problems. From here, the special characteristics of MONF problems may be investigated. We may then design specialized solution algorithms for MONF problem by combining the major algorithmic strategies of both MCDM and network flows.
CHAPTER 4
A SEARCH OF THE MONF LITERATURE

4.1 Introduction

In this chapter, a discussion of the literature relevant to multiple objective network flow problems is presented. The discussion provides a comprehensive listing of the MONF literature to the present. The survey provided here is an expansion of earlier searches presented by Current and Min (1986), Current and Marsh (1993), Ulungu and Teghem (1991, 1994) and ReVelle, Cohon, and Shobrys (1981). The major objective in presenting a comprehensive listing of the MONF literature is to identify and provide evidence of gaps in the research to date. Those references that are directly relevant to the research of this dissertation are identified and discussed in more detail. In the next section, a comprehensive listing is provided of published works related to MONF. The chapter will then conclude by listing some gaps in MONF theory that are targeted with this research.

4.2 Application of Multiple Criteria Decision Aid to Network Flow Modeling:

Despite the large amount of research in both the fields of multiple objective linear programming and network optimization, there are relatively few works combining these two areas. This is surprising since the addition of multiple criteria to a single objective network optimization problem seems a natural extension towards improving how close a
model mirrors reality for many applications. A few authors have presented surveys
enumerating research related to MONF. Current and Min (1986) presented a taxonomy
and annotated bibliography of transportation network design and routing problems, which
was updated by Current and Marsh (1993). At the first level, the taxonomy categorizes
each research example according to its application area. These Application areas include:
Shortest Path Problem, Transportation Problem, Assignment Problem, Transshipment
Problem, Vehicle Routing Problem, Optimal Network Design Problem, and Generalized
Network Problem.

The taxonomy further divides these categories according to whether solution
methods used are exact or whether they incorporate heuristics. An exact method uses
algorithms which are guaranteed to find a final solution and do not stop until the final
solution is found. Heuristic searches on the other hand are procedures that are repeated
only until a satisfactory solution is found. They may also be based on guidelines that are
thought to be true in general but may not be guaranteed to always be true.

In the third level of the taxonomy, the research is classified based on the multiple
objective method used: those that utilize some type of generating technique versus
preference based techniques. Here, generating techniques are those which seek to
generate either all or a portion of the efficient set and then search through the points
generated to find a solution. Preference based techniques attempt to use information
solicited from the DM to model the DM's utility function. Then a search is carried out to
find the solution of greatest utility. The resulting taxonomy and bibliography provide a
resource with which to begin a search of the MONF literature.
Ulungu and Teghem (1991, 1994) provide literature surveys that describe related research. Ulungu and Teghem (1991) lists research into the multiple objective shortest path problem. Ulungu and Teghem (1994), provides a much broader list of multiple objective combinatorial optimization problems, of which network applications are a subset. ReVelle, Cohon, and Shobrys (1981) provide a review of Multiple Objective Facility Location applications, many of which can also be modeled as networks.

For this literature survey, we are concerned with the minimum cost network flow problem and its related applications. The taxonomies of Current and Min (1986) and Current and Marsh (1993) will be followed with minor alterations. The first level of the taxonomy is shown in Figure 4.1. Following Current and Min, this level is divided into categories based on specific applications. The categories include only those applications that can be modeled as minimum cost network flows. Thus the categories of optimal network design problem, and generalized network problem have been removed and replaced with the Minimum Cost Network Flow category. In addition, a category for Location Problems and one for Scheduling has been included. For these last two categories, only research that uses network modeling has been included.

Figure 4.1. First level of The Taxonomy
The second and third levels of the Current and Min taxonomy are used without alteration. An example of the Minimum Cost Network Flow category appears in Figure 4.2.

Figure 4.2. An Example of Taxonomy levels two and three.

The literature survey is organized based on the respective node in the taxonomy in the fashion of Current and Min. Categories for which no papers were found are not listed. The works more relevant to the dissertation research are described. As is the case with most categories described here, the number of multiple objectives that are considered in many papers can be further classified as two, three or a general number with two objective problems the most frequent. In this way, 233 articles related to multiple objective network flow problems are described.

1. A. 1. Shortest Path problem, exact algorithm, generating technique

Murthy and Olson (1994) apply exact, generating techniques to the shortest path problem with two objectives. Problems that incorporate three objectives are studied by Martins (1984b), and Current, ReVelle, and Cohon (1985a, and 1985b).


I. A. 2. Shortest Path problem, exact algorithm, preference based technique


I. B. 1. Shortest Path problem, heuristic algorithm, generating technique

As well as providing several exact algorithms for specific two objective shortest path problems, Hansen (1980) describes a heuristic based method for another. Warburton (1987) introduces the heuristic of $\varepsilon$-efficiency for approximating the efficient set of the multiple objective shortest path problem for a non-specific number of objectives.

I. B. 2. Shortest Path problem, heuristic algorithm, preference based technique

Bronzini and Sherman (1983) present a weighted MOLP approach for a two objective shortest path problem. Bard and Bennett (1991) present a heuristic for determining the path that maximizes the expected utility of a stochastic network.

II. A. 1. Transportation problem, exact algorithm, generating technique

Algorithms and research for the transportation problem with general number of objectives are provided by Lawrence and Burbridge (1976), Diaz (1978 and 1979), Isermann (1979), and Sutcliffe, Board, Cheshire (1984), and Shi (1995). Current (1993) provides an overview of prior research into transportation networks that highlights the need for multiple objective formulations.

II. A. 2. Transportation problem, exact algorithm, preference based technique


II. B. 1. Transportation problem, heuristic algorithm, generating technique


II. B. 2. Transportation problem, heuristic algorithm, preference based technique

Ringuest and Rinks (1987) suggest an interactive, weighted MOLP approach to the multiple objective transportation problem with a general number of objectives. Li (1988) presents an expert system for the evaluation of multiple objective transportation networks. Bit, Biswal, and Alam (1992 and 1993a,b,c) apply fuzzy programming to several multiple objective transportation problems. Chalam (1994) apply a fuzzy goal programming approach.

III. A. 1. Assignment problem, exact algorithm, generating technique

Malhotra, Bhatia and Puri (1982) investigate the bicriteria assignment problem. In White (1984), a weighted MOLP approach is used in solving an assignment problem with three objectives. Other authors who present research on the multiple objective assignment problem include Lee and Clayton (1972), Lawrence, Reeves, and Lawrence (1984), McClure and Wells (1987), and Liang and Thompson (1987).

III. A. 2. Assignment problem, exact algorithm, preference based technique


III. B. 1. Assignment problem, heuristic algorithm, generating technique


III. B. 2. Assignment problem, heuristic algorithm, preference based technique

Ignizio, Palmer, and Murphy (1982) apply a non-linear goal programming approach towards the design of a distributed computer architecture system. Murphy and Ignizio (1984) and Malakooti and D'Souza (1987) solve multiple objective quadratic assignment problems for distributed computing and facility layout problems, respectively.

IV. A. 1. Transshipment problem, exact algorithm, generating technique

Arthur and Lawrence (1982) present a goal programming model for a general, multiple objective logistics planning problem. A constrained MOLP approach is used by
Osleeb and Ratick (1983) and Ratick (1983) for two coal handling applications. Hemaida and Kwak (1994) apply goal programming to a bicriteria model.

**IV. A. 2. Transshipment problem, exact algorithm, preference based technique**

Moore, Taylor and Lee (1978) and Ogryczak, Studzinski, and Zorychta (1989), apply goal programming formulations to the multiple objective transshipment problem.

**V. A. 1. Vehicle Routing problem, exact algorithm, generating technique**


**V. B. 1. Vehicle Routing problem, heuristic algorithm, generating technique**

V. B. 2. Vehicle Routing problem, heuristic algorithm, preference based technique


VI. A. 1. Minimum Cost Flow problem, exact algorithm, generating technique

More recently Ehrgott (1990) and Sun (2000) have investigated various aspects of MONF. These include solution methods for MONF, advanced warm start routines for sub-problem testing, and strategies for generating integer solutions.

VI. A. 2. Minimum Cost Flow problem, exact algorithm, preference based technique


VI. B. 1. Minimum Cost Flow problem, heuristic algorithm, generating technique


VI. B. 2. Minimum Cost Flow problem, heuristic algorithm, preference based technique

VII. A. 1. Location problem, exact algorithm, generating technique


VII. A. 2. Location problem, exact algorithm, preference based technique


VII. B. 1. Location problem, heuristic algorithm, generating technique


VII. B. 2. Location problem, heuristic algorithm, preference based technique

Rao and Saraswati (1988) present a fuzzy set theoretic approach for a multiple objective facility location problem on a network. In Chaudry and Kincaid (1990), the Rao and Saraswati paper is discussed, pointing out several omissions and mistakes.

VIII. A. 1. Scheduling problem, exact algorithm, generating technique


VIII. A. 2. Scheduling problem, exact algorithm, preference based technique


VIII. B. 1. Scheduling problem, heuristic algorithm, generating technique

VIII. B. 2. Scheduling problem, heuristic algorithm, preference based technique

Khattab and Choobineh (1992) look at a multiattribute heuristic for single resource project scheduling.

4.3 Conclusion

While an impressive list of research has been conducted and presented in the area of multiple objective network flow problems, several points can be considered which lead to opportunities for research:

• Most of the existing research in this area has targeted the specific cases of two and three objective network models.

• Most of the research involving a general number of objectives involves the application of goal programming methods.

• Only a few papers are presented which deal with the more sophisticated interactive techniques of multiple objective programming.

Nevertheless, the works cited, especially the few works regarding models that have a general number of objectives, provide a sound basis for further exploration into the application of more sophisticated MOLP methods to the MONF problem.
CHAPTER 5

SOLUTION CHARACTERISTICS FOR MULTIPLE OBJECTIVE NETWORK FLOW PROBLEMS

5.1 Introduction

A characterization of the efficient and nondominated sets is important in the background for an understanding of decision models such as the MONF problem. This is necessary because one must recognize a potential solution to a problem when it is encountered. Also, in the case of multiple alternatives one must be able to evaluate and distinguish among them.

An understanding of solution characteristics can be important for other reasons as well. Important insights may be gained into how to move from initial or intermediate states to final solution states. Clues leading toward the development of solution algorithms with improved computational efficiency may also be obtained.

The simplex algorithm for LP provides a significant and germane example of the importance of characterizing possible solutions to a mathematical program. By understanding that solutions to LP can be completely described by extreme point solutions (either by a single extreme point or a convex combination of extreme points), a robust algorithm that searches through extreme point solutions only was derived.

In this chapter, the MONF problem is described along with a study of solution characteristics. A distinction is made between networks that allow non-integer or
continuous flows and those with only integer valued flows. The different possible solutions that may be obtained for each problem class are discussed and characterized.

5.2 The Multiple Objective Network Flow Problem

Multiple objective network flow (MONF) problems can be viewed as a sub-class of MOLP. As such, they may be modeled as:

\[
\begin{align*}
\text{min:} \quad & c^1 x = z_1 \\
\vdots \quad & \\
\text{min:} \quad & c^k x = z_k \\
\text{s.t.:} \quad & x \in S
\end{align*}
\]

The feasible region, \( S \), is defined to correspond to the allowed flows along arcs within a network. The vector \( x \in \mathbb{R}^n \) is a vector of decision variables. Each component of \( x \) represents a flow over an arc of the network. In general, \( S \) will be defined in the mathematical model using a set of divergence constraints, one for each node in the corresponding network, and a set of capacity constraints, one for each arc of the network as with the MCNF. Thus, \( S \) can be modeled as:

\[
S = \left\{ x \in \mathbb{R}^n \left| \sum_{(j,i,j) \in A} x_{ij} - \sum_{(j,j,i) \in A} x_{ji} = b_i ; 0 \leq x_{ij} \leq u_{ij} ; \forall (i,j) \in A \right. \right\}.
\]

In this formulation, the components of vector \( x \) are described as \( x_{ij} \), representing the flow on the arc from node \( i \) to node \( j \) of the network; the \( b_i \) are values representing the supply (positive valued) or demand (negative valued) at each node \( i \); and the \( u_{ij} \) represent the capacity of the arcs \( (i, j) \) in the network. Variations in network formulation exist such as un-capacitated networks (each \( u_{ij} = \infty \)), lower bounds other than zero or restricting arc flows to integer values.
The basic theory of MOLP described in Chapter 2 holds for the MONF class of problems. Thus, for MONF there is usually no single solution that simultaneously optimizes all of the objectives. Instead, there is a potentially large set of possible solutions in criterion space called the nondominated set. It is from within this set that the sophisticated methods of MOLP search to find the DMs’ most preferred solution. It will be of interest to describe the characteristics of this set for MONF.

Many characteristics of MONF are inherited from the network theory described in Chapter 3. The structure of the feasible region is defined by the structure of the network. For single objective MCNF problems, this network structure has been utilized to great advantage. The special topological structure of networks has allowed for solution procedures that are more than 100 times faster than traditional linear programming methods (Kennington and Helgason (1980)). These speeds are possible in the single objective case largely due to two factors. First, movements within the feasible region correspond to movements on the network graph. This allows for data structures that can be used very proficiently. Second, if all problem parameters, the coefficients and RHS values, are integer valued then the optimal solution in the single objective case is known to also be integer valued. This can improve the computational speed and also the numerical consistency of the calculations.

The primary interest in MONF research is whether or not the network structure may be exploited to improve computation speed when applying multiple objective methods. While the advantages of the structure of network flow problems have been extensively explored in the single objective case, there seems to be a lack of progress for
the multiple objective case. This stems from the much greater additional complexity introduced with the addition of multiple objectives.

5.3 Continuous and Integer Variables

For many practical applications of network flow models, the solutions sought are understandably integer valued. Recall from Chapter 3 that for single objective MCNF models the optimal solution will be integer valued as long as all coefficient and RHS values are integer. For this reason, the usual single objective MCNF formulation does not restrict the variables to integer values. In contrast, there is no corresponding guarantee for the MONF problem that a particular nondominated solution will be integer. When the MONF variables are allowed to vary continuously a large portion of the nondominated set may be non-integer. Therefore, when only integer solution values are desired we will need to add the further restriction that all decision variables are integer. This moves the MONF formulation from the realm of a LP with multiple objectives into the realm of integer programming (IP) with multiple objectives. Contrary to intuition, the combinatorial nature of IP makes the search for the DM's most preferred solution more difficult.

For an IP we can generate an LP relaxation by taking the same objective function (or functions) and same structural constraints. We then relax the requirement that variables are integer and replace it with the appropriate continuous constraints. For MONF, this means we simply allow the decision variables to vary continuously from 0 to their respective capacity values.
Example 5.1 - Consider the biobjective MONF in Figure 5.1 with decision variables restricted to integer values:

\[ [u_{ij}, c^1_{ij}, c^2_{ij}] \]

Figure 5.1 - A Biobjective, Integer Flow Network

Figure 5.2 - Feasible Region in Criterion Space and LP Relaxation

The discrete points shown in Figure 5.2 represent the feasible region in criterion space. The dashed lines represent the feasible region of the LP relaxation. Notice that the dashed lines form a convex hull around the feasible region of the original integer valued
feasible region. All points that lie on the boundary of the convex hull from $z^1$ to $z^2$ to $z^3$ are nondominated.

Figure 5.2 illustrates some of the difficulties in using an LP relaxation as an aid in modeling a problem with integer valued variables. When using the relaxed feasible region, The DM may choose points that are not integer valued. While rounding the DM’s selected point to the nearest integer may seem like a reasonable step, there is no guarantee that the nearest integer solution is the best integer solution. Indeed, there is no guarantee that the nearest integer solution is even feasible.

In the next couple of sections, observations for the MONF with both continuous and integer valued decision variables are discussed.

5.4 Observations for the Continuous Case

For the MONF that is an MOLP, the decision variables are allowed to vary within linear constraints forming a feasible region in decision space that is a bounded, convex set. When the objectives are also linear, the convexity of the feasible region holds for criterion space. For these problems, the nondominated set resides on the boundary of the feasible region in criterion space. The nondominated set may consist of extreme and non-extreme points. In the case with continuous variables, all nondominated points can be shown to be supported by hyperplanes derived from the objectives. Points in the nondominated set may then be classified as either extreme points or supported non-extreme points. For MONF, the decision variables represent flows on arcs. Observations related to the network, the efficient set, and the nondominated set for the MONF with continuous arc
flows are described here. These observations are made under the assumption that the null vector condition as described in Steuer (1989) does not hold.

**Observation 1:** The feasible region $S$ for the MONF problem with continuous variables is a closed, bounded, convex set.

As discussed in Chapter 2, the linear constraints of MONF form a feasible region that is a closed, convex polytope. In addition, regardless of whether or not capacity bounds are present on the arcs of the network, the finite supplies and demands of the network bound above the values of all flows. Thus, for MONF, the feasible region is bounded. Here, the feasible region is formed by the intersection of a finite number of hyperplanes from the divergence (equality) constraints of each network node and closed half-spaces from the lower and upper capacity (inequality) constraints on the arcs. Therefore, $S$ is closed, bounded, and convex.

The feasible region $S$ includes a finite number of extreme points. Extreme points by definition reside on the boundary of the $S$. Since the set is convex, for all $x \in S$ there exist points $x^1, x^2 \in S$ such that $x = (1 - \lambda)x^1 + \lambda x^2$ for $\lambda \in [0,1]$. With this it may be shown that every point in $S$ may be written as a convex combination of extreme points. For single objective MCNF it is known that if an optimal solution exists, then there exists an optimal extreme point.

**Observation 2:** Extreme points in decision space correspond to spanning trees on a network.

Kennington and Helgason (1980) provide foundation and proof.
**Observation 3:** If all network parameters are integer, then extreme point solutions of networks are integer valued.

These observations are well known from network theory as described in Chapter 3. Since extreme points of decision space correspond to basic solutions of the LP formulation for the network and basic solutions of the LP formulation correspond to spanning trees of the network it follows that the extreme points in decision space correspond to spanning trees of the network. This correspondence between basic solutions of the LP and spanning trees allows us to search for extreme point solutions by conducting a search of spanning trees on the network. It has been used as the basis for many single objective MCNF solution procedures. It is also this correspondence that allows for the use of special network related data structures resulting in high-speed algorithms. Both observations 2 and 3 have been stated as theories and proofs may be found in such texts as Kennington and Helgason (1980) and Ahuja, Magnanti and Orlin (1993).

**Observation 4:** Vector-Maximum theory of MOLP holds for the set of efficient solutions for the MONF with continuous variables.

As discussed in Steuer (1989) and in Chapter 2 a vector maximum problem is one in which the overall goal is to maximize a vector. In the case of MONF, instead of maximizing a vector we wish to minimize a vector of criterion values. This does not pose a problem since maximization objectives may be easily transformed to equivalent minimization objectives. Also, the feasible region of the MONF with continuous variables is defined by linear constraints, as is the feasible region of the MOLP. The MONF with
continuous variables represents a class of problems that are basically a special subset of MOLP. Thus, theory that holds for MOLP will hold for the MONF with continuous variables. Vector maximum theory in particular provides several implications that will be important to this work.

For MOLP, including MONF, the set of interest in decision space is the *efficient set*. If an efficient solution exists then at least one extreme point is efficient. Efficient extreme points correspond to *efficient bases* of the LP formulation of $S$. By finding efficient bases, we also find efficient extreme points.

Two efficient extreme points are said to be *adjacent* if one can be reached from the other by moving along an edge of the efficient set. This corresponds to removing a basic variable in the current basis and replacing it with an appropriate nonbasic variable. Given an efficient extreme point, one can determine whether or not there are adjacent efficient extreme points and the appropriate pivots to make to reach the adjacent efficient points. These are known as *efficient pivots*.

**Observation 5:** *For the MONF with continuous variables, there are no unbounded edges so efficient extreme points and selected convex combinations of efficient extreme points can be used to characterize the efficient set.*

As discussed above, the feasible region for MONF is closed and bounded. While unbounded edges may be recognized using vector maximum theory, their absence provides one less item of concerned. Thus for the MONF with continuous variables, efficient extreme points and selected convex combinations of efficient extreme points can fully characterize the efficient set.
**Observation 6:** For the MONF with continuous variables, efficient points may be extreme, lie in the interior of an edge, or lie on the interior of an efficient f-facet of the efficient set.

This follows from the possible efficient solutions of MOLP less the possibility of unbounded edges. It is worth stating to provide a clear picture of the solution set through which we will be searching.

**Observation 7:** For the MONF with continuous variables, efficient points on the interior of a facet of the efficient set which are not extreme (facet dimension greater than 0) may not be integer valued.

Network theory only promises that basic solutions of the LP formulation will be integer valued if all problem parameters, objective and constraint coefficients as well as RHS values, are integer. The one-to-one correspondence between basic solutions and spanning trees of the network extends this guarantee to spanning trees of the network. For the MONF problem with continuous variables, the efficient set will usually include solutions which are not extreme and do not correspond to a spanning tree. For these solutions, there is no guarantee that they will be integer valued.

**Example 5.2** - Consider the following MONF formulation for the network of Figure 5.3. Figure 5.4 graphs the feasible region $S$ in decision space and illustrates several of the observations above. The MONF formulation for this example is
The network of Figure 5.3 was chosen for illustration because of its small size. For most network algorithms, parallel arcs would not be allowed, and we have seen in chapter 3 how they may be eliminated through a simple transformation. Also, while larger networks are more likely to occur in practice, it is quite difficult to visualize vector spaces higher than three dimensions. In Figure 5.4, the feasible region for the network is mapped in decision space. The criterion vectors are shown as dark arrows emanating from the origin.

For this network, we observe that the feasible region is a closed, convex subset of a plane in decision space with six extreme points. It is evident in the figure that any point in the feasible region may be written as a convex combination of two or more of the extreme points. With the given criterion vectors, the entire feasible region is efficient.
Three of the solutions have been highlighted. Solution $\mathbf{x}^1 = (3, 5, 0)$ is an efficient extreme point. Note that this solution has all integer values and corresponds to a spanning tree. For this spanning tree, the $x_1$ arc is basic, the $x_2$ arc is non-basic at its upper bound, and the $x_3$ arc is non-basic at its lower bound of zero. Solution $\mathbf{x}^2 = (0, 4, 4)$ represents an efficient solution that resides on an edge of the feasible region. Notice that it does not correspond to a spanning tree since both arcs $x_2$ and $x_3$ are between their bounds. Solution $\mathbf{x}^2$ also illustrates the notion that points other than extreme points may be integer valued. Solution $\mathbf{x}^3 = (4.8, 1.5, 2.2)$ represents a solution that resides on a maximal facet of the efficient set. Note that with non-integer flows allowed, this point is both feasible and efficient.

![Figure 5.4 - Feasible Region for Network of Figure 5.3](image)

$\mathbf{c}^1 = (1, 0, 1)$  
$\mathbf{c}^2 = (0, 1, 0)$  
$\mathbf{x}^1 = (3, 5, 0)$  
$\mathbf{x}^2 = (0, 4, 4)$  
$\mathbf{x}^3 = (4.8, 1.5, 2.2)$
**Observation 8:** Movement along an edge of the efficient set from one efficient extreme point to another (an efficient pivot), corresponds to movement in the network from one spanning tree $T_1$ to another spanning tree $T_2$. This movement can be characterized by a simple cycle, which is fundamental with respect to both $T_1$ and $T_2$.

Recall from the network theory described in Chapter 3 that movement between adjacent spanning trees occurs when one arc in a tree $T_1$ is removed and replaced by another arc to form an adjacent tree $T_2$. The addition of the new arc to $T_1$ forms a simple cycle, which is fundamental to $T_1$. Alternatively, the addition of the removed arc to $T_2$ forms a cycle that is fundamental to $T_2$. The cycle in both cases is the same. Since spanning trees in the network correspond to extreme points of the efficient set, movement along an edge of the efficient set between two adjacent efficient extreme points corresponds to movement between two spanning trees. Thus an edge of the efficient set can be related to a fundamental cycle with respect to both $T_1$ and $T_2$.

**Observation 9:** For the MONF with continuous variables, efficient points on the relative interior of an edge of the efficient set, which might not be integer valued, correspond to flows which occur in the network while moving from one spanning tree to another using a fundamental cycle with respect to the adjacent spanning trees.

This observation follows directly from observation 8 and the fact that points on an edge of the efficient set represent convex combinations of the extreme points of the edge. The implication is that we may reach efficient points that lie on the relative interior of an edge of the efficient set by adjusting the flows on a fundamental cycle with respect to two efficient spanning trees. At this point, efficient solutions which are extreme and those
which lie on edges of the efficient set have been related to the network. An illustration of
the movement along an edge of the efficient set of the previous network from extreme
point \( x^1 \) to an adjacent extreme point \( x^3 \) is provided in figures 5.5 and 5.6.

In Figure 5.5 we have the feasible region of the network of Figure 5.3 with three
points highlighted along an edge of the feasible region. Solution \( x^1 = (0, 5, 3) \) and \( x^3 = (0, 3, 5) \) are adjacent extreme points of the efficient set and \( x^2 = (0, 3.6, 4.4) \) is a point which
lies along the edge between them. Figure 5.6 shows a representation of the flows on the
network corresponding to each of these points. Here, arcs with flows at the upper or
lower bound are represented as dashed lines. When the current solution is basic these arcs
are considered nonbasic. Solid lines represent arcs with flows that are between their
respective bounds. Note that only one arc is required to form a spanning tree for the
current network.

The solution \( x^1 \) is basic with the arc corresponding to \( x_3 \) forming the spanning tree
\( T_1 \). The solution \( x^3 \) is basic with the arc corresponding to \( x_2 \) forming the spanning tree \( T_3 \).
The cycle formed by the \( x_2 \) and \( x_3 \) arcs is the unique cycle that is fundamental to both \( T_1 \)
and \( T_3 \). A total change in flow of \( \Delta = 2 \) units is required to move from solution \( x^1 \) to
solution \( x^3 \). The counterclockwise curved arrow within the fundamental cycle represents
this flow change. Solution \( x^2 \) corresponds to a change of flow to the fundamental cycle of
1.4 units.
Observation 10: Assuming nondegeneracy, the set of efficient bases is connected.

This observation has been stated as theory and proofs are provided by Zeleny (1982) and Steuer (1989) among others. This provides a starting point for developing
algorithms that will find all efficient extreme points for MOLP and by extension MONF.

With this result, beginning with an arbitrary efficient extreme basis one could list all adjacent feasible bases and then pivot to each in turn. At each efficient basis, new adjacent feasible bases may be determined and added to the list. Once all bases in the list have been tested for adjacent efficient bases and no new ones are found, then the set of efficient bases is complete.

Each spanning tree in the network of a MONF corresponds to a basis and defines a unique flow \( x \). If the flow satisfies the flow bounds \( 0 \leq x_{ij} \leq u_{ij} \) for every arc \((i, j)\) of the network, then the spanning tree represents a feasible basis, otherwise it is infeasible. A spanning tree \( T \) is degenerate if \( x_{ij} = 0 \) or \( x_{ij} = u_{ij} \) for some arc \((i, j)\) in \( T \), and nondegenerate otherwise. In a nondgenerate spanning tree, \( 0 < x_{ij} < u_{ij} \) for every arc \((i, j)\) in \( T \). Assuming nondegeneracy, there is a one-to-one correspondence between the set of efficient bases and the set of efficient extreme points.

**Observation 11:** When degenerate efficient bases exist, degenerate efficient pivots may be required before moving to adjacent efficient extreme points.

Degeneracy is an important theoretical and practical consideration when dealing with network flow models. Computational studies have been reported that show that as many as 90% of the pivot operations in commonplace networks can be degenerate (Ahuja, Magnanti, and Orlin (1993)). With such a high rate of potential degeneracy, the handling of degenerate pivots must be considered.
Consider the three objective network of Figure 5.7 with a capacity of 2 units on for each arc. The spanning trees in Figure 5.8 represent three efficient bases for this network. Labels are provided to represent the amount and direction of the flow on the arc. The arc between nodes 3 and 5 in Tree 1 is drawn as a dashed line because this arc is nonbasic at capacity.
Figure 5.8 illustrates that when degeneracy occurs it is possible that not all bases of adjacent efficient extreme points are adjacent efficient bases. Trees 1 and 2 are bases for the same efficient extreme point. Since no change in flow is required when pivoting between Tree 1 and Tree 2 these bases are degenerate. Tree 3 represents an efficient basis for a new extreme point. Since the basis of Tree 3 may be reached from Tree 2 in one pivot (arc (3, 5) goes out while arc (4, 5) enters the basis with a flow change of 2 units) these two bases are adjacent and thus the two extreme points are also adjacent. However, since two pivots are required to move from the basis of Tree 1 to the basis of Tree 3, these two bases are not adjacent.

This observation has implications for any attempt to list the entire sets of efficient bases and efficient extreme points. The primary conclusion is that unless all known adjacent efficient bases have been tested, even those that result from degenerate pivots, it is not clear that all efficient bases will be found.

Observation 12: Higher order efficient f-facets (dimension greater than 1) may be characterized by convex combinations of efficient extreme points.

An individual f-facet of the efficient set is by definition a closed, convex set. Thus when a facet includes extreme points (always the case with linear constraints) then any point within the facet may be expressed as some convex combination of the facets extreme points.

Note that this does not imply that all facets that can be expressed by convex combinations of efficient extreme points are efficient. The feasible region in decision space of a minimization MOLP shown in Figure 5.9 illustrates this point. The points $x^1$, $x^2$, ...
$x^2$, and $x^3$ are efficient extreme points as illustrated by their domination sets. However, not all of the facets generated by these extreme points are efficient facets. The edge between $x^1$ and $x^2$ and the edge between $x^2$, and $x^3$ are efficient 1-facets. However, the 1-facet represented by the edge between $x^1$ and $x^3$ contains only two efficient points, namely the extreme points. Similarly, the 2-facet made up of the entire feasible region contains many points which are inefficient.

\[
x^1 = (12, 2) \\
x^2 = (7, 4) \\
x^3 = (5, 10) \\
e^1 = (1, 5) \\
e^2 = (2, -3)
\]

Figure 5.9 - Efficient and Non-Efficient Facets

**Observation 13:** Extreme points and edges of $S$ in decision space do not necessarily map to extreme points and edges of $Z$ in criterion space.
Assuming that the number of objectives is less than the number of decision variables, facets of the efficient set in decision space may correspond to facets of the nondominated set in criterion space of equal dimension or less. This could occur due to a collapsing effect that takes place when mapping an object of higher dimension to a space of lower dimension. This effect was described by Dauer (1987). This implies that points which are extreme points of the efficient set in decision space may be extreme points of the nondominated set, lie along edges of the nondominated set, or occur in the interior of a facet of the nondominated set when mapped into objective space. Dauer and Gallagher (1996) demonstrated that there is a one-to-one correspondence between maximally efficient facets in decision space and maximally nondominated facets in criterion space.

Figure 5.10 - The Feasible Region in Criterion Space
This is seen if we map the feasible region of Example 5.2 into criterion space as shown in Figure 5.10. The result of this mapping is that the two objective functions collapse the planar feasible region of decision space to a line segment in 2-space. The endpoints of the line segment, $z^1$ and $z^3$, represent the objective values that correspond to the highest and lowest edges of the efficient set in decision space respectively. The criterion vector $z^2$ corresponds to the objective values of the remaining efficient extreme points and their feasible convex combinations. Note that these two efficient extreme points do not map to nondominated extreme points although their criterion values remain integer.

**Observation 14:** The size of the efficient and nondominated sets tends to grow as the problem size in terms of the number of network nodes, the number of network arcs, the capacities on the arcs and the size of the criterion cone defined by the objectives increases.

The first three items listed are problem parameters that affect the size of the entire feasible region and any subsets thereof. The size of the criterion cone, as demonstrated in Chapter 2, can effect how much of the feasible region is efficient thus affecting the sizes of the efficient and nondominated sets. Computational experience that provides evidence of this observation will be presented in Chapter 9.

**5.5 Integer Case**

If arc flows are restricted to integer values, the MONF becomes a MOIP. With all $x_i \geq 0$ and integer it is now possible to obtain a new category of solutions. Observations for the MONF with integer flows are described here.
Recall from Chapter 3, that for MOIP it is possible that there exist nondominated solutions that lie on the boundary of the convex hull of the feasible region in criterion space. These points are said to be supported by hyperplanes that may be generated by vectors within the relative interior of the positive orthant of criterion space. These supported nondominated points lie on the nondominated set of the relaxed feasible region. For integer problems, it is also possible to obtain nondominated solutions that do not lie on the nondominated set of the relaxed feasible region. These points are known as unsupported nondominated points. As such, while these points are not dominated by any feasible point, they are dominated by some convex combination of other nondominated points. The inverse images of supported nondominated points in criterion space are known as \textit{supported efficient points}. Similarly, the inverse images of unsupported nondominated points in criterion space are known as \textit{unsupported efficient points}.

**Observation 15:** \textit{For the MONF with integer variables, nondominated points may be supported or unsupported.}

This was illustrated with Example 5.1. The feasible region $\mathcal{Z}$ is repeated in Figure 5.11. The gray lines represent the nondominated set for the relaxed feasible region. Points that lie along these lines are supported nondominated. Domination sets are shown to illustrate which points are unsupported nondominated.

Finding unsupported points is difficult for two main reasons. First, many methods that are used to search the nondominated set employ movements that occur on the boundary of the feasible region. For sets that have a boundary that is disconnected, such as those restricted to integer solutions, such methods may stall upon reaching a point of
discontinuity. Second, weights are often used as a means of developing a relationship between the competing objectives. The relationship is then used to assess the relative utility of possible solutions. For these methods, the weights work much like the parametric values that occur in a convex combination of points. Thus, unsupported points will never appear since they will be dominated by some convex combination of extreme points. For methods that use weights in this way, it would make no difference that the point determined by the convex combination might not be feasible.

Figure 5.11 - Feasible Region in Criterion Space with Domination Sets

Observation 16: For the MONF with integer variables, solutions which lie on the convex hull of the feasible region may be extreme, lie in the interior of an edge, or lie on the interior of a higher order facet of the convex hull.
**Observation 17:** Integer efficient solutions that are members of the convex hull of the feasible region fit observations described for the MONF with continuous variables.

The convex hull of the feasible region in decision space has a structure that is equivalent to the boundary of the feasible region of MONF with continuous variables. So, efficient solutions that are members of the convex hull can be characterized in an equivalent manner to those on feasible region boundary for the continuous MONF.

**Observation 18:** The number of supported nonextreme efficient points along an edge of the convex hull efficient set is equal to Δ-1. Where Δ is equal to the total change along the fundamental cycle in the network when moving from one corresponding spanning tree to another.

Recall that efficient extreme point solutions correspond to spanning trees in the network and that in the presence of all integer problem coefficients will always consist of integer flows. Recall also that movement along an edge of the efficient set between one efficient extreme point to another corresponds to the change in a fundamental cycle when moving from one spanning tree (basic solution) to another. Let $x^1$ and $x^2$ represent the two efficient extreme point solutions and let $x_{ij}$ represent the nonbasic variable that becomes basic as movement is made along the adjoining edge from $x^1$ to $x^2$. If Δ represents the total change in $x_{ij}$, then Δ is integer and there are Δ-1 integer values between 0 and Δ. Each of these Δ-1 integer values corresponds to an integer flow solution between $x^1$ and $x^2$.

**Observation 19:** Unsupported nondominated points in criterion space lie in a special domination set associated with groups of supported nondominated points.
Let \( z^i \) and \( z^j \) be two supported nondominated points in criterion space for an MONF, then \( z^*_v \) is defined to be the *nadir point associated with points* \( z^i \) and \( z^j \). Also, define \( Z^v_z \) to be the domination set in criterion space associated with \( z^*_v \). Let \( \text{rel } Z^v_z \) denote the relative interior of \( Z^v_z \). Figure 5.12 illustrates these concepts for an integer MONF.

![Figure 5.12 - Nondominated Regions in Criterion Space](image)

In Figure 5.12 the dashed line represents a supporting hyperplane for the feasible region in criterion space. The points \( z^1 \), \( z^2 \) and \( z^3 \) are then supported, integer nondominated solutions for the MONF. The points \( z^*_{12} \) and \( z^*_{23} \) are the nadir points associated with points \( z^1 \) and \( z^2 \), and \( z^2 \) and \( z^3 \), respectively. The domination sets emanate
from $z_{12}^\nu$ and $z_{23}^\nu$. Notice that the points along the boundary are dominated by the associated supported nondominated solutions. However, if a point happens to lie within the relative interior of one of the domination sets it would not be dominated by the associated nondominated solutions. The points on the boundary of the convex hull for the feasible region define a region in which a nondominated point may lie. While not fully explored in this research, this observation may potentially be useful in placing bounds on the region of criterion space where unsupported nondominated points may be found.

This observation is similar to those discussed in Chapter 2 relating to unsupported points in decision space. One slight differences is that while the interior angle of the domination cone may vary in decision space, in criterion space it will always be $90^\circ$.

**Observation 20:** The number of unsupported efficient points, like supported efficient points, can depend on the size and shape of the feasible region as determined by the number of nodes, arcs and capacities on the network as well as the size of criterion cone.

We have seen with Observation 12 and in Chapter 2 that the efficient and nondominated sets tend to grow as the problem parameters increase. Intuitively, it would seem that the likelihood of unsupported efficient and nondominated solutions would increase as the size of these sets increase. It is not clear that this is necessarily true. Variations in the criterion cone may have a great effect on the proportion of efficient and nondominated points that are unsupported.

The criterion cone generated by the objectives may affect these proportions in several ways. As with continuous networks, as the cone widens out, there is a greater potential for all of the solutions categories to be included in the efficient and
nondominated sets. Also, a widening criterion cone corresponds to narrow domination sets in decision space raising the likelihood that the domination set may penetrate into the interior of the convex hull. In addition, the objective function coefficients are the primary factor affecting the mapping of the feasible region from decision space to criterion space. The nature of the criterion cone may contribute to a collapsing of unsupported efficient points as they are mapped to criterion space. An investigation into some of these issues is discussed in the next chapter.

5.6 Conclusion

In this chapter, we have introduced the multiple objective network flow problem along with observations about its structure for the cases of continuous and integer flows. From the observations and empirical investigation that were discussed several conclusions may be reached:

In the chapters that follow, the investigation into the MONF problem is carried further. The observations discussed here are used to develop specialized algorithms for generating all efficient extreme point solutions to MONF as well as searching the efficient set for a decision makers most preferred solution. In addition studies are presented that seek to improve our knowledge of the nondominated set and search capabilities for larger, more practical MONF problems.
CHAPTER 6
SIZE AND PROPORTIONS OF THE MONF EFFICIENT SET

As we have seen, there is interest in the nondominated and efficient sets of the MONF as the subsets of the feasible regions in their respective contexts. The observations of the previous sections have illustrated some important characteristics of these sets and their composition. In this chapter, we describe an empirical study designed to provide further insight into the nondominated set and by extension the efficient set. Namely, a study of how the size and composition of the nondominated set varies with changes in several MONF parameters is conducted and presented.

6.1 A Naïve Algorithm

Ideally, when studying the nondominated set it would be desirable to have the entire set at our disposal. For the MONF with continuous variables, the nondominated set is infinite and therefore impossible to enumerate. Nevertheless, the nondominated set for this case is easier to characterize due to its continuous nature. It would suffice in this case to enumerate the nondominated extreme points and their relationship to one another. Then any portion of the remainder of the nondominated set can be determined as a convex combination of some extreme points.

Obtaining a complete listing of the nondominated set for the MONF restricted to integer variables may at first glance seem easier. After all, since finite supplies as well as
capacity and non-negativity constraints bound the feasible region of the MONF the number of feasible integer solutions is finite. Finite however does not mean small and manageable. As mentioned in the previous section, additional complexity is added to the integer case by the possibility of unsupported nondominated points. Thus, for the integer case, some portions of the nondominated set may not be characterized using convex combinations of nondominated extreme points. Still, the finite nature of the feasible region and nondominated set makes the idea of a complete enumeration of integer nondominated solutions interesting.

A simple, naïve algorithm for obtaining a complete enumeration of the nondominated set for the integer MONF is possible. Here the idea is to exhaustively generate and examine one-by-one every possible combination of integer values that is feasible for the network. From the feasible combinations, a list of the nondominated solutions is compiled. This list may then be examined to determine and count the extreme points, supported nonextreme points and unsupported points.

The combinations will be generated in a systematic manner. As each combination is generated, it is first examined to verify its feasibility. Feasible solutions are then compared against the current list of nondominated solutions. If the solution under investigation is found to be dominated, it is discarded and the algorithm will move on to the next combination. If the new solution is nondominated, it is added to the list and any solutions currently in the list that are dominated by the new solution are removed. Once all feasible combinations are checked the final list will contain all nondominated solutions. The list may then be examined and relevant statistics acquired.
6.2 Limitations of the Naïve Algorithm

This strategy is simple enough and due to the finite nature of the set under investigation it is guaranteed to work, eventually. It would be naïve to think, however, that this would provide a practical method for characterizing the nondominated set. Several problems exist which limit the practical application of this strategy to real world problems.

First, how does one generate a feasible combination to begin with? A MONF has two types of constraints, capacity constraints that limit the arc flows and divergence constraints that require the inflow of a node to balance with the outflow. Any combination generated must satisfy both sets of constraints. For the implementation of the naïve algorithm to be described here, flow combinations are generated which satisfy arc capacities. The flow combination is then checked for feasibility of the divergence constraints. While this makes the generation of a combination easier, many combinations, which are divergence infeasible, may be unnecessarily generated.

Second, how many combinations will need to be evaluated and how long will this take? Consider a network with \( m \) arcs and capacities \( u_1, \ldots, u_m \). Taking only arc capacities into account, the number of combinations to be evaluated is equal to \( \prod_{i=1}^{m} u_i \). Obviously, the total number of combinations can get very large very quickly as the number of arcs or the arc capacities increase.

Finally, the size of the nondominated set may itself be quite large. In some unusual cases, such as Example 5.2 or when the null vector condition holds, it may be as large as
the feasible region itself. This may pose storage problems in addition to the large time requirements for analyzing a large nondominated set.

These practical limitations provide additional impetus for developing more sophisticated algorithms for generating the efficient and nondominated sets. One such strategy will be explored in the next chapter. Fortunately, some simple theory of networks may be applied to improve our naïve strategy for solving networks of reasonable size.

6.3 Improving the Naïve Algorithm

A first step is to reduce the number of combinations that need to be evaluated. Recall that the divergence constraints of the LP formulation for the network are represented by a system of $m$ linear, equality constraints with $n$ variables. Generally, $m \leq n$. It can be shown that for network flow problems that as long as the total supply and demand net to zero, $\sum_{i=1}^{m} b_i = 0$, then this system of constraints is consistent. For any such consistent system of equations, a result from linear algebra states that a set of $m$ variables may be chosen that form a basis of the system. The remaining $n-m$ variables are known as free or nonbasic variables. If the rank of the constant coefficient matrix for such a system is less than $n$, then such a system may have an infinite number of solutions. Nonbasic variables are called free because they do not depend on the values of the basis variables. On the other hand, if values are given for the nonbasic variables then the values of basic variables will be unique. Thus the values of basic variables depend on the values of the nonbasic variables. For our purposes, this means that once a basis is identified, we will only need to generate possible combinations of the nonbasic variables and then calculate
the corresponding basic variable values. This can reduce the number of combinations that will be evaluated significantly. Feasibility still needs to be verified since there is no guarantee that the derived basic variable values will be capacity feasible or nonnegative.

For our implementation of the naïve algorithm, we will generate all possible combinations of the nonbasic variables that are feasible within their capacity and non-negativity constraints and then check for feasibility of the calculated basic variables. So it is important to choose our basic and nonbasic variables wisely so that a minimum number of combinations need to be generated. This may be accomplished by choosing the group of nonbasic variables that have the lowest capacities from the available variables. Recall that for network flow problems, a basis of the mathematical model corresponds to a spanning tree of the network. One way to choose nonbasic variables of low capacity is to place into the basis the arcs with higher capacities. This is equivalent to finding the spanning tree of maximum capacity. We can do this by employing a simple maximum spanning tree algorithm with the capacities as arc costs.

For each feasible combination of arc flows generated, the objective function values for the new solution will be calculated and compared with those stored in a candidate list of nondominated solutions from previous combinations. Not only will it be necessary here to determine whether or not the new flow is nondominated, but if the new flow is in fact nondominated it may also be necessary to remove solutions in the list that may now be dominated by the new flow. Because of the dynamic nature of the nondominated list that may expand and contract as the list develops, an appropriate data structure needs to be chosen. For the implementation described here, the nondominated solutions will be stored as a linked list.
Once all feasible combinations of arc flows have been determined and evaluated and the nondominated list is finalized, the next task will be to analyze the generated nondominated set. Here we will wish to count the number of nondominated points which are extreme, supported nonextreme, or unsupported. Steuer (1989) provides a test for determining the status of nondominated solutions based on the fact that unsupported points are dominated by some convex combination of other nondominated solutions. This test, which is in the form of a linear program, is stated by Steuer as follows. Let\[ I_N = \{ j \mid z^j \in N \} \]represent a set of indices for each of the nondominated solutions. Then for each nondominated solution in the set we test by solving the following LP:\[
\max \left\{ \sum_{j=1}^{k} r_i = v \right\} \\
\text{s.t.} \quad \sum_{j \in I_N - \{h\}} \lambda_j z^j_i - r_i = z^h_i, \quad 1 \leq i \leq k, \\
\lambda_j = 1, \quad j \in I_N - \{h\} \\
\lambda_j, r_i \geq 0, \quad \text{all }
\]
If it turns out that the LP is inconsistent then the point being tested is supported extreme. The objective function value will be \( v = 0 \) when the point is supported non-extreme and when \( v > 0 \) the point is unsupported. One such test will be carried out for each point in the nondominated set.

6.4 The Improved Naïve Algorithm

**Step 1**: Find a spanning tree of maximum capacity; and set the \( N \), the number of nondominated solutions, to zero.

**Step 2**: If all feasible combinations of nonbasic arc flows have been checked go to Step 9.

**Step 3**: Generate a feasible combination of nonbasic arc flows.
Step 4: Calculate basic arc flows and check feasibility of basic arcs. If feasible go to Step 4. Otherwise, go to step 2.

Step 5: Let $z^0$ represent the calculated objective values.

Step 6: If the nondominated candidate list is empty, add $z^0$ solution to the list, set $N=1$ and go to Step 2; Otherwise, go to Step 7.

Step 7: For each vector $z^i$ currently in the candidate list, compare the new vector, $z^0$. If the $z^0$ is dominated by $z^i$, stop the comparison and go to Step 2. If the $z^i$ is dominated by $z^0$, then remove $z^i$, let $N = N - 1$, and continue with next vector $z^{i+1}$ in list.

Step 8: Reaching this Step means that the new vector is nondominated, so add it to the candidate list and let $N = N + 1$. Go to Step 2.

Step 9: For each vector in nondominated list, conduct the Steuer test for supported, supported extreme, or unsupported. Increment appropriate count based on the outcome of the test.

6.5 Experimental Method

While the naïve algorithm may not be suitable for practical use with large real-world problems, it does afford an opportunity to experiment with various MONF problem parameters. Using this algorithm, an empirical experiment was conducted to see the effect that changes in MONF problem parameters have on the size and composition of the nondominated set. For this experiment, random problems were generated while varying parameters which effect network and objective cone sizes. These parameters include the number of network nodes, the arc density, the total network supply and demand, the number of objectives and the size of the criterion cone.
The problems were generated at random using a variation of the NETGEN algorithm of Klingman, Napier, and Stutz (1974). Specifically, the NETGEN algorithm, a popular generator of single objective network flow problems, was modified to allow for the random generation of multiple objectives. This modification provided for the specification of the number of objectives, lower and upper bounds on objective coefficient values, and a parameter used to control the percentage of costs that will equal the desired upper bound. By adjusting the range between the lower and upper bound along with the percentage of costs that are allowed to vary, one may control to some extent the size of the criteria cone.

For this experiment all networks with integer flows only were generated with one supply and one demand node. The main network parameters were varied according to the factor levels and settings shown in Table 6.1. Parameters that affect the size of the network include the number of nodes and arcs in the network, the total supply and demand. The number of objectives and the size of the criterion cone generated by the objectives were also varied. The values of the arc factor levels used were selected to provide low, medium and high-density networks and they varied according to the number of nodes in the network. Arc capacities were chosen at random with a reasonable percentage allowed to match the total supply to ensure network feasibility. Adjusting the range from which the objective function coefficients are randomly chosen varied the size of the criterion cone. Generally, the wider the range, the wider the criterion cone. This range was varied by keeping the upper bound on the range constant at a value of 10 units and changing the lower range by the values shown in the table.
The networks were generated using factor combinations determined using a three level, full factorial experimental design. Two repetitions were conducted for each experimental combination for a total of 486 networks tested. For each network, a total enumeration of the nondominated set was obtained. From this enumeration, the total number of feasible solutions, the total number of nondominated solutions, the total number of nondominated extreme solutions, the total number of nondominated supported solutions, and the total number of nondominated unsupported solutions were obtained and recorded. These values will serve as the dependent variables for our analysis. The generation of random networks, the Naïve algorithm and compilation of results were carried out using routines written in Visual Basic 5.0 on Pentium based computers. The total number of feasible solutions for each of the problems investigated ranged from 16 to 92,897,280.

### Table 6.1 - Factor Levels for Empirical Analysis

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Factor Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>a. Nodes</td>
<td>4</td>
</tr>
<tr>
<td>b. Arcs</td>
<td></td>
</tr>
<tr>
<td>(6 nodes)</td>
<td>8</td>
</tr>
<tr>
<td>(8 nodes)</td>
<td>10</td>
</tr>
<tr>
<td>(10 nodes)</td>
<td>12</td>
</tr>
<tr>
<td>c. Objectives</td>
<td>2</td>
</tr>
<tr>
<td>d. Supply</td>
<td>4</td>
</tr>
<tr>
<td>e. Minimum Cost</td>
<td>10</td>
</tr>
</tbody>
</table>

6.6 Experimental Results
The purpose of this experiment is to determine which problem parameters have an effect on the structure of the efficient set and to what extent. Since variation needs to be taken into account in making these determinations, analysis of variance (ANOVA) has been selected as the statistical method to interpret the experimental data. Developed by Sir Ronald Fisher in the 1930s, ANOVA is a statistically based decision tool for detecting any differences in average performance of groups of items tested. The decisions, rather than using pure judgement take variation into account. The ANOVA model may be written as:

\[ Y_{ijklmn} = \mu_{ijklmn} + \varepsilon_{ijklmn} \]

For this model, the \( Y_{ijklmn} \) represents the response variable for the \( i \)th level of factor A, the \( j \)th level of factor B, the \( k \)th level of factor C, and so on. The \( \mu_{ijklmn} \) are parameters estimated by the treatment means, and the \( \varepsilon_{ijklmn} \) represents error terms which are independent N(0, \( \sigma \)). The indices vary to correspond with the levels used for each of the factors. For this experiment, three different levels were used for each factor.

For our analysis, the network factors listed in Table 6.1 will serve as our controlled, independent variables and the counts of the various nondominated set categories will serve as the dependent variables of interest. Several interesting results can be seen from this data which are described next.

Separate analyses, each carried out using a similar strategy, are discussed for each of the dependent variables. Initially, an ANOVA model incorporating all possible factors along with 2- and 3-factor interactions was fit. F-test values for the factors were used to determine which of the factors and interaction effects were significant. Plots of the means were inspected for further evidence of interaction. In addition, normal probability plots,
and plots of the residuals versus the fitted values were checked to make sure that the model assumptions of independent, equal, and normal error terms were met. These plots showed that for all dependent variables the error variances tended to increase proportional to the estimated mean. Fortunately, the employment of equal sample sizes for each treatment minimizes the effect of unequal variances on the ANOVA model (Neter, Wasserman, and Kutner (1990)). A logarithmic transformation \( Y' = \log Y \) for each response variable was used to stabilize the error variances further and correct for lack of normality. Using the transformed response, additional ANOVA models were investigated and diagnosed to check further departures from assumptions. For each analysis, only the final ANOVA results are presented along with any relevant graphs and statistics.

### 6.6.1 ANOVA for Number of Feasible Solutions

The number of feasible solutions for a network is the first dependent variable investigated. For the problem set, the number of total number of feasible solutions ranged from 1 to 2,773,707. The ANOVA table for the logarithmic transformation of this variable is shown in Table 5.2. Since all ANOVA tables will be presented in this form a brief summary of the table will be given here. The column label **Source** lists the main factors and interactions whose effect on the independent variable was found to be significant. The **DF** lists the degrees of freedom for each factor. **SS** and **V** stand for sum of squares and variance, respectively. With ANOVA, the sum of square for a factor gives an indication of the portion of the total sum of squares that can be attributed to the factor. In other words, it provides a measure with which we may compare the relative effect of each factor. The variance is calculated by dividing the sum of squares by the degrees of
freedom. The F column presents a value for the F-distribution used in testing the significance of a particular factor's contribution to the overall variance. Related to F is the p-value that is presented in the column labeled p. This value is the probability that a given F statistic would be obtained assuming that there is no significant contribution from the factor. Thus low p-values, which usually correspond to higher values of the F-statistic, give a good indication that the factor does provide a significant contribution to the overall variation.

Table 6.2 - ANOVA for Number of Feasible Solutions (Log Transformed)

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>SS</th>
<th>V</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>2</td>
<td>92.598</td>
<td>46.299</td>
<td>220.38</td>
<td>0.000</td>
</tr>
<tr>
<td>Arcs</td>
<td>2</td>
<td>259.592</td>
<td>129.796</td>
<td>613.47</td>
<td>0.000</td>
</tr>
<tr>
<td>Supply</td>
<td>2</td>
<td>85.919</td>
<td>42.9595</td>
<td>203.4</td>
<td>0.000</td>
</tr>
<tr>
<td>Node×Arc</td>
<td>4</td>
<td>2.508</td>
<td>0.627</td>
<td>2.98</td>
<td>0.019</td>
</tr>
<tr>
<td>Node×Supply</td>
<td>4</td>
<td>7.732</td>
<td>1.933</td>
<td>9.1</td>
<td>0.000</td>
</tr>
<tr>
<td>Arc×Supply</td>
<td>4</td>
<td>11.777</td>
<td>2.94425</td>
<td>13.96</td>
<td>0.000</td>
</tr>
<tr>
<td>Error</td>
<td>459</td>
<td>98.315</td>
<td>0.214194</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>485</td>
<td>558.441</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The ANOVA output shown in Table 6.2 indicates that only the main factors of number of nodes, number of arcs and network supply have any effect in determining the number of feasible solutions for the MONF. In addition, only interactions between these main factors are significant. The main factors of number of objectives and the cone angle show no significant impact. This is not surprising since the entire feasible region is defined by the network structure as determined by these variables. It does however increase confidence in the validity of the experimental data.

For the logarithmic transformation results shown here, the main factors and interactions account for 82.3% of the total variation with the number of arcs having the
most impact by accounting for 46.5% of the variation. The number of nodes and the
supply account for nearly the same proportion of variation with 16.6% and 15.3%,
respectively. It is interesting to note that in addition to the main effects all possible
interactions between these three main effects contribute a statistically significant
proportion of the total variation.

6.6.2 ANOVA for Number of Nondominated Solutions

For the problem set, the number of total number of nondominated solutions ranged
from 1 to 1058. While the number of objectives and the cone angle have no effect on the
size of the feasible region, the ANOVA model results presented in Table 6.3 indicate that
these two factors contribute the most to the variation in the number of nondominated
points. The cone angle main effect accounts for approximately 26.0% of the total
variation while the number of objectives main effect accounts for around 25.3%. All
significant factors account for 64% of the total variation in number of nondominated
points.

Table 6.3 - ANOVA for Number of Nondominated Solutions (Log Transformed)

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>SS</th>
<th>V</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>2</td>
<td>8.1588</td>
<td>4.0794</td>
<td>21.07</td>
<td>0.000</td>
</tr>
<tr>
<td>Arcs</td>
<td>2</td>
<td>10.6244</td>
<td>5.3122</td>
<td>25.35</td>
<td>0.000</td>
</tr>
<tr>
<td>Objectives</td>
<td>2</td>
<td>70.7771</td>
<td>35.38855</td>
<td>165.79</td>
<td>0.000</td>
</tr>
<tr>
<td>Supply</td>
<td>2</td>
<td>5.9475</td>
<td>2.97375</td>
<td>13.63</td>
<td>0.000</td>
</tr>
<tr>
<td>Cone Angle</td>
<td>2</td>
<td>72.8076</td>
<td>36.4038</td>
<td>170.06</td>
<td>0.000</td>
</tr>
<tr>
<td>Arcs×Cone Angle</td>
<td>4</td>
<td>3.0582</td>
<td>0.76455</td>
<td>3.69</td>
<td>0.006</td>
</tr>
<tr>
<td>Objs.×Cone Angle</td>
<td>4</td>
<td>7.3612</td>
<td>1.8403</td>
<td>8.54</td>
<td>0.000</td>
</tr>
<tr>
<td>Error</td>
<td>467</td>
<td>100.6248</td>
<td>0.2154707</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>485</td>
<td>279.3596</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.6.3 ANOVA for Number of Nondominated Extreme Solutions
For the problem set, the number of total number of nondominated extreme solutions ranged from 1 to 226. The output in Table 6.4 shows that the same main effect and interaction factors significant in the ANOVA for the number of nondominated solutions are also significant for that of the number of nondominated extreme points although with slight changes in the contribution of each. Here, the number of objectives has the greatest effect contributing about 30.8% of the variation in number of objectives. The contribution that was provided by the cone angle for this model was 13.9%. It is interesting to note that while small the contribution of the interaction between the number of arcs and the cone angle was significant for both the number of nondominated solutions and the number of nondominated extreme points.

Table 6.4 - ANOVA for Number of Nondominated Extreme Points (Log Transformed)

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>SS</th>
<th>V</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>2</td>
<td>3.4396</td>
<td>1.7198</td>
<td>13.26</td>
<td>0.000</td>
</tr>
<tr>
<td>Arcs</td>
<td>2</td>
<td>3.1572</td>
<td>1.5786</td>
<td>11.36</td>
<td>0.000</td>
</tr>
<tr>
<td>Objectives</td>
<td>2</td>
<td>44.7834</td>
<td>22.3917</td>
<td>152.9</td>
<td>0.000</td>
</tr>
<tr>
<td>Supply</td>
<td>2</td>
<td>1.8333</td>
<td>0.91665</td>
<td>6.11</td>
<td>0.002</td>
</tr>
<tr>
<td>Cone Angle</td>
<td>2</td>
<td>20.2156</td>
<td>10.1078</td>
<td>68.87</td>
<td>0.000</td>
</tr>
<tr>
<td>Arcs×Cone Angle</td>
<td>4</td>
<td>1.4612</td>
<td>0.3653</td>
<td>2.53</td>
<td>0.040</td>
</tr>
<tr>
<td>Objs×Cone Angle</td>
<td>4</td>
<td>1.5222</td>
<td>0.38055</td>
<td>2.58</td>
<td>0.037</td>
</tr>
<tr>
<td>Error</td>
<td>467</td>
<td>68,096</td>
<td>0.1473439</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>485</td>
<td>145.222</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.6.4 ANOVA for Number of Nondominated Supported, Nonextreme Solutions

For the problem set, the number of total number of nondominated supported, nonextreme solutions ranged from 0 to 515. Note, that the maximum value for this range is higher than that for nondominated extreme points. From Table 6.5 it is seen that the significant factors and interactions contributing to the variation in the number of
nondominated, nonextreme solutions together contribute approximately 58.5% of the total variation. Here again, the number of objectives and the cone angle main effects contribute the greatest proportions with 18.1% and 20.9%, respectively. A greater contribution is made by an interaction effect, namely the interaction between the number of objectives and the cone angle, than in any previous ANOVA model. This interaction effect contributes roughly 10.5% of the total variation in number of nondominated, supported extreme points.

Table 6.5 - ANOVA for Number of Nondominated Supported Points (Log Transformed)

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>SS</th>
<th>V</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>2</td>
<td>1.3822</td>
<td>0.6911</td>
<td>4.53</td>
<td>0.011</td>
</tr>
<tr>
<td>Arcs</td>
<td>2</td>
<td>4.2348</td>
<td>2.1174</td>
<td>11.62</td>
<td>0.000</td>
</tr>
<tr>
<td>Objectives</td>
<td>2</td>
<td>38.2318</td>
<td>19.1159</td>
<td>99.31</td>
<td>0.000</td>
</tr>
<tr>
<td>Supply</td>
<td>2</td>
<td>4.5173</td>
<td>2.25865</td>
<td>11.95</td>
<td>0.000</td>
</tr>
<tr>
<td>Cone Angle</td>
<td>2</td>
<td>44.2383</td>
<td>22.11915</td>
<td>117.67</td>
<td>0.000</td>
</tr>
<tr>
<td>Nodes×Arcs</td>
<td>4</td>
<td>2.0066</td>
<td>0.50165</td>
<td>2.58</td>
<td>0.037</td>
</tr>
<tr>
<td>Nodes×Supply</td>
<td>4</td>
<td>1.3182</td>
<td>0.32955</td>
<td>1.77</td>
<td>0.134</td>
</tr>
<tr>
<td>Arcs×Cone Angle</td>
<td>4</td>
<td>2.412</td>
<td>0.603</td>
<td>3.44</td>
<td>0.009</td>
</tr>
<tr>
<td>Objs×Cone Angle</td>
<td>4</td>
<td>22.1691</td>
<td>5.542275</td>
<td>28.74</td>
<td>0.000</td>
</tr>
<tr>
<td>Supply×Cone Angle</td>
<td>4</td>
<td>2.841</td>
<td>0.71025</td>
<td>3.7</td>
<td>0.006</td>
</tr>
<tr>
<td>Error</td>
<td>455</td>
<td>87.4375</td>
<td>0.1921703</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>485</td>
<td>210.7888</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.6.5 ANOVA for Number of Nondominated Unsupported Solutions

For the problem set, the number of total number of nondominated unsupported solutions ranged from 0 to 740. Note, that the maximum value for this range is higher than those for the nondominated extreme points and the nondominated supported categories. The ANOVA results for the logarithmic transformation of the number of nondominated unsupported solutions is presented in Table 6.6. The most model includes
the most main and interaction effects as being significant than any of the previous models. Together the factors included here contribute to 72.5% of the total variation. As might be expected, once again the two major contributors are the main effects of the number of objectives and the cone angle. Here the cone angle contributes to 31.5% of the variation while the number of objectives contributes to 12.2%. The interaction between these two factors contributes and additional 6.2% of the variation. It is also interesting to note that contribution of the number of arcs is also significant at 6.7%.

Table 6.6 - ANOVA for Number of Nondominated Unsupported Points (Log Transformed)

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>SS</th>
<th>V</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>2</td>
<td>10.6172</td>
<td>5.3086</td>
<td>30.96</td>
<td>0.000</td>
</tr>
<tr>
<td>Arcs</td>
<td>2</td>
<td>20.1636</td>
<td>10.0818</td>
<td>54.45</td>
<td>0.000</td>
</tr>
<tr>
<td>Objectives</td>
<td>2</td>
<td>36.4599</td>
<td>18.22995</td>
<td>95.88</td>
<td>0.000</td>
</tr>
<tr>
<td>Supply</td>
<td>2</td>
<td>3.6852</td>
<td>1.8426</td>
<td>9.99</td>
<td>0.000</td>
</tr>
<tr>
<td>Cone Angle</td>
<td>2</td>
<td>94.2295</td>
<td>47.11475</td>
<td>252.33</td>
<td>0.000</td>
</tr>
<tr>
<td>Nodes×Arcs</td>
<td>4</td>
<td>2.8074</td>
<td>0.70185</td>
<td>3.37</td>
<td>0.010</td>
</tr>
<tr>
<td>Nodes×Objs.</td>
<td>4</td>
<td>2.8225</td>
<td>0.705625</td>
<td>3.72</td>
<td>0.005</td>
</tr>
<tr>
<td>Nodes×Cone Angle</td>
<td>4</td>
<td>5.3985</td>
<td>1.349625</td>
<td>7.54</td>
<td>0.000</td>
</tr>
<tr>
<td>Arcs×Objs.</td>
<td>4</td>
<td>3.497</td>
<td>0.87425</td>
<td>4.46</td>
<td>0.002</td>
</tr>
<tr>
<td>Arcs×Cone Angle</td>
<td>4</td>
<td>10.702</td>
<td>2.6755</td>
<td>14.84</td>
<td>0.000</td>
</tr>
<tr>
<td>Objs.×Cone Angle</td>
<td>4</td>
<td>18.5418</td>
<td>4.63545</td>
<td>24.26</td>
<td>0.000</td>
</tr>
<tr>
<td>Supply×Cone Angle</td>
<td>4</td>
<td>2.7733</td>
<td>0.693325</td>
<td>3.64</td>
<td>0.006</td>
</tr>
<tr>
<td>Arcs×Objs.×C. Angle</td>
<td>8</td>
<td>3.5551</td>
<td>0.4443875</td>
<td>2.34</td>
<td>0.018</td>
</tr>
<tr>
<td>Error</td>
<td>439</td>
<td>83.4895</td>
<td>0.1901811</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>485</td>
<td>298.7426</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.6.6 Additional Results

Another interesting result may be seen in Figures 6.1 and 6.2. In these figures, plots of the number of supported points versus the average cone angle and the number of unsupported points versus the average cone angle are presented. The distributions of both
appear very similar. It is interesting to note here that no supported or unsupported points appear for problems where the average cone angle is less than 40 degrees. In addition, the numbers for both seem to peak around an average cone angle of 80 degrees.

Figure 6.1 - Plot of Number of Supported Points versus Average Cone Angle

Figure 6.2 - Plot of Number of Unsupported Points versus Average Cone Angle

6.7 Conclusion
This investigation introduced a Naïve Algorithm for finding all nondominated solutions of the MONF problem. While the Naïve algorithm is guaranteed to list all possible nondominated solutions for any MONF problem, its major disadvantage is that the solution time becomes impractical very quickly as the size of the network increases. Nevertheless, the Naïve algorithm allowed us to conduct an investigation of how structural parameters of MONF are related to the size and shape of the set of potential solutions.

From the observations of Chapter 5 and empirical investigation several conclusions may be reached:

- The size of the feasible region and the sizes of the efficient and nondominated sets can get large quickly for MONF. Thus, there is a need for effective solution procedures for searching the solution set in a reasonable amount of time.

- The presence of nonextreme supported and unsupported nondominated points to a need for a MONF solution method that can reach these points. Existing network flow solution methods lead primarily to extreme point solutions only. There are available several MOLP methods that can find nonextreme solutions. It may be possible that these may be adapted to take advantage of the special structure MONF.

- One characteristic that might be possible to exploit is the correspondence between edges of the efficient set and fundamental cycles on the network. By taking advantage of special data structures associated with networks, movements between bases using fundamental cycles can be very fast. This may prove useful for general movements around the efficient set in particular when moving from extreme point to extreme point and approaching points which lie along edges of the efficient set.
The size of the feasible region is affected only by structural network parameters including the number of nodes, the number of arcs and the total supply/demand and is not affected by the objectives of the problem. Additional network characteristics such as arc capacities and the shape of the network itself may also play a role.

The number of objectives and the average angle of the cone generated by the objectives are the primary factors affecting the size of the nondominated set.

For certain average cone angles, the presence of nonextreme nondominated solutions was not detected. Possible heuristics may be developed based on cone size to indicate whether or not there is a need to search for nonextreme solutions for a given MONF.

A comparison of the range of values obtained for each category demonstrates that it is possible that more of the nondominated set is unsupported than supported for the integer MONF.
CHAPTER 7

FINDING ALL EFFICIENT EXTREME POINTS OF MONF

7.1 Introduction

In many situations involving problems with multiple objectives it is desirable to have an approach for listing the entire sets of nondominated and efficient solutions. A complete listing may provide further insights into the characteristics of these sets. Thus, this capability is important for research into MOLP. In some cases, a listing may provide a beginning point for searching out the decision-makers' most preferred solution.

In this chapter, we describe a vector maximum algorithm for enumerating all efficient extreme points of the MONF problem that combines vector maximum theory with the simplex on a graph algorithm. This algorithm is an improvement over the Naïve algorithm of Chapter 5. Then, the supporting theory for implementation of the algorithm is discussed. We then describe a computer implementation of the algorithm called MONFSOLV and its features. Finally, an empirical test is described comparing the performance of the algorithm with a popular general MOLP solver.

7.2 Overview of Algorithm

Several authors have investigated methods for listing the sets of efficient bases, efficient extreme points, and nondominated extreme points. These include Evans and Steuer (1973), Isermann (1977), Steuer (1977), Yu and Zeleny (1975), and Zionts and Wallenius (1980).
A general strategy for characterizing the efficient set is to begin by locating an initial efficient extreme point and use it to start a list. A systematic search for efficient extreme points can then be carried out in which at each point in the list all adjacent efficient extreme points are determined. Adjacent efficient extreme points that have not
been previously determined are added to the list. These will be visited in turn. The search will end when all points in the list have been visited and no new efficient points that are connected by an efficient edge are found. A flowchart of the algorithm is shown in Figure 7.1.

While stating a search strategy can be difficult, carrying it out requires several steps that could be equally complex. First, a method is needed for moving to and between basic feasible solutions. It will be shown that this may be accomplished on the network by utilizing the simplex on a graph method. Second, at each efficient extreme point that is visited, a test is needed to determine which adjacent extreme points are efficient. This may be accomplished with a network adaptation of the Evans-Steuer (Steuer (1989)) test for nonbasic variable efficiency. Finally, there are several practical considerations such as an efficient method of handling the lists of efficient and nondominated points. These and other considerations will be discussed in the next sections.

7.3 MONF Formulation Revisited

Recall from Chapter 5 that the MONF problem may be formulated as

\[
\begin{align*}
\min & \{ c'x = z_1 \} \\
\min & \{ c^2x = z_2 \} \\
& \ddots \\
\min & \{ c^kx = z_k \} \\
\text{s.t.} & \quad x \in S \\
\end{align*}
\]

\[
S = \left\{ x \in R^n \mid \sum_{(j,i) \in A} x_{ij} - \sum_{(j,i) \notin A} x_{ji} = b_i; \ 0 \leq x_{ij} \leq u_{ij}; \ \forall (i,j) \in A \right\}.
\]
In this formulation there are \( k \) linear minimization objectives. There are \( n \) decision variables, \( x \in \mathbb{R}^n \). Each \( x_{ij} \) represents the amount of flow on the arc from node \( i \) to node \( j \). The feasible region is formed by \( m \) divergence constraints corresponding to the balance of flow at each of the \( m \) nodes and the \( m \) capacity constraints. We assume that the total of all RHS values for the divergence constraints is zero, \( \sum_{i=1}^{m} b_i = 0 \), since this is necessary for feasibility. The \( u_{ij} \) represents the flow capacity on the arc from node \( i \) to node \( j \).

**Example 7.1:** Consider the network of Figure 7.2 that will be used as an example for much of the chapter.

![Figure 7.2 - A 3-Objective MONF](image)

The MONF formulation for this network is
7.4 Vector Maximum Theory

In the introduction to the chapter it was mentioned that the algorithm described could be classified as a vector maximum algorithm. In this section, we define a vector algorithm and describe some theoretical foundations.

A vector maximum problem attempts to maximize a vector. Strictly speaking with MONF, we are interested in minimizing the vector of objective function values

\[
\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{bmatrix}
\]

In reality, this is not a problem since minimizing a given vector is equivalent to maximizing the negative of the vector.

The difficulty arises for MONF, as discussed in Chapter 2, as in most cases there is no single point which simultaneously minimizes all components of the vector. The concept of efficiency is applied to generalize vector maximization to the MONF. The
following definitions and theorems are derived from Steuer (1989) to provide a foundation for vector maximum theory related to MONF.

Recall from Chapter 2 that for minimization problems a point $\bar{x} \in S$ is efficient if and only if there does not exist another $x \in S$ such that $Cx \leq C\bar{x}$ and $C\bar{x} \neq Cx$. If the feasible region $S$ of MONF has an efficient point, then at least one extreme point of $S$ is efficient. Also recall that if $\bar{x} \in S$ minimizes the weighted sums $\text{LP} \min \{\lambda^T Cx \mid x \in S\}$ where $\lambda \in \Lambda$, then $\bar{x}$ is an efficient point. If $\bar{x}$ is an extreme point, then at least one basis $B$ associated with $\bar{x}$ is an efficient basis. In other words, $B$ is an efficient basis if and only if $B$ is an optimal basis of the weighted sums $\text{LP}$ for some $\lambda \in \Lambda$. This provides a correspondence between an efficient extreme point and an efficient basis. So if we can find all efficient bases, we can find all efficient extreme points and vice versa. Recall from Chapter 3 that bases also correspond to spanning trees on a network. In addition, Steuer (1989) shows that all efficient bases are connected via efficient pivots. If we can find one efficient basis, we should be able to find all others by somehow following these connections.

We may also define the reduced cost matrix $W$ for a basis $B$. Let $C_B$, $C_N$ and $N$ represent the basic columns of the criterion matrix $C$, the nonbasic columns of $C$, and the nonbasic columns of the constraint matrix $A$, respectively. Then $W = C_N - C_B B^{-1} N$. The matrix $W$ is composed of the reduced costs, also known as shadow prices, that are calculated in a simplex tableau for a given basis $B$. The number of rows in the matrix corresponds to the $k$ objectives while the number of columns equals $(n - m)$, the number of nonbasic variables.
The reduced cost of an arc \((i, j)\), denoted \(c_{ij}^\pi\), is calculated based on node potentials. The node potentials are equivalent to the dual variables for the LP formulation. Denoted \(\pi_i\), there is one node potential for each node in the network. We can arbitrarily set one of the node potentials. It is customary to set the node potential of the root to zero, \(\pi_1 = 0\). Given two nodes \(i\) and \(j\) connected by arc \((i, j)\) with cost \(c_{ij}\), the node potentials are related by \(\pi_j = \pi_i + c_{ij}\). After setting \(\pi_1 = 0\), we can work our way through the spanning tree calculating the other node potentials.

The reduced cost for a nonbasic arc may be calculated using \(c_{ij}^\pi = c_{ij} - \pi_i + \pi_j\).

To determine whether or not a given basis for the network is optimal, we check to see whether the basis satisfies the following optimality conditions:

\[
c_{ij}^\pi \geq 0 \text{ for every arc } (i, j) \text{ with } x_{ij} = 0
\]

\[
c_{ij}^\pi \geq 0 \text{ for every arc } (i, j) \text{ with } x_{ij} = u_{ij}.
\]

The reduced cost matrix \(W\) for the MONF is then made up of the reduced costs of nonbasic arcs at zero or at capacity flow. We can then partition \(W\) into two parts, letting \(W^+\) consist of the columns of \(W\) which correspond to arcs with flows at capacity and \(W^-\) consist of the columns of \(W\) which correspond to arcs with flow at zero.

The reduced cost matrix for the weighted sums LP is given by \(\lambda^T W\). This implies that a basis \(B\) is efficient if and only if the system of inequalities

\[
\lambda^T W^+ \geq 0
\]

\[
\lambda^T W^- \leq 0
\]

\[
\lambda > 0
\]

is consistent.
Two bases, $\mathbf{B}$ and $\hat{\mathbf{B}}$, are adjacent if it only takes one pivot to obtain one from the other. If $\mathbf{B}$ is an efficient basis and $x_{ij}$ is a nonbasic arc, then $x_{ij}$ is an efficient nonbasic variable with respect to $\mathbf{B}$ if and only if there is a $\lambda$ such that

$$\lambda W^- \leq 0, \lambda W^+ \geq 0, \text{and } \lambda w_{ij} = 0$$

where $w_{ij}$ is the column of $\mathbf{W}$ associated with arc $x_{ij}$. If $\mathbf{B}$ is an efficient basis and $x_{ij}$ is an efficient nonbasic variable with respect to $\mathbf{B}$, then any feasible pivot to make $x_{ij}$ basic is an efficient pivot with respect to $\mathbf{B}$ and $x_{ij}$. If $\mathbf{B}$ is an efficient basis, then any efficient pivot from $\mathbf{B}$ results in an adjacent efficient basis $\hat{\mathbf{B}}$ and the edge between their associated extreme points is efficient.

So far, efficient bases are associated with efficient extreme points and we can move from one efficient basis to an adjacent efficient basis via an efficient pivot. The following theorems provide the means for detecting which nonbasic variables are efficient with respect to a given basis $\mathbf{B}$ for MONF. They are based on the Evans-Steuer test for nonbasic variable efficiency (Steuer (1989)).

**Theorem 7.1** (Nonbasic variable with zero flow): Let $x_{ij} = 0$ be nonbasic with respect to $\mathbf{B}$. Let $\mathbf{W}$, with partitions $\mathbf{W}^+$ and $\mathbf{W}^-$, be the reduced cost matrix of $\mathbf{B}$. Then, all feasible pivots from the $x_{ij}$ entering column are efficient pivots if and only if the subproblem

$$\max \left\{ e^T v \right\}$$

$$s.t. W^+ y^+ - W^- y^- + w_{ij} \delta - Iv = 0$$

$$0 \leq y \in R^{n-m}$$

$$0 \leq \delta \in R$$

$$0 \leq v \in R^k$$

has an optimal objective function value of zero.
Proof: If $B$ is an efficient basis, then $x_j$ is an \textit{efficient nonbasic variable with respect to $B$} if and only if there is a $\lambda$ such that $\lambda^T W^- \leq 0$, $\lambda^T W^+ \geq 0$, and $\lambda w^j = 0$ where $w^j$ is the column of $B$ associated with arc $x_{ij}$. For $x_{ij} = 0$, $\lambda w^j = 0 \Rightarrow (w^j)^T \lambda = 0 \Rightarrow (w^j)^T \lambda \geq 0$.

For $x_{ij}$ is efficient with respect to $B$ if and only if

$$\min \{0^T \lambda\}$$

$$s.t. \quad W^+ \lambda \geq 0$$
$$W^- \lambda \leq 0$$
$$(w^j)^T \lambda = 0$$
$$I \lambda \geq e$$
$$\lambda \geq 0$$

Rearranging and substituting we have

$$\min \{0^T \lambda\}$$

$$s.t. \quad -W^+ \lambda \leq 0$$
$$W^- \lambda \leq 0$$
$$-(w^j)^T \lambda \leq 0$$
$$-I \lambda \leq e$$
$$\lambda \geq 0$$

The dual is then

$$\max \{e^T v\}$$

$$s.t. \quad -W^+ y^+ + W^- y^- - w^j \delta - Iv \geq 0$$
$$y \in R^{n-m}$$
$$\delta \in R$$
$$v \in R^k$$

Adding surplus and rearranging the constraints yields
The vector $t$ may be removed since the objective function may increase by setting $t_i = 0$.

**Theorem 7.2** (Nonbasic variable with capacity flow): Let $x_{ij}$ be a nonbasic variable at capacity flow with respect to $B$. Let $W$ be the reduced cost matrix associated with $B$ which may be partitioned into $W^+$ and $W^-$. Then, all feasible pivots that can be made from the $x_{ij}$ entering column are efficient pivots if and only if the subproblem

$$
\max \left\{ e^T v \right\} \\
\text{s.t.} \quad W^+ y^+ - W^- y^- + w^\| \delta + Iv + It = 0 \\
0 \leq y \in R^{n-m} \\
0 \leq \delta \in R \\
0 \leq v \in R^k
$$

has an optimal objective function value of zero.

**Proof:** If $B$ is an efficient basis, then $x_j$ is an efficient nonbasic variable with respect to $B$ if and only if there is a $\lambda$ such that $\lambda W^- \leq 0$, $\lambda W^+ \geq 0$, and $\lambda w^j = 0$ where $w^j$ is the column of $B$ associated with arc $x_{ij}$. For $x_{ij} = 0$, $\lambda w^j = 0 \Rightarrow (w^j)^T \lambda = 0 \Rightarrow (w^j)^T \lambda \leq 0$.

For $x_{ij}$ is efficient with respect to $B$ if and only if

$$
\min \left\{ 0^T \lambda \right\} \\
\text{s.t.} \quad W^+ \lambda \geq 0 \\
W^- \lambda \leq 0 \\
(w^j)^T \lambda = 0 \\
I \lambda \geq e \\
\lambda \geq 0
$$
Rearranging and substituting we have

\[
\begin{align*}
\min \left\{ 0^T \lambda \right\} \\
\text{s.t.} & \quad -W^{+T} \lambda \leq 0 \\
& \quad W^{-T} \lambda \leq 0 \\
& \quad (w^{ij})^T \lambda \leq 0 \\
& \quad -I \lambda \leq e \\
& \quad \lambda \geq 0
\end{align*}
\]

The dual is then

\[
\begin{align*}
\max \left\{ e^T v \right\} \\
\text{s.t.} & \quad -W^+ y^+ + W^- y^- + w^{ij} \delta - Iv \geq 0 \\
0 \leq y & \in R^{n-m} \\
0 \leq \delta & \in R \\
0 \leq v & \in R^k
\end{align*}
\]

Adding surplus and rearranging the constraints yields

\[
\begin{align*}
\max \left\{ e^T v \right\} \\
\text{s.t.} & \quad W^+ y^+ - W^- y^- - w^{ij} \delta + Iv + It = 0 \\
0 \leq y & \in R^{n-m} \\
0 \leq \delta & \in R \\
0 \leq v & \in R^k
\end{align*}
\]

The surplus vector \( t \) may be removed since we can always increase the objective function by setting \( t_i = 0 \). 

These two theorems provide single objective LPs that may be used to test whether a nonbasic variable is efficient with respect to \( B \). The only differences in the single objective LPs are the signs on the \( w^{ij} \delta \) terms of the constraints. If the nonbasic arc has zero flow, the sign will be positive. If the nonbasic arc has capacity flow, the sign will be negative.
Finally, if \( \overline{B} \) and \( \hat{B} \) are efficient bases where one may be obtained from the other by performing only efficient pivots, the \( \overline{B} \) and \( \hat{B} \) are \textit{connected}. By using the subproblem test in the Theorem 6.1 and pivoting among all of the efficient bases, we can identify all efficient extreme points of \( S \). This leads to the algorithm of Figure 7.1.

\subsection*{7.5 Step 1: Finding Initial Efficient Basis}

The definitions and theory of the preceding section provide a means of listing all efficient extreme points. From an initial efficient extreme point, we test for all adjacent efficient extreme points and add them to the list. Then, for each of the efficient extreme points in the list we again test for adjacent efficient extreme points and add any new ones found to the list. We continue in this manner until all efficient extreme points are found. Since all efficient extreme points of the efficient set are edge connected we are guaranteed to eventually find all efficient extreme points. To take advantage of this, however, we must find an initial efficient extreme point from which to start. In this section, we discuss methods for obtaining the initial efficient extreme point.

\subsubsection*{7.5.1 Initial Objective Function}

One method of finding an initial efficient extreme point is to form a single objective MCNF that is guaranteed to have a solution that is efficient and then solve the MCNF. The weighted sums LP provides the foundation for allowing us to do this. Remember that if \( \overline{x} \in S \) minimizes the weighted sums LP \( \min \{ \overline{\lambda} Cx \mid x \in S \} \) where \( \overline{\lambda} \in \Delta \), then \( \overline{x} \) is an efficient point. Here, \( \Delta = \left\{ \lambda \in \mathbb{R}^k \mid \lambda_i > 0, \sum_{i=1}^k \lambda_i = 1 \right\} \). To form the weighted sums LP for the MONF problem, we simply need to select an appropriate vector of weights \( \overline{\lambda} \in \Delta \).
We then multiply the weight vector by the criterion matrix $C$ to obtain the single objective for the weighted sums LP. If $S$ is bounded, the solution of the weighted sums LP will provide an initial efficient solution for the original MOLP.

Notice that the components of the weighting vector are strictly positive. This ensures that the single objective provides a gradient which points into the relative interior of the criterion cone. So we are free to choose any weighting vector $\lambda \in \Delta$. A simple option would be to choose a large weight for one of the objectives and relative small weights for the other objectives. We may then scale the weights to integer values, if we wish to ensure that the resulting criterion values are integer.

**Example 7.2:** For the MONF of Example 7.1, the criterion matrix $C$ is

$$
C = \begin{bmatrix}
2 & 3 & 3 & 1 & 1 & 2 & 3 \\
3 & 1 & 2 & 1 & 2 & 1 & 1 \\
1 & 2 & 1 & 1 & 3 & 1 & 3 \\
\end{bmatrix}.
$$

A weighting vector $\lambda \in \Delta$ that puts more weight on the first objective could be chosen to be

$$
\lambda = \begin{bmatrix}
.98 \\
.01 \\
.01 \\
\end{bmatrix}.
$$

We can scale the weighting vector so that all components are integer by multiplying the vector times a scaling value of $r = 100$.

$$
\lambda_r = r\lambda = 100 \begin{bmatrix}
.98 \\
.01 \\
.01 \\
\end{bmatrix} = \begin{bmatrix}
98 \\
1 \\
1 \\
\end{bmatrix}.
$$

We then calculate $\lambda_r C$ as
The network with this weighted sums objective is shown in Figure 7.3.

\[ \lambda_r^t \mathbf{C} = \begin{bmatrix} 2 & 3 & 3 & 1 & 1 & 1 & 2 & 3 \\ 3 & 1 & 2 & 1 & 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 1 & 3 & 1 & 3 & 1 \end{bmatrix} \]

\[ = \begin{bmatrix} 200 & 297 & 297 & 100 & 103 & 100 & 200 & 297 \end{bmatrix} \]

7.5.2 All-Artificial-Arcs Method

Once the weighted sums LP objective has been determined we may use any MCNF solution algorithm to find the optimal point which is also efficient. For the solver described later in this chapter, the *Simple-on-a-Graph* method (Kennington and Helgason (1980)), also called *Network Simplex* method, was chosen. The mechanics of the Network Simplex will be discussed in the next section, but in order to begin the method requires an initial feasible solution. One method for obtaining an initial feasible solution is known as the method of *All-Artificial-Arcs*.
Using the method of All-Artificial-Arcs is similar to the use of the artificial variables in the Simplex method to provide an initial basis in the tableau. First, an artificial node, $A$, is added to the network. This node will serve as a transshipment node in the initial basis. Then artificial directed arcs are added to the network. Costs and capacities are assigned to the artificial arcs in such a way that they are guaranteed not to appear in the final solution if the original network is feasible. Depending on how the costs are assigned, the MCNF may be solved using a method similar to the *Two-Phase* method of linear programming or a method similar to the *Big-M* method of linear programming (Bazaraa and Jarvis (1977)). Because it is fairly simple to implement, we will use the Big-M method. With this method, the costs for the artificial arcs are set very high so that these arcs are "driven out" by the lower cost arcs of the original network.

For each supply node $i$ with $b_i > 0$, construct an artificial arc from node $i$ to the artificial node. Set the arc flow and the arc capacity to $b_i$. Set the cost of the node to 0. For each transshipment node $i$ with $b_i = 0$, construct an artificial arc from the artificial node to node $i$. Set the arc flow to zero and the arc capacity to $\infty$. Set the arc cost to a high value relative to the arc costs of the original network. For each demand node $i$ with $b_i < 0$, construct an artificial arc from the artificial node to node $i$. Set the arc flow to $-b_i$ and the arc capacity to $b_i$. Set the arc cost to a high value relative to the arc costs of the original network.

**Example 7.3:** The network of Figure 7.3 with the addition of an artificial node and artificial arcs is shown in Figure 7.4.
7.5.3 The Network Simplex Method

In Chapter 3, it was shown how each basis for the MCNF corresponds to a spanning tree on the network. The Network Simplex algorithm utilizes this fact to allow pivoting between bases to be carried out on the network itself. As such, beginning with an initial feasible solution, the method maintains a feasible spanning tree structure at each step, and successively transforms it into an improved spanning tree structure until the network flow is optimal.

Two additional concepts are necessary to represent a tree structure, the root and an appropriate data structure. The root may be selected as any node in the network. For the majority of this work, node 1 will be the chosen as the root by default. The spanning
tree is usually drawn as hanging from the root. The root has the additional benefit as an artificial variable in the LP formulation. The addition of the root variable to the LP formulation makes the constraint matrix $A$ have full row rank (Ahuja, Magnanti, and Orlin (1993)). For any feasible solution the root variable will be basic with zero flow. Thus, any spanning tree in the network plus the root can represent a basic solution.

A data structure for a spanning tree must provide enough information to describe the tree in a format appropriate for computer storage. In addition, the data structure can affect the performance of the algorithm, so careful selection is required. Two pieces of information that are often stored and used to describe a spanning tree are the predecessor and the thread. Together, the predecessor and thread provide enough information to reconstruct a spanning tree. Each node in a spanning tree has a unique path from itself to the root. Starting at a given node and tracing the path back to the root, the first node encountered is the predecessor. The thread defines a sequence of nodes that walks or threads its way through the nodes of a tree, starting at the root node, and visiting nodes in a "top-to-bottom" order, and finally returning to the root. An example of a tree and its predecessor/thread data structure are provided in Figure 7.5.
The steps of the Network Simplex method are

**Step 1:** Determine an initial feasible tree structure.

**Step 2:** Let $x$ represent the vector of flows and calculate $\pi$, the vector of node potentials for the tree.

**Step 3:** If some nontree arc violates the optimality condition, go to Step 4. Otherwise, end with the optimal flows.

**Step 4:** Select an entering arc $(k, l)$ that is violating its optimality condition.

**Step 5:** Add arc $(k, l)$ to the tree and determine the leaving arc $(p, q)$.

**Step 6:** Update the tree and update the flow vector $x$ and the vector of node potentials, $\pi$. Go to Step 3.
For Step 1, the method of All-Artificial-Arcs determines an initial feasible spanning tree. The initial flow vector $\mathbf{x}$ has zero flow for all arcs in the original network and positive flow on artificial arcs directed into or out of supply/demand nodes.

At Step 2, the vector of node potentials is calculated. In Step 3, the arcs that are not in the current tree are checked to see if they may enter the basis. This is accomplished by calculating the reduced cost for each arc. If a nonbasic arc with zero flow has a nonnegative reduced cost, then it is a candidate for entering the basis. Also, if a nonbasic arc with flow at capacity has a nonpositive reduced cost, then it is a candidate for entering the basis. If no arcs satisfy these conditions, then the current basis is optimal and the algorithm stops.

There may be several arcs that are candidates for entering the spanning tree based on the conditions above. In Step 4, one of the arcs, $(k, l)$, is selected. There are several possible rules for selecting an entering arc $(k, l)$. The rule used here, known as Dantzig's pivot rule, says to select at each iteration the arc with the most violation of the optimality conditions to enter the tree.

In Step 5, the selected nonbasic arc is added to the tree forming a pivot cycle. When this is done, there is no longer a tree and an arc in the pivot cycle must be selected to leave. The selection of the leaving arc is conducted in a way similar to the ratio test of the LP Simplex method. If the entering arc $(k, l)$ was initially at its lower bound of 0, the pivot cycle flow will change in a manner that would increase the flow for arc $(k, l)$. If the entering arc $(k, l)$ was initially at its upper bound of $u_{kl}$, the pivot cycle flow will change in a manner that would decrease the flow for arc $(k, l)$. The leaving arc, $(p, q)$, is then selected as the arc which first reaches its upper or lower bound as the flow changes. It is
possible that the arc \((p, q)\) chosen to leave the basis is the same as the arc \((k, l)\) that was chosen to enter. When this occurs, the succeeding pivot will result in a tree that has the same basic arcs but different arc flows. The amount of flow change in the pivot cycle is denoted by \(\delta\), and is calculated based on the current flow of arc \((p, q)\) or

\[
\delta_{pq} = \begin{cases} 
  u_{pq} - x_{pq} & \text{if } x_{pq} \text{ increases} \\
  x_{pq} & \text{if } x_{pq} \text{ decreases}
\end{cases}
\]

In Step 6, the tree is updated by removing the leaving arc \((p, q)\), adjusting the flows and calculating the new node potentials.

**Example 7.4:** Calculations for solving the network of Figure 7.4 are shown in Figure 7.6.

The calculations begin with the initial spanning tree, in section 0 of Figure 7.6, consisting of the nodes and the artificial arcs. With the cost for arc \((A, 5)\) at 1000, the current value of the weighted sum objective function is \(z_{WS} = 5000\). The node potentials are indicated near each node.

The initial spanning tree includes none of the original network arcs so they are considered nonbasic. Currently, there is 0 flow on all nonbasic arcs. The reduced costs for the nonbasic arcs are shown in the table next to the spanning tree. Notice that for the initial spanning tree, all of the nonbasic arcs that begin at node one have positive reduced costs. Since the flow is zero for each of these arcs, all three are candidates to enter the tree. Arc \((1, 2)\) is selected since its reduced cost is greatest. The dashed arrow from node 1 to node 2 represents the arc to enter the tree.
Figure 7.6 - Iterating to Initial Efficient Extreme Point
When arc (1, 2) enters the tree, the flow should be increased in the pivot cycle formed by arc (1, 2) in a counterclockwise direction. Notice that arc (A, 2) currently has a zero flow. If there were a counterclockwise flow increase the flow on arc (A, 2) would drop below zero which is infeasible. Therefore, arc (A, 2) should leave the basis with $\delta = 0$. In this case, the entering arc became basic at its lower bound while the leaving arc became nonbasic at its lower bound. We did remove an artificial arc that no longer needs to be considered. The updated tree is shown in section 1 of the Figure 7.6.

Calculations continue in this way until the optimal spanning tree is reached. This is shown in section 5 of the Figure 7.6. Note that after the fourth pivot, in section 4, arc (1, 2) has a positive reduced cost but is not selected to enter the spanning tree in the next iteration. This is because during the fourth pivot the flow on this arc was adjusted to its upper bound of 4. Recall that an arc that is nonbasic at its upper bound does not satisfy the optimality condition if its reduced cost is negative.

The calculations end in section 5 of Figure 7.6 with the optimal spanning tree. The optimal flow vector is $x = [4 \ 1 \ 0 \ 0 \ 4 \ 0 \ 1 \ 0]$ and the value of the weighted sum objective function is $z_{WS} = 1709$.

### 7.6 Step 2: Storing Basis Codes

Each time that a new efficient basis is found it will need to be saved in a list. Each entry in the list will need to contain enough information to allow us to construct each spanning tree for the stored bases. For this, we will discuss a coding scheme that can store the specifications for each spanning tree efficiently. Also, the algorithm will need to search the list to see if a candidate basis already exists in the list, and it will need to keep
track of which bases have not been tested. An efficient data structure for these tasks will be presented.

### 7.6.1 Coding a Basis

For each basis, the primary information to store is which arcs included in the spanning tree corresponding to the basis. Since there may be many arcs in each basis and the total number of efficient bases may be large, an efficient method of storing and retrieving this information is needed. To store which arcs are included in a spanning tree we may use a binary-to-decimal coding scheme.

First, we need to assign to each arc an index value. If we list the arcs in a forward star manner as discussed in Chapter 4, then a natural indexing is to map each arc to the set of natural numbers. Then we can assign a binary value to each of the arcs to indicate whether it is included in the spanning tree. If the arc is included in the spanning tree, a 1 is assigned. Otherwise, a 0 is assigned to the arc. The result will provide a string of binary digits. If we begin our index values at 1, we will include 0 as a placeholder on the right end of the binary string.

While the coding will be stored as a binary string in the computer, it is convenient to think of the coding as its corresponding decimal value. Moving between the binary coding and the decimal coding is carried out using binary to decimal conversion techniques.

**Example 7.5:** The arcs of the initial efficient basis of Example 7.4 may be coded as follows. First, an index is assigned to each arc in the network. Then binary values are assigned to each arc depending on whether or not the arc is in the spanning tree. These
binary values along with a dummy 0 in the $2^0$ position are used to construct a string of binary digits.

<table>
<thead>
<tr>
<th>index</th>
<th>arc</th>
<th>basic code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$x_{12}$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$x_{13}$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>$x_{14}$</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>$x_{23}$</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>$x_{25}$</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>$x_{34}$</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>$x_{35}$</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>$x_{45}$</td>
<td>0</td>
</tr>
</tbody>
</table>

The binary code may be converted to a decimal value by remembering that each digit of the binary number stands for a power of two. The binary string for this spanning tree may be converted to a decimal value as

$$010101100 = 0 \times 2^8 + 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 = 172_{10}.$$  

We can convert the decimal value back to binary using successive division by 2. The remainder after each division will provide the binary digit.

### 7.6.2 Capacity Codes

For most MOLP problems, a simple coding of the basic variables would be sufficient for reconstructing a given basis. However, for network flow problems with arc capacities it is insufficient. This is because the nonbasic arcs on a network may be at 0 flow or at capacity flow. This makes it possible for the exact same arcs to be present in more than one basic solution.

**Example 7.6:** Consider the network and the two adjacent spanning trees in Figure 7.7. Both of the trees are feasible and adjacent. We can obtain spanning tree (2) from (1) by
adjusting flows along the cycle when arc (2, 4) enters the tree. The flow adjustment results in arc (2, 4) becoming nonbasic at its upper bound. Thus, we have two different basic solutions with the same basic arcs in the spanning tree.

![Figure 7.7 - A Network and Two Spanning Trees](image)

We can code and store the index of arcs that are nonbasic at capacity in the same way that we coded the basic variables. This will require additional storage and processing to keep track of these capacity codes.

### 7.6.3 List Data Structures

With the MONF, a list of efficient basic solutions is generated. As each new efficient basic solution is tested for efficient pivots, one or more adjacent basic solutions may be found. These bases can be identified with unique numerical codes based on the basic variables. If an adjacent efficient corner point solution is new then it should be added to the list. In order to determine if a solution under consideration is new, the current list needs to be searched.
Among the most effective of the techniques for improving the time of sorting and searching involves a data structure called a Binary Search Tree (BST) (Decker (1989)). Here we examine the use of a BST data structure and its adaptation for the MONF.

We have seen that a tree is a hierarchical data structure for which each element may have at most one predecessor but may have many successors. in the tree. A tree that can have at most \( k \) children per node is known as a \( k \)-ary tree. A tree that can have at most two children per node is called a 2-ary tree or more often a binary tree.

A main advantage of a binary tree data structure is that if the data modeled using the tree has an inherent linear order, then this order can be incorporated into the structure of the tree. A binary search tree is a binary tree with the property that the value stored at a node is greater than the values of data stored at any node in its left subtree, and smaller than that of any node in its right subtree. An example of a binary search tree is shown in Figure 7.8. Note a search for the data point coded 21 requires only three comparisons.

In general, the time required to perform a search on a binary search tree depends on the depth of the node for which one is searching. A binary tree of \( n \) nodes has a depth \( d \), with \( \log_2(n) < d \leq n - 1 \). In the best case then, a search on a binary search tree will take time \( \mathcal{O}(\log_2(n)) \) and in the worst case it will take time \( \mathcal{O}(n) \). In practice, it turns out that the depth of most binary trees is not too much worse than a depth proportional to \( \log_2(n) \) (Decker (1989)).
In computer implementations of binary search trees, the structure of the tree is usually specified using pointers. The tree is referred to via a pointer to its root node (which is nil if the tree is empty). Each node contains fields for the data and at least two pointers, one pointer each for the left and right children of the node. Again, if a pointer to a child is nil then no child exists in the respective direction from the node. When using a programming language, such as Fortran, which does not support pointers, a natural idea is to store the data as an array in which two of the array fields serve as pointers. A node will be represented by a cell containing the associated data, a cell for a pointer to the left child and a cell for the pointer to the right child. Pointers that are stored as array fields are often called *cursors*. The cursor representation for the binary search tree of Figure 7.8 is displayed in Figure 7.9.

Several operations are usually carried out on a binary search tree. These include: tree creation, searching for particular data, updating the data in a node, inserting data into
the tree, and deletion of data from the tree. For the MONF, we are basically building a binary search tree as new bases are found with no plans for changing or deleting the data once entered. Therefore, the MONF algorithm will require only three of these procedures: create, search, and insert. In addition, thread and predecessor pointers will be included in order to search the tree in a fashion that should minimize the number of pivots required when crashing between bases.

<table>
<thead>
<tr>
<th>Binary Tree Node</th>
<th>Data</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

Figure 7.9 - Cursor Representation of Binary Search Tree

**Example 7.7:** The initial efficient basis found earlier can be stored in the initial list data structure of Figure 6.10. Thread and predecessor pointers are discussed in the next sections.
7.7 Steps 3 through 7: Finding Adjacent Efficient Bases

After obtaining and recording an efficient spanning tree, we then determine if there are any adjacent feasible spanning trees. This is accomplished by using the reduced costs of the nonbasic arcs and the subproblem tests of Theorems 7.1 and 7.2. Each nonbasic arc $x_j$ is tested one-by-one to determine if $x_j$ is nonbasic efficient with respect to the current basis $B$.

When a variable $x_j$ is found to be nonbasic efficient, the leaving variable and change of flow are determined so that the basis and capacity codes may be calculated. A search of the master list is conducted. If the basis under consideration is not found in the list, it is new and is added to the list. Otherwise, it is discarded and testing continues with the next nonbasic variable.

Once all nonbasic variables for a spanning tree have been tested, we check the list to determine if there are any bases that have not yet been tested. If there are no more bases to test, the algorithm stops with all efficient bases found. If there are more to test, we then crash to the next basis. To crash to a basis means to obtain its spanning tree based on the stored coded values. The thread pointer in the data structure will be used to indicate which basis should be tested next.

As the search is carried out for all efficient bases of the feasible region, we can think of each basis found as a node on a search tree. The predecessor/thread data

<table>
<thead>
<tr>
<th>Index</th>
<th>Basis Code</th>
<th>Capacity Code</th>
<th>Left Pointer</th>
<th>Right Pointer</th>
<th>Thd</th>
<th>Pred</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>172</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
structure can help us keep track of how the search tree grows, where the search has already been, and which bases are yet to be tested. Predecessor and thread pointers were chosen in order to provide an efficient means of traversing the search tree. This traversal is carried out as we move to each basis in the list.

An example of how the predecessor/thread data structure is to be maintained is shown in Figure 7.11. Each child in the tree was found when its parent was tested for adjacent efficient bases. Maintaining the pointers in this way minimizes the average number of pivots during crashing. Moving down the tree, from a parent to a child, requires only one pivot. Moving back up the tree often only requires one or two pivots to reach an untested basis.

![Figure 7.11 - Search Tree and Data Structure](image)

The following example illustrates testing a basis for adjacent efficient bases.

**Example 7.8:** The initial efficient basis found in Example 7.4 and coded and stored in Examples 7.5 and 7.7, is shown in Figure 7.12.
Figure 7.12 - Initial Efficient Basis and Nonbasic Variable Reduced Costs

Using the reduced costs, we can now formulate the subproblem LP for the first nonbasic arc, \( x_{12} \), as

\[
\begin{align*}
\text{max:} & \quad v_1 + v_2 + v_3 = z \\
\text{st.} & \quad 2y_1 + 2y_2 + y_3 + y_4 - 2\delta + v_1 = 0 \\
& \quad -3y_1 + 2y_4 + 3\delta + v_2 = 0 \\
& \quad y_1 + y_2 + 2y_3 - 3y_4 - \delta + v_3 = 0 \\
& \quad y_i, v_i, \delta \geq 0 \text{ for all } i
\end{align*}
\]

The coefficients for this formulation are shown in a standard Simplex tableau in Table 7.1. Notice that since there are no positive reduced cost values for the nonbasic variables in the tableau, the current basis for the subproblem is optimal with an objective function value of \( z = 0 \). So, the nonbasic arc \( x_{12} \) is efficient with respect to the current basis.

Next, we must determine which arcs would leave the basis when arc (1, 2) enters. When arc (1, 2) enters the basis, it will form a pivot cycle that will include the basic arcs (1, 3), (3, 5), and (2, 5). Since arc (1, 2) currently has a capacity flow of \( x_{12} = 4 \), the flow would change in a clockwise direction as indicated in Figure 7.13.
Table 7.1: Simplex Tableau for Initial Efficient Extreme Point

<table>
<thead>
<tr>
<th>( c_i )</th>
<th>( y_1 )</th>
<th>( y_2 )</th>
<th>( y_3 )</th>
<th>( y_4 )</th>
<th>( \delta )</th>
<th>( y_1 )</th>
<th>( y_2 )</th>
<th>( y_3 )</th>
<th>RHS</th>
<th>Basic ( \pi ) Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>( \pi_1 ) = 1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>-2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \pi_1 ) = 1</td>
</tr>
<tr>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \pi_2 ) = 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>-3</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>( \pi_3 ) = 1</td>
</tr>
<tr>
<td>( z_i )</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( c_i - z_i )</td>
<td>0</td>
<td>-3</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Recalling the capacity values from Figure 6.2, we see that a flow change of \( \Delta = 2 \), both arc (1, 3) and arc (3, 5) reach their capacity of 3 units of flow. So either may leave the basis. In addition, arc (2, 5) is at capacity, so we may leave the flow as is, with \( \Delta = 0 \) and arc (2, 5) leaving the basis. The trees and their basis codes are shown in Figure 6.14.
Arc (1,3) leaves at capacity: Basis code = 170  
Cap. code = 4

Arc (3,5) leaves at capacity: Basis code = 46  
Cap. code = 128

Arc (2,5) leaves at capacity: Basis code = 142  
Cap. code = 32

Figure 7.14 - Efficient Bases Adjacent to Initial Efficient Point

The results of adding these new bases to our master list is shown in Figure 7.15.

<table>
<thead>
<tr>
<th>Index</th>
<th>Basis Code</th>
<th>Capacity Code</th>
<th>Left Pointer</th>
<th>Right Pointer</th>
<th>Thd</th>
<th>Pred</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>172</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>170</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>46</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>142</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 7.15 - The Master List After Testing First Nonbasic Variable at Initial Point

It turns out that after the testing is complete arc (4, 5) is efficient, while arcs (2, 3) and (3, 4) are not efficient with respect to the initial basis. After determining the new bases derived from the test for arc (4, 5), the Master List appears as in Figure 7.16.
<table>
<thead>
<tr>
<th>Index</th>
<th>Basis Code</th>
<th>Capacity Code</th>
<th>Left Pointer</th>
<th>Right Pointer</th>
<th>Thd</th>
<th>Pred</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>172</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>170</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>46</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>142</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>420</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>300</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>424</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 7.16 - The Master List After Completing Tests of Initial Point

7.8 Step 9: Crashing to Next Basis

The process of moving from one basis to another, in the fewest pivots possible, is known as *crashing*. Consider the search tree in Figure 7.17. Following the thread, it eventually becomes necessary to crash from basis 6 to basis 3. It is possible that these two bases are not adjacent. In this section, two strategies for crashing are described. Both strategies attempt to take advantage of the network structure of the problem.

Figure 7.17 - Crashing from Basis 6 to Basis 3
7.8.1 Pivoting on the Network Method

The objective of crashing is to move from one basic spanning tree for the network to another in the minimum number of pivots. Let unwanted arcs be those arcs in the current spanning tree that should not appear in the next spanning tree. Let wanted arcs be those arcs in the next spanning tree that do not appear in the current spanning tree.

Suppose that a crash is needed from current basis $B_1$ to the next basis $B_2$. We can define the unwanted arc set as $A_u = \{(i, j) \in A_{B_1} | (i, j) \notin A_{B_2}\}$ and the wanted arc set as $A_w = \{(i, j) \in A_{B_2} | (i, j) \notin A_{B_1}\}$. The number of unwanted arcs is equal to the number of wanted arcs since the number of arcs in any spanning tree for the network is constant.

One strategy that utilizes the network structure is to conduct a series of pivots on the network. Each pivot would pivot a wanted arc into the tree until the next basis is reached.

A crashing algorithm based on this idea is described as:

**Step 1:** Decode the basis and capacity codes for the next basis. Create a list of wanted arcs.

**Step 2:** If there exists a wanted arc that is not in the current basis, go to Step 3. Otherwise go to Step 6.

**Step 3:** Select a wanted arc from the list to enter the basis.

**Step 4:** Select an arc to leave the basis as in Network Simplex algorithm.

**Step 5:** Pivot and update the network as in Network Simplex algorithm.

**Step 6:** Stop.

**Example 7.9:** Suppose that we want to crash from the current basis to the next basis shown in Figure 7.18. Both of these are efficient bases for the network of Example 7.1.
This crash requires two pivots as shown in the sequence of Figure 7.19. From the current basis we can select wanted arc \((4, 5)\) to enter. For this arc to enter the tree, there needs to be a counterclockwise adjustment of flow. A flow change of 2 units in the pivot cycle would reduce arc \((1, 2)\) and arc \((2, 5)\) to zero flow, making them both candidates to leave the basis. Since, arc \((2, 5)\) is an unwanted arc it is removed and the tree is updated to the middle basis.

From the middle basis, wanted arc \((1, 3)\) is selected to enter. This requires a counterclockwise adjustment of flow in the pivot cycle. A flow adjustment of 1 unit will take arcs \((1, 4)\) and \((4, 5)\) to capacity making them both candidates for leaving the tree. Since, arc \((1, 4)\) is unwanted with respect to the next basis, it is removed. After the tree is updated we are at the next basis with all flows correct.
While this method is intuitive and allows for the use of Network Simplex routines for network adjustments, it suffers from one major flaw. Namely, there is no guarantee that the flows will be correct when the next basis is reached. Consider the following example.

**Example 7.10:** Suppose that we want to crash from the current basis to the next basis shown in Figure 7.20.
Note what could happen during this crash as illustrated in the pivot sequence shown in Figure 7.21. Here, wanted arc (1, 3) is first selected to enter the bases. A counterclockwise flow adjustment of 2 units will make unwanted arc (2, 5) nonbasic at a capacity of 4 units. After adjusting the network, the middle basis is obtained.

In the middle basis of Figure 7.21, the wanted arc (4, 5) is chosen to enter the basis. A counterclockwise flow adjustment of 1 unit will make unwanted arc (3, 5) nonbasic with 0 units of flow. After the network is updated, the third tree in Figure 7.21 is obtained.
Notice that while the correct arcs are present in the basis, the arc flows do not match that of the desired next basis. In addition, there is no clear way to adjust the flows with a simple pivot.

This problem complicates matters. Pivoting on the network is desirable in order to take advantage of the network structure. However, the present algorithm is not guaranteed to reach the desired basis with all correct flows. What is needed is another way to create a basis using only the information stored in the basis and capacity codes. An alternative is discussed next.

### 7.8.2 Triangularizing a Basis

Recall that the MONF also has its MOLP formulation. In particular, the divergence constraint equations may be written in matrix form as $Ax = b$, where $A$ is an $m \times n$ matrix, $x$ is an $n$-vector, and $b$ is an $n$-vector. When basic variables are chosen we may partition both the constraint matrix $A$ and the variable $x$ into two parts.
corresponding to the basic and nonbasic variables. Let $B$ be a $m \times m$ matrix consisting of the columns of the $A$-matrix which correspond to the basic variables. Let $N$ be a $m \times (n-m)$ matrix consisting of the columns of the $A$-matrix which correspond to the nonbasic variables. Then we may rewrite the matrix form of the constraint equations as $Bx_B + Nx_N = b$. Rearranging, we have $Bx_B = b +Nx_N$.

By definition, the columns of $B$ form a basis for the system of equations if and only if these columns are linearly independent. One problem with the standard form of LP constraints for network flow problems is that there are not $m$ columns of the constraint matrix that are linearly independent. So we cannot choose $m$ variables so that the matrix can have full row rank. One way to correct this problem is to add another column to the $A$ matrix which is linearly independent to all other columns and then make sure that this column is in the basis.

The standard method for adding this linearly independent column is to use the root node. To the root node is added an additional arc which is directed outward and does not connect to another node. This node will never carry flow but it will provide another equation to add to the balance equation of the root node. By keeping a variable corresponding to this arc in the basis at all time, we ensure that the basis always has full row rank.

**Example 7.11:** The matrix form of the constraint equations, with the addition of a root variable $r$ for the network of Example 1 is given by
For the basis with code 270 and capacity code 128 from Figure 7.20, the basis matrix $B$ and $N$ can be selected and the matrix equation may be rearranged in $Bx = b + NxN$ form as

$$
\begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
-1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & -1 & 0 & -1 & 0 & 1 & 1 & 0 \\
0 & 0 & -1 & 0 & 0 & -1 & 0 & 1 \\
0 & 0 & 0 & 0 & -1 & 0 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_{12} \\
x_{13} \\
x_{14} \\
x_{23} \\
x_{25} \\
x_{34} \\
x_{35} \\
x_{45} \\
r
\end{bmatrix} =
\begin{bmatrix}
5 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
$$

Since, the $B$ matrix has full row rank when the values of the nonbasic variables are known, the values of the basic variables may be calculated. Often, the $B$ matrix is not in a form that is amenable for making this calculation efficiently. It helps to rearrange the $B$ matrix so that it is triangular. This is called triangularizing the basis.

An algorithm based on that of Kennington and Helgason (1980) may be described as:

**Step 0:** Let $i = 1$. 
**Step 1:** Given a basis and its associated spanning tree, find an node $k$ that is an endpoint of the tree and let $e_s$ be the arc in the tree that is incident to the node. If the endpoint is the root, go to Step 4.

**Step 2:** Let the $i$th row of the $B$ matrix correspond to node $k$ and let the $i$th column of $B$ correspond to arc $e_s$.

**Step 3:** Reduce the tree by removing the node $k$ and the incident arc $e_s$ and go to step 1.

**Step 4:** Let the $n$th row of the $B$ matrix correspond to the root and let the $n$th column of $B$ correspond to $e^n$.

**Example 7.12:** The next basis spanning tree with basis code = 270 and capacity code = 128 is repeated in Figure 7.22.

![Figure 7.22 - Spanning Tree for Next Basis](image)

Choosing the endpoints of the tree in the order of 2-3-5-4 endpoint we have the following matrix equation after triangularizing the basis
With the flows on nonbasic arcs $x_{23} = 0$, $x_{25} = 4$, $x_{34} = 0$, and $x_{35} = 0$ (known because of the capacity code), we can calculate flows on the basic arcs in the following order:

\[-x_{12} = 0 - x_{23} - x_{25} = 0 - 0 - 4 \Rightarrow x_{12} = 4\]
\[-x_{13} = 0 + x_{23} - x_{34} - x_{35} = 0 + 0 + 0 + 0 \Rightarrow x_{13} = 0\]
\[-x_{45} = -5 + x_{25} + x_{35} = -5 + 4 + 0 \Rightarrow x_{45} = 1\]
\[x_{45} - x_{14} = 0 - x_{34} = 0 - 0 \Rightarrow x_{14} = x_{45} = 1\]
\[x_{12} + x_{13} + x_{14} + r = 5 \Rightarrow r = 5 - 4 - 0 - 1 = 0\]

### 7.9 MONFSOLV Software and Features

The program MONFSOLV was developed to search for and count, all efficient extreme points and all nondominated extreme points for MONF problems. Written in Fortran, MONFSOLV combines vector maximum theory with Network Simplex theory to provide a tool for conducting further research into the nature of the efficient and nondominated sets for MONF. Testing described in the next section demonstrates that by taking advantage of the network structure, the MONFSOLV software is able to enumerate the efficient set more quickly than the more general MOLP solver ADBASE (Steuer (1995)). A users’ manual for MONFSOLV is provided in Appendix A.

MONFSOLV provides several features to aid in a study of the efficient and nondominated sets. These include:

- **Random problem generation:** MONFSOLV provides a multiple objective minimum cost network random problem generator. The generator is an extension of a popular
single objective network flow generator, NETGEN, by Klingman, Napier, and Stutz (1974). The user may specify parameters for the network which determine the number and type of nodes, the total supply, the total number of arcs, the capacity range on the arcs, the percentage of arcs with capacity at the maximum value in the specified range, the number of objectives, the range of values for objective function coefficients (indirectly specifies size of the criterion cone), and the proportion of arcs to get the maximum value in the range of costs.

- **Two input formats:** In addition to random problems, MONFSOLV will work with existing problems that are stored in one of two different file formats. First, the MONFSOLV format is typical of the standard format for MCNF solvers. This format incorporates a node list and an arc list to specify problem parameters. The second input file format uses the ADBASE input file specification.

- **Several Output options:** The user of MONFSOLV may choose to obtain one or more of the following items in the output files:
  
  - **Problem specifications** - This includes a listing of the run options chosen and the random problem generator specifications.
  
  - **Network specification** - The program can save the network problem parameters in either MONFSOLV or ADBASE format. This is useful for saving the problem parameters for a randomly generated MONF or for conversion of between MONFSOLV and ADBASE file formats.
  
  - **Efficient bases** - Prints all criterion and flow values for each efficient basis.
  
  - **Nondominated vectors** - A listing of all nondominated extreme points.
Cumulative data and statistics - Prints the number of efficient extreme points, the number of nondominated extreme points, and solver performance measures.

Listing of the basis and capacity codes - Prints the master listing containing coded bases and their nonbasic arcs at capacity.

- User specified weights - The user may specify weights to use in place of default values when calculating the weighted sums objective for the initial efficient extreme point.

- SONF mode - The solver may be used to solve single objective MCNF problems.

- Degenerate pivot option - The user specifies whether or not the solver will add to the master list adjacent efficient solutions with no flow change.

- Crashing option - Two crashing options are available. The first uses pivoting on the graph backed up by triangularizing the basis. If cycling or incorrect flows are obtained through pivoting, the next basis is obtained through triangularizing the basis. The second crash option uses triangularizing, exclusively.

The current version of MONFSOLV is capable of solving multiple objective network flow problems with up to 300,000 efficient bases.

7.10 Empirical Results

As a test of the efficacy of MONFSOLV and a demonstration of its usefulness for research, an experiment was designed and conducted. A description of the experiment and its results are described in this section.

7.10.1 Experimental Design and Data
For this experiment, the random problem generator of MONFSOLV was used to create 45 problems. The problem parameters, which include the number of nodes, number of arcs, and number of objectives, varied based on a factorial design using the values listed in Table 7.2. The experimentation was performed on the SGI Origin Complex, an SGI rack-mount Origin 2000 with 24×300 MHz MIPS R12000 processors with 4MB cache memory and 8 GB of system memory, at the University of Georgia. In addition to the parameters shown in the table, the range of values for the objective function coefficients were chosen so that the size of the criterion cone varied randomly between 10° and 100°, based on measurement of the Average Cone Angle. Three replications were obtained for each of the problem parameter combinations shown in Table 7.2. When referring to the problem parameters in the text, we will denote by \( n \times m \times k \) a problem with \( n \) nodes, \( m \) arcs and \( k \) objectives.

Table 7.2 - Experimental Problem Parameter Values

<table>
<thead>
<tr>
<th>Combination Number</th>
<th>Nodes ((n))</th>
<th>Arcs ((m))</th>
<th>Objectives ((k))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>40</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>40</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>30</td>
<td>50</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>30</td>
<td>50</td>
<td>6</td>
</tr>
<tr>
<td>13</td>
<td>30</td>
<td>60</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>30</td>
<td>60</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>30</td>
<td>60</td>
<td>6</td>
</tr>
</tbody>
</table>
Each of the 45 random problems obtained were solved for all efficient extreme points and all nondominated extreme points in five different ways. The first four methods utilized various MONFSOLV run options as described in the next section. The fifth utilized the ADBASE software, which is a well-known solver for general MOLP, by Steuer (1989). These last results were obtained in order to compare the run times for the two programs. Of the 45 total problems, two resulted in a total number of efficient extreme points that is too large for the current array structure of ADBASE. These two data points are used in all analyses that are related to MONFSOLV alone. They are discarded for the run time comparison with ADBASE. The sizes of the efficient sets for the random problems as measured by the total number of efficient extreme points ranged from 4 to 349,321. As none of the networks are considered very large compared to those used in practice, it can be seen that the size of the efficient set may get large very quickly relative to the size of the problem.

7.10.2 Comparison of MONFSOLV Run Options

There are two run options for MONFSOLV for which it was uncertain how the program would perform best. The first is how the program should react to adjacent efficient basis that would result from "degenerate" pivots. These are pivots for which arcs are exchanged in the spanning trees but there is no flow change. One option, denoted D1, is allows these points to be added to the master list when they are found. The second option, denoted D2, is to only add adjacent efficient basis to the list that result in a
network flow change. It is thought that not adding them to the list might improve solution
time but may risk missing some points.

The second run option of interest is related to the method of crashing. Recall that
earlier we discussed to possibilities for crashing. One option, denoted C1 is to perform all
.crash pivots on the network unless cycling or incorrect flows result. When cycling or
incorrect flows are obtained, the next basis would be obtained using triangularization. For
this option, the primary crash method is pivoting with triangularization used only for
backup. The second crash option, denoted C2, would be to use triangularization
exclusively. The goal here is to determine if one method is more efficient then the other in
terms of solution time.

Using MONFSOLV, solutions were obtained for each of the four combinations of
crash and degeneracy options. All combinations worked in that they found the exact same
number of efficient and nondominated extreme points. The Table 7.3 presents the average
of the CPU time in seconds for each combination.

<table>
<thead>
<tr>
<th></th>
<th>D1</th>
<th>D2</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>52.60</td>
<td>42.73</td>
</tr>
<tr>
<td>C2</td>
<td>53.25</td>
<td>43.85</td>
</tr>
</tbody>
</table>

A factor effects chart for these values is presented in Figure 7.23. The chart
indicates very little change in CPU time contributed by the choice of crashing method.
There is some improvement however due to the method of handling degenerate adjacent
bases.
185

Figure 7.23 – Experimental Data


The results of a balanced ANOVA to check whether or not these factors are significant effecting CPU is presented in Table 7.4. The model employed used the CPU time as the response and the factors network size, number of objectives, degeneracy mode, and crash mode. The size of the network relates to the combinations of nodes and arcs. As seen in Table 7.2 there are 6 different combinations of node and arc values. Each combination was assigned a factor level for the ANOVA. Similarly, there were three factor levels for the number of objectives, and two each for the degeneracy and crash method factors. A simple additive linear model was assumed with no interactive effects.

Here we focus on the p-values, which can be used to indicate whether or not a factor's effect on the response is significant. The p-values test the hypothesis that the average response is equal for all levels of a given factor. For a given confidence coefficient \( \alpha \), a p-value \( \leq \alpha \) indicates that the averages are not equal and the factor's effect on the response variable is significant. The ANOVA shows that the factors of network size and number of objectives are significant at all levels of confidence (p-value = 0.000). However, the method of crashing factor is not significant for any reasonable value of \( \alpha \). The degeneracy method factor does not appear to have a great effect, but it is significant for an \( \alpha = 0.33 \). This implies that we may have 67% confidence that the factor is significant.

From this we may conclude that the degeneracy method D2, to only add adjacent efficient basis to the list that result in a network flow change, provides the best performance in terms of CPU time. Even given the relatively low confidence in the degeneracy factors effect on CPU time, this is not a risky conclusion since using the
program solves the problems correctly regardless of the mode employed. For all further tests, the output results from runs conducted using the C1-D2 combination. In other words, we will perform the runs using the pivoting method with triangularization backup to crash while not adding degenerate pivots to the master list.

Table 7.4 - Analysis of Variance for CPU Time with Run Mode Factors

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>SS</th>
<th>MS</th>
<th>F</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Size</td>
<td>4</td>
<td>301363</td>
<td>72341</td>
<td>17.75</td>
<td>0.000</td>
</tr>
<tr>
<td>Number of Objectives</td>
<td>2</td>
<td>193439</td>
<td>96720</td>
<td>22.79</td>
<td>0.000</td>
</tr>
<tr>
<td>Degeneracy Method</td>
<td>1</td>
<td>4180</td>
<td>4180</td>
<td>0.98</td>
<td>0.322</td>
</tr>
<tr>
<td>Crash Method</td>
<td>1</td>
<td>36</td>
<td>36</td>
<td>0.01</td>
<td>0.927</td>
</tr>
<tr>
<td>Error</td>
<td>171</td>
<td>725793</td>
<td>4244</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>179</td>
<td>1224811</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.24 - Factor Effects Chart of CPU Time

7.10.3 MONFSOLV vs. ADBASE CPU Times
The next question to consider is whether or not MONFSOLV, which is specialized for network flow models, realizes improvement in solution time as compared to ADBASE, which is a general MOLP solver. At first glance, the answer is yes. Recall, that the sample size here is 43 because there are two less CPU data points due to an ADBASE array size constraint. The average solution time for ADBASE is 60.50 seconds compared to an average of 39.04 seconds for MONFSOLV. If the difference is statistically significant, this is an estimated 35% improvement in terms of CPU time. An ANOVA table taking into account the problem size, the number of objectives and the use of the two different programs is provided in Table 7.5. The p-value of 0.063 indicates that the difference in factor level means due to the use of the two different programs is significant for an $\alpha = 0.10$, or with 90% confidence.

Table 7.5 - Analysis of Variance for CPU Time with Program Factor

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>SS</th>
<th>MS</th>
<th>F</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of Network</td>
<td>4</td>
<td>233975</td>
<td>58494</td>
<td>7.36</td>
<td>0.000</td>
</tr>
<tr>
<td>Number of Objectives</td>
<td>2</td>
<td>133057</td>
<td>66528</td>
<td>8.37</td>
<td>0.000</td>
</tr>
<tr>
<td>Program Used</td>
<td>1</td>
<td>27702</td>
<td>27702</td>
<td>3.49</td>
<td>0.063</td>
</tr>
<tr>
<td>Error</td>
<td>80</td>
<td>635756</td>
<td>7947</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>87</td>
<td>1030490</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7.10.4 Relationship of Output Results to Problem Size Parameters

The ANOVA results indicate that the numbers of nodes and arcs in the network affects the size of the efficient set. Also indicated is a significant effect from the number of objectives. In the chart of Figure 7.25 the total number of efficient extreme points is
plotted against the size categories. As shown in Table 7.2, there are five different combinations of total number of nodes and arcs. On the chart these have been coded 1 through 5 in increasing order. Here we can see that both the average size of the efficient set and the variation in efficient size increase as the network size increases.

The different shapes of the data points on the chart indicate the number of objectives for the problems. With the exception of one outlier, the size of the efficient set tends to increase as the number of objectives increase.

Figure 7.25 - Total Efficient Extreme Points vs. Network Size Category

Recall that another fact that would seem to affect the size of the efficient set is the size of the criterion cone. To measure this, ADBASE calculates an average cone angle for the criterion cone. The average cone angle is calculated by first taking the objectives a
pair at a time and calculating the angle between them. These angles are then averaged and
the average is used as an indicator of the size of the cone.

A plot of the total number of efficient points versus the average cone angle is
presented in Figure 7.26. In this chart, the size of the efficient set and variation in the set
sizes tend to increase with increasing average cone angle up to a point, then drops off.
Further study of the size of the efficient set in relation to problem parameters for the
general MOLP will be described in Chapter 9.

![Efficient Extreme Points vs. Average Criterion Cone Angle](image)

Figure 7.26 - Total Efficient Extreme Points vs. Average Criterion Cone Angle

**7.10.5 Number of Efficient Extreme Points vs. Number of Nondominated Points**

Another interesting comparison is between the sizes of the efficient and
nondominated sets of extreme points. In most cases, the total number of efficient extreme
points is large compared to the total number of nondominated extreme points. On average
the number of nondominated extreme points is 3.5\% of the number of efficient extreme points. The difference becomes more pronounced for larger sets of efficient extreme points. For problems with over 1000 efficient extreme points the average number of nondominated extreme points as a percentage of the number of efficient extreme points drops to 0.26\%.

7.11 Conclusion

The tests show that MONFSOLV is capable of listing all efficient and nondominated extreme points for the MONF. Several conclusions may be drawn from the test results.

- MONFSOLV works best when adding to the master list only those adjacent efficient bases resulting from a nondegenerate pivot with flow change. While the results presented do not indicate any problems, this must be used with caution since there is a possibility that some efficient points could be missed.

- The MONFSOLV software requires less CPU time than ADBASE when solving the same MONF. The time savings are due to the specialized nature of MONFSOLV for minimum cost network formulations. However, the time savings are small relative to the time savings of single objective network flow solvers over general-purpose LP solvers. While MONFSOLV can probably be made to run faster as it is improved, it will probably never come close to the hundred times improvement of single objective network solvers. While the single objective MCNF solver carries out all optimization steps on the network, this is not the case for MONFSOLV. Network operations only make up a portion of the work required, namely finding the initial efficient basis and
then crashing to other efficient bases. Another large part of the work involved is in testing each efficient basis. Consider a MONF with 30 nodes and 60 arcs. At each efficient basis there will be 29 basic arcs and 31 nonbasic arcs. This means that at each basis 31 subproblem tests will need to be carried out, one for each of the nonbasic arcs. Due to the structure of the subproblem test, pivots are not carried out on the network. If the problem has 1000 efficient extreme points, then there will be a total of 31,000 subproblem tests. This is compared to the 1000 uses of network optimization features, one for finding the initial efficient basis and one for each subsequent crash. Further improvements to MONFSOLV should focus on improving the speed of the subproblem test or reducing the number of subproblem tests required.

- The size of the MONF, in terms of number of nodes, number of arcs, number of objectives, and the criterion cone, does affect the sizes of the sets of efficient and nondominated extreme points. As might be expected, the sizes of the sets tend to increase as the problem sizes increase but the variation in set sizes increase as well. Further study of the nature of this relationship will be explored in Chapter 8.

- There is a large difference between size of the set of efficient extreme points and the size of the set of nondominated extreme points for MONF. Two causes for this come to mind. First, the differences could be due to degeneracy. Network flow formulations are notorious for their large amounts of degenerate solutions. Another cause may lie in the compression effects that occur when the efficient set from $n$-space is mapped to the lower dimensional $k$-space of criterion space.

With this initial study, MONFSOLV has proven itself to be a capable tool in studying the MONF problem. It should prove valuable as further research is conducted.
CHAPTER 8

AN INTERACTIVE SOLUTION METHOD FOR MONF

The primary motivation for understanding mathematical models such as MONF is the hope of applying them as aids in decision making. As we have seen, a complication that arises for models with multiple objectives is that there is usually no single solution that optimizes all objectives. For such models, we attempt to aid the DM in searching for the “best” or most preferred solution in the face of the trade-offs that must be made between the competing objectives.

In this chapter, we present a method for providing decision aid when the MONF problem is applicable. The method employs a sophisticated method of MOLP decision aid, the Interactive Weighted Tchebycheff method. In addition, the method attempts to exploit the special structure of the network formulation in order to improve the runtime efficiency of the computational phases of the method. The chapter begins with motivation for interactive sampling methods when working with multiple objectives in general and the MONF in particular. Then, supporting theory for the algorithm is presented. Finally, a software implementation of the method is described along with examples and comments.

8.1 Motivation for an Interactive Sampling Method

For any MOLP, including the MONF problem, the DM’s most preferred solution in terms of criteria space will reside in the nondominated set. This is because for any
point that is dominated there will be a point which is at least as good in terms of all objectives and better for at least one objective. So the nondominated set in criteria space and its associated efficient set in decision space represent all points which are “candidates” for the DM’s most preferred solution. Since the nondominated and efficient sets are related, we can carry out our search in one or both of these sets.

One approach to solving the MOLP is to enumerate the entire nondominated set, present it to the DM, and let the DM choose the best solution. Once, the best nondominated solution is chosen, the values of the decision variables for a corresponding efficient point may be found. A considerable problem with this approach is that, as we have seen, the nondominated set can be too large for the DM to evaluate effectively.

An alternative is to sample the nondominated set in some way. With sampling, we can present only a “good” subset of the nondominated set to the DM from which to choose. For this approach to work, we would need some way to generate the sample. Unfortunately, an single sample we could generate could not be guaranteed to include the overall “best” solution.

The more sophisticated approaches to solving the MOLP utilize sampling in an iterative fashion. They present a sample of nondominated points to the DM and solicit information about the most preferred solution in the sample. This information is then used to generate another sample of nondominated points in a progressively smaller neighborhood around the preferred point. Since these approaches require interaction with the DM in order to narrow the samples presented, they have come to be known as interactive procedures.
Research and presentation of interactive MOLP methods have been carried out by Benayoun, De Montgolfier, Tergny, and Larichev (1971), Geoffrion, Dyer, and Feinberg (1972), Zionts and Wallenius (1980), Steuer (1977), Steuer (1980), Steuer and Choo (1983), and Korhonen and Laakso (1996) among others. Surveys which include descriptions of several of these interactive MOLP procedures are presented in Steuer (1989) and Vinke (1992). Not only does the sampling techniques used in interactive methods provide for efficient searches but the interaction between the DM and the analysts typically provides a deeper understanding of the problem itself. Interactive procedures have turned out to be the most successful of the methods employed in solving MOLP. Ali (1990), Calvete and Mateo (1995a and 1995b), Ehrgott (1999) and Sun (2000) have conducted research in applying interactive methods to network flow problems.

8.2 Theoretical Background

One popular interactive methods is the Interactive Weighted Tchebycheff methods. In the next few sections, this method and related theory will be discussed. In particular, the attractive features of each method and their limitations will be described. In addition, we will look at the possibility of exploiting the network structure within this method.

8.2.1 Tchebycheff Theory

The weighted sums approach to solving an MOLP was introduced in Chapter 2. Each time a weighted sums program is solved, it is convenient to think in terms of probing the feasible region. In other words, we can think of the gradient of the current weighted sums objective as a half-ray, anchored at the origin and pointing in the direction of
increasing objective function value. Along this half-ray and perpendicular to it, we can picture a set of hyperplanes. Each hyperplane represents a set of points in the solution space with identical objective function values. These hyperplanes are known as level curves or contours. A level curve of a linear expression is the locus of points which satisfies a given RHS value. When minimizing an objective, we are looking for the point in the feasible region with the lowest valued level curve.

Level curves in criterion space are illustrated in Figure 8.2. In this figure, we see a feasible region in criterion space and a weighted sums minimization objective $\bar{z}$. The nondominated set consists of all points on the boundary of the feasible region between $z^1$ and $z^2$, between $z^2$ and $z^3$, and between $z^3$ and $z^4$, inclusive. Level curves for $z$ are shown for $z = 2$, $z = 3$, and $z = 4$. Here, the lowest valued level curve that intersects the feasible region has a value of $\bar{z} = 3$ and intersects at extreme point $z^2$. For this example,
weight vectors may be found which would result in either of \( z^1, z^2, z^3, \) or \( z^4 \) as providing the minimum weighted sums objective value in the feasible region.

The level curves for the weighted sums probes are convex sets, and the intersection with the feasible region will also be convex sets. Thus, these types of probes will identify only nondominated extreme points or entire nondominated \( f \)-facets. If the intersection with the feasible region is an efficient \( f \)-facet, the weighted sums probe considers all to have the same objective function value when in reality each point in the \( f \)-facet will correspond to a different criteria vector for the original problem. Thus, the probes of the weighted sums are not good at identifying nonextreme nondominated solutions. In addition, for the integer case the level curves would intersect with only those points which lie on the convex hull of the feasible region. So, unsupported nondominated points would not even be recognized as alternative solutions. Clearly, if it is desired to be able to sample the entire nondominated set and not just nondominated extreme points, then a different type of probe is needed.

One probe, based on the Augmented Weighted Tchebycheff Metric, has been used successfully in several interactive MOLP procedures (Steuer (1989)). Recall from Chapter 2, The family of weighted \( L_p \)-metrics is given by

\[
\|x - y\|_p^\lambda = \left[ \sum_{i=1}^n \lambda_i \left( |x_i - y_i| \right)^p \right]^{1/p}, \quad p \in \{1, 2, 3, \ldots\} \cup \{\infty\}.
\]

A weighted \( L_p \)-metric is used as a distance measure between the vectors \( x \) and \( y \) with different weights assigned to different vector components. A component \( i \) with a higher weight will have more impact on the value of the metric than one with a lower value. The value of \( p \) determines how the metric is calculated.
The Weighted Tchebycheff metric is the weighted $L_p$-metric with $p = \infty$. With a given vector $\mathbf{z}^{**}$ and $\lambda \in \Lambda = \left\{ \lambda \in \mathbb{R}^k \mid \lambda_i \geq 0, \sum_{i=1}^{k} \lambda_i = 1 \right\}$, 

$$
\|\mathbf{z} - \mathbf{z}^{**}\|^{\lambda}_{\infty} = \max_{i=1,\ldots,k} \left| \mathbf{z}_i - \mathbf{z}_i^{**} \right|
$$

is a member of the family of weighted Tchebycheff metrics that measure the distance between $\mathbf{z} \in \mathbb{R}^k$ and $\mathbf{z}^{**}$. The augmented weighted Tchebycheff metric is defined to be

$$
\|\mathbf{z} - \mathbf{z}^{**}\|^{\lambda}_{\infty} = \|\mathbf{z} - \mathbf{z}^{**}\|^{\lambda}_{\infty} + \rho \sum_{i=1}^{k} \left| \mathbf{z}_i - \mathbf{z}_i^{**} \right|
$$

where $\rho$ is a sufficiently small number. Contours of the weighted Tchebycheff and augmented weighted Tchebycheff metric are shown in Figure 8.3 for a vector $\mathbf{z}^{**}$.

![Figure 8.3 - Weighted Tchebycheff and Augmented Weighted Tchebycheff Contours](image_url)

With $\mathbf{z}^{**}$ as the reference point, the dashed rectangle represents a weighted Tchebycheff contour. In other words, all points that lie on the boundary of the dashed rectangle are the same distance from $\mathbf{z}^{**}$ according to a weighted Tchebycheff metric.
Similarly, the octagon defined by the solid line represents an augmented weighted Tchebycheff contour. All points that lie on the boundary of the solid-lined octagon are the same distance from \( z^{**} \) according to an augmented weighted Tchebycheff metric. The amount of expansion in the contour from the rectangle to the octagon is determined by the value of \( \rho \). Since \( \rho \) is usually chosen to be small, the amount of augmentation in the figure is exaggerated. Steuer (1989) provides the following definitions. They are modified here for the MOLP with minimization objectives.

Let \( z^{**} \leq z \) and \( \lambda \in \Lambda \). Then, \( z \) is a \textit{definition point} of the \( \|z - z^{**}\|_\infty^\lambda \) contours if and only if

\[
\lambda_i = \begin{cases} 
1 & \text{if } \bar{z}_i \neq z_i^{**} \text{ for all } i \\
1 & \text{if } \bar{z}_i = z_i^{**} \\
0 & \text{if } \bar{z}_i \neq z_i^{**} \text{ but } \exists j \in \mathbb{R} \text{ s.t. } \bar{z}_j = z_j^{**}
\end{cases}
\]

Let \( z^{**} \leq z \). Then \( z \) is a \textit{vertex} of a given \( \|z - z^{**}\|_\infty^k \) or \( \|z - z^{**}\|_\infty^k \) contour if and only if \( z \) is an extreme point of the closed convex set in \( \mathbb{R}^k \) whose boundary is the contour.

Let \( z \in \mathbb{R}^k \) be a definition point of a \( \|z - z^{**}\|_\infty^k \) or \( \|z - z^{**}\|_\infty^k \) contour. Then, the line segment connecting \( z^{**} \) to \( z \) is the \textit{diagonal} of the contour.

In Figure 8.3, \( z \) is both a vertex and a definition point for both contours. The arrow from \( z^{**} \) to \( z \) is the diagonal for both of the contours. The vector \( z^{**} \) is usually chosen as the criterion vector of the MOLP discussed in Chapter 2. The manner in which \( z^{**} \) is constructed for minimization objectives means that only points on the upper right
portion of the contour will intersect the feasible region. Figure 8.3 illustrates how this
portion of the augmented weighted Tchebycheff contour can be extended in the direction
of the diagonal to probe the feasible region.

To implement the probing of the feasible region $Z$ in criterion space, we have the

**augmented weighted Tchebycheff program**

$$
\begin{align*}
\min \quad & \left\{ \alpha + \rho \sum_{i=1}^{k} (z_i - z_i**) \right\} \\
\text{s.t.} \quad & \alpha \geq \lambda (z_i - z_i**) \quad 1 \leq i \leq k \\
& f_i(x) = z_i \quad 1 \leq i \leq k \\
& x \in S
\end{align*}
$$

The solution for this program will be a vector $(\bar{x}, \bar{z}, \bar{z})$ with $\bar{z}$ the closest feasible point in
criterion space and $\bar{x}$ is the corresponding image in decision space. The variable $\alpha$ is used
to minimize the maximum weighted distance between $z_i$ and $z_i**$. Steuer (1989) shows
that the solution of the augmented weighted Tchebycheff program will be nondominated.

Figure 8.4 shows how the augmented weighted Tchebycheff program may be used
to probe the feasible region in criterion space. Here, $\bar{z}$ is a nondominated solution in the
interior of an efficient edge. In Figure 8.5 the augmented weighted Tchebycheff program
intersects an unsupported nondominated criterion vector for a feasible region restricted to
integer values. The dashed line in Figure 8.5 represents the convex hull of the feasible
region. Notice also that the contour of the metric does not necessarily intersect the
feasible region at the vertex.
8.2.2 The Interactive Weighted Tchebycheff Method for MOLP

The Interactive Weighted Tchebycheff method (IWT) follows a progressive probing strategy. At each iteration, samples of progressively smaller subsets of nondominated points are presented to the decision maker. These samples consist of $P$
representative points that are generated using the augmented weighted Tchebycheff program. The subsets of the nondominated region are indirectly determined by systematic reductions of weight space. It should be noted that with IWT, the weights have no bearing on the relative importance of the objectives.

A pseudocode for the IWT is presented in Figure 8.6. The procedure is described as follows.

Step 1: Specify the sample size $P$ of nondominated vectors to present to the DM on each iteration; the expected number of iterations $t$; and $r$, a $\Lambda$-reduction factor that determines by how much the weight space is reduced on each iteration.

Step 2: Obtain a payoff table and calculate $z^{\text{min}}$ and $z^{\text{max}}$. Obtain a $z^{**}$ ideal criterion vector.

Step 3: Normalize the objective functions.

Step 4: Set the iteration counter $h$ to 0 and set $l_i^1 = 0$ and $u_i^1 = 0$, for all $i$.

Step 5: Increment the $h = h + 1$; and create the weight space

$$\Lambda^h = \left\{ \lambda \in R^k \left| \lambda_i \in [l_i^h, u_i^h], \sum_{i=1}^k \lambda_i = 1 \right. \right\}.$$

Step 6: Obtain $50k$ randomly generated weight vectors from $\Lambda^h$.

Step 7: Select the $2P$ most dispersed weighting vectors from the $50k$.

Step 8: Solve each of the $2P$ augmented weighted Tchebycheff programs that are associated with the $2P$ weighting vectors.

Step 9: Reduce the number of criterion vectors to present to the DM to the $P$ most different and present them to the DM.
algorithm IWT
begin
  specify the sample size \( P \), the number of iterations \( t \); and a \( \Lambda \)-reduction factor \( r \);
  construct a payoff table by minimizing each criterion separately over \( Z \);
  let \( z_i^* = z_g \) of the payoff matrix for \( i = 1, \ldots, k \) and let \( z_i^{**} = z_i^* - \varepsilon_i; \varepsilon_i > 0 \);
  for \( i = 1 \) to \( k \) do
      normalize objective \( k \);
      let \( k_i = 0 \) and let \( \mu_k = 1 \);
  end;
  let \( k = 0 \),
  while \( k < t \) and DM wishes to continue do
    let \( k = k + 1 \);
    let \( \Lambda^k = \{ \lambda \in \mathbb{R}^k \mid \lambda_i \in \mathcal{P}(\sum^k_{i=1} \lambda_i = 1) \} \);
    for \( j = 1 \) to \( 50k \) do let \( \mathcal{A} \) be a randomly chosen \( \lambda \in \Lambda^k \);
    choose the \( 2P \) most different weight vectors from the \( 50k \) weight vectors;
    for \( j = 1 \) to \( 2P \) do let \( x, \tilde{x} \) solve augmented weighted Tchebycheff program
      \[
      \min \left\{ \alpha + \rho \sum_{i=1}^k \left( z_i^{**} - z_i^* \right) \left| \alpha \geq \lambda_i \left( z_i^{**} - z_i^* \right); f_i(x) = z_i; x \in S \right. \right\};
      \]
    choose the \( P \) most different criterion vectors from the \( 2P \) LP solutions;
    let \( z^k = \) the DMs most preferred vector from the \( P \) solutions;
    for \( i = 1 \) to \( k \) do
      if \( z_i^k = z_i^{**} \) for all \( i \) then let \( \lambda_i^k = \frac{1}{\sum_{i=1}^k (z_i^{**} - z_i^k)} \);
      if \( z_i^k = z_i^{**} \) then let \( \lambda_i^k = 1 \);
      if \( z_i^k \neq z_i^{**} \) but there exists a \( j \) such that \( z_j^k = z_j^{**} \) then let \( \lambda_i^k = 0 \);
    end;
    let \( \lambda_i^k = (\lambda_1^k, \lambda_2^k, \ldots, \lambda_k^k) \);
    if \( k < t \) then
      let \( f = \rho \);
      for \( i = 1 \) to \( k \) do
        if \( \lambda_i^k - f / 2 \leq 0 \) then let \( \lambda_i^{k+1} = 0 \) and let \( \mu_i^{k+1} = f \);
        if \( \lambda_i^k - f / 2 \geq 1 \) then let \( \lambda_i^{k+1} = 1 - f \) and let \( \mu_i^{k+1} = 1 \);
        else \( \lambda_i^{k+1} = \lambda_i^k - f / 2 \) and let \( \mu_i^{k+1} = \lambda_i^k + f / 2 \);
      end;
    end;
    let \( (x^k, z^k) \) represent the final solution;
  end;
end;

Figure 8.6 - Pseudocode for Interactive Weighted Tchebycheff Method
Step 10: Let \( z^h \) represent the DM’s most preferred criterion vector from the sample.

Step 11: If the DM wishes to stop early, go to Step 16. Otherwise, go to Step 12.

Step 12: Let \( \lambda^h \) be the \( \lambda \)-vector whose components are given by

\[
\lambda^h_i = \begin{cases} 
1 & \text{if } z^h_i = z_i^{**} \\
0 & \text{if } z^h_i \neq z_i^{**} \text{ but } \exists j > z^h_j = z_j^{**} \\
1 - \left( \frac{1}{(z^h_i - z_i^{**})} \sum_{i=1}^{k} \frac{1}{(z^h_i - z_i^{**})} \right)^{-1} & \text{if } z^h_i \neq z_i^{**} \text{ for all } i
\end{cases}
\]

Step 13: With \( \lambda^h \) the weighting vector computed in Step 12, calculate new weight space bounds as

\[
[l_i^{h+1}, u_i^{h+1}] = \begin{cases} 
[0, r^h] & \text{if } \lambda^h_i - \frac{r^h}{2} \leq 0 \\
[1 - r^h, 1] & \text{if } \lambda^h_i + \frac{r^h}{2} \geq 1 \\
\lambda^h_i - \frac{r^h}{2}, \lambda^h_i + \frac{r^h}{2} & \text{otherwise}
\end{cases}
\]

where \( r^h \) is the reduction factor raised to the \( h \)th power.

Step 14: If \( h < t \), go to Step 5. If \( h \geq t \), go to Step 15.

Step 15: If the DM wishes to continue, go to Step 5. Otherwise, go to Step 16.

Step 16: Stop and present the final solution as \((x^h, z^h)\).

As with the weighted sums, the initial parameters for the IWT, \( t \), 50\( k \), 3\( P \), and \( P \) are rules of thumb and can be chosen by the DM. In particular, the expected number of iterations, \( t \), is treated as an estimate since the DM is given the option to continue or stop at the end of each iteration. We also see that filtering is used to obtain samples of
maximum dispersal. In Step 12, the weighting vector is calculated to be more properly associated with the selected nondominated criterion vector $z^h$. This is because the nondominated point found may not intersect at the vertex of the Tchebycheff contour.

The main advantage of the IWT is that its probes can locate any point in the nondominated set not just extreme points. This means that a large and significant portion of the nondominated set does not need to be overlooked. A disadvantage for the MONF is that the augmented weighted Tchebycheff program introduces additional constraints that are not of the network variety. This means that it may be difficult to take advantage of the network structure with high-speed algorithms. The $2k$ extra constraints, 2 for each objective including the objective function itself and a minimax constraint, are considered to be side constraints. An algorithm for solving networks with side constraints is discussed in Section 8.2.4.

8.2.3 Filtering

A task common to both the weighted sums and IWT is to filter samples of vectors to maximally dispersed subsets. Filtering is required twice in both of the algorithms, first to obtain a sample of representative weighting vectors for use in subproblem programs and then to obtain a smaller sample of representative criterion vectors to present to the DM. Since filtering is not a trivial task a brief presentation of a filtering algorithm from Steuer (1989) will be presented.

Given a set of vectors $V$, the goal of filtering is to select a subset of size $P$ consisting of vectors from the set that are the most different from one another. To measure the difference between two vectors we have the family of weighted $L_p$-metrics.
With \( p \) representing the metric parameter, \( q \) the number of components in the vectors, and \( \pi_i \), a range equalization weight associated with the \( i \)th component, the distance between two vectors \( \mathbf{v}^l \) and \( \mathbf{v}^m \) is given by the weighted \( L_p \)-metric

\[
\| \mathbf{v}^l - \mathbf{v}^m \|_p^p = \left[ \sum_{i=1}^{q} \left( \pi_i |v^l_i - v^m_i| \right)^p \right]^{1/p}.
\]

The range equalization weights, \( \pi_i \), are used to ensure that the vector components with the largest ranges do not bias the filtering process. To compute the range equalization weights, we first find the range of each component of the vectors in \( V \) as

\[
R_i = \max_{v \in V} \{v_i\} - \min_{v \in V} \{v_i\}.
\]

The range equalization weights are then calculated using

\[
\pi_i = \frac{1}{R_i} \left[ \sum_{j=1}^{q} \frac{1}{R_j} \right]^{-1}.
\]

In order to determine which vectors should be placed in the final subset and which should be discarded, a filtering relationship will be used to compare the distances between the points. We employ a test distance \( d \) to compare a vector \( \mathbf{v}^l \) that is currently retained by the filter with a vector \( \mathbf{v}^m \) that is under consideration using the relationship

\[
\left[ \sum_{i=1}^{q} \left( \pi_i |v^l_i - v^m_i| \right)^p \right]^{1/p} < d.
\]

If the distance between the two vectors is greater than or equal to \( d \), the two points are significantly different and the new vector is a candidate for being retained in the subset. If the difference is less than \( d \), the new vector is not a candidate for the subset. The algorithm that will be used here will choose from all vectors that are significantly different the vector that is closest.
The number of points retained by the filter is dependent on the value of the test distance $d$. Unfortunately, it is not possible to know what value should be used \textit{a priori} to running the filtering algorithm. This means that several values of $d$ might need to be tried until a subset of the desired size is obtained. To accomplish this, we choose an initial value for $d$ and filter the vectors. Then, based on the number of vectors retained for the subset a new $d$ is calculated and used for the next pass through the filter. Filtering is continued in this manner until a subset of the desired size is obtained. Pseudocode for the \textit{Filter} algorithm is presented in Figure 8.7.

The method described can only guarantee that the subset obtained is approximately maximally dispersed. This will be satisfactory for our purposes since ensuring that the subset obtained is the most dispersed requires much more computation.

\textbf{8.2.4 Networks with Side Constraints}

The mathematical formulation for the augmented weighted Tchebycheff program includes additional constraints to those of the network LP formulation. In network optimization theory additional constraints are known as \textit{side constraints}. Side constraints tend to complicate matters in network optimization. The addition of side constraints makes the special purpose network flow programming algorithms inapplicable. Nevertheless, there is interest in at least partially exploiting the network structure with specialized methods. In this section, a method developed by Kennington and Helgason (1980) is described.

The matrix for the pure MONF with no side constraints may be formulated as

\[
\begin{align*}
\min \quad & Cx = z \\
\text{s.t.} \quad & Ax = b \\
& 0 \leq x \leq u
\end{align*}
\]
algorithm Filter

begin

specify test distance \(d\) and the number of maximally dispersed vectors desired \(n\);
let \(\Delta d = d/2\), \(sflag = 0\), \(lflag = 0\), and \(icounter = 0\);
let \(V\) = number of vectors in original list, \(\text{vlist}\);
let number of retained vectors \(r = 0\);

while \(r < n\),

let \(icounter = icounter + 1\);
create empty list of retained vectors;
select vector \(\mathbf{v}^1\) as current vector and add to list of retained vectors;
let number of vectors retained or discarded \(c = 0\);
set \(\text{status}(1) = 1\), \(c = c + 1\), and \(r = r + 1\);

while \(c < V\),

for \(i = 1\) to \(V\) do

let \(\text{min} = \text{large value}\)
if \(\text{status}(i) = 0\) then

calculate distance from current vector to \(\text{vlist}(i)\);
if \(\text{distance} < \Delta d\) then

set \(\text{status}(i) = -1\)
\(c = c + 1\);
end;
end;
select \(\mathbf{v}^*\) with \(\text{status}(i) = 0\) and lowest total distance from current retained vectors;
let \(\text{status}(i) = 1\) and let \(\mathbf{v}^*\) be current vector;
let \(c = c + 1\), and \(r = r + 1\);
end;
if \(r < n\) then

if \(r > n\) then \(lflag = 1\);
if \(r < n\) then \(sflag = 1\);
if \(lflag = 1\) and \(sflag = 1\) then

let \(\Delta d = \Delta d/2\);
else

if \(icounter = 1\) then let \(\Delta d = 2^*\Delta d\).
end;
if \(r > n\) then

let \(d = d + \Delta d\);
else

let \(d = d - \Delta d\);
end;
end;
end;
return dispersed subset of filtered vectors;
end.
Where $\mathbf{x}$ is an $n$-vector of decision variables representing arc flows, $\mathbf{C}$ is a $k \times n$ matrix of objective function coefficients, and $\mathbf{z}$ is a $k$-vector of criterion values. $\mathbf{A}$ is an $m \times n$ node-arc incidence matrix of constraint coefficients and $\mathbf{b}$ is an $m$-vector of RHS values comprised of the node supplies. The equation $\mathbf{Ax} = \mathbf{b}$ is made up of the node balance equations from a minimum cost flow network. In addition, $\mathbf{u}$ is an $n$-vector of arc capacities.

A network with additional side constraints may be formulated as

$$
\min \quad \mathbf{Cx} + \mathbf{Dy} = \mathbf{z}
$$

s.t.

$$
\begin{align*}
\mathbf{Ax} + \mathbf{er} &= \mathbf{b} \\
\mathbf{Sx} + \mathbf{Py} &= \mathbf{f} \\
0 \leq \mathbf{x} \leq \mathbf{u}, \quad 0 \leq \mathbf{y} \leq \mathbf{v}
\end{align*}
$$

The vector $\mathbf{y}$ represents non-network decision variables with the vector of upper limits $\mathbf{v}$. The matrix $\mathbf{D}$ is composed of the objective function coefficients for the additional variables. The matrix equation $\mathbf{Sx} + \mathbf{Py} = \mathbf{f}$ is made up of additional constraints on $\mathbf{x}$ and $\mathbf{y}$. It is assumed that the network defined by $\mathbf{Ax} = \mathbf{b}$ is connected with total supply equal to total demand, $\sum \mathbf{b} = 0$. It is also assumed that the matrix $[\mathbf{S} | \mathbf{P}]$ has full row rank.

The $\mathbf{er}$ term in the formulation represents the root variable added to give the network constraint matrix full row rank. None of these assumptions are restrictive since if any are not true the model may be altered through simple transformations to make them true.

The full constraint matrix

$$
\overline{\mathbf{A}} = \begin{bmatrix}
\mathbf{A} & \mathbf{e} \\
\mathbf{S} & \mathbf{P}
\end{bmatrix}
$$

will have full row rank.
Kennington and Helgason (1980) describe a specialization of the simplex method which partitions the basis so that a portion of the basis always corresponds to a rooted spanning tree of a network. All calculations involving the tree component of the basis may then be carried out using labeling operations instead of matrix multiplication. In particular, they show that every basis for \( \bar{A} \) may be placed in the form

\[
\bar{B} = \begin{bmatrix} B & C \\ D & F \end{bmatrix}
\]

where \( B \) is a submatrix of \([A | e]\) and \( \text{Det}(B) \neq 0 \). An important result of this is that if both \( \bar{B} \) and \( B \) are invertible, then

\[
\bar{B}^{-1} = \begin{bmatrix}
B^{-1} + B^{-1}CQ^{-1}DB^{-1} & -B^{-1}CQ^{-1} \\
-DB^{-1}Q^{-1} & Q^{-1}
\end{bmatrix}
\]

where \( Q = F - DB^{-1}C \). \( Q \) is called the working basis.

The basis inverse and the working basis are important because they allow us to take advantage of the partial network structure of the problem. The goal is to speed up the simplex operations of pricing, ratio testing and updating. To do this, at least a portion of the simplex matrix operations of pricing and the ratio test can be replaced with labeling operations on the network. In addition, for some pivots on the MONF the updates to the basis can be improved through special operations on the working basis \( Q \).

### 8.2.4 Warm Starting

One aspect of the IWT that necessitates the bulk of the computation is that at each iteration, 2P Tchebycheff programs are formed and solved. The only differences from one formulation to the next are due to changing the constraints that result from changing \( \lambda \)-
vectors. So that we may use the optimal basis of the most recent as an advanced start the
following warm starting procedure may be employed. This procedure is based on a
strategy developed by Jeff Kennington and Bruce Patty.

Each Tchebycheff program may be formulated with the following structure:

$$\begin{align*}
\min & : \{ \quad \rho e^T w + \alpha \quad \} \\
\text{s.t.} : & \quad Ax = b \\
& \quad Cx - Iz = 0 \\
& \quad Iz + Iw = z^{**} \\
& \quad Lw - \epsilon \alpha \leq 0
\end{align*}$$

where $x \geq 0$ and integer, $z$ is unrestricted, $w \geq 0$, and $\alpha$ is unrestricted.

By substituting $\alpha = \alpha^+ - \alpha^-$ and $z = z^+ - z^-$ and inverting $L$, we have

$$\begin{align*}
\min & : \{ \quad \rho e^T w + (\alpha^+ - \alpha^-) \quad \} \\
\text{s.t.} : & \quad Ax = b \\
& \quad Cx - I(z^+ - z^-) = 0 \\
& \quad I(z^+ - z^-) + Iw = z^{**} \\
& \quad Iw - L^1 e(\alpha^+ - \alpha^-) \leq 0
\end{align*}$$

In order for the solution of problem $P_1$ to be used to warm start problem $P_2$, we can form

$$\begin{align*}
\min & : \{ \quad \rho e^T w + (\alpha^+ - \alpha^-) + M(\alpha^+ - \alpha^-) \quad \} \\
\text{s.t.} : & \quad Ax = b \\
& \quad Cx - I(z^+ - z^-) = 0 \\
& \quad I(z^+ - z^-) + Iw = z^{**} \\
& \quad Iw - L_1^1 e(\alpha^+ - \alpha^-) - L_2^1 e(\alpha^+ - \alpha^-) \leq 0
\end{align*}$$

Then to warm start problem $P_2$, we form and solve

$$\begin{align*}
\min & : \{ \quad \rho e^T w + M(\alpha^+ - \alpha^-) + (\alpha^+ - \alpha^-) \quad \} \\
\text{s.t.} : & \quad Ax = b \\
& \quad Cx - I(z^+ - z^-) = 0 \\
& \quad I(z^+ - z^-) + Iw = z^{**} \\
& \quad Iw - L_1^1 e(\alpha^+ - \alpha^-) - L_2^1 e(\alpha^+ - \alpha^-) \leq 0
\end{align*}$$
To warm start problem $P_3$, we then form and solve

$$\min: \ \{ \rho x^T w + (\alpha^+ - \alpha^-) + M(\alpha^+ - \alpha^-) \}$$

subject to:

$$Ax = b$$

$$Cx - I(z^+ - z^-) = 0$$

$$I(z^+ - z^-) + Iw = z^{**}$$

$$Iw - L_3^{-1} e(\alpha^+ - \alpha^-) - L_2^{-1} e(\alpha^+ - \alpha^-) \leq 0$$

This continues with the formulation for problem $i + 1$ as

$$\min: \ \{ \rho x^T w + (\alpha^+ - \alpha^-) + M(\alpha^+ - \alpha^-) \}$$

subject to:

$$Ax = b$$

$$Cx - I(z^+ - z^-) = 0$$

$$I(z^+ - z^-) + Iw = z^{**}$$

$$Iw - L_3^{-1} e(\alpha^+ - \alpha^-) - L_2^{-1} e(\alpha^+ - \alpha^-) \leq 0$$

8.3 A Tchebycheff Approach for MONF

The methods and theories of Section 8.2 may be combined to provide an interactive procedure for MONF problems. Each iteration, $2P$ Tchebycheff models will be formed and solved for their optimal solutions. Each model will correspond to a different weighting vector from a set generated using the filtering algorithm. The solution of each of these $2P$ models will exploit the network structure in order to improve individual model solution time. The $2P$ solutions will then be filtered down to $P$ solutions that will then be presented to the DM. Based on the DM’s most preferred solution, intervals will be calculated to contract weight space. The contracted weight space will be employed in the next iteration to generate the set of weighting vectors. In addition to the potential improvements in solution time due to exploitation of the network structure, the warm starting strategy may be employed to improve the overall solution time for each Tchebycheff iteration.
8.4 The TchebyMONF Implementation

In order to test the validity of the methods described above for the MONF problem, a set of FORTRAN programs were developed. Collectively, these programs are known as TchebyMONF. The TchebyMONF routines combine Tchebycheff theory with a Side Constrained Minimum Cost Network solution approach to provide a research and practical tool for use in solving MONF problems. Basic features of the TchebyMONF package are described in this section. This is followed by an illustrative example in Section 8.5. Both the Fortran code and a users’ manual for TchebyMONF are provided in Appendix C and D, respectively.

The TchebyMONF package is composed of N Fortran programs: TchebySTART, WEIGHTS, FILTER, and TchebySIDE.

- **TchebySTART**: This routine is used to initialize the Tchebycheff algorithm. Using this routine the MONF problem parameters are read from an input file. TchebySTART will solve the network for an initial efficient basic solution. Then by solving for each objective function a payoff matrix is generated. Using the payoff matrix, a $z^{**}$ ideal criterion vector is obtained. The ideal criterion vector and the node and arc lists for an initial efficient basic solution are stored in an intermediate file for use with the TchebySIDE procedure.

- **WEIGHTS**: Similar to the LAMBDA program from Steuer’s Adbase package (Steuer (1989)), the WEIGHTS program is used to generate a large population of initial weighting vectors. Input to WEIGHTS is provided with a file that includes the
number of weights to generate, the intervals from which to generate the weights, and an option which sets the method used to generate the weights.

- **FILTER**: Similar to the FILTER program from Steuer’s Adbase package (Steuer (1989)), the FILTER program is used to filter the weights provided by the WEIGHTS program to a representative subset.

- **TchebySIDE**: Performing the bulk of the calculations, TchebySIDE forms and solves a MONF Tchebycheff model for each of the weights generated by the FILTER program. For each MONF Tchebycheff model, the solution is obtained using a Side Constrained Network Algorithm. A warm starting strategy is used to begin the solution of each successive MONF Tchebycheff model. Input to TchebySIDE includes the output of both TchebySTART and FILTER. The output consists of a file containing a list of P criterion vectors.

### 8.5 A MONF Example

The following Packet Routing model was presented in Chapter 1. It is used here as an illustrative example of typical session using TchebyMONF. This problem was chosen primarily because its small size facilitates brief explanation. However, TchebyMONF has been designed to handle networks containing up to 1000 nodes and arcs.
Step 1: Calculate payoff table and $z^{**}$. Let $Z_1$ and $Z_2$ represent the objective function values corresponding to the arc costs in Figure 8.8. The payoff table may be calculated as shown in Table 8.1. These values may be obtained using the TchebySTART software.

**Table 8.1 – Payoff Table**

<table>
<thead>
<tr>
<th></th>
<th>$Z_1$</th>
<th>$Z_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Z_1$</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>$Z_2$</td>
<td>26</td>
<td>12</td>
</tr>
</tbody>
</table>

From Table 8.1, we can see that the minimum ("best") feasible value for $Z_1$ provides a value of 17 units and the minimum ("best") feasible value for $Z_2$ provides a value of 12 units. Using this information, we can compute a utopian point, $z^{**}$, as

$$z^{**} = (16, 11).$$

Step 2: Using the interval $[l_i^{(1)}, u_i^{(1)}] = [0,1]$ and the WEIGHTS procedure, generate $50k\lambda$-vectors for the first iteration of Tchebycheff formulations from

$$\Lambda^{(h)} = \left\{ \lambda \in R^k \left| \sum_{i=1}^{k} \lambda_i = 1 \right. \right\}. \text{ Filter these weight vectors (using FILTER)}$$

to $3P$ most different.
Step 3: Form and solve 3P Weighted-Tchebycheff Network problems using TchebySIDE.

Step 4: Filter the 3P nondominated criterion solution vectors from Step 3 down to the P most different.

Step 5: Present the P nondominated solution vectors to the decision maker. The decision maker should select their most preferred.

Table 8.2 shows the result of the first set (P = 5) of nondominated solutions along with their corresponding λ-vectors. In order to ensure consistency of selection, we assume that the decision maker selects a criterion vector that minimizes the utility function $U = -2Z_1Z_2$. Based on this utility function, the criterion vector indicated by the shading in the table would be selected.

Table 8.2 – Criterion Vectors from Iteration 1.

<table>
<thead>
<tr>
<th>Weights</th>
<th>Criterion Vectors</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>λ₁</td>
<td>λ₂</td>
<td>Z₁</td>
<td>Z₂</td>
<td>U</td>
</tr>
<tr>
<td>0.0177</td>
<td>0.9823</td>
<td>26.0000</td>
<td>12.0000</td>
<td>-624.0000</td>
</tr>
<tr>
<td>0.2126</td>
<td>0.7874</td>
<td>23.8292</td>
<td>13.1139</td>
<td>-624.9861</td>
</tr>
<tr>
<td>0.4407</td>
<td>0.5593</td>
<td>21.0414</td>
<td>14.9724</td>
<td>-630.0806</td>
</tr>
<tr>
<td>0.6315</td>
<td>0.3685</td>
<td>19.0808</td>
<td>16.2795</td>
<td>-621.2502</td>
</tr>
<tr>
<td>0.8293</td>
<td>0.1707</td>
<td>17.3656</td>
<td>17.6344</td>
<td>-612.4639</td>
</tr>
</tbody>
</table>

Step 6: Reduce weight space around the λ-vector corresponding to the DM’s selection using

$$[l_i^{h+1}, u_i^{h+1}] = \begin{cases} 
[0, r^h] & \text{if } \lambda_i^h - \frac{r^h}{2} \leq 0 \\
[1 - r^h] & \text{if } \lambda_i^h - \frac{r^h}{2} \geq 1 \\
[\lambda_i^h - \frac{r^h}{2}, \lambda_i^h + \frac{r^h}{2}] & \text{otherwise}
\end{cases}$$
with \( r = 0.27 \) and \( h = 1 \). The calculated intervals, which will be used in generating \( \lambda \)-vectors for the second iteration are shown in Table 8.3.

Table 8.3 – Weight Vector Intervals for Iteration 2.

<table>
<thead>
<tr>
<th>Intervals</th>
<th>( \lambda_1 )</th>
<th>( \lambda_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_i )</td>
<td>0.236</td>
<td>0.494</td>
</tr>
<tr>
<td>( h_i )</td>
<td>0.506</td>
<td>0.764</td>
</tr>
</tbody>
</table>

Steps 2 through 6 may be repeated a set number of iterations, until there is no significant improvement, or the DM wishes to stop. Criterion vectors for 2 additional iterations are shown in Tables 8.4 and 8.5. Notice how quickly this problem has converged to solutions that fall within a small interval. While this can be affected by the size of the model, in terms of number of nodes, arcs and objectives, this convergence depends primarily on the sample size generated, the weight space reduction factor and the precision desired by the DM.

Table 8.4 – Criterion Vectors from Iteration 2.

<table>
<thead>
<tr>
<th>Weights</th>
<th>( \lambda_1 )</th>
<th>( \lambda_2 )</th>
<th>Criterion Vectors</th>
<th>( Z_1 )</th>
<th>( Z_2 )</th>
<th>( U )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda_1 )</td>
<td>( \lambda_2 )</td>
<td>( Z_1 )</td>
<td>( Z_2 )</td>
<td>( U )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2403</td>
<td>0.7597</td>
<td>23.4603</td>
<td>13.3598</td>
<td>-626.8496</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2765</td>
<td>0.7235</td>
<td>22.9919</td>
<td>13.6721</td>
<td>-628.6937</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3932</td>
<td>0.6068</td>
<td>21.5781</td>
<td>14.6146</td>
<td>-630.7106</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4380</td>
<td>0.5620</td>
<td>21.0714</td>
<td>14.9524</td>
<td>-630.1359</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5023</td>
<td>0.4977</td>
<td>20.3757</td>
<td>15.4162</td>
<td>-628.2319</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 8.5 – Criterion Vectors from Iteration 3.

<table>
<thead>
<tr>
<th>Weights</th>
<th>Criterion Vectors</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_1$</td>
<td>$\lambda_2$</td>
<td>$Z_1$</td>
<td>$Z_2$</td>
<td>$U$</td>
</tr>
<tr>
<td>0.3578</td>
<td>0.6422</td>
<td>21.9922</td>
<td>14.3385</td>
<td>-630.6718</td>
</tr>
<tr>
<td>0.3812</td>
<td>0.6188</td>
<td>21.7171</td>
<td>14.5219</td>
<td>-630.7486</td>
</tr>
<tr>
<td>0.4038</td>
<td>0.5962</td>
<td>21.4565</td>
<td>14.6956</td>
<td>-630.6344</td>
</tr>
<tr>
<td>0.4129</td>
<td>0.5871</td>
<td>21.3530</td>
<td>14.7647</td>
<td>-630.5398</td>
</tr>
<tr>
<td>0.4289</td>
<td>0.5711</td>
<td>21.1728</td>
<td>14.8848</td>
<td>-630.3058</td>
</tr>
</tbody>
</table>

8.7 Conclusion

The procedures and software discussed in this chapter provide a method for decision aid consistent with the more sophisticated, interactive methods employed in multiple criteria decision-making. They employ results from network flow theory that have the potential for significantly improving the solution time for large MONF models.

In addition, a foundation is provided for exploring an interesting combination of decision aid that has received relatively little attention to date. Several future directions may be explored. These include:

- **Finding nearest efficient, integer solution** – Many practical problems that may be modeled as networks require that the ultimate solution chosen be integer. Using the IWT method combined with the network with side constraints algorithm will typically produce solutions on the convex hull of the network feasible region. This means that non-integer solutions are the norm. So far, no guaranteed algorithm for moving to the nearest nondominated efficient solution has been developed.

- **Combine algorithm with Interactive Weighted Sums (IWS)**: The IWT may be viewed as a complementary approach to the IWT. The IWT sub-problems may be solved as pure network flow models taking full advantage of the very fast
algorithms developed for their solution. A major disadvantage of IWS is that it is only capable of generating corner point solutions. Steuer (1995) describes a method in which the IWT and IWS may be combined. In early stages of the combined method, IWS may be used to narrow the solution space under consideration. In later stages, the IWT may then be used to focus on an optimal, possibly non-corner point solution.

• **Empirical testing with real-world problems:** The major testing thus far with both the TchebyMONF package and the MONFSOLV software discussed in Chapter 7 has been with randomly generated problems. Applications of network models in such areas as telecommunications networks, computer network design, and financial decision making should be of great interest.

• **Further work with the unique formulation of the Tchebycheff LP:** The network simplex with side constraints algorithm of Kennington and Helgason that has been used for this work attempts to take advantage of the partial network structure of the LP. The Tchebycheff LP formulated for TchebyMONF retains this partial network structure. In addition, there may be further structural characteristics of the Tchebycheff LP that will allow for further algorithmic improvements. One such structure is that of the D-matrix described for the side constraint algorithm. Notice in the example that this matrix may be partitioned into two matrices, one incorporating the constraint matrix coefficient of the MONF model and the other consisting entirely of zeroes. This and other characteristics may lead to improvements from further factorization of the basis matrix.
These items will form the basis for further work in this area. The remaining chapters in this dissertation will explore the characteristic and solution of large MOLP models. Since practical network flow applications are typically very large in terms of number of nodes and arcs, the results should be applicable.
CHAPTER 9

ANALYSIS OF NUMBER OF EFFICIENT EXTREME POINTS
VERSUS PROBLEM SIZE

9.1. Introduction

We have seen that a significant area of endeavor that falls under the rubric of
Multiple Criteria Decision Making (MCDM) is research into the computational properties
and characteristics of problems in multiple objective linear programming. Authors
including Benson and Sayin (1997), Breslawski and Zionts (1992), Climaco and Antunes
Korhonen, Salo and Steuer (1997), Mavrotas, Diakoulaki and Assimacopoulos (1998),
Reeves and Reid (1988), Sayin (1998), Strijbosch, van Doorne and Selen (1991), and
Zionts and Wallenius (1980) have studied the computational characteristics and properties
of the efficient set. Some of the topics that have been examined include:

• Enumerating all efficient extreme points.
• Obtaining payoff tables and minimum criterion values over the efficient set.
• Optimizing an arbitrary function over the efficient set.
• Obtaining discrete representations of the efficient set.
• Computing the subsets of efficient extreme points that define the maximally efficient
  facets of an MOLP.
Such research has led to an increased understanding of the efficient set and challenges involved in the development of effective procedures (see the methods reviewed in Gardiner and Steuer (1994) and references therein) for searching the efficient set for a most preferred solution.

Because, on the research envelope, the number of efficient extreme points of an MOLP continues to be an open question as computers become more powerful, the purpose of this chapter is to explore the nature of the statistical relationship between the number of all efficient extreme points in problems that may have up to 100,000 efficient extreme points and the problem-size parameters of an MOLP. The regression analysis of this chapter would also be of value for dynamic array size allocation in programs that process efficient extreme points. It would provide a means for predicting the number of efficient extreme points and the CPU time to compute them where such information is necessary for carrying out or validating the results of computational studies.

In Section 9.2, 9.3 and 9.4, we discuss the random problem generator employed, the experimental runs conducted, and the observations obtained. In Sections 9.5 through 9.8, we discuss the model building process, proposed regression equations, and the ability of the models to predict the mean responses for new observations. Section 9.9 consists of concluding remarks.

9.2. Random Problem Generation

The program ADBASE (Steuer (1998)) was the workhorse software of this investigation. ADBASE computes all efficient extreme points of an inputted MOLP. Employing the program's random problem generator, ADBASE was used to solve
randomly generated MOLPs for all efficient extreme points. All of the ADBASE runs were conducted on a University of Georgia mainframe computer because of the availability of a comfortable editing environment and multiple processors to run the streams of experiments on an around-the-clock basis. All times have been scale-adjusted to equivalent 400 MHz Pentium II times for more meaningful benchmark purposes.

Quantities necessary to configure the random problem generator are:

- **K** number of objectives
- **M** number of main constraints
- **N** number of structural variables
- **ADEN** percentage nonzero **A**-matrix density
- **CDEN** percentage nonzero **C**-matrix density
- **[LC, UC]** bounds on the interval containing the integers from which the nonzero **C**-matrix elements are uniformly drawn
- **[LA, UA]** lower and upper bounds on the interval containing the integers from which the nonzero **A**-matrix elements are uniformly drawn
- **[LB, UB]** lower and upper bounds on the interval containing the integer from which the **b**-vector elements are uniformly drawn
- **IDISP** a nonnegative value used to displace the midpoint of the feasible region defined by the main constraints away from the origin to the point \((IDISP, IDISP, IDISP) \in \mathbb{R}^n\).

In the above, the quantities CDEN, LA, UA, LB, UB and IDISP are not considered problem-size parameters because their purpose is merely to fashion reasonably constructed randomly generated MOLPs (details about the quantities are given in Steuer
Hence, the values of these quantities were held fixed throughout the experiments at the levels shown on the right-hand side of Table 8.1. On the other hand, the quantities K, M, N, ADEN and the difference between LC and UC, are known, or strongly suspected, as having a material effect on the number of efficient extreme points. The difference between LC and UC is denoted GAP in this research and is adjusted by holding UC constant and varying LC in a controlled fashion. These quantities are called a problem-size parameters. The values of the problem-size parameters were varied over those values shown on the left-hand side of Table 9.1.

<table>
<thead>
<tr>
<th>Problem-Size Parameters</th>
<th>Values Used</th>
<th>Constants</th>
<th>Values Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>2, 3, 4, 5, 6, 7, 8</td>
<td>CDEN</td>
<td>100</td>
</tr>
<tr>
<td>M</td>
<td>10, 20, 30, 40, 50, 60</td>
<td>UC</td>
<td>20</td>
</tr>
<tr>
<td>N</td>
<td>25, 50, 75, 100, 125</td>
<td>[LA, UA]</td>
<td>[-20, 20]</td>
</tr>
<tr>
<td>ADEN</td>
<td>20, 40, 60, 80, 100</td>
<td>[LB, UB]</td>
<td>[50, 100]</td>
</tr>
<tr>
<td>LC</td>
<td>-20, -10, 0, 10</td>
<td>IDISP</td>
<td>20</td>
</tr>
</tbody>
</table>

9.3. Experimental Procedures

The data were obtained using a designed experiment that varied the problem-size parameters to fit an elliptical equation because of the interaction of the parameters. The goal was to obtain data for as many combinations of the parameter values as possible. Certain combinations of the parameters were infeasible or impractical as MOLPs and were eliminated beforehand. These include problem parameters which would necessarily result in infeasible MOLP problems. In addition, some combinations of the parameters created problems that were too large or took too long to solve for the available computing resources. These combinations were also eliminated. In the end, data were obtained for
2556 of the possible combinations of random problem generator parameters. Two replications were generated at random for each of these 2556 combinations resulting in a total of 5112 data points in order to get better information about patterns in the error variances. The number of efficient extreme points in the 5112 MOLPs range from 1 to 119,876 with solution times varying from 0.02 to 1247.82 Pentium seconds. A second set of data consisting of 2030 data points was generated to use in validating the predictive capability of the regression models selected.

For each data point, the quantities \( K, M, N, \text{ADEN} \) and \( \text{LC} \) were recorded along with the \( \text{GAP} \) and a cone-size measure \( \text{ACONEA} \). The output variables representing the number of efficient extreme points, \( \text{KEEXT} \), and the time for problem solution, \( \text{CPUTIM} \), were also recorded. \( \text{ACONEA} \) is designed to provide a more accurate measure of the size of the criterion cone than can be obtained by using the \( \text{LC} \) and \( \text{UC} \) bounds. The \( \text{ACONEA} \) measure is a function of the average angle between the gradients of the objective functions. Larger values for \( \text{ACONEA} \) correlate with larger cone sizes. While the data reveals that \( \text{ACONEA} \) is highly correlated (\( R = 0.986 \)) with the \( \text{GAP} \), \( \text{ACONEA} \) provides a problem size parameter that is related to the generated problem itself. Also, \( \text{ACONEA} \) can be calculated from any problem specifications used in practice.

### 9.4. Initial Observations

Since the number of initial predictor variables is not large, a full exploration of relationships and possible strong interactions was undertaken. The values of the predictor variables under consideration were determined based on the experimental design.
An initial regression analysis was performed using KEEXT as the response variable and the variables: K, M, N, ADEN, and GAP as predictor variables. The results are shown in Figure 9.1. A normal probability plot of the residuals is shown in Figure 9.2 and a plot of the standardized residuals versus the fitted values is shown in Figure 9.3.
Figure 9.2- Normal Probability Plot of Residuals (Minitab 10.51)

Figure 9.3 - Standardized Residuals vs. Fitted Values (Minitab 10.51)
It is of interest to see how ACONEA would work as a substitute in the equation.

As previously described, ACONEA is a measure of the angles between the objectives and is therefore more applicable to the general MOLP. The initial results of a regression using the ACONEA variable to replace GAP are shown in Figures 9.4 through 9.6.

The regression equation is
\[
K_{EEXT} = -12658 + 1705 K + 95.9 M + 1.58 N + 16.1 ADEN + 90.1 ACONEA
\]

<table>
<thead>
<tr>
<th>Predictor</th>
<th>Coef</th>
<th>Stdev</th>
<th>t-ratio</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>-12658.2</td>
<td>579.3</td>
<td>-18.63</td>
<td>0.000</td>
</tr>
<tr>
<td>K</td>
<td>1705.40</td>
<td>71.08</td>
<td>23.99</td>
<td>0.000</td>
</tr>
<tr>
<td>M</td>
<td>95.855</td>
<td>7.646</td>
<td>12.54</td>
<td>0.000</td>
</tr>
<tr>
<td>N</td>
<td>1.584</td>
<td>3.570</td>
<td>0.44</td>
<td>0.657</td>
</tr>
<tr>
<td>ADEN</td>
<td>16.071</td>
<td>4.134</td>
<td>3.89</td>
<td>0.000</td>
</tr>
<tr>
<td>ACONEA</td>
<td>90.066</td>
<td>4.182</td>
<td>21.54</td>
<td>0.000</td>
</tr>
</tbody>
</table>

\( s = 8372 \quad R\text{-sq} = 15.2\% \quad R\text{-sq(adj)} = 15.1\% \)

Analysis of Variance

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SS</th>
<th>MS</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>5</td>
<td>64078835712</td>
<td>12815767552</td>
<td>182.87</td>
<td>0.000</td>
</tr>
<tr>
<td>Error</td>
<td>5106</td>
<td>3.57842E+11</td>
<td>70082632</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>5111</td>
<td>4.21921E+11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SEQ SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>1</td>
<td>23459758464</td>
</tr>
<tr>
<td>M</td>
<td>1</td>
<td>7106672832</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>116334768</td>
</tr>
<tr>
<td>ADEN</td>
<td>1</td>
<td>875396288</td>
</tr>
<tr>
<td>ACONEA</td>
<td>1</td>
<td>32510474240</td>
</tr>
</tbody>
</table>

Figure 9.4 - Initial Regression Output with ACONEA (Minitab 10.51)
229

**Figure 9.5 - Normal Probability Plot of Residuals (Minitab 10.51)**

---

**Figure 9.6 - Standardized Residuals vs. Fitted Values (Minitab 10.51)**

---
A comparison of the regression analysis using GAP and the regression analysis using ACONEA shows that there is very little difference in the results. The $R^2$ value for the ACONEA regression is only slightly higher than that of the GAP regression. In addition, the coefficients of the other predictor variables are nearly the same and the plots appear virtually identical in shape. This is not surprising considering the high correlation between these two alternative variables. Thus, it is reasonable to replace the GAP variable with ACONEA in the analysis and this will be done for all further analysis.

At first glance, the initial results do not appear very promising. The $R^2$ value, which gives the amount of variation in the number of extreme points that can be explained by the predictor variables, is very low at $R^2 = 15.2\% \ (R^2_{adj.} = 15.1\%)$. The standard deviation estimate $s = 8372$ and $\text{MSE} = 70082632$ are high. In addition, the high p-value for the variable $N$ indicates that there is insufficient evidence to indicate that this variable should be included in the model. This runs counter to our experience. In order to improve the prediction criteria values, we will investigate interaction effects and higher order variables in the model.

The normal probability plot and the plot of standardized residuals shown in Figures 9.5 and 9.6 give an indication of an inherent problem with the current model. A basic assumption of regression analysis is that the probability distributions of the response variable (KEEXT) are normal and have homogeneous variance, regardless of the levels of the predictor variables. The normal probability plot of Figure 9.5 indicates that the error terms are not normally distributed. The coefficient of correlation between the ordered residuals and their expected values under normality is only $R = 0.708$. The plot of Figure 8.6 clearly shows that the assumption of constant variance does not hold true as the
variance increases drastically with higher values of KEEXT. This condition is known a heteroscedasticity. Heteroscedasticity is inherent when the response in regression analysis is functionally related to the mean. Remedial measures for the departure from normality and non-homogenous variance can include transformations of variables or the use of a weighted least squares procedure.

Several other first-order regression models were fit and various residual plots obtained to investigate interaction effects and possible transformation of variables. One example, shown in Figure 9.7, indicates that a logarithmic or exponential transformation of either the response variable or the K variable may be necessary. Other transformation and interaction effects were also noticed.

![Partial Regression Plot for K](image)

Figure 9.7 - Partial Regression Plot for K (Minitab 10.51)
Figure 9.8 - Initial Regression Output for CPU times (Minitab 0.51)

An initial regression was also carried out using the Pentium CPU times as the response variable. The results were slightly better than the regression for number of extreme points with an $R^2 = 20.9\%$ and all variables seem to provide a significant contribution to the model. However, the same violations in model assumptions are indicated. The regression results, normal probability plot and residual plot is shown in Figures 9.8 through 9.10.

The regression equation is:

$$CFUTIM = -357.34 + 43.7 K + 21.19 M + 1.01 N + 0.341 ADEN + 1.87 ACONEA$$

<table>
<thead>
<tr>
<th>Predictor</th>
<th>Coef</th>
<th>Stdev</th>
<th>t-ratio</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>-357.34</td>
<td>13.14</td>
<td>-27.20</td>
<td>0.000</td>
</tr>
<tr>
<td>K</td>
<td>43.733</td>
<td>1.374</td>
<td>31.82</td>
<td>0.000</td>
</tr>
<tr>
<td>M</td>
<td>2.1894</td>
<td>0.1478</td>
<td>14.81</td>
<td>0.000</td>
</tr>
<tr>
<td>N</td>
<td>1.00922</td>
<td>0.06904</td>
<td>14.62</td>
<td>0.000</td>
</tr>
<tr>
<td>ADEN</td>
<td>0.34071</td>
<td>0.07993</td>
<td>4.26</td>
<td>0.000</td>
</tr>
<tr>
<td>ACONEA</td>
<td>1.86751</td>
<td>0.08086</td>
<td>23.10</td>
<td>0.000</td>
</tr>
</tbody>
</table>

$s = 161.9 \quad R$-sq $= 20.9\% \quad R$-sq(adj) $= 20.9\%$

Analysis of Variance

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SS</th>
<th>Mean SS</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>5</td>
<td>35427740</td>
<td>7085548</td>
<td>270.41</td>
<td>0.000</td>
</tr>
<tr>
<td>Error</td>
<td>5106</td>
<td>133791848</td>
<td>26203</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>5111</td>
<td>169219584</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SS</th>
<th>Q</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>1</td>
<td>13942955</td>
<td>13942955</td>
<td>13942955</td>
</tr>
<tr>
<td>M</td>
<td>1</td>
<td>2816030</td>
<td>2816030</td>
<td>2816030</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>4296133</td>
<td>4296133</td>
<td>4296133</td>
</tr>
<tr>
<td>ADEN</td>
<td>1</td>
<td>395173</td>
<td>395173</td>
<td>395173</td>
</tr>
<tr>
<td>ACONEA</td>
<td>1</td>
<td>13977501</td>
<td>13977501</td>
<td>13977501</td>
</tr>
</tbody>
</table>
Figure 9.9 - Normal Probability Plot of Residuals for CPU times (Minitab 10.51)

Figure 9.10 - Standardized Residuals vs. Fitted Values for CPU times (Minitab 10.51)
9.5. Model Building and Selection

When building a prediction model, there are several aspects of model building that should be considered. These include: the proper selection of variables and the proper functional form in which they should enter the model; the verification of model assumptions; the possible remedial measures when assumptions are violated; whether or not the model should include an intercept term; and the proper criteria for selecting good models. In this section, we discuss these modeling aspects.

For this analysis, the variables for data collection were chosen based on the author's theoretical knowledge and experience. Once the initial pool of variables has been selected and data have been obtained and verified, the building of the model can begin. This model building process will result in the selection of variables to appear in the final model. The variables will include some of those from the initial pool and possibly variable transformations such as interaction and quadratic variables. Based on the possibility of interactions indicated by regression and partial residual plots, all two-variable cross product terms and quadratic terms for each of the predictor variables were added to our pool of variables under investigation.

Another consideration for our model is the question of whether or not the intercept term should be included in the model. A model without an intercept term reflects a relationship between the variables that will pass through the origin. It seems reasonable that a MOLP with problem size parameters equal to zero will have zero extreme points. While this may be the case in reality, our model will be used for prediction within the range of the data points obtained.
As it turns out, a simple logarithmic transformation of the response variable improves the situation. In this case, the transformation corrects for the heteroscedasticity of the data and decreases the strength of most interactive effects. To perform this transformation, the natural logarithm of the KEEXT variable was taken for each record in the data set to create a new variable called LNY. Using this variable as the response variable, an analysis was performed regressing this variable against the initial set of predictor variables. The results are shown in Figures 9.11 through 9.13.

The regression equation is

\[ \text{LNY} = -4.11 + 0.940 \text{ K} + 0.0732 \text{ M} + 0.0101 \text{ N} + 0.0109 \text{ ADEN} + 0.0412 \text{ ACONEA} \]

<table>
<thead>
<tr>
<th>Predictor</th>
<th>Coef</th>
<th>Std Err</th>
<th>t-ratio</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>-4.1055</td>
<td>0.1204</td>
<td>-34.11</td>
<td>0.000</td>
</tr>
<tr>
<td>K</td>
<td>0.94039</td>
<td>0.01260</td>
<td>74.66</td>
<td>0.000</td>
</tr>
<tr>
<td>M</td>
<td>0.073185</td>
<td>0.001355</td>
<td>54.02</td>
<td>0.000</td>
</tr>
<tr>
<td>N</td>
<td>0.0100765</td>
<td>0.0006327</td>
<td>15.93</td>
<td>0.000</td>
</tr>
<tr>
<td>ADEN</td>
<td>0.0106655</td>
<td>0.0007325</td>
<td>14.33</td>
<td>0.000</td>
</tr>
<tr>
<td>ACONEA</td>
<td>0.0412054</td>
<td>0.0007410</td>
<td>55.61</td>
<td>0.000</td>
</tr>
</tbody>
</table>

\[ s = 1.483 \quad R^2 = 61.4\% \quad R^2(\text{adj}) = 61.4\% \]

Analysis of Variance

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SS</th>
<th>MS</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>5</td>
<td>17991.2</td>
<td>3570.2</td>
<td>1626.04</td>
<td>0.000</td>
</tr>
<tr>
<td>Error</td>
<td>5106</td>
<td>11236.2</td>
<td>2.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>5111</td>
<td>29227.4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SEQ</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>1</td>
<td>5701.9</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>1</td>
<td>4569.5</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>266.5</td>
<td></td>
</tr>
<tr>
<td>ADEN</td>
<td>1</td>
<td>426.5</td>
<td></td>
</tr>
<tr>
<td>ACONEA</td>
<td>1</td>
<td>6804.8</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9.11 - Regression Output for Log Transformation (Minitab 10.51)
Figure 9.12 - Normal Probability Plot of Residuals for Log Transform (Minitab 10.51)

Correlation of RESI1 and NORMS = 0.999

Figure 9.13 - Standardized Residuals vs. Fitted Values of LNY(Minitab 10.51)
Several improvements can be noted in the model. First, the $R^2$ value has improved from 15.2% to 61.4% indicating that a relatively large amount of the variation in the new response variable can be explained by the regression relationship. Second, the normal probability plot of Figure 9.12 demonstrates that the residuals are approximately normally distributed. Third, the plot of standardized residuals in Figure 9.13 shows that while the variance still increases somewhat with increasing LNY, this tendency has been reduced.

Parallel results for the natural logarithm transformation of CPUTIM, denoted LNCPU, are shown in Figures 9.14 to 9.16. For this regression analysis, the $R^2$ value is 67.7%.

The regression equation is

$$\text{LNCPU} = -0.50 + 1.17 K + 0.0001 M + 0.0329 N + 0.0102 \text{ADEN} + 0.0401 \text{ACONEA}$$

<table>
<thead>
<tr>
<th>Predictor</th>
<th>Coef</th>
<th>StdDev</th>
<th>t-ratio</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>-0.5020</td>
<td>0.1248</td>
<td>-66.76</td>
<td>0.000</td>
</tr>
<tr>
<td>K</td>
<td>1.17259</td>
<td>0.01306</td>
<td>89.73</td>
<td>0.000</td>
</tr>
<tr>
<td>M</td>
<td>0.000101</td>
<td>0.001405</td>
<td>57.02</td>
<td>0.000</td>
</tr>
<tr>
<td>N</td>
<td>0.0029048</td>
<td>0.0006560</td>
<td>50.16</td>
<td>0.000</td>
</tr>
<tr>
<td>ADEN</td>
<td>0.002105</td>
<td>0.0007595</td>
<td>13.44</td>
<td>0.000</td>
</tr>
<tr>
<td>ACONEA</td>
<td>0.0401361</td>
<td>0.0007683</td>
<td>52.24</td>
<td>0.000</td>
</tr>
</tbody>
</table>

$s = 1.538$  \hspace{1cm} R-sq = 67.7\%  \hspace{1cm} R-sq(adj) = 67.7\%

Analysis of Variance

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SS</th>
<th>MS</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>5</td>
<td>25302.2</td>
<td>5060.4</td>
<td>1138.82</td>
<td>0.000</td>
</tr>
<tr>
<td>Error</td>
<td>5106</td>
<td>12080.8</td>
<td>2.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>5111</td>
<td>37383.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>1</td>
<td>6631.1</td>
</tr>
<tr>
<td>M</td>
<td>1</td>
<td>4818.7</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>5021.5</td>
</tr>
<tr>
<td>ADEN</td>
<td>1</td>
<td>374.6</td>
</tr>
<tr>
<td>ACONEA</td>
<td>1</td>
<td>6456.2</td>
</tr>
</tbody>
</table>

Figure 9.14 - Regression Output for Log Transformation of CPU Time(Minitab 10.51)
Figure 9.15 - Normal Probability Plot of Residuals for Log Transform (Minitab 10.51)

<table>
<thead>
<tr>
<th>NORMS</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.5+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>40</td>
<td>++++</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>++++</td>
<td>0.0+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>++++</td>
<td>0.0+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>++++</td>
<td>0.0+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>++++</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>x7+8+9</td>
<td>x152</td>
<td>x x</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Correlation of RESI1 and NORMS = 0.996

Figure 9.16 - Standardized Residuals vs. Fitted Values of LNY(Minitab 10.51)

<table>
<thead>
<tr>
<th>SR331</th>
<th>2.5+</th>
<th>x</th>
<th>42x3599533</th>
<th>x 23477+</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>x x2</td>
<td>23x43x57+7+9+46+75x++</td>
<td>x47+954x4 233 x</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>3x754243533x5567996++402422</td>
<td>3x65555 3 xx xx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>332563+++9864933 x x</td>
<td>x3x32235+94695++5+7x2xx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>x3x32432234x24x27x3xxx x</td>
<td>xxxixxx42 ixx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>x x x</td>
<td>- x x x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FITS1</th>
<th>-2.5</th>
<th>0.0</th>
<th>2.5</th>
<th>5.0</th>
<th>7.5</th>
<th>10.0</th>
</tr>
</thead>
</table>
With these results in mind, all further analysis was performed using LNY and LNCPU as the response variables. The focus was then turned towards additional predictor variable selection and refinement of the model. Additional variables analyzed included quadratic terms for all of the predictor variables as well as interaction effects. While the R² value was used as the primary criterion for model selection, other factors such as the number and type of variables in the model, the behavior of residuals, and the final predictive value of the model were taken into account.

Figure 9.17 provides a graphical comparison of the R² values for various models under review. These values were obtained by performing a Best-Subsets Regression for all 20 first-order, second-order, and first-order interaction variables. Best-Subsets Regression is a method whereby certain regression values, in this case R², are calculated for models using possible combinations of the predictor variables. The statistical analysis package used to support this analysis employs a technique known as the Hamiltonian Walk (Minitab Release 10 (1995)). In this technique all 2^m - 1 possible subsets may be "visited" in 2^m - 1 steps. Each subset in the Hamiltonian Walk differs from the previous subset by the addition or deletion of only one variable. This allows each successive R² value to be efficiently derived from its predecessor.

The efficient method allowed for the R² value to be calculated for all 1,048,575 possible subsets. Only the top three subsets for each number of predictor variables are displayed in the chart. Note that the R² value will always increase as more variables are added to the model. This results in a trade-off between the number of variables in the model and a higher R² value. This trade-off is represented in the Figure 9.17 as a
nondominated frontier. A common rule of thumb is to choose a model at a point where adding further variables does not significantly improve the result.

Figure 9.17 - Plot of Adjusted R2 versus The Number of Model Variables

9.6 Regression Model for Number of Efficient Extreme Points

The following refined regression model was selected

\[
\text{LNY} = -4.43 + 1.26 \text{ K} + 0.135 \text{ M} - 0.148 \text{ K}^2 - 0.00162 \text{ M}^2 + 0.0140 \text{ K} \times \text{M} + 0.00361 \text{ K} \times \text{N} + 0.0112 \text{ K} \times \text{ACONEA} + 0.0115 \text{ ADEN}
\]

based on the trade-off between its high R² value and the number of variables along with the fact that each of the controllable variables of the random problem generator are included in the model. It is located on the graph for Figure 9.17 as the point with the highest R² (R² = 81.4%) value plotted above the eight on the horizontal axis. The model
will enable us to make predictions on the number of MOLP extreme points for various parameter combinations.

The regression results are shown in Figures 9.18 through 9.20. For this model, the $R^2$ value indicates that 81.4% of the variation of LNY is explained using the predictor variables of the formula. The p-values indicate that the coefficients are significant. The plots show that the residuals are near normally and randomly distributed.

The regression equation is

$$LNY = -4.43 + 1.26 K + 0.135 M - 0.148 K2 - 0.00162 M2 + 0.0140 KM + 0.00361 KN + 0.0112 KAC + 0.0115 ADEN$$

<table>
<thead>
<tr>
<th>Predictor</th>
<th>Coef</th>
<th>Stdev</th>
<th>t-ratio</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>-4.4345</td>
<td>0.1474</td>
<td>-30.09</td>
<td>0.000</td>
</tr>
<tr>
<td>K</td>
<td>1.26221</td>
<td>0.05397</td>
<td>23.39</td>
<td>0.000</td>
</tr>
<tr>
<td>M</td>
<td>0.134883</td>
<td>0.005238</td>
<td>25.75</td>
<td>0.000</td>
</tr>
<tr>
<td>K2</td>
<td>-0.147708</td>
<td>0.004757</td>
<td>-31.05</td>
<td>0.000</td>
</tr>
<tr>
<td>M2</td>
<td>-0.00162312</td>
<td>0.00006222</td>
<td>-26.08</td>
<td>0.000</td>
</tr>
<tr>
<td>KM</td>
<td>0.0140118</td>
<td>0.0005969</td>
<td>23.47</td>
<td>0.000</td>
</tr>
<tr>
<td>KN</td>
<td>0.0036148</td>
<td>0.0001044</td>
<td>34.61</td>
<td>0.000</td>
</tr>
<tr>
<td>KAC</td>
<td>0.0111783</td>
<td>0.0001233</td>
<td>90.66</td>
<td>0.000</td>
</tr>
<tr>
<td>ADEN</td>
<td>0.0115441</td>
<td>0.0005091</td>
<td>22.68</td>
<td>0.000</td>
</tr>
</tbody>
</table>

$s = 1.030 \quad R$-sq = 81.4\% \quad R$-sq(adj) = 81.4\%

Analysis of Variance

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SS</th>
<th>MS</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>8</td>
<td>23708.9</td>
<td>2963.6</td>
<td>2791.03</td>
<td>0.000</td>
</tr>
<tr>
<td>Error</td>
<td>5103</td>
<td>5418.5</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>Total</td>
<td>5111</td>
<td>29127.4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SEQ</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>1</td>
<td>5701.9</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>1</td>
<td>4669.5</td>
<td></td>
</tr>
<tr>
<td>K2</td>
<td>1</td>
<td>3037.9</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>1</td>
<td>795.4</td>
<td></td>
</tr>
<tr>
<td>KM</td>
<td>1</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>KN</td>
<td>1</td>
<td>343.8</td>
<td></td>
</tr>
<tr>
<td>KAC</td>
<td>1</td>
<td>8610.4</td>
<td></td>
</tr>
<tr>
<td>ADEN</td>
<td>1</td>
<td>546.1</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9.18 - Regression Output for Final Log Transformation Model (Minitab 10.51)
An additional data set of 2030 observations was generated in order to validate the model. When fitting the chosen model to the validation set, the $R^2$ value was found to be 82.0% with coefficients close to those of the model. A mean squared prediction error MSPR was calculated to be 1.217 which is very close to the MSE =1.2 of the regression fit. If the mean squared prediction error is fairly close to MSE based on the regression fit to the model-building set, then the error mean square MSE for the selected regression model is not seriously biased and gives an appropriate indication of the predictive ability of the model.

Figure 9.19 - Plot of Residuals versus Fitted Values for Final Log Transformation Model (Minitab 10.51)
In Figure 9.21, a plot of adjusted $R^2$ values versus number of predictor variables generated from a best subset regression for CPU time is presented. The plot represents a nondominated frontier similar to that of Figure 9.17. Here, we see again that an equation using 8 predictor variables represents a reasonable tradeoff between number of predictor variables and best adjusted $R^2$. With an adjusted $R^2$ of 84.3%, the following represents the best 8 predictor variable regression equation for the natural log of CPU times:

$$
\text{LNCPU} = -4.77 + 0.140 M - 0.0758 K^2 - 0.00172 M^2 - 0.000183 ACONEA^2 + 0.0172 K*M + 0.00914 K*N + 0.0159 K*ACONEA + 0.00288 K*ADEN
$$
The regression output and related plots for this equation are presented in Figures 9.22 through 9.24. The p-values indicate that all predictor variables make a significant contribution to $R^2$ while the charts show reduction in heteroscedasticity.

![Figure 9.21 - Plot of Adjusted R2 versus The Number of Model Variables for CPU Time](image)

It is interesting to see if the same predictor variables may be used for both LNY and LNCPU. Choosing the predictor variables used in our final LNY regression equation, the corresponding equation for LNCPU is

$$\text{LNCPU} = -7.58 + 1.09K + 0.159M - 0.135K^2 - 0.00176M^2 + 0.0121KM + 0.00872KN + 0.0114K\text{ACONEA} + 0.0110\text{ADEN}.$$
The regression equation is
\[
\text{LNCPU} = -4.77 + 0.140 M - 0.0750 K2 - 0.00172 M2 - 0.000163 AC2 + 0.0172 KM \\
+ 0.00914 KN + 0.0159 KAC + 0.00288 KADEN
\]

<table>
<thead>
<tr>
<th>Predictor</th>
<th>Coef</th>
<th>Stdev</th>
<th>t-ratio</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>-4.7686</td>
<td>0.0851</td>
<td>-56.02</td>
<td>0.000</td>
</tr>
<tr>
<td>M</td>
<td>0.1400</td>
<td>0.0052</td>
<td>26.86</td>
<td>0.000</td>
</tr>
<tr>
<td>K2</td>
<td>-0.0757</td>
<td>0.0020</td>
<td>-37.33</td>
<td>0.000</td>
</tr>
<tr>
<td>M2</td>
<td>-0.0017</td>
<td>0.0006</td>
<td>-26.69</td>
<td>0.000</td>
</tr>
<tr>
<td>AC2</td>
<td>-0.0002</td>
<td>0.0000</td>
<td>-18.38</td>
<td>0.000</td>
</tr>
<tr>
<td>KM</td>
<td>0.0172</td>
<td>0.0005</td>
<td>34.05</td>
<td>0.000</td>
</tr>
<tr>
<td>KN</td>
<td>0.0091</td>
<td>0.0001</td>
<td>66.46</td>
<td>0.000</td>
</tr>
<tr>
<td>KAC</td>
<td>0.0159</td>
<td>0.0002</td>
<td>65.80</td>
<td>0.000</td>
</tr>
<tr>
<td>KADEN</td>
<td>0.0029</td>
<td>0.0001</td>
<td>24.67</td>
<td>0.000</td>
</tr>
</tbody>
</table>

\[ s = 1.071 \quad R^2 = 84.3\% \quad R^2(adjust) = 84.3\% \]

Analysis of Variance

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SS</th>
<th>MS</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>8</td>
<td>31529.6</td>
<td>3941.2</td>
<td>3435.91</td>
<td>0.000</td>
</tr>
<tr>
<td>Error</td>
<td>5103</td>
<td>5853.4</td>
<td>1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>5111</td>
<td>37383.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SEQ</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>1</td>
<td>1317.3</td>
<td></td>
</tr>
<tr>
<td>K2</td>
<td>1</td>
<td>952.5</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>1</td>
<td>1065.1</td>
<td></td>
</tr>
<tr>
<td>AC2</td>
<td>1</td>
<td>4555.0</td>
<td></td>
</tr>
<tr>
<td>KM</td>
<td>1</td>
<td>1418.8</td>
<td></td>
</tr>
<tr>
<td>KN</td>
<td>1</td>
<td>7814.8</td>
<td></td>
</tr>
<tr>
<td>KAC</td>
<td>1</td>
<td>5121.0</td>
<td></td>
</tr>
<tr>
<td>KADEN</td>
<td>1</td>
<td>799.7</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9.22 - Regression Output for Log CPU Time Equation (Minitab 10.51)
Figure 9.23 - Normal Probability Plot for Log CPU Time (Minitab 10.51)

Figure 9.24 - Plot of Standardized Residuals vs. Fitted Values for Log CPU Time

(Minitab 10.51)
The regression equation is
\[ \text{INCPU} = -7.58 + 1.09 \times K + 0.159 \times M - 0.135 \times K2 - 0.00176 \times M2 + 0.0121 \times KM + 0.00872 \times KN + 0.0114 \times KAC + 0.0110 \times ADEN \]

<table>
<thead>
<tr>
<th>Predictor</th>
<th>Coef</th>
<th>Stdev</th>
<th>t-ratio</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>-7.5753</td>
<td>0.1557</td>
<td>-48.66</td>
<td>0.000</td>
</tr>
<tr>
<td>K</td>
<td>1.09222</td>
<td>0.05701</td>
<td>19.16</td>
<td>0.000</td>
</tr>
<tr>
<td>M</td>
<td>0.159210</td>
<td>0.005533</td>
<td>28.77</td>
<td>0.000</td>
</tr>
<tr>
<td>K2</td>
<td>-0.135258</td>
<td>0.005025</td>
<td>-26.92</td>
<td>0.000</td>
</tr>
<tr>
<td>M2</td>
<td>-0.00175745</td>
<td>0.00006573</td>
<td>-26.74</td>
<td>0.000</td>
</tr>
<tr>
<td>KM</td>
<td>0.0120727</td>
<td>0.0006306</td>
<td>19.15</td>
<td>0.000</td>
</tr>
<tr>
<td>KN</td>
<td>0.0087174</td>
<td>0.0001103</td>
<td>79.02</td>
<td>0.000</td>
</tr>
<tr>
<td>KAC</td>
<td>0.0114148</td>
<td>0.0001303</td>
<td>87.64</td>
<td>0.000</td>
</tr>
<tr>
<td>ADEN</td>
<td>0.0109984</td>
<td>0.0005378</td>
<td>20.45</td>
<td>0.000</td>
</tr>
</tbody>
</table>

\[ s = 1.089 \quad R\text{-sq} = 83.8\% \quad R\text{-sq(adj)} = 83.8\% \]

Analysis of Variance

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SS</th>
<th>MS</th>
<th>F</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>8</td>
<td>31336.2</td>
<td>3917.0</td>
<td>3305.63</td>
<td>0.000</td>
</tr>
<tr>
<td>Error</td>
<td>5103</td>
<td>6046.0</td>
<td>1.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>5111</td>
<td>37383.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9.8 Prediction of Mean Response and New Observations

An important use of a regression equation is the ability to use the model to predict the mean response or a new observation for a given combination of the predictor variables.
variables. Predictions can be in the form of a point estimate or a confidence interval. A point estimate provides one number as the predicted value of the response variable. This predicted value is a rough approximation of where the response variable might lie. The point estimate for the mean response at a given combination of predictor variables is equivalent to the point estimate of a new observation. A confidence interval goes a step further by taking the estimated variation into account. A confidence interval provides two numbers between which the mean response or the new observation is predicted. While centered at the same value, the point estimate, the confidence interval for a new observation is typically wider than that for the mean response. Equations for calculating point estimates and confidence intervals are fairly standard and detailed discussion of them can be found in a text on linear models such as (Neter et al. 1990). A quick summary of regression prediction calculations is provided here.

The calculation of a point estimate is less complicated than the calculation for a confidence interval. Let the values of the predictor variables for a given combination be denoted as $X_{h1}, \ldots, X_{h,p-1}$, and the point estimate of the mean response for the response variable be denoted as $\hat{Y}_h$. We can define the vector $X_h$ as

\[
X_h = \begin{bmatrix}
1 \\
X_{h1} \\
\vdots \\
X_{h,p-1}
\end{bmatrix}.
\]

The point estimate can be estimated as $\hat{Y}_h = X_h^T b$ where $b$ is the row vector of the estimated model coefficients. In other words, the point estimate for the mean response
and a new observation can be calculated by incorporating a given combination of the
predictor variables into the regression model and cranking out the result.

A 1-\( \alpha \) confidence interval for a mean response is calculated using
\[
\hat{Y}_h \pm t(1-\alpha/2; n-p)s\{\hat{Y}_h\}.
\]
Here, \( \hat{Y}_h \) is the point estimate as described above; \( t(1-\alpha/2; n-p) \) is a value from the
student’s t-distribution; and \( s\{\hat{Y}_h\} \) is the estimated standard deviation for the estimated
mean response. The estimated variance \( s^2\{\hat{Y}_h\} \) is given by
\[
s^2\{Y_h\} = X_h^T s^2\{b\} X_h = MSE\left(X_h^T (X^T X)^{-1} X_h\right).
\]
Here, \( MSE \) is the mean squared error from the model calculated during the regression
analysis, and the matrix \( X \) is the matrix of predictor variable values from the model-
building set.

A 1-\( \alpha \) prediction interval for a new observation is calculated using
\[
\hat{Y}_h \pm t(1-\alpha/2; n-p)s\{\bar{Y}_{h(new)}\}.
\]
Here, \( \hat{Y}_h \) is the point estimate as described above; \( t(1-\alpha/2; n-p) \) is a value from the
student’s t-distribution; and \( s\{\bar{Y}_{h(new)}\} \) is the estimated standard deviation for the estimated
new response observation. The estimated variance \( s^2\{\bar{Y}_{h(new)}\} \) is given by
\[
s^2\{\bar{Y}_{h(new)}\} = MSE + s^2\{\hat{Y}_h\} = MSE + X_h^T s^2\{b\} X_h = MSE\left(1+X_h^T (X^T X)^{-1} X_h\right).
\]

Recall that the response variable of interest, KEEXT, has been transformed using
the natural logarithm to obtain the response variable LNY which is used in the final
regression model. Therefore, to make inferences about the values of KEEXT for various
combinations of the predictor variables, we must first make inferences for LNY and then convert these predictions with a reverse transformation. This is accomplished using

\[
\text{LNY} = \ln(\text{KEEXT})
\]

\[
\text{KEEXT} = e^{\text{LNY}}.
\]

Using the calculations described above, confidence intervals for the mean number of MOLP extreme points and prediction intervals for new observations of the number of MOLP extreme points are presented in Table 9.2. In addition, corresponding intervals for the CPU time is presented in Table 9.3. For both tables, the regression equations contained the exact same predictor variables.

Table 9.2: Inferences for the Number of MOLP Extreme Points (KEEXT).

<table>
<thead>
<tr>
<th>K</th>
<th>M</th>
<th>N</th>
<th>ADEN</th>
<th>ACONTA</th>
<th>Point Estimate</th>
<th>95% C.I. for Mean</th>
<th>95% P.I. for New Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>25</td>
<td>100</td>
<td>90</td>
<td>10.1</td>
<td>(9.1, 11.2)</td>
<td>(1.3, 76.4)</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>125</td>
<td>100</td>
<td>90</td>
<td>244.8</td>
<td>(220.8, 271.5)</td>
<td>(32.4, 1851.4)</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>25</td>
<td>100</td>
<td>90</td>
<td>58.8</td>
<td>(53.9, 64.1)</td>
<td>(7.8, 443.9)</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>125</td>
<td>100</td>
<td>90</td>
<td>4116.7</td>
<td>(3708.6, 4569.2)</td>
<td>(544.3, 31132.3)</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>25</td>
<td>100</td>
<td>90</td>
<td>254.2</td>
<td>(232.9, 277.5)</td>
<td>(33.6, 1921)</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>100</td>
<td>100</td>
<td>90</td>
<td>20247.8</td>
<td>(18560.7, 22086)</td>
<td>(2679.3, 153001)</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>25</td>
<td>100</td>
<td>90</td>
<td>818.5</td>
<td>(744.9, 899.4)</td>
<td>(108.3, 6187)</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>25</td>
<td>100</td>
<td>90</td>
<td>60457.7</td>
<td>(54479.9, 67124.2)</td>
<td>(7993.6, 457256.5)</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>25</td>
<td>100</td>
<td>90</td>
<td>19614</td>
<td>(17714.4, 2172)</td>
<td>(259.4, 14832.9)</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>25</td>
<td>100</td>
<td>75</td>
<td>92790.5</td>
<td>(82149.8, 104799.1)</td>
<td>(12256.4, 702429.3)</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>25</td>
<td>100</td>
<td>90</td>
<td>3498.2</td>
<td>(3114.1, 3929.2)</td>
<td>(462.2, 26473.5)</td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>25</td>
<td>100</td>
<td>75</td>
<td>31166.5</td>
<td>(27973.1, 34721)</td>
<td>(4120, 235743.6)</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>25</td>
<td>100</td>
<td>90</td>
<td>4642.9</td>
<td>(4007.4, 5378.7)</td>
<td>(612.3, 35207)</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>25</td>
<td>100</td>
<td>15</td>
<td>2232.3</td>
<td>(1862.9, 2723)</td>
<td>(286, 17140.5)</td>
</tr>
</tbody>
</table>

Table 9.3: Inferences for the CPU Time (CPUTIM)
9.8 Concluding Remarks

From the tables in the previous section, it is clear that the numbers of MOLP extreme points (KEEXT) and the CPU times vary a great deal over the range of data in the study. Several interesting items can be seen in these tables:

1. The number of objectives (K) seems to have the greatest affect on the number of extreme points. This is evident in the regression equation as K appears in five of the eight predictor variables. In Table 9.2, it can be seen that varying the number of objectives from two to eight while keeping all other variables constant yields an increase in the point estimate for the predicted mean response from 10.1 extreme points to 4642.9. Similarly, the point estimate for the predicted mean response for CPU time varies from 0.5 to 383.8 Pentium seconds as shown in Table 9.3.

2. The next greatest effect on the number of extreme points is the number of constraints (M). M appears as part of three of the eight predictor variables. Evidence of this in Table 9.2 can be seen with the predictions for five objectives. Notice that with all else constant,
increasing the number of constraints from 10 to 50 results in an increase in the point estimate for the mean number of extreme points from 818.5 to 60457.7. The point estimate for the CPU time for 5 objectives varies from 47 to 4517 Pentium seconds.

3. While the effects of the three variables, N, ADEN, and ACONEA, do not appear as dramatic, they are significant. (A regression for LNY performed with only K and M terms yielded an $R^2$ of only 48.8%) These effects are noticeable in the charts.

4. As usual with regression analysis, tighter bounds on the prediction for mean response are available compared to the bounds for prediction of new observations. In this case, the difference is striking for larger problems. The large prediction intervals are a result of the large variance in number of extreme points for large problems.

Regression analysis is an important and popular method for statistical analysis of experimental data. Regression analysis has a long track record in obtaining useful information concerning the relationships between experimental and observational variables. In this chapter, it has been demonstrated how regression analysis can be useful in gaining insight into the nature of the efficient set for MOLPs. In particular, we have shown how knowledge of MOLP characteristics can give an a priori estimate of the size of the efficient set. Estimates of this sort can prove useful to researchers who wish to experiment using random problems of specific size. In addition, an a priori estimate of the size of the efficient set may be useful to analysts in choosing methods for reducing this potential numbers of solutions for specific problem instances. Future research into this area may include refinement of the statistical model or the development of theoretical models for the number of MOLP extreme points for given problem parameters.
CHAPTER 10
CONE DECOMPOSITION FOR THE SOLUTION OF EFFICIENT EXTREME POINTS IN PARALLEL

10.1 Introduction:

An important solution strategy, at least in the research arena, for MOLP as well as the subclass of problems known as MONF is to generate the entire set of efficient solutions. The main drawback of generating the entire set of efficient solutions is that the set can be very large. The generation of the efficient set can become time and memory intensive and fast algorithms are highly desirable. The ability to use multiple processors for computing has allowed for the improvement of many algorithms. Often, this is accomplished by dividing the work over several processors that then perform their tasks separately. The solutions for the processors are finally combined to determine the final solution set for the original problem. This process is known as parallelization.

In this chapter, we describe an idea for a parallel method of obtaining the solution set of a MOLP. The method is based on decomposition of the criterion cone (the cone generated by gradients of the objective functions of the MOLP). This cone is an important part of the theory involving vector maximum algorithms used with MOLPs. In Section 10.2, we discuss the how theory of vector maximum algorithms and characteristics of the efficient set are related to the concept of the criterion cone. In Section 10.3, we will describe an idea for parallelizing the vector maximum algorithm via
cone decomposition. In Sections 10.4 and 10.5, we will describe some computational tests and results. Finally, in Section 10.6, we will discuss the results of the experimentation and comment on future directions.

10.2 Basic Vector Maximization Theory:

We have seen that in general, an MOLP can be formulated as follows:

$$\max \{ Cx = z | x \in S \}$$

Here, \( z \) represents a vector of objective values to be maximized, \( C \) is a matrix of objective function coefficients, \( x \) represents a vector of decision variables and \( S \) can be defined as:

$$S = \{ x \in \mathbb{R}^n | A x = b, x \geq 0, b \in \mathbb{R}^m \}$$

Here, \( A \) is a matrix of constraint coefficients that is assumed to be of full row rank and \( b \) is a vector of RHS coefficients. \( S \) is typically called the feasible region. The overall goal is to find a point in \( S \) such that the values in the vector \( z \) are simultaneously maximized.

MOLPs can be considered as a subset of a larger set of problems called vector-maximum problems. In this sense, the idea is to maximize the \( z \)-vector of objective values. Since, with an MOLP, there is usually not one solution that simultaneously maximizes the values of the \( z \)-vector, the problem becomes one of characterizing the efficient set. Let \( \Lambda \) denote the set of weighting vectors such that:

$$\Lambda = \{ \lambda \in \mathbb{R}^n | \lambda_i > 0, \sum_{i=1}^k \lambda = 1 \}.$$ 

It has been shown that if \( x^* \in S \) is efficient, then there exists a \( \lambda^* \in \Lambda \) such that \( x^* \) is a maximum solution of \( \max \{ \lambda^* C x | x \in S \} \). Therefore, each efficient point can be thought of as maximizing at least one vector which is a weighted combination of the objective
functions. When vector maximum algorithms are used with MOLPs, they are used to maximize these weighted vectors.

A mathematical concept that is particularly useful when using a vector maximum algorithm with an MOLP is that of a cone. Let \( v \in V \subset R^n, V \neq \emptyset \). Then for all scalars \( \alpha \geq 0 \), \( V \) is a cone if and only if \( \alpha v \in V \). In addition, if we consider a set of \( k \)-vectors \( \{v^1, v^2, \ldots, v^k\} \) and a set \( V \) where \( V = \left\{ v \in R^n \left| v = \sum_{i=1}^{k} \alpha_i v^i, \alpha_i \geq 0 \right. \right\} \), then \( V \) consists of all nonnegative linear combinations of the \( v^i \) and is a convex cone generated by the set \( \{v^1, v^2, \ldots, v^k\} \). The \( v^i \) are called generators of the cone \( V \).

The cone concept can be related to an MOLP as follows. First, think of the gradient \( c^i \) of each objective function as a \( k \)-vector emanating from the origin. The set of gradients \( \{c^1, c^2, \ldots, c^k\} \) generate a convex cone called the criterion cone. Let \( v^* = \lambda^* C \), where \( \lambda^* \in \Lambda \), then \( v^* \) is in the relative interior of the criterion cone. Since \( v^* = \lambda^* C \) is a weighted combination of the objective function gradients, each efficient point maximizes at least one vector that is a member of the criterion cone. Therefore, vector maximum algorithms are used to maximize vectors in the relative interior of the criterion cone. It is important to note that the relationship between efficient extreme points and weighting vectors from the interior of the criterion cone is not one-to-one. Each efficient extreme point generally will maximize many vectors from the criterion cone.

10.3. Decomposition of the Criterion Cone

Recall that the essence of parallelizing a problem is to divide the work of an algorithm into smaller parts that can be worked on separately by multiple processors.
With this in mind, the main idea for this research is to cut the criterion cone into smaller sections or sub-cones and then solve for each sub-cone with different processors. The resulting solutions for each sub-cone problem can then be combined into a solution for the original larger problem. In this section, we discuss this idea, the potential benefits and the potential difficulties.

Consider the following simple 2-variable and 2-objective MOLP which is graphically illustrated in Figure 10.1:

\[
\begin{align*}
\text{Max:} & \quad x_1 + 6x_2 \\
\text{Max:} & \quad 7x_1 + x_2 \\
\text{s.t.:} & \quad x_1 \leq 8 \\
& \quad x_2 \leq 8 \\
& \quad 3x_1 + x_2 \leq 26 \\
& \quad x_1 + x_2 \leq 12 \\
& \quad x_1 + 3x_2 \leq 26 \\
& \quad x_1 \geq 0; \quad x_2 \geq 0
\end{align*}
\]

In Figure 10.1, the vectors \(c^1\) and \(c^2\) represent the gradients defined by the two objective functions and the bounded area represents the feasible region defined by the constraints. The points \(x^1\) through \(x^6\) represent the extreme points on the boundary of the feasible region. Recall that points in the efficient set can be found using vectors which lie in the relative interior of the cone generated by the objective function gradients. It is relatively easy to see in the figure that the efficient set for this problem includes all points on the boundary of the feasible region that lie between the extreme points \(x^2\) and \(x^5\). The efficient set here includes and can be characterized by the extreme points \(x^2, x^3, x^4,\) and \(x^5\).
Suppose now that we introduce a new vector, \( c^* \), in order to divide the criterion cone into two sub-cones. Here, vectors in the relative interior of sub-cone 1 defined by the gradients \( c_1 \) and \( c^* \) will lead us to the efficient extreme points \( x_2, x_3, \) and \( x_4 \). Similarly, vectors in the relative interior of the second sub-cone 2 defined by the gradients \( c^* \) and \( c_2 \) will lead us to the efficient extreme points \( x_4 \) and \( x_5 \).

Two important points are illustrated by this example. First, we see here that we can divide the efficient set into smaller subsets using cone decomposition. If it turns out that the time to solve a problem is directly related to the number of extreme points, then in the ideal case we could cut our solution time for this MOLP in half. This could result in significant improvements in the solution time for large problems. Second, the subsets are apparently not mutually exclusive since the point \( x_4 \) appears in both subsets. Thus the ideal case will not be possible for most problems.
Another complication that could arise using the decomposition strategy shown in Figure 10.1 is that there is a possibility that some efficient extreme point solutions of the original problem could be missed. This problem arises because the vector maximum algorithm uses vectors from the relative interior of a criterion cone. This means that points corresponding to weights falling on the boundary of the cone may not be indicated as part of the solution set. While this is proper for the solution of the original problem, if we divide the cone using a vector from the interior of the original criterion cone, we in effect are defining this vector to be on the boundary of the sub-cones. It is possible then that efficient extreme points of the original problem corresponding to the weights of the subdividing vector may not be found when solving for the sub-cone MOLPs.

The idea of sub-dividing the criterion cone of a 3-objective MOLP can also be conveniently illustrated using a weight-space graph such as that used in the software TRIMAP. TRIMAP (Climaco and Altunes (1989), and Climaco, Altunes, and Alves, (1993)) is an interactive linear programming software package for solving tri-criteria MOLPs. In TRIMAP, the weighting space is shown as in Figure 10.2. The weights $\lambda_1$, $\lambda_2$, and $\lambda_3$ are associated with each objective of the problem. In the TRIMAP-style graph of Figure 10.2, the weight space is decomposed into sections. Each section corresponds to the set of all weighting vectors that are associated with a particular efficient extreme point solution. The map of Figure 10.2 illustrates the weight space for a tri-criteria problem with 48 efficient extreme points. The dashed lines in the figure show one possible method of dividing the weight space into sub-problems. This type of subdivision would result if a new vector were created by simple adding together the original vectors.
In this figure, we can see that we can indeed create sub-problems that will include substantially smaller numbers of extreme points than the original. For instance, the weights of Section I represent 19 extreme points, the weights of Section II represent 20 extreme points, and the weights of Section III represent 20 extreme points. The phenomenon of overlap is also illustrated here. The overlapping points correspond to the decomposition sections that lie in two or more of the sub-sections. From the figure and the number of efficient extreme points that correspond to each subsection it is evident that there are at least 11 efficient extreme points that will be found when solving two or more of the sub-problems. Also in the figure, the number of extreme points found from each sub-section is relatively balanced. This may not be the case in general.

![Figure 10.2 - TRIMAP style representation of 3 Objective MOLP weight space](image)

Based on these ideas, several questions come to mind. How much of a gain in solution time can we expect for various problem sizes? What is the extent of possible overlap particularly for larger problems when parallel implementations are of interest?
How balanced are the numbers of efficient extreme points over the cone subdivisions and would this effect our decomposition strategy? Are there different strategies for dividing the criterion cone in order to reduce overlap and reduce the chance of missing extreme points? We address these questions along with others when describing our empirical research in the following sections.

10.4. Experimental Design

In order to study the effects of decomposition of the criterion cone, an empirical experiment was designed and implemented. The program ADBASE was used in this experiment to generate and solve MOLPs for efficient extreme points. ADBASE is a vector maximization code that has achieved widespread distribution and has been extensively used in the study of MOLPs. Its features include, among others, a random problem generator and a solver for computing all efficient extreme points and all unbounded efficient edges. For this experiment, the random problem generator was used to create MOLP problems with varying problem parameters. The problem sizes used are listed in Table 10.1. The experimentation was performed on an IBM-3270 mainframe at the University of Georgia.

<table>
<thead>
<tr>
<th>MOLP Parameter</th>
<th>Values Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Objectives</td>
<td>3, 4, 5</td>
</tr>
<tr>
<td>Number of Constraints</td>
<td>20, 30, 40, 50, 60</td>
</tr>
<tr>
<td>Number of Variables</td>
<td>10, 25, 50, 75, 100</td>
</tr>
</tbody>
</table>
For each combination of the problem parameters, a number of original problems were generated and solved for all efficient extreme points. In addition, each original problem was decomposed into several sub-problems that were also solved for all efficient extreme points. Data recorded for each MOLP/Sub-problem set includes: the number of original problem efficient extreme points; the CPU solution time for the original problem; the number of efficient extreme points for each sub-problem; and the CPU solution time for each sub-problem. In this way, data was collected for 998 MOLP/Sub-problem sets.

The sub-problem decomposition for each of the MOLPs are generated in a simple manner. First, adding together the normalized original criterion vectors created a “central” criterion vector. Then replacing one of the original objectives with the new objective vector one-at-a-time generates a number of sub-problems equal to the original number of objectives. Thus for an original n-objective MOLP, n sub-problems are generated. As we will see later, other strategies are possible and may be more desirable.

10.5. Computational Results

In order to answer the questions of interest, several measures are calculated based on the data collected. These measures are based on the number of objectives and CPU time for the original MOLP and for the corresponding sub-problems. The following variables will be useful in understanding the definitions of these measures:

\[ N = \text{number of original MOLP objectives} \]
\[ \text{EXT}_{i} = \text{number of efficient extreme points for the original MOLP i} \]
\[ \text{EXT}_{i,j} = \text{number of efficient extreme points for sub - problem j of MOLP i} \]
\[ \text{CPU}_{i} = \text{solution time for the original MOLP i} \]
\( \text{CPU}_{ij} = \text{solution time sub - problem } j \text{ of MOLP } i \)

A measure of the gain in time possible by solving the sub-problems in parallel can be determined as follows:

\[
\text{PERCENT TIME GAIN} = \frac{\text{CPU}_i - \text{MAX(\text{CPU}_{i,k})}_{k=1}^j}{\text{CPU}_i}
\]

This formula determines the possible gain in time due to parallelizing the problem by first subtracting the longest sub-problem time from the original time. In order to compare the time gain for problems of different sizes the potential gain is divided by the original problem time to calculate a percentage. In Figure 10.3, this measure is charted against the original number of extreme points. The data for this measure appears quite promising as can be seen in the Figure. The percentage time gain quickly increases greater than 30\% for modest sized problems and for large problems the gain ranges from 50\% to 70\%.

![Figure 10.3 - Percentage Gain in Solution Time Vs. Number of Original Efficient Extreme Points](image_url)
To ensure maximum efficiency when solving a MOLP in parallel, it would be nice if the original problem efficient extreme points were spread out in a balanced fashion over the sub-cones. For example, for a MOLP that is divided into 3 sub-problems, it would be desirable for about one-third of the efficient extreme points to appear for each sub-problem. For sub-problems when this is not the case the problem could be said to be in imbalance. This imbalance could also be of interest in describing the general characteristics of MOLP solution sets. Imbalance would most likely occur when certain areas of the solution set are denser with solutions than other areas.

A measure of imbalance can be defined in a manner that is similar to the statistical concept of variance. For the imbalance measure, we can calculate a difference between the actual percentage of efficient extreme points in a sub-cone and the percentage that would be there under perfect balance. We then square these differences and find an average of these squared differences. The calculation is as follows:

$$\text{IMBALANCE} = \frac{\sum_{k=1}^{N} \left( \frac{\text{EXT}_{i,k}}{\text{EXT}_i} - \frac{1}{N} \right)^2}{N}$$

In the graph of Figure 10.4, we can see that the imbalance over the sub-cones tends to improve for large problems.

Recall from the earlier discussion that one potential complication is related to overlapping solution sets for the MOLP sub-problems. When there is overlap in these solution sets, the processors will essentially be solving for the overlapping points multiple times. Thus in the presence of overlapping points our time gain is not ideal. We can measure the number of overlapping points with the following measures:
TOTAL OVERLAP = \left( \sum_{k=1}^{j} EXT_{i,k} \right) - EXT_i

PERCENT OVERLAP = \frac{\text{TOTAL OVERLAP}}{\sum_{k=1}^{j} EXT_{i,k}}

Figure 10.4 - Sub-Problem Imbalance Vs. Number of Original Efficient Extreme Points

Here, we are measuring the total overlap for each of the problems and it is then converted into percentages for comparisons between problem sizes. Figures 10.5 and 10.6 show the relationship of these two measures versus the number of original problem extreme points for the data. As the charts illustrate, the total number of overlapping points increases with the problem size as measured by the number of original efficient extreme points. On the other hand, these numbers as a percentage of the total sub-problem efficient extreme points show a decrease as the problem size increases. Also of interest are the definite differences between these measures for the different values of number of MOLP objectives. The charts illustrate that overlap increases at a greater rate with the number of objectives.
Another potential complication that seems opposite to that of overlap is the potential for missing some of the original problem efficient extreme points when solving the sub-problems. In order to study the extent of this potential problem, counts were taken of the number of times that particular efficient extreme points were found by the
sub-cones for each 3- and 4-objective MOLP. Average counts for each problem size are shown in Table 10.2. The average number of points that were not found in any sub-problems is shown the column marked zeros. The fact that both values are greater than zero attests to the fact that it is possible to miss an efficient extreme point with the current strategy. The relatively low values demonstrate that this will occur for only a very small percentage of the problems.

Table 10.2 - Average Number of Overlap Counts.

<table>
<thead>
<tr>
<th>MOLP Size</th>
<th>Zeros</th>
<th>Ones</th>
<th>Twos</th>
<th>Threes</th>
<th>Fours</th>
<th>Fives</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 Objectives</td>
<td>0.01</td>
<td>974.98</td>
<td>75.03</td>
<td>1.49</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>4 Objectives</td>
<td>0.07</td>
<td>6727.19</td>
<td>1414.41</td>
<td>110.31</td>
<td>2.81</td>
<td>NA</td>
</tr>
</tbody>
</table>

The possibility exists that other decomposition strategies could obtained reduced solution times while also reducing the possible complicating factors discussed above. In order to test this, an alternative strategy was formed for three objective problems. A graphical representation of the strategy is shown in Figure 10.7. For this strategy, we form three new criterion vectors instead of one. Each of the new criterion vectors is formed by a simple addition of two normalized versions of the original criterion vectors. Then four sub-problems are formed using three of the now six objectives in such a way as to correspond to the Figure. With this strategy, it is hoped that there will be no missed efficient extreme points while we obtain at least as good if not better performance on the other measures. Indeed, since we are dividing the overall cone into more, smaller sections we should see an improvement in the time gain.
Using the new strategy, data for 374 three-objective MOLPs were obtained representing all combinations of the problem size parameters. The Figures, 10.8 through 10.10, provide a comparison between the new decomposition strategy for three objective MOLPs and the previous strategy. While a small improvement in the percentage gain in solution time is seen, the measures of total overlap and imbalance show no significant difference. More notably, no efficient extreme points were missed for any MOLP solved using the new strategy.

Figure 10.7 - Alternative 3-Objective Decomposition Strategy
Figure 10.8 - Comparison of Percentage Gain in Solution Time

Figure 10.9 - Comparison of Total Overlap
10.6. Concluding Remarks

The results presented here provide positive evidence of the potential of the cone decomposition strategy for parallelizing large MOLPs. Gains in solution time greater than 50% are possible. From these data we see that as problem size increases:

- the percentage time gain increases.
- the total sub-problem overlap increases.
- the percentage of sub-problem overlap decreases.
- the number of extreme points are more balanced across the sub-problems.

In addition, an alternative cone decomposition has been found which eliminates the complication of missing efficient extreme points while maintaining the potential time savings. Further research on this topic could include an actual parallel implementation as well as more exploration of alternative decomposition strategies. Since MONF are generally quite large, this strategy may also be applied with the specialized MONF codes presented in earlier chapters.
CHAPTER 11

CONCLUSION

The research presented here has dealt with various aspects of the Multiple Objective Network Flow (MONF) problem and related issues of MOLP. The primary purpose of the research has been to understand the nature of an important class of decision models and how these models are affected when the attempt is made to bring them closer to reality with the inclusion of multiple objectives. This has involved the combination of two areas of decision science research, network flows and multiple criteria decision making (MCDM). The approach has been to combine the sophisticated interactive procedures of MCDM with the high-speed algorithms of network flow theory that take advantage of the network flow model’s special structure. In addition, the approach has been to support the suppositions made with empirical research and computational experience. In this chapter, a summarization of the major questions answered and a discussion of further questions that have arisen conclude the dissertation. Included are discussions of ideas for future research and possible practical applications.

11.1 Research Summary

The presentation of the research in this dissertation follows a progression that began with a study of the nature of MONF problems using networks with a low number of nodes and arcs and then on to solution methods designed for larger networks. The
quickly expanding nature of the efficient set then led to studies related to MOLPs with very large efficient sets and how to deal with them. In this section, the various conclusions reached at each stage of this progression are discussed.

11.1.1 Solution Characteristics for MONF problems

In order to understand what would be involved in solving MONF, characteristics of the typical MONF solution set were investigated in Chapter 5. Several interesting aspects of the solution set were presented.

As expected, the nature of the solution set varies depending on whether flows on the network are restricted to integer values or are they allowed to take on continuous values. For both cases the solution set usually consists of more than a single solution. This is due to the usual trade-offs that exist between the multiple objectives. Because of this, the search for the DMs’ preferred solution is conducted in two phases. In the first phase, efficient solutions are found and in the second a search among the efficient solutions is conducted.

For the case of continuous flows, the nondominated set resides on the boundary of the feasible region in criterion space. In this case, all nondominated points are supported by hyperplanes derived from the objectives. Points in the nondominated set may then be classified as either extreme points or supported non-extreme points. Several observations were reported about the MONF with continuous variables.

- Observation 1: The feasible region S for the MONF problem with continuous variables is a closed and bounded, convex set.
• Observation 2: Extreme points in decision space correspond to spanning trees on a network.

• Observation 3: If all network parameters are integer, then extreme point solutions of networks are integer valued.

• Observation 4: Vector-Maximum theory of MOLP holds for the set of efficient solutions for the MONF with continuous variables.

• Observation 5: For the MONF with continuous variables, there are no unbounded edges so efficient extreme points and selected convex combinations of efficient extreme points can be used to characterize the efficient set.

• Observation 6: For the MONF with continuous variables, efficient points may be extreme, lie in the interior of an edge, or lie on the interior of an efficient f-facet of the efficient set.

• Observation 7: For the MONF with continuous variables, efficient points on the interior of a facet of the efficient set which are not extreme (facet dimension greater than 0) may not be integer valued.

• Observation 8: Movement along an edge of the efficient set from one efficient extreme point to another (an efficient pivot), corresponds to movement in the network from one spanning tree T1 to another spanning tree T2. This movement can be characterized by a simple cycle, which is fundamental with respect to both T1 and T2.

• Observation 9: For the MONF with continuous variables, efficient points on the relative interior of an edge of the efficient set, which might not be integer valued, correspond to flows which occur in the network while moving from one spanning
tree to another using a fundamental cycle with respect to the adjacent spanning
trees.

- Observation 10: Assuming nondegeneracy, the set of efficient bases is connected.
- Observation 11: When degenerate efficient bases exist, degenerate efficient pivots
  may be required before moving to adjacent efficient extreme points.
- Observation 12: Higher order efficient f-facets (dimension greater than 1) can be
  characterized by convex combinations of efficient extreme points.
- Observation 13: Extreme points and edges of S in decision space do not
  necessarily map to extreme points and edges of Z in criterion space.
- Observation 14: The size of the efficient and nondominated sets tends to grow as
  the problem size in terms of the number of network nodes, the number of network
  arcs, the capacities on the arcs and the size of the criterion cone defined by the
  objectives increases.

The primary conclusion from these observations is that the nondominated set for
the MONF with continuous variables shares characteristics of the MOLP in general. Since
this is the case, sophisticated interactive methods from the MOLP literature are applicable
when searching for a solution to the MONF problem. As many of these methods employ
solution methods from LP, the specialized network LP methods for network flow
problems are also applicable. The combination of network LP methods with the
sophisticated interactive methods of MOLP should result in improved algorithmic
efficiency in the MOLP solution methods. This was explored in later chapters.
As with LP models and contrary to intuition, the MONF constrained to integer variables introduced some interesting twists and difficulties. Observations for this case include:

- **Observation 15:** For the MONF with integer variables, nondominated points may be supported or unsupported.
- **Observation 16:** For the MONF with integer variables, solutions which lie on the convex hull of the feasible region may be extreme, lie in the interior of an edge, or lie on the interior of a higher order facet of the convex hull.
- **Observation 17:** Integer efficient solutions that are members of the convex hull of the feasible region fit observations described for the MONF with continuous variables.
- **Observation 18:** The number of supported nonextreme efficient points along an edge of the convex hull efficient set is equal to D-1. Where D, is equal to the total change along the fundamental cycle in the network when moving from one corresponding spanning tree to another.
- **Observation 19:** Unsupported nondominated points in criterion space lie in a special domination set associated with groups of supported nondominated points.
- **Observation 20:** The number of unsupported efficient points like supported efficient points can depend on the size and shape of the feasible region as determined by the number of nodes, arcs and capacities on the network as well as the size of criterion cone.

The integer case introduces several difficulties. First, some solutions, namely unsupported solutions, exist that may not be found using MOLP methods that rely on
traditional weighting vectors. In addition, while we may hope to treat the nondominated set as continuous, find a potential solution, and then move to the nearest nondominated, integer solution, no dependable method has yet been developed to do this. The proportion of the nondominated set that may be unsupported was explored in this research. Methods for moving from a solution of a continuous nondominated set to the nearest integer solution may is a subject for future research.

For both cases, the observations also demonstrate that the size of the nondominated and efficient sets tend to increase dramatically along with increases in problem size parameters. The relationship between these parameters and the size of the solution sets was explored and is discussed in section 11.1.5.

11.1.2 Size and Proportions of the MONF Efficient Set

In Chapter 6, we described an initial, empirical study designed to provide further insight into the nondominated set and by extension the efficient set. A study of how the size and composition of the nondominated set varies with changes in several MONF parameters was conducted and presented. This study utilized a naïve algorithm for generating all feasible solutions of randomly generated MONF problems. Due to the combinatorial nature of MONF, this study was restricted to small problems by network programming standards. Nevertheless, some interesting aspects of the MONF nondominated set were noted.

Using the naïve algorithm, empirical experiments were conducted by varying several model parameters, solving random models for each combination of model parameters, and recording the number and type of feasible solutions for each model.
Model parameters that were varied for each combination include the number of network nodes, the arc density, the total network supply and demand, the number of objectives and the size of the criterion cone. Only solutions that result from integer flows were considered.

From the observations and empirical investigation that were discussed several conclusions were reached. First, the size of the feasible region and the sizes of the efficient and nondominated sets can get large quickly for MONF. Thus, there is a need for effective solution procedures for searching the solution set in a reasonable amount of time. Second, the presence of nonextreme supported and unsupported nondominated indicates a need for a MONF solution method that can reach these points. Existing network flow solution methods lead primarily to extreme point solutions only. Third, it may be possible to exploit the correspondence between edges of the efficient set and fundamental cycles on the network. By taking advantage of special data structures associated with networks, movements between bases using fundamental cycles can be very fast. This may prove useful for general movements around the efficient set in particular when moving from extreme point to extreme point and approaching points which lie along edges of the efficient set. Fourth, the size of the feasible region is affected only by structural network parameters including the number of nodes, the number of arcs and the total supply/demand and is not affected by the objectives of the problem. Additional network characteristics such as arc capacities and the shape of the network itself also play a role. Fifth, the number of objectives and the average angle of the cone generated by the objectives are the primary factors affecting the size of the nondominated set. For certain average cone angles, the presence of nonextreme nondominated solutions was not detected. Possible
heuristics may be developed based on cone size to indicate whether or not there is a need to search for nonextreme solutions for a given MONF. Finally, a comparison of the range of values obtained for each category demonstrates that it is possible that more of the nondominated set is unsupported than supported for the integer MONF.

11.1.3 Finding the MONF Efficient Set

To provide the capability of finding all efficient extreme points of the MONF efficient set, the software MONFSOLV was developed and described in Chapter 7. While impractical as a tool for working with a DM to provide a solution for a MONF problem, the software was created as a research tool for use in further study of the MONF efficient and nondominated sets. Tests of MONFSOLV show that it is capable of listing all efficient and nondominated extreme points for the MONF.

Several conclusions were drawn from the test results. It was found that MONFSOLV works best when adding to the master list only those adjacent efficient bases resulting from a nondegenerate pivot with flow change. While the results presented do not indicate any problems, this must be used with caution since with the presence of degenerate solutions there is a possibility that the efficient set is not connected and some points could be missed.

It was also shown that the MONFSOLV software requires less CPU time than ADBASE when solving the same MONF. The time savings are due to the specialized nature of MONFSOLV for minimum cost network formulations. However, the time savings are small relative to the time savings of single objective network flow solvers over general-purpose LP solvers. While MONFSOLV can probably be made to run faster as it
is improved, it will probably never come close to the hundred times improvement of single objective network solvers. While the single objective MCNF solver carries out all optimization steps on the network, this is not the case for MONFSOLV. Network operations only make up a portion of the work required, namely finding the initial efficient basis and then crashing to other efficient bases. The subproblem tests for efficiency make up the majority of LP program models solved in the algorithm. Since these subproblem tests do not have a network structure, no improvement in solution time is seen with the addition of network flow methods. Further improvements to MONFSOLV will focus on improving the speed of the subproblem test or reducing the number of subproblem tests required.

As seen with the Naïve algorithm tests, the tests using MONFSOLV also indicate that the size of the MONF, in terms of number of nodes, number of arcs, number of objectives, and the criterion cone, does affect the sizes of the sets of efficient and nondominated extreme points. As might be expected, the sizes of the sets tend to increase as the problem sizes increase but the variation in set sizes increase as well. Also, there is a large difference between size of the set of efficient extreme points and the size of the set of nondominated extreme points for MONF. Two causes for this come to mind. First, the differences could be due to degeneracy. Network flow formulations are notorious for their large amounts of degenerate solutions. Another cause may lie in the compression effects that occur when the efficient set from $n$-space is mapped to the lower dimensional $k$-space of criterion space. Additional studies of the effects of degeneracy on the MONF efficient and nondominated sets may prove useful.

11.1.4 Interactive Solution for MONF
In Chapter 8, an interactive solution algorithm for MONF and software based on this algorithm were described. This algorithm combined the Interactive Weighted Tchebycheff (IWT) procedure with network flow methods to provide a tool for decision aid consistent with the more sophisticated, interactive methods employed in multiple criteria decision-making. They employ results from network flow theory that have the potential for significantly improving the solution time for large MONF models. In addition, a foundation is provided for exploring an interesting combination of decision aid that has received relatively little attention to date. Several future directions may be explored. These include:

- Finding nearest efficient, integer solution: Using the IWT method combined with the network with side constraints algorithm will typically produce solutions on the convex hull of the network feasible region. This means that non-integer solutions are the norm. So far, no guaranteed algorithm for moving to the nearest nondominated efficient solution has been developed.

- Combine algorithm with Interactive Weighted Sums (IWS): The IWS may be viewed as a complementary approach to the IWT. The IWS sub-problems may be solved as pure network flow models taking full advantage of the very fast algorithms developed for their solution. A major disadvantage of IWS is that it is only capable of generating corner point solutions. Steuer (1995) describes a method in which the IWT and IWS may be combined. In early stages of the combined method, IWS may be used to narrow the solution space under consideration. In later stages, the IWT may then be used to focus on an optimal, possibly non-corner point solution.
• Empirical testing with real-world problems: The major testing thus far with both the TchebyMONF package and the MONFSOLV software discussed in Chapter 7 has been with randomly generated problems. Applications of network models in such areas as telecommunications networks, computer network design, and financial decision making should be of great interest.

• Further work with the unique formulation of the Tchebycheff LP: The network simplex with side constraints algorithm of Kennington and Helgason that has been used for this work attempts to take advantage of the partial network structure of the LP. The Tchebycheff LP formulated for TchebyMONF retains this partial network structure. In addition, there may be further structural characteristics of the Tchebycheff LP that will allow for further algorithmic improvements.

11.1.5 Size of MOLP Efficient Set versus Problem Size

As noted earlier, the efficient set of MONF problems, as well as the more general class of MOLP problems tends to increase in size as various problem parameters increase. In Chapter 9, we described the development of a model for characterizing the relationship between the size of the efficient set and the problem parameters that may affect it. The approach was to develop a regression model that may be used to analyze the effect of each problem parameter on efficient set size. It was hoped that the model would aid in forecasting the size of the efficient set for a given MOLP.

While the predictive ability of the model is not as powerful as hoped, the development of the model itself has led to several interesting insights concerning the efficient set size and its relationship to problem parameters. For instance, the number of
objectives in the model seems to have the greatest affect on the number of extreme points with the next greatest effect on the number of extreme points is the number of model constraints. The effects of the number of model variables, the zero-density of the constraint coefficient matrix, and the size of the criterion cone as measured by average cone angle, are not as dramatic, but they are significant.

As usual with regression analysis, tighter bounds on the prediction for mean response are available compared to the bounds for prediction of new observations. In this case, the difference is striking for larger problems. The large prediction intervals are a result of the large variance in number of extreme points for large problems. Methods for handling this increasingly large variance will need to be included in future work.

11.1.6 A Parallel Strategy for MOLP

Given the large efficient and nondominated sets for large MOLP and MONF models. Strategies for reducing the solution times for large problems are of interest. One such strategy was presented and explored in Chapter 10. There we described an idea for a parallel method of obtaining the solution set of a MOLP. The method is based on decomposition of the criterion cone. By decomposing the criterion cone, we effectively divide the efficient into smaller overlapping subsets. We may then solve for each subset on parallel processors and then combine the results. While total processing time (the sum of the processing times for all of the processors) may increase, the time to reach a solution in terms of real-time will decrease.

The results of the tests presented in Chapter 10 provide positive evidence of the potential of the cone decomposition strategy for parallelizing large MOLPs. Gains in
solution time greater than 50% are possible. From the data presented, we see that as problem size increases: the percentage time gain increases; the total sub-problem overlap increases; the percentage of sub-problem overlap decreases; and the number of extreme points are more balanced across the sub-problems.

Further research on this topic could include an actual parallel implementation as well as more exploration of alternative decomposition strategies. Since MONF are generally quite large, this strategy may also be applied with the specialized MONF codes presented in earlier chapters.

11.2 Concluding Remarks

With this research, we have laid the groundwork for exploring an interesting and important class of decision model. The theory and empirical evidence described should provide a solid foundation for future work in this area.
BIBLIOGRAPHY


MONFSOLV USER GUIDE

MONFSOLV is a program for solving multiple objective minimum cost network flow (MONF) problems for all efficient extreme points. Using a vector maximum approach, it is closely modeled on the general multiple objective solver ADBASE by R. E. Steuer. While ADBASE is capable of solving any multiple objective linear program (MOLP) for all efficient extreme points and unbounded edges, the network specialization that is incorporated into MONFSOLV currently allows for the solution of MONF problems up to three times faster than ADBASE. Information required for the use of MONFSOLV is described here.

A.1 Features of MONFSOLV

The program MONFSOLV was developed to search for and count, all efficient extreme points and all nondominated extreme points for MONF problems. Written in Fortran, MONFSOLV combines vector maximum theory with network simplex theory to provide a tool for conducting further research into the nature of the efficient and nondominated sets for MONF.

MONFSOLV provides several features to aid in a study of the efficient and nondominated sets. These include:

Two input formats: MONFSOLV will work with existing problems that are stored in one of two different file formats. First, the MONFSOLV format is typical of the standard format for network flow solvers. This format incorporates a node list and an arc list to specify problem parameters. The second input file format uses the ADBASE input file specification.

Several output options: The user of MONFSOLV may choose to obtain one or more of the following items in the output files:

User specified weights: The user may specify weights to use in place of default values when calculating the weighted sums objective for the initial efficient extreme point.

SONF mode: The solver may be used to solve single objective MCNF problems.

Choice of execution options: The user may choose from several execution options. One option regulates how degenerate network pivots are counted in the search for efficient extreme points. The second determines the method used in crashing from one efficient extreme point to the next.

A.2 MONFSOLV Input Files

At least two input files are required to run MONFSOLV. A file with extension Q is required to input execution parameters. A file with extension I is required to input problem specifications for a single or multiple objective network flow problem. A third
input file may be used to specify a weight vector used to find an initial efficient extreme point.

A.2.1 The Q File

An example of a Q file is presented in Figure A.1. Each value is stored in the Q file so that it may be read using the \texttt{FORMAT(I8X,I7)} Fortran format statement.

```
****---1-----2---3---4--- MONFSOLV **************
1. IDPROB 1 (PROBLEM IDENTIFICATION NUMBER)
2. INPUT TYPE 0 (0=RANDOM, 1=NETWORK, 2=ABBASR)
3. MODS 0 (0=MONF, 1=SONF, 2=RAND. GEN.)
4. WEIGHTS 0 (0=DEFAULT, 1=USER SUPPLIED)
5. DEGFLG 0 (0=DEGENERATE PIVOTS, 1=NO DEG. PIVOTS)
6. CRSFLG 0 (0 = CRASH1, 1=CRASH2)

***PRINT OPTIONS************
7. PRINT(1) 0 (0=NOTHING, 1=FILE VALUES)
8. PRINT(2) 0 (0=NOTHING, 1=PROB-NET, 2=PROB-ABBASR)
9. PRINT(3) 0 (0=NOTHING, 1=EFFICIENT EASES)
10. PRINT(4) 0 (0=NOTHING, 1=MOMDOMINATED VECTORS IN 2 FILE)
11. PRINT(5) 1 (0=NOTHING, 1=CUMULATIVE DATA AND TIMES)
12. PRINT(6) 0 (0=NOTHING, 1=ACCUMULATING COUNTS)
13. PRINT(7) 0 (0=NOTHING, 1=BASIS AND CAPACITY CODES)
14. INCA 1000 (INCREMENT TO PRINT ACCUMULATING COUNTS)
15. NUP 300000 (UPPER LIMIT ON EFFICIENT POINTS)

***VALUES FOR RANDOM PROBLEM GENERATION************
16. ISeed 22715 (RANDOM NUMBER GENERATOR SEED=1-32000)

-NODE PARAMETERS-------
17. NODES 34 (TOTAL NUMBER OF NODES)
18. NSRCs 3 (TOTAL NUMBER OF SOURCES-PURE AND TRANS.)
19. NSNKS 3 (TOTAL NUMBER OF SINKS-PURE AND TRANS.)
20. NTSRCS 2 (TOTAL NUMBER OF TRANSHIPMENT SOURCES)
21. NTSNKS 2 (TOTAL NUMBER OF TRANSHIPMENT SINKS)
22. NSPPM 94 (TOTAL SUPPLY/DEMAND)

-ARC PARAMETERS--------
23. NARCS 81 (TOTAL NUMBER OF ARCS)
24. KNINCAP 47 (LOWER BOUND OF ARC CAPACITY RANGE)
25. KNXCAP 94 (UPPER BOUND OF ARC CAPACITY RANGE)
26. KDCAP 100 (PERCENTAGE OF CAPACITATED ARCS)

-OBJECTIVE PARAMETERS------
27. MOBS 3 (NUMBER OF OBJECTIVES)
28. MINCST -10 (LOWER BOUND OF ARC COST RANGE)
29. MAXCST 10 (UPPER BOUND OF ARC COST RANGE)
30. MPMAX 20 (PERCENTAGE OF SKELETON ARCS TO GET MAXCST)

****---1-----2---3---4--- MONFSOLV **************
```

Figure A.1 - MONFSOLV Q File

Items 1 through 6 provide execution parameters for MONFSOLV.

1. IDPROB Used to set an ID for problem identification.
2. **INPUT TYPE**  
Sets the input mode of MONFSOLV. Use 0 to generate a problem at random; 1 to input file in MONFSOLV network format; and 2 to input file in ADBASE format.

3. **MODE**  
Sets the execution mode for MONFSOLV. Set to 0 to solve for all efficient and nondominated extreme points; 1 to solve as a single objective network flow problem. For a MONF, this option stops with the first efficient extreme points; 2 to generate and output a random problem without solving.

4. **WEIGHTS**  
Set to 0 to use default weights to find first efficient extreme point and 1 to read weights from a W input file.

5. **DEGFLG**  
Sets mode for handling adjacent efficient extreme points that result from degenerate network pivots. Set to 0 to add all adjacent efficient extreme points to master list; 1 to add only nondegenerate adjacent efficient extreme points to master list. Note: Both methods will record all of the same points, but in a different order.

6. **CRSFLG**  
Sets mode for crashing from one efficient extreme point to the next in the master list. Use 0 to use pivots on network with triangulation of the basis for backup; 1 to use triangularization of basis method only.

Items 7 through 13 are used to specify the desired output. For each of these items, set to 0 if output is not desired and set to 1 if it is desired.

7. **PRINT(1)**  
Prints Q file values to the S file.

8. **PRINT(2)**  
Prints MONF problem parameters to an N file. Use 1 to print in MONFSOLV format. Use 2 to print in ADBASE format.

9. **PRINT(3)**  
Prints all efficient basis including decision and objective variable values to the S file.
10. PRINT(4)  Prints all nondominated vectors to a Z file.
11. PRINT(5)  Prints cumulative data and run time statistics to the S file.
12. PRINT(6)  Prints a running total of efficient bases to S file.
13. PRINT(7)  Prints all master list codes to the S file.
14. INCR  Increment to print accumulating counts.
15. NUP  Upper limit on the number of efficient points.

Items 16 through 30 are used to specify parameters that govern the structure of randomly generated problems.
16. ISEED  Provides a seed for the random number generator.
17. NODES  Specifies the total number of nodes in the network.
18. NSRCS  Specifies the total number of source nodes, including both pure and transshipment source nodes.
19. NSNKS  Specifies the total number of sink nodes, including both pure and transshipment sink nodes.
20. NTSRCS  Specifies the total number of transshipment source nodes.
21. NTSNKS  Specifies the total number of transshipment sink nodes.
22. NSPDM  Specifies the total supply for the network. Note: supply = demand.
23. KARCS  Specifies the total number of arcs in the network.
24. KMINCAP  Specifies the lower bound of range from which to generate arc capacity values.
25. KMAXCAP  Specifies the upper bound of range from which to generate arc capacity values.
26. KPCAP  Specifies the percentage of arcs that have capacity set to upper bound.

27. MOBJS  Specifies the total number of objectives.

28. MINCST  Specifies the lower bound of range for selecting arc cost values.

29. MAXCST  Specifies the upper bound of range for selecting arc cost values.

30. MPMAX  Specifies the percentage of skeleton arcs in network to be set to upper bound of cost range.

A.2.2 The I File - MONFSOLV Format

An example of an I file in MONFSOLV network format for a network with 8 nodes, 13 arcs, and 2 objectives is presented in Figure A.2. The file is constructed as follows:

Line 1:  Values specifying the number of nodes N, the number of arcs A, and the number of objectives K for the network. This line is created to be compatible with the FORMAT(3I5) Fortran format statement.

Next N lines:  Node list specifying the node number and the supply (positive value) or demand (negative value). Compatible with the FORMAT(I5,I5) Fortran format statement.

Next A lines:  Arc list specifying the arc number, the FROM node, the TO node, and the CAPACITY for each arc. Compatible with the FORMAT(I5,I5,I5,I5) Fortran format statement.
Next K lines: List of objective function coefficients for each arc. Compatible with the `FORMAT(10I5)` Fortran format statement.

```
8  13  2
1  10
2  0
3  0
4  0
5  0
6  0
7  -3
8  -7
1  1  2  4
2  1  3  2
3  2  4  3
4  2  6  3
5  3  2  2
6  3  4  4
7  3  7  2
8  4  5  4
9  5  6  3
10  5  7  3
11  6  7  2
12  6  8  3
13  7  8  4
5  2
2  5
2  1
5  2
2  1
1  2
2  10
1  1
1  2
2  1
2  1
5  2
2  5
```

Figure A.2 - A MONFSOLV I File

### A.2.1 The W File

An additional input file, called a W file, may be used to input a user-specified weighting vector. This weighting vector is used to combine the multiple objectives into a
single objective for use in finding the initial efficient extreme point. The weights are chosen from

$$\Delta = \left\{ \lambda \in R^k \mid \lambda_i > 0, \sum_{i=1}^{k} \lambda_i = 1 \right\}.$$ 

To use a W file, enter one weight per line to be compatible with the `FORMAT(I6)` Fortran format statement.

### A.3 MONFSOLV Output

Output from MONFSOLV is provided in 3 files. The primary output is provided in the S file, which will be used for most execution of the program. Additional files include the N file and the Z file.

#### A.3.1 The S File

The S file provides the primary output for MONFSOLV. The makeup of the S file depends on which Print options are selected in the Q file. With all options selected the S file includes: Q file values, all efficient basis with decision and objective variable values to the S file; cumulative data and run time statistics; and all master list codes.

#### A.3.2 The N File

If PRINT(2) is set to 1 or 2, an N file will be created that contains the MONF problem specification. If the option is set to 1, the N file will be in MONFSOLV network format. If the option is set to 2, the N file will be in ADBASE format. If the setting is 0, no N file is created. This option is useful in converting an I file of one format into an N file of the other. The N file may then be used as an input I file by simply renaming the file extension and setting the appropriate input mode.
A.3.3 The Z File

Setting PRINT(4) directs MONFSOLV to create a Z file containing a list of all nondominated criterion vectors.

A.4 Exec File

For most implementations of Fortran, an executable script file is necessary to define input and output files and to load and execute the primary program. An example script file, called MJ, for UNIX platform is shown in Figure A.3.

```bash
#!/bin/sh

# Script to run MONFSOLV from object
in -sf $1.q  fort.25
in -sf $1.i  fort.26
in -sf $1.w  fort.27
in -sf $2.a  fort.30
in -sf $2.n  fort.31
in -sf $2.d  fort.32
in -sf $2.z  fort.33
echo 'Commencing MONFSOLV execution......' `date`
monfsolv
echo 'MONFSOLV execution completion.......' `date`
rm -f fort.25
rm -f fort.26
rm -f fort.27
rm -f fort.30
rm -f fort.31
rm -f fort.32
rm -f fort.33
```

Figure A.3 - Script File for MONFSOLV Execution

This script is invoked using two arguments. The arguments specify the filenames to use for the input and the output files. For example, to solve a problem with input files p1.q and p1.i and placing the output in files p2.n, p2.s, and p2.z use the command

mj p1 p2
C MONFSOLV3 BY CRAIG PIERCY
C FEBRUARY 28, 2001
C MULTIPLE OBJECTIVE NETWORK FLOW SOLVER
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES, ARCS, OBJS
INTEGER NDNUM(1000), SUPPLY(1000)
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER COSTS(1000, 10)
INTEGER PRED(1000), THD(1000), TFL(1000), DUAL(1000), M(1000)
INTEGER ROOT, FLOW(1000), TFLG(1000)
INTEGER PRICES(1000, 10), DUALS(1000, 10)
INTEGER IDPROB, INPFLG, WGTFLG, PRTFLG(7), ISEED
INTEGER MODE, CRSFLG, INCR, DEGFLG
INTEGER NSRCS, NSNKS, NTSRCS, NTSNKS, NSPDM
INTEGER KMINCAP, KMAXCAP, KPCAP
INTEGER MINCST, MAXCST, MMAX
INTEGER WSUM(10), WSCOST(1000)
INTEGER BCODE(30), NEXT, BL(400000, 33), TOTBAS
INTEGER ZLIST(400000, 12), NEXTZ, TOTALZ
INTEGER CCODE(30), CL(400000, 30)
INTEGER PRBAS, NCRASH, NCR1
INTEGER PRICE(1000), NARC, DELTA, ARCS1, NODES1
INTEGER G1, G2, BIGD, NFLG(1000), W, MU, BIGK, OUT
INTEGER CAPFLAG(1000)
REAL STIME, CTIME
COMMON /NET1/ NODES, ARCS, OBJS
COMMON /NET2/ NDNUM, SUPPLY
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /NET4/ PRED, THD, TFL, DUAL, M
COMMON /NET5/ ROOT, FLOW, TFLG
COMMON /NET6/ PRICES
COMMON /QPARM1/ IDPROB, INPFLG, WGTFLG, PRTFLG, ISEED
COMMON /QPARM2/ MODE, CRSFLG, INCR, NUP
COMMON /RAND1/ NSRCS, NSNKS, NTSRCS, NTSNKS, NSPDM
COMMON /RAND2/ KMINCAP, KMAXCAP, KPCAP
COMMON /RAND3/ MINCST, MAXCST, MMAX
COMMON /WEIGHT1/ WSUM, WSCOST
COMMON /LIST1/ BCODE, NEXT, BL, TOTBAS
COMMON /LIST2/ ZLIST, NEXTZ, TOTALZ
COMMON /LIST3/ CCODE, CL
COMMON /CDATA1/ PRBAS, STIME, NCRASH
COMMON /CDATA2/ NCR1, CTIME
COMMON /LAB1/ IQFI, IIFI, IWFI, ISFI, ICFI, INF, IZFI
INTEGER FSTAT
INTEGER*8 NUP
5 FORMAT(1X, 'NUMBER OF EFFICIENT POINTS BEYOND MAX')

APPENDIX B
MONFSOLV FORTRAN CODE
C INITIALIZE STAT VARIABLES AND READ PARAMETERS FROM QFILE
CALL XTERNL
CALL INITIALIZE
CTIME = 0.0
TOTBAS=0
PRBAS=0
NCR1 = 0
NCRASH = 0
WRITE(ISFI,10)
WRITE(ISFI,20) IDPROB
IF (PRTFLG(1).EQ.1) THEN
   CALL PRINTQ
END IF
C GET THE NETWORK (EITHER RANDOM, NETWORK, OR ADBASE FORMATS)
CALL GETNET
IF (PRTFLG(2).EQ.1) THEN
   CALL PRINTNET
END IF
IF (PRTFLG(2).EQ.2) THEN
   CALL PRINTANET
END IF
STIME= MCLOCK()
C IF ONLY GENERATING RANDOM FILE, THEN GOTO END
IF (MODE.EQ.2) GO TO 2000
C IF SOLVING A NETWORK, THEN CONTINUE
IF (MODE.NE.2) THEN
C GET FIRST EFFICIENT BASIS
   CALL FIRSTBASIS
C IF SINGLE OBJECTIVE PRINT OUTPUT TO S FILE AND QUIT
   IF (MODE.EQ.1) THEN
      TOTBAS = TOTBAS+1
      CALL PRINTSOL
      GO TO 2000
   END IF
C IF SOLVING MONF, THEN CONTINUE
   IF (MODE.EQ.0) THEN
C CALCULATE PRICES FOR FIRST BASIS
      CALL PRCCAL
      TOTBAS = TOTBAS+1
      IF (PRTFLG(3).EQ.1) THEN
         CALL PRINTSOL
      END IF
C FIND CODE FOR ARCS IN BASIS
      CALL CODING(ARCS,TFLG,BCODE)
C CREATE BASIS LIST AND PUT FIRST CODE IN LIST
      CALL CLIST
C FIND CODE FOR NONBASIC ARCS AT CAPACITY
      DO 93 JJ=1,ARCS
         IF (((FLOW(JJ).EQ.CAP(JJ)).AND.(TFLG(JJ).EQ.0)) THEN
            CAPFLAG(JJ)=1
         ELSE
            CAPFLAG(JJ)=0
         END IF
      CONTINUE
C CREATE LIST FOR CODES OF NONBASIC ARCS AT CAPACITY
      CALL CAPLIST(CAPFLAG)
C CREATE LIST OF Z VALUES
CALL ZMAKE
C TEST CURRENT SOLUTION FOR EFFICIENT PIVOTS AND ADD NEW TO LIST
100 CALL EFFSRCH(FSTAT)
   IF (PRTFLG(6).EQ.1) THEN
      IF (MOD(TOTBAS,INCR).EQ.0) THEN
         WRITE(ISFI,6) TOTBAS
      END IF
   END IF
   IF (TOTBAS.GT.NUP) THEN
      WRITE(ISFI,5)
      GOTO 2000
   END IF
   IF (FSTAT.EQ.1) THEN
      C IF THERE ARE MORE BASES IN LIST, THEN CRASH TO NEXT BASIS
      CALL CRASH
      GOTO 100
   END IF
110 END IF
2000 IF (PRTFLG(5).EQ.1) THEN
   CALL TIDYUP
END IF
   IF (PRTFLG(4).EQ.1) THEN
      CALL PRINTZ(IDPROB,ZLIST,TOTALZ,OBJS)
   END IF
   IF (PRTFLG(7).EQ.1) THEN
      CALL PRCODES
   END IF
STOP
END
C
SUBROUTINE PRINTZ(IDPROB,ZLIST,TOTALZ,OBJS)
C PRINTS A LIST OF NONDOMINATED VECTORS TO Z FILE
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER IDPROB,TOTALZ,OBJS
INTEGER ZLIST(400000,12)
INTEGER IQFI,IIFI,IWFI,ISFI,ICFI,INFI,IZFI
COMMON /LAB1/ IQFI,IIFI,IWFI,ISFI,ICFI,INFI,IZFI
70 FORMAT(3I6)
80 FORMAT(I4,I6,I2,5(I12))
85 FORMAT(6X,I2,5(I12))
WRITE(IZFI,70) IDPROB,OBJS,TOTALZ
DO 100 J = 1,TOTALZ
   ITOP = 5
   IF (OBJS.LT.5) ITOP = OBJS
   WRITE(IZFI,80) IDPROB,J,1,(ZLIST(J,I),I=1,ITOP)
   NROW = 2
   NLEFT = OBJS - 5
90 IF (NLEFT.GT.5) THEN
   ITOP = 5
   IF (NLEFT.LT.5) ITOP = NLEFT
   WRITE(IZFI,85) NROW,(ZLIST(J,(I+(NROW-1)*5)),I=1,ITOP)
   NROW = NROW+1
   NLEFT = NLEFT-5
   GOTO 90
END IF
100 CONTINUE
RETURN
END
SUBROUTINE PRCODES
C PRINTS A LIST OF BASIS AND CAPACITY CODES TO C FILE
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
COMMON /LAB1/ IQFI,IIFI,IWFI,ISFI,ICFI,INF,IIZFI
INTEGER IQFI,IIFI,IWFI,ISFI,ICFI,INF,IIZFI
INTEGER BCOD(30),NEXT,BL(400000,33),TOTBAS
INTEGER CCOD(30),CL(400000,30)
COMMON /LIST1/ BCOD,NEXT,BL,TOTBAS
COMMON /LIST3/ CCOD,CL
COMMON /NET1/ NODES,ARCS,OBJS
90 FORMAT(1X)
100 FORMAT(1X,'BASIS CODES')
105 FORMAT(1X,'CAPACITY CODES')
110 FORMAT(1X,I5,I5,5I10)
120 FORMAT(6X,I5,5I10)
WRITE(ICFI,90)
WRITE(ICFI,100)
NTOP = ARCS/30
IF (MOD(ARCS,30).GT.0) NTOP = NTOP+1
DO 200 I=1,TOTBAS
   N2 = 0
   NSTP = 0
   K = 1
220 N1 = N2+1
   IF ((NTOP-5).LE.0) THEN
      N2 = N2 + NTOP
      NSTP = 1
   ELSE
      N2 = N2+5
   END IF
   NTOP = NTOP - 5
   IF (K.EQ.1) THEN
      WRITE(ICFI,110) I,K,(BL(I,J), J = N1,N2)
   ELSE
      WRITE(ICFI,110) K,(BL(I,J), J = N1,N2)
   END IF
   K = K+1
   IF (NSTP.EQ.0) GOTO 220
200 CONTINUE
WRITE(ICFI,90)
WRITE(ICFI,105)
NTOP = ARCS/30
IF (MOD(ARCS,30).GT.0) NTOP = NTOP+1
DO 300 I=1,TOTBAS
   N2 = 0
   NSTP = 0
   K = 1
320 N1 = N2+1
   IF ((NTOP-5).LE.0) THEN
      N2 = N2 + NTOP
      NSTP = 1
   ELSE
      N2 = N2+5
   END IF
   NTOP = NTOP - 5
   IF (K.EQ.1) THEN
      WRITE(ICFI,110) I,K,(CL(I,J), J = N1,N2)
   ELSE
      WRITE(ICFI,110) K,(CL(I,J), J = N1,N2)
C
SUBROUTINE INITIALIZE
C THIS SUBROUTINE Initializes count, iteration and timing variables,
C and reads in program parameters from the Q file
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER IDPROB,INPFLG,WGTFLG,PRTFLG(7),ISEED,DEGFLG
INTEGER NODES,ARCS,OBJ
INTEGER NSRCS,NSNKS,NTSRCS,NTSNKS,NSPDM
INTEGER KMINCAP,KMAXCAP,KPCAP
INTEGER MINCST,MAXCST,MPMAX
INTEGER IQFI,IIFI,IWFI,ISFI,ICFI,INFI,IZFI
COMMON /LAB1/ IQFI,IIFI,IWFI,ISFI,ICFI,INFI,IZFI
COMMON /QPARM1/ IDPROB,INPFLG,WGTFLG,PRTFLG,ISEED
COMMON /QPARM2/ MODE,DEGFLG,CRSFLG,INCR,NUP
INTEGER CRSFLG,INCR
INTEGER*8 NUP
COMMON /NET1/ NODES,ARCS,OBJ
COMMON /RAND1/ NSRCS,NSNKS,NTSRCS,NTSNKS,NSPDM
COMMON /RAND2/ KMINCAP,KMAXCAP,KPCAP
COMMON /RAND3/ MINCST,MAXCST,MPMAX
10 FORMAT(1X)
20 FORMAT(18X,I7)
READ(IQFI,10)
READ(IQFI,20) IDPROB
READ(IQFI,20) INPFLG
READ(IQFI,20) MODE
READ(IQFI,20) WGTFLG
READ(IQFI,20) DEGFLG
READ(IQFI,20) CRSFLG
READ(IQFI,10)
DO 100 I = 1,7
   READ(IQFI,20) PRTFLG(I)
100 CONTINUE
READ(IQFI,20) INCR
READ(IQFI,20) NUP
IF (INPFLG.EQ.0) THEN
   READ(IQFI,10)
   READ(IQFI,20) ISEED
   READ(IQFI,10)
   READ(IQFI,20) NODES
   READ(IQFI,20) NSRCS
   READ(IQFI,20) NSNKS
   READ(IQFI,20) NTSRCS
   READ(IQFI,20) NTSNKs
   READ(IQFI,20) NSPDM
   READ(IQFI,10)
   READ(IQFI,20) ARCS
   READ(IQFI,20) KMINCAP
   READ(IQFI,20) KMAXCAP
   READ(IQFI,20) KPCAP
   READ(IQFI,10)
   READ(IQFI,20) OBJ
   READ(IQFI,20) MINCST
   READ(IQFI,20) MAXCST
READ(IQFI,20) MPMAX
END IF
RETURN
END

C
SUBROUTINE GETNET
C THIS SUBROUTINE GETS THE PROBLEM COEFFICIENTS. THE METHOD USED
C DEPENDS ON THE INPUT TYPE SWITCH IN THE Q FILE.
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER IDPROB,INPFLG,WGTFLG,PRTFLG(7),ISEED
COMMON /QPARM1/ IDPROB,INPFLG,WGTFLG,PRTFLG,ISEED
IF (INPFLG.EQ.0) THEN
  CALL RANDNET
END IF
IF (INPFLG.EQ.1) THEN
  CALL READNET
END IF
IF (INPFLG.EQ.2) THEN
  CALL READANET
END IF
NASIZE = 1000
RETURN
END

C
SUBROUTINE FIRSTBASIS
C THIS SUBROUTINE FINDS THE FIRST EFFICIENT BASIS AND INITIALIZES THE
C THE BOOKKEEPING SYSTEM.
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER IDPROB,INPFLG,WGTFLG,PRTFLG(7),ISEED
INTEGER NODES,ARCS,OBJS
INTEGER ARCNUM(1000),FROM(1000),TO(1000),CAP(1000)
INTEGER COSTS(1000,10)
INTEGER PRED(1000),THD(1000),TFL(1000),DUAL(1000),M(1000)
INTEGER ROOT,FLOW(1000),TFLG(1000)
INTEGER WSUM(10),WSCOST(1000)
INTEGER PRICE(1000),NARC,DELTA,ARCS1,NODES1
INTEGER ITER,LAST,K
INTEGER G1,G2,BIGD,NFLG(1000),W,MU,BIGK,OUT
COMMON /QPARM1/ IDPROB,INPFLG,WGTFLG,PRTFLG,ISEED
COMMON /QPARM1/ IDPROB,INPFLG,WGTFLG,PRTFLG,ISEED
COMMON /QPARM1/ IDPROB,INPFLG,WGTFLG,PRTFLG,ISEED
COMMON /QPARM1/ IDPROB,INPFLG,WGTFLG,PRTFLG,ISEED
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /WEIGHT1/ WSUM,WSCOST
COMMON /WEIGHT1/ WSUM,WSCOST
COMMON /WEIGHT1/ WSUM,WSCOST
COMMON /WEIGHT1/ WSUM,WSCOST
COMMON /WEIGHT1/ WSUM,WSCOST
COMMON /WEIGHT1/ WSUM,WSCOST
COMMON /WEIGHT1/ WSUM,WSCOST
COMMON /NET4/ PRED,THD,TFL,DUAL,M
COMMON /NET4/ PRED,THD,TFL,DUAL,M
COMMON /NET4/ PRED,THD,TFL,DUAL,M
COMMON /NET4/ PRED,THD,TFL,DUAL,M
COMMON /NET4/ PRED,THD,TFL,DUAL,M
COMMON /NET4/ PRED,THD,TFL,DUAL,M
COMMON /NET4/ PRED,THD,TFL,DUAL,M
COMMON /LAB1/ IQFI,IIFI,IWFI,ISFI,ICFI,INFI,IZFI
ITER=1
LAST = 0
ARCS1=ARCS
NODES1=NODES
CALL WEIGHTSUM
CALL ARTARCS(WSCOST)
CALL DUALCAL(ROOT,NODES,ARCS,THD,M,WSUM,PRED,DELTA)
90  CALL PRICING(NODES,ARCS,PRED,TO,DUAL,WSUM,TFLG,PRICE)
CALL SELECT(ARCS1,FLOW,CAP,TFLG,PRICE,NARC,DELTA,LAST)
LAST = NARC
IF (DELTA.EQ.0) GOTO 140
CALL RATTEST(NARC,G1,G2,BIGD,NFLG,W,MU,BIGK,OUT)
CALL UPDATE(PRICE,NARC,DELTA,G1,G2,BIGD,NFLG,W,MU,BIGK,OUT)
GOTO 90
140  K = 0
DO 160 I = ARCS1+1,ARCS  
  IF (FLOW(I).GT.0.001) CALL ERRO(12)  
  IF (TFLG(I).EQ.1) K = K+1  
160 CONTINUE  
  IF (K.GT.1) CALL ARTOUT(ARCS1,NODES1,K)  
DO 170 I = 1,NODES1  
  IF (THD(I).EQ.NODES) THEN  
    THD(I) = THD(NODES)  
    GOTO 180  
  END IF  
170 CONTINUE  
180 NODES=NODES1  
  ARCS=ARCS1  
RETURN  
END

C SUBROUTINE ARTOUT(ARCS1,NODES1,K)
C SETS UP ALL ARTIFICIAL ARCS FOR INITIAL BASIS
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES,ARCS,OBJS
INTEGER ARCNUM(1000),FROM(1000),TO(1000),CAP(1000)
INTEGER COSTS(1000,10)
INTEGER PRED(1000),THD(1000),TFL(1000),DUAL(1000),M(1000)
INTEGER ROOT,FLOW(1000),TFLG(1000)
INTEGER CNODE,FLAG(1000),PNODE,PTR
INTEGER FNODE,W,II,NARC
INTEGER I,J,K,RTHD(1000)
INTEGER ARCS1,NODES1
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /NET4/ PRED,THD,TFL,DUAL,M
COMMON /NET5/ ROOT,FLOW,TFLG
COMMON /LAB1/ IQFI,IIFI,IWFI,ISFI,ICFI,INFI,IZFI
K = K-1
90 I = ARCS1  
100 I = I+1  
  IF (I.GT.ARCS) GOTO 90  
  IF (TFLG(I).NE.1) GOTO 100  
  IF (ABS(M(NODES)).EQ.I) GOTO 100  
  IF (FROM(I).NE.NODES) CNODE = FROM(I)  
  IF (TO(I).NE.NODES) CNODE = TO(I)  
  J = 0
120 J = J+1  
  DO 110 II = 1,NODES  
    FLAG(II) = 0  
 110 CONTINUE  
  NARC = 0  
  W = 0  
  IF (J.GT.ARCS1) GOTO 100  
  IF (TFLG(J).NE.1) THEN  
    FNODE = FROM(J)  
130   FLAG(FNODE) = 1  
    IF (PNODE.EQ.ROOT) GOTO 135  
    FNODE = PRED(FNODE)  
    GOTO 130  
135 FNODE = TO(J)  
140   IF (FLAG(FNODE).EQ.1) THEN  
    W = FNODE  
    GOTO 145  
  END IF
FLAG(FNODE) = 1
FNODE = PRED(FNODE)
GOTO 140
END IF
145 IF ((FLAG(CNODE).EQ.1).AND.(FLAG(NODES).EQ.1)) NARC = J
IF (NARC.EQ.0) GOTO 120
TFLG(I) = 0
TFLG(J) = 1
K = K-1
IF (K.NE.0) GOTO 100
C SET PRED AND M DATA STRUCTURE
K = 0
DO 150 I = 1,NODES
   PRED(I) = 0
   M(I) = 0
   THD(I) = 0
   RTHD(I)=0
   TFL(I) = 0
150 CONTINUE
DO 155 I=1,ARCS1
   FLAG(I) = 0
155 CONTINUE
M(1) = 0
PRED(1) = 0
FLAG(1)=1
190 DO 200 I = 1,ARCS1
   IF(TFLG(I).EQ.1) THEN
      IF ((FLAG(FROM(I)).EQ.1).AND.(FLAG(TO(I)).EQ.0)) THEN
         PRED(TO(I))=FROM(I)
         M(TO(I))=I
         FLAG(TO(I))=1
         K=K+1
      ELSE
         IF ((FLAG(TO(I)).EQ.1).AND.(FLAG(FROM(I)).EQ.0)) THEN
            PRED(FROM(I))=TO(I)
            M(FROM(I))=-I
            FLAG(FROM(I))=1
            K=K+1
         END IF
      END IF
   END IF
200 CONTINUE
IF (K.LT.NODES1-1) GOTO 190
C SET THREAD AND REVERSE THREAD
DO 300 I=0,NODES1
   THD(I) = 0
   RTHD(I) = 0
300 CONTINUE
PNODE = ROOT
PTR = ROOT
DO 400 I=1,NODES1-1
   DO 450 J = 1,NODES1
      IF (PRED(J).EQ.PNODE) THEN
         THD(J)=THD(PTR)
         RTHD(THD(PTR))=J
         THD(PTR)=J
         RTHD(J)=PTR
         PTR=J
      END IF
450 CONTINUE
C
SUBROUTINE WEIGHTSUM
C CALCULATES THE WEIGHTED SUM OBJECTIVE
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER IDPROB,INPFLG,WGTFLG,PRTFLG(7),ISEED
INTEGER NODES,ARCS,OBJS
INTEGER ARCNUM(1000),FROM(1000),TO(1000),CAP(1000)
INTEGER COSTS(1000,10)
INTEGER WSUM(10),WSCOST(1000)
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /QPARM1/ IDPROB,INPFLG,WGTFLG,PRTFLG,ISEED
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /WEIGHT1/ WSUM,WSCOST
20 FORMAT(I6)
INTEGER I,J,WSUM1
IF (OBJS.EQ.1) THEN
  WSUM(1) = 1
  GO TO 1000
END IF
IF (WGTFLG.EQ.0) THEN
  WSUM1 = 100
  DO 200 I = 2,OBJS
    WSUM1 = WSUM1-1
    WSUM(I) = 1
  200 CONTINUE
  WSUM(1) = WSUM1
END IF
IF (WGTFLG.EQ.1) THEN
  DO 210 J=1,OBJS
    READ(IWFI,20) WSUM(J)
  210 CONTINUE
END IF
1000 DO 220 J = 1,ARCS
  WSCOST(J) = 0
  DO 230 I = 1,OBJS
    WSCOST(J) = WSCOST(J)+COSTS(J,I)*WSUM(I)
  230 CONTINUE
  CONTINUE
220 CONTINUE
RETURN
END
C
SUBROUTINE ARTARCS(SCOST)
C ADDS ARTIFICIAL ARCS AND AN ARTIFICIAL NODE
C FOR A NETWORK WITH SINGLE OBJECTIVE
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES, ARCS, OBJS
INTEGER NDNUM(1000), SUPPLY(1000)
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER COSTS(1000, 10)
INTEGER PRED(1000), THD(1000), TFL(1000), DUAL(1000), M(1000)
INTEGER ROOT, FLOW(1000), TFLG(1000)
COMMON /NET1/ NODES, ARCS, OBJS
COMMON /NET2/ NDNUM, SUPPLY
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /NET4/ PRED, THD, TFL, DUAL, M
COMMON /NET5/ ROOT, FLOW, TFLG
INTEGER SCOST(1000)
INTEGER ARTNODE, I, K, ARCN
ROOT = 1
ARTNODE = NODES + 1
NODES = ARTNODE
DO 400 I = 1, NODES
  PRED(I) = 0
  THD(I) = 0
  M(I) = 0
  TFL(I) = 0
  DUAL(I) = 0
400 CONTINUE
DO 410 I = 1, ARCS
  FLOW(I) = 0
  TFLG(I) = 0
410 CONTINUE
SUPPLY(ARTNODE) = 0
NDNUM(ARTNODE) = ARTNODE
DO 510 I = 1, (NODES - 1)
  SUPPLY(ARTNODE) = SUPPLY(ARTNODE) + SUPPLY(I)
510 CONTINUE
PRED(ARTNODE) = 1
THD(ARTNODE) = 2
DUAL(ARTNODE) = 0
DO 520 K = 1, (NODES - 1)
  ARCS = ARCS + 1
  ARCNUM(ARCS) = ARCS
  IF (SUPPLY(K) .GT. 0) THEN
    M(K) = -ARCS
    FROM(ARCS) = K
    TO(ARCS) = ARTNODE
    SCOST(ARCS) = 0
    CAP(ARCS) = 99999
    FLOW(ARCS) = SUPPLY(K)
    TFLG(ARCS) = 1
  ELSE
    M(K) = ARCS
    FROM(ARCS) = ARTNODE
    TO(ARCS) = K
    SCOST(ARCS) = 99999
    CAP(ARCS) = 99999
    FLOW(ARCS) = -SUPPLY(K)
    TFLG(ARCS) = 1
  END IF
IF (K .EQ. 1) THEN
  PRED(K) = 0
  THD(K) = ARTNODE
ELSE
  PRED(K) = ARTNODE
THD(K) = K + 1
TFL(K) = FLOW(ABS(M(K)))
IF((K + 1).EQ.ARTNODE) THD(K) = 1
END IF

520 CONTINUE
M(ARTNODE) = -M(ROOT)
TFL(ARTNODE) = FLOW(ABS(M(ROOT)))
M(ROOT) = 0
RETURN
END

SUBROUTINE DUALCAL(R, N, A, T, M2, C, P, D)
SUBROUTINE DUALCAL(R, N, A, T, M2, C, P, D)
SUBROUTINE DUALCAL(R, N, A, T, M2, C, P, D)

C CALCULATES DUALS FOR THE NETWORK FOR A GIVEN COST VECTOR.
C
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER R, N, A
INTEGER T(N), M2(N), P(N), D(N)
INTEGER C(A)
INTEGER CUR, J, SIGN, C2
D(R) = 0
CUR = T(R)
DO 620 J = 2, N
IF (M2(CUR).GT.0) THEN
SIGN = -1
ELSE
SIGN = 1
END IF
C2 = C(ABS(M2(CUR)))
D(CUR) = SIGN*C2 + D(P(CUR))
CUR = T(CUR)
620 CONTINUE
RETURN
END

SUBROUTINE PRICING(N, A, F, T, D, C, TF, P)
SUBROUTINE PRICING(N, A, F, T, D, C, TF, P)
SUBROUTINE PRICING(N, A, F, T, D, C, TF, P)

C CALCULATES A PRICING VECTOR FOR A GIVEN COST VECTOR.
C
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER N, A
INTEGER F(A), T(A), C(A), TF(A), P(A)
INTEGER D(N)
INTEGER I
DO 100 I = 1, A
IF (TF(I).EQ.1) THEN
P(I) = 0
ELSE
P(I) = D(F(I)) - D(T(I)) - C(I)
END IF
100 CONTINUE
RETURN
END

SUBROUTINE SELECT(A, F, C, TF, P, NEW, DEL, LA)
SUBROUTINE SELECT(A, F, C, TF, P, NEW, DEL, LA)
SUBROUTINE SELECT(A, F, C, TF, P, NEW, DEL, LA)

C SELECTS THE ARC TO ENTER THE BASIS.
C
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER A, NEW, DEL, I, LA
INTEGER F(A), C(A), TF(A), P(A)
NEW = 0
DEL = 0
N1 = LA + 1
N2 = A
DO 200 J = 1,2
DO 100 I=N1,N2
   IF (TF(I).NE.1) THEN
      IF (P(I).GT.0) THEN
         IF (F(I).EQ.0) THEN
            NEW = I
            DEL = 1
            GOTO 110
         END IF
      ELSE
         IF (P(I).LT.0) THEN
            IF (F(I).EQ.C(I)) THEN
               NEW = I
               DEL = -1
               GOTO 110
            END IF
         END IF
      END IF
   END IF
100 CONTINUE
N1 = 1
N2 = LA
200 CONTINUE
110 RETURN
END

C SUBROUTINE RATTEST(EDGE,G1,G2,BIGD,NFLG,W,MU,BIGK,OUT)
C PREFORMS RATIO TEST TO DETERMINE ARC TO REMOVE FROM BASIS
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES,ARCS,OBJ
INTEGER ARCNUM(1000),FROM(1000),TO(1000),CAP(1000)
INTEGER COSTS(1000,10)
INTEGER PRED(1000),THD(1000),TFL(1000),DUAL(1000),M(1000)
INTEGER ROOT,FLOW(1000),TFLG(1000)
INTEGER G1,G2,BIGD,W,MU,BIGK,OUT
COMMON /NET1/ NODES,ARCS,OBJ
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /NET4/ PRED,THD,TFL,DUAL,M
COMMON /NET5/ ROOT,FLOW,TFLG
INTEGER NFLG(1000)
INTEGER I,CNODE ,EDGE
BIGK = -1
MU = 0
DO 100 I = 1,NODES
   NFLG(I) = 0
100 CONTINUE
   IF (FLOW(EDGE).EQ.0) THEN
      G1 = -1
      G2 = 1
   ELSE
      G1 = 1
      G2 = -1
   END IF
BIGD = CAP(EDGE)
CNODE = FROM(EDGE)
110 NFLG(CNODE) = 1
   IF (CNODE.NE.ROOT) THEN
      CNODE = PRED(CNODE)
   GOTO 110
END IF
   CNODE = TO(EDGE)
200 IF (NFLG(CNODE).EQ.1) THEN
      W = CNODE
      GOTO 300
END IF
   NFLG(CNODE)=1
IF ((M(CNODE)*G2).GT.0) GOTO 210
IF ((CAP(ABS(M(CNODE)))-TFL(CNODE)).GT.BIGD) THEN
   CNODE= PRED(CNODE)
   GOTO 200
ELSE
   BIGD = CAP(ABS(M(CNODE)))-TFL(CNODE)
   MU = 2
   BIGK = CNODE
   CNODE = PRED(CNODE)
   GOTO 200
END IF
210 IF (TFL(CNODE).GT.BIGD) THEN
   CNODE = PRED(CNODE)
   GOTO 200
ELSE
   BIGD = TFL(CNODE)
   MU = 2
   BIGK = CNODE
   CNODE = PRED(CNODE)
   GOTO 200
END IF
300 CNODE = FROM(EDGE)
310 IF (CNODE.EQ.W) GOTO 400
IF ((M(CNODE)*G1).GT.0) GOTO 320
IF ((CAP(ABS(M(CNODE)))-TFL(CNODE)).GT.BIGD) THEN
   CNODE = PRED(CNODE)
   GOTO 310
ELSE
   BIGD = CAP(ABS(M(CNODE)))-TFL(CNODE)
   MU = 1
   BIGK = CNODE
   CNODE = PRED(CNODE)
   GOTO 310
END IF
320 IF (TFL(CNODE).GT.BIGD) THEN
   CNODE = PRED(CNODE)
   GOTO 310
ELSE
   BIGD = TFL(CNODE)
   MU = 1
   BIGK = CNODE
   CNODE = PRED(CNODE)
   GOTO 310
END IF
400 IF (BIGK.GT.0)THEN
   OUT=M(BIGK)
ELSE
   OUT = 0
END IF
RETURN
END

C SUBROUTINE UPDATE(PCOST, EDGE, DEL, G1, G2, BIGD, NFLG, W, MU, BIGK, OUT)
C UPDATES THE NETWORK BASIS STRUCTURE FOR ENTERING AND LEAVING ARC

IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER PRED(1000), THD(1000), TFL(1000), DUAL(1000), M(1000)
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER COSTS(1000, 10)
INTEGER ARCNUM, FROM, TO, CAP, COSTS
COMM /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMM /NET4/ PRED, THD, TFL, DUAL, M
COMM /NET5/ ROOT, FLOW, TFLG
INTEGER EDGE, G1, G2, BIGD, W, MU, BIGK, OUT
INTEGER NFLG(1000), PCOST(1000), DEL
INTEGER D1, D2, K, D, G
INTEGER BETA, SIG, DPRIME, Q, QPRIME
INTEGER LAMBDA, I, J, KPRIME, OLDFLOW
INTEGER MS, Z, X, R, O
IF (BIGD.NE.CAP(EDGE)) GOTO 310
IF (DEL.LT.0) THEN
  FLOW(EDGE) = 0
ELSE
  FLOW(EDGE) = CAP(EDGE)
END IF
D1 = FROM(EDGE)
D2 = TO(EDGE)
K = 1
D = D1
G = G1*BIGD
230 IF (D.EQ.W) GOTO 290
IF (M(D).GT.0) THEN
  BETA = 1
ELSE
  BETA = -1
END IF
TFL(D) = TFL(D) - G*BETA
FLOW(ABS(M(D))) = TFL(D)
NFLG(D) = 0
D = PRED(D)
GOTO 230
290 IF (K.NE.1) GOTO 910
K = 2
D = D2
G = G2*BIGD
GOTO 230
310 SIG = PCOST(EDGE)
TFLG(EDGE) = 1
TFLG(ABS(OUT)) = 0
O = OUT*DEL
IF (((O.GT.0).AND.(MU.EQ.1)).OR.((O.LT.0).AND.(MU.EQ.2))) THEN
  FLOW(ABS(OUT)) = CAP(ABS(OUT))
ELSE
  FLOW(ABS(OUT)) = 0
END IF
IF (FLOW(EDGE).EQ.CAP(EDGE)) THEN
  DPRIME = CAP(EDGE) - BIGD
ELSE
  DPRIME = BIGD
END IF
IF (MU.EQ.1) THEN
  Q = FROM(EDGE)
  QPRIME = TO(EDGE)
  LAMBDA = -EDGE

SIG = -SIG
G = G1*BIGD
ELSE
    Q = TO(EDGE)
    QPRIME = FROM(EDGE)
    LAMBDA = EDGE
    G = G2*BIGD
END IF
I = Q
J = PRED(Q)
KPRIME = PRED(BIGK)
410 OLDFLOW = TFL(I)
    DUAL(I)=DUAL(I)+SIG
    TFL(I) = DPRIME
    IF (M(I).GT.0) THEN
        BETA = 1
    ELSE
        BETA = -1
    END IF
    MS = ABS(M(I))
    M(I) = LAMBDA
    FLOW(ABS(M(I))) = TFL(I)
    NFLG(I) = 0
    Z=I
    X = THD(I)
520 IF ((X.EQ.ROOT).OR.(NFLG(PRED(X)).EQ.1)) GOTO 610
    DUAL(X)=DUAL(X)+SIG
    Z = X
    X = THD(X)
    GOTO 520
610 R = J
620 IF (THD(R).EQ.I) GOTO 710
    R = THD(R)
    GOTO 620
710 IF (I.EQ.BIGK) GOTO 790
    DPRIME = OLDFLOW-G*BETA
    LAMBDA = -BETA*MS
    THD(R)=X
    THD(Z)=J
    R=I
    I=J
    J=PRED(J)
    PRED(I)=R
    GOTO 410
790 THD(R)=X
    THD(Z)=THD(QPRIME)
    THD(QPRIME)=Q
    PRED(Q)=QPRIME
    IF (MU.EQ.1) THEN
        D1 = KPRIME
        D2 = QPRIME
        K = 1
    ELSE
        D1 = QPRIME
        D2 = KPRIME
        K = 1
    END IF
780 IF (D.EQ.W) GOTO 880
IF (M(D).GT.0) THEN
    BETA = 1
ELSE
    BETA = -1
END IF
TFL(D) = TFL(D) - G*BETA
FLOW(ABS(M(D))) = TFL(D)
NFLG(D) = 0
D = PRED(D)
GOTO 830
880 IF (K.NE.1) GOTO 910
    K = 2
    D = D2
    G = G2*BIGD
    GOTO 830
910 I = W
920 NFLG(I) = 0
    IF (I.EQ.ROOT) GOTO 970
    I = PRED(I)
    GOTO 920
970 RETURN
END

SUBROUTINE TIDYUP
C THIS SUBROUTINE FINALIZES COUNT, ITERATION AND TIMING VARIABLES,
C AND WRITES TO APPROPRIATE OUTPUT FILES
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER PRBAS
INTEGER BCODE(30), NEXT, BL(400000, 33), TOTBAS, NCRASH, NCR1
REAL STIME, CTIME
INTEGER ZLIST(400000, 12), NEXTZ, TOTALZ
INTEGER IQFI, IIFI, IWFI, ISFI, ICFI, INFI, IZFI
COMMON /LIST2/ ZLIST, NEXTZ, TOTALZ
COMMON /CDATA1/ PRBAS, STIME, NCRASH
COMMON /CDATA2/ NCR1, CTIME
COMMON /LIST1/ BCODE, NEXT, BL, TOTBAS
COMMON /LAB1/ IQFI, IIFI, IWFI, ISFI, ICFI, INFI, IZFI
REAL CPUTIM, CRTIME
10 FORMAT(1X, '***** CUMULATIVE PROBLEM STATS *****')
20 FORMAT(1X)
30 FORMAT(5X, 'CPU TIME = ', F10.2)
40 FORMAT(5X, 'TOTAL EFFICIENT BASES = ', I10)
50 FORMAT(5X, 'TOTAL NONDOMINATED EXTREME POINTS = ', I10)
60 FORMAT(5X, 'AVERAGE CRASH TIME = ', F10.6)
70 FORMAT(5X, 'NUMBER OF CALLS TO CRASH2 = ', I10)
WRITE(ISFI, 10)
WRITE(ISFI, 20)
CPUTIM = 1.0*(MCLOCK() - STIME)/100
CRTIME = 1.0*(CTIME/NCR1)/100
WRITE(ISFI, 30) CPUTIM
WRITE(ISFI, 40) TOTBAS
WRITE(ISFI, 50) TOTALZ
WRITE(ISFI, 60) CRTIME
WRITE(ISFI, 70) NCRASH
RETURN
END

SUBROUTINE READNET
C GETS THE PROBLEM COEFFICIENTS FROM A FILE
C THAT IS IN A STANDARD NETWORK INPUT FORMAT.
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES, ARCS, OBJS
INTEGER NDNUM(1000), SUPPLY(1000)
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER COSTS(1000, 10)
INTEGER IQFI, IIFI, IWFI, ISFI, ICFI, INFI, IZFI
COMMON /NET1/ NODES, ARCS, OBJS
COMMON /NET2/ NDNUM, SUPPLY
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /LAB1/ IQFI, IIFI, IWFI, ISFI, ICFI, INFI, IZFI

C LOCAL VARIABLES
INTEGER I, J
100 FORMAT (3I5)
110 FORMAT (I5, I5)
120 FORMAT (I5, I5, I5, I5)
130 FORMAT (10I5)

C INITIALIZE ARRAYS
NODES = 0
ARCS = 0
OBJS = 0
READ(IQFI, 100) NODES, ARCS, OBJS
DO 205 I = 1, NODES
   NDNUM(I) = 0
   SUPPLY(I) = 0
205 CONTINUE
DO 210 I = 1, ARCS
   ARCNUM(I) = 0
   FROM(I) = 0
   TO(I) = 0
   CAP(I) = 0
   DO 200 J = 1, OBJS
      COSTS(I, J) = 0
200 CONTINUE
210 CONTINUE

C READ INPUT FILE (26)
DO 220 I = 1, NODES
   READ(IIFI, 110) NDNUM(I), SUPPLY(I)
220 CONTINUE
DO 230 I = 1, ARCS
   READ(IIFI, 120) ARCNUM(I), FROM(I), TO(I), CAP(I)
230 CONTINUE
DO 240 I = 1, ARCS
   READ(IIFI, 130) (COSTS(I, J), J = 1, OBJS)
240 CONTINUE
RETURN
END

C SUBROUTINE READANET
C GETS THE PROBLEM COEFFICIENTS. FROM A FILE
C THAT IS IN ADBASE INPUT FORMAT.
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER IDPROB, INPFLG, WGTFLG, PRTFLG(7), ISEED
INTEGER NODES, ARCS, OBJS
INTEGER NDNUM(1000), SUPPLY(1000)
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER COSTS(1000, 10)
INTEGER IQFI, IIFI, IWFI, ISFI, ICFI, INFI, IZFI
COMMON /NET1/ NODES, ARCS, OBJS
COMMON /QPARM1/ IDPROB, INPFLG, WGTFLG, PRTFLG, ISEED
COMMON /NET2/ NDNUM, SUPPLY
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /LAB1/ IQFI, IIFI, IWFI, ISFI, ICFI, INFI, IZFI
INTEGER I, J, NCAPS(1000,1000), A(1000,1000), AE
INTEGER BE, CK, NCAP, STOT

100 FORMAT(1X)
110 FORMAT(8I8)
120 FORMAT(I8)
130 FORMAT(4(I3,12X))
140 FORMAT(4(I3,3X,I12))

READ(IIFI, 100)
READ(IIFI, 110) IDPROB, NOBJS, NVARS, IK, IE, IS, IPASE0, NGRAYS
OBJS=NOBJS
ARCS=NVARS
NODES=IE
DO 205 I = 1, NODES
   NDNUM(I) = 0
   SUPPLY(I) = 0
205 CONTINUE

DO 210 I = 1, ARCS
   ARCNUM(I) = 0
   FROM(I) = 0
   TO(I) = 0
   CAP(I) = 0
   DO 200 J = 1, OBJS
      COSTS(I, J) = 0
200 CONTINUE
210 CONTINUE

DO 400 I = 1, ARCS
   ARCNUM(I) = I
400 CONTINUE

DO 410 I = 1, NODES
   NDNUM(I) = I
410 CONTINUE

READ(IIFI, 120) NCAP
IF (NCAP.GT.0) THEN
   READ(IIFI, 130) (I, J, NCAPS(I, J), K = 1, NCAP)
END IF
DO 500 I = 1, ARCS
   CAP(I) = 100
500 CONTINUE

READ(IIFI, 120) NCAP
IF (NCAP.GT.0) THEN
   READ(IIFI, 140) (I, CAP(I), K = 1, NCAP)
END IF
DO 510 I = 1, ARCS
   DO 520 J = 1, NODES
      A(I, J) = 0
520 CONTINUE
510 CONTINUE

READ(IIFI, 120) AE
IF (AE.GT.0) THEN
   READ(IIFI, 130) (I, J, A(I, J), K = 1, AE)
END IF
READ(IIFI, 120) BE
IF (BE.GT.0) THEN
   READ(IIFI, 140) (I, SUPPLY(I), K = 1, BE)
END IF
STOT=0
DO 512 I = 1, BE
   STOT=STOT+SUPPLY(I)
512 CONTINUE
DO 514 I=NODES,1,-1
   IF(STOT.GT.0) THEN
      SUPPLY(I)=-SUPPLY(I)
      DO 513 J=1,ARCS
         A(I,J)=-A(I,J)
         STOT=STOT+SUPPLY(I)
      CONTINUE
   ELSE
      GOTO 515
   ENDIF
514 CONTINUE
515 DO 530 I=1,ARCS
   DO 540 J=1,NODES
      IF (A(J,I).EQ.1) THEN
         FROM(I)=J
      END IF
      IF (A(J,I).EQ.-1) THEN
         TO(I)=J
      END IF
   CONTINUE
530 CONTINUE
540 CONTINUE
READ(IIFI,100)
READ(IIFI,100)
READ(IIFI,120) CK
   IF (CK.GT.0) THEN
      READ(IIFI,130) (I,J,COSTS(J,I),K=1,CK)
   END IF
DO 550 I=1,OBJS
   DO 555 J=1,ARCS
      COSTS(J,I)=-COSTS(J,I)
   CONTINUE
550 CONTINUE
RETURN
END

C
SUBROUTINE PRINTQ
C PRINTS Q FILE PARAMETERS TO S FILE
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER IDPROB,INPFLG,WGTFLG,PRTFLG(7),ISEED
INTEGER NODES,ARCS,OBJS
INTEGER NSRCS,NSNKS,NTSRCS,NTSNKS,NSPDM
INTEGER KMINCAP,KMAXCAP,KPCAP
INTEGER MINCST,MAXCST,MPMAX
COMMON /QPARM1/ IDPROB,INPFLG,WGTFLG,PRTFLG,ISEED
COMMON /QPARM2/ MODE,DEGFLG,CRSFLG,INCR,NUP
INTEGER*8 NUP
INTEGER CRSFLG,INCR
INTEGER IQFI,IIFI,IWFI,ISFI,ICPI,INFI,IZFI
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /RAND1/ NSRCS,NSNKS,NTSRCS,NTSNKS,NSPDM
COMMON /RAND2/ KMINCAP,KMAXCAP,KPCAP
COMMON /RAND3/ MINCST,MAXCST,MPMAX
COMMON /LAB1/ IQFI,IIFI,IWFI,ISFI,ICPI,INFI,IZFI
10 FORMAT(2X,'**************************************************')
20 FORMAT(2X,'1. IDPROB ',I7)
30 FORMAT(2X,'2. INPUT TYPE',I7)
40 FORMAT(2X,'3. MODE ',I7)
50 FORMAT(2X,'4. WEIGHTS ',I7)
60 FORMAT(2X,'5. PRINT(1) ',I7)
SUBROUTINE PRINTNET
    C PRINTS A PROBLEM FILE IN NETWORK FORMAT
    IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
    INTEGER NODES,ARCS,OBJS
    INTEGER NDNUM(1000),SUPPLY(1000)
    INTEGER ARCNUM(1000),FROM(1000),TO(1000),CAP(1000)

    WRITE(ISFI,10)
    WRITE(ISFI,20) IDPROB
    WRITE(ISFI,30) INPFLG
    WRITE(ISFI,40) MODE
    WRITE(ISFI,50) WGTFLG
    WRITE(ISFI,60) PRTFLG(1)
    WRITE(ISFI,70) PRTFLG(2)
    WRITE(ISFI,80) PRTFLG(3)
    WRITE(ISFI,90) PRTFLG(4)
    WRITE(ISFI,100) PRTFLG(5)
    WRITE(ISFI,110) PRTFLG(6)
    WRITE(ISFI,115) PRTFLG(7)
    IF (INPFLG.EQ.0) THEN
        WRITE(ISFI,120) ISEED
        WRITE(ISFI,130) NODES
        WRITE(ISFI,140) NSRCS
        WRITE(ISFI,150) NSNKS
        WRITE(ISFI,160) NTSRCS
        WRITE(ISFI,170) NTSNKS
        WRITE(ISFI,180) NSPDM
        WRITE(ISFI,190) ARCS
        WRITE(ISFI,200) KMINCAP
        WRITE(ISFI,210) KMAXCAP
        WRITE(ISFI,220) KPCAP
        WRITE(ISFI,230) OBJS
        WRITE(ISFI,240) MINCST
        WRITE(ISFI,250) MAXCST
        WRITE(ISFI,260) MPMAX
    END IF
    WRITE(ISFI,10)
    RETURN
END
SUBROUTINE PRINTSOL
C PRINTS THE CURRENT FLOW AND COST VALUES TO SFILE
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES,ARCS,OBJS
INTEGER ARCNUM(1000),FROM(1000),TO(1000),CAP(1000)
INTEGER COSTS(1000,10)
INTEGER ROOT,FLOW(1000),TFLG(1000)
INTEGER TOTBAS,NDNUM(1000)
INTEGER PRBAS,NCRASH,NCR1
REAL STIME,CTIME
INTEGER IDPROB,INPFLG,WGTFLG,PRTFLG(7),ISEED
INTEGER PRED(1000),THD(1000),TFL(1000),DUAL(1000),M(1000)
INTEGER IQFI,IIFI,IWFI,ISFI,ICFI,INFI,IZFI
COMMON /NET4/ PRED,THD,TFL,DUAL,M
COMMON /QPARM1/ IDPROB,INPFLG,WGTFLG,PRTFLG,ISEED
COMMON /CDATA1/ PRBAS,STIME,NCRASH
COMMON /CDATA2/ NCR1,CTIME
COMMON /NET5/ ROOT,FLOW,TFLG
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /NET2/ NDNUM,SUPPLY
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /LAB1/ IQFI,IIFI,IWFI,ISFI,ICFI,INFI,IZFI
INTEGER TCOSTS(10),C(1000),TC(10)
5 FORMAT(1X)
10 FORMAT(1X,'SOLUTION: ',I10)
20 FORMAT(6X,'ARC',3X,'FROM',4X,'TO',3X,'FLOW')
30 FORMAT(5X,I4,3X,I4,2X,I4,2X,I5)
40 FORMAT(5X,I4,3X,I4,2X,I4,2X,I5,1X,'B')
50 FORMAT(1X)
60 FORMAT(6X,'OBJ.',3X,'TOTAL')
70 FORMAT(7X,I3,2X,I10)
WRITE(ISFI,5)
IF (PRTFLG(3).EQ.1) THEN
PRBAS=PRBAS+1
WRITE(ISFI,10) PRBAS
WRITE(ISFI,5)
READ(ISFI,20)
DO 200 I=1,ARCS
   IF (TFLG(I).EQ.1) THEN
      WRITE(ISFI,40) ARCNUM(I),FROM(I),TO(I),FLOW(I)
   ELSE
      WRITE(ISFI,30) ARCNUM(I),FROM(I),TO(I),FLOW(I)
   END IF
200 CONTINUE
END IF
IF (PRTFLG(3).EQ.1) THEN
   WRITE(ISFI,50)
   WRITE(ISFI,60)
END IF
TC(1)=0
DO 210 I=1,OBJS
   DO 220 J = 1,ARCS
      TC(I) = TC(I)+COSTS(J,I)*FLOW(J)
   220 CONTINUE
   IF (PRTFLG(3).EQ.1) THEN
      WRITE(ISFI,70) I,TC(I)
   END IF
   TC(I+1)=0
210 CONTINUE
RETURN
END

C
SUBROUTINE PRINTANET
C WRITES NETWORK PARAMETERS TO AN ADBASE INPUT FILE
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER IDPROB,INPFLG,WGTFLG,PRTFLG(7),ISEED
INTEGER NODES,ARCS,OBJS
INTEGER NDNUM(1000),SUPPLY(1000)
INTEGER ARCNUM(1000),FROM(1000),TO(1000),CAP(1000)
INTEGER COSTS(1000,10)
INTEGER IQFI,IIFI,IWFI,ISFI,ICFI,INFI,I2FI
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /QPARM1/ IDPROB,INPFLG,WGTFLG,PRTFLG,ISEED
COMMON /NET2/ NDNUM,SUPPLY
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /LAB1/ IQFI,IIFI,IWFI,ISFI,ICFI,INFI,I2FI
INTEGER I,J,IS,IFASE0,NGRAYS,TEMP1(1000,3),TEMP2(1000,2)
INTEGER AS,BS,A(1000,1000)
100 FORMAT(1X,"PROBLEM ",I8," IN ADBASE FORMAT")
110 FORMAT(8I8)
120 FORMAT(8I8)
130 FORMAT(4(2I3,I12))
140 FORMAT(4(I3,3X,I12))
WRITE(INFI,100) IDPROB
IS=0
IFASE0=0
NGRAYS=0
WRITE(INFI,110) IDPROB,OBJS,ARCS,ARCS,NODES,IS,IFASE0,NGRAYS
WRITE(INFI,120) ARCS
WRITE(INFI,130) (I,I,1,I=1,ARCS)
WRITE(INFI,120) ARCS
WRITE(INFI,140) (I,CAP(I),I=1,ARCS)
DO 500 J=1,ARCS
   DO 510 I=1,NODES
      A(I,J)=0
510 CONTINUE
500 CONTINUE
500 CONTINUE
DO 520 J=1,ARCS
A(FROM(J),J)=1
A(TO(J),J)=-1
520 CONTINUE
DO 523 I=1,NODES
IF (SUPPLY(I).LT.0) THEN
DO 524 J=1,ARCS
A(I,J)=-A(I,J)
524 CONTINUE
END IF
523 CONTINUE
K=0
DO 525 I=1,NODES
DO 530 J=1,ARCS
IF (A(I,J).NE.0) THEN
K=K+1
TEMP1(K,1)=I
TEMP1(K,2)=J
TEMP1(K,3)=A(I,J)
END IF
530 CONTINUE
525 CONTINUE
WRITE(INFI,120) K
WRITE(INFI,130) (TEMP1(L,1),TEMP1(L,2),TEMP1(L,3),L=1,K)
K=0
DO 540 J=1,NODES
IF (SUPPLY(J).NE.0) THEN
K=K+1
TEMP2(K,1)=J
TEMP2(K,2)=ABS(SUPPLY(J))
END IF
540 CONTINUE
WRITE(INFI,120) K
WRITE(INFI,130) (TEMP2(L,1),TEMP2(L,2),L=1,K)
AS=0
WRITE(INFI,120) AS
WRITE(INFI,120) AS
K=0
DO 550 J=1,OBJS
DO 560 I=1,ARCS
K=K+1
TEMP1(K,1)=J
TEMP1(K,2)=I
TEMP1(K,3)=-COSTS(I,J)
560 CONTINUE
550 CONTINUE
WRITE(INFI,120) K
WRITE(INFI,130) (TEMP1(L,1),TEMP1(L,2),TEMP1(L,3),L=1,K)
WRITE(INFI,120) AS
RETURN
END

C SUBROUTINE RANDNET
C GENERATES PROBLEM COEFFICIENTS FOR A RANDOM PROBLEM.
C IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER IDPROB,INPFLG,WGTFLG,PRTFLG(7),ISEED
INTEGER NODES,ARCS,OBJS
INTEGER NSRCS,NSNKS,NTSRCS,NTSNKS,NSPDM
INTEGER KMINCAP,KMAXCAP,KPCAP
INTEGER MINCST, MAXCST, MPMAX
INTEGER NDNUM(1000), SUPPLY(1000)
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER COSTS(1000, 10)
INTEGER IPRED(1000)
INTEGER IFROM(1000), ITO(1000), ICOST(1000), IFLAG(1000), LSNKS(1000)
INTEGER CURFROM, CURTO, TEMP(15), PTR, ARCN
COMMON /QPARM1/ IDPROB, INPFLG, WGTFLG, PRTFLG, ISEED
COMMON /NET1/ NODES, ARCS, OBJ
COMMON /RAND1/ NSRCS, NSNKS, NTSRCS, NTSNKS, NSPDM
COMMON /RAND2/ KMINCAP, KMAXCAP, KPCAP
COMMON /RAND3/ MINCST, MAXCST, MPMAX
COMMON /NET2/ NDNUM, SUPPLY
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /RNET1/ IPRED, IST
COMMON /RNET2/ NSRT
COMMON /RNET3/ IFROM, ITO, ICOST, IFLAG, LSNKS
COMMON /RNET4/ NDSLFT, KCNARCS, NNPDSR, KNTL, ARCN
COMMON /RAN1/ MULT, MODUL, I15, I16, J RAN
COMMON /PTR1/ NFTSPTR

50 FORMAT('ERROR: NUMBER OF NODES TOO LARGE FOR MONFSOLV')
      N ASIZE = 1000
      IF (NODES.GT.NASIZE) CALL ERRO(1)
      IF (ARCS.GT.NASIZE) CALL ERRO(1)
      IF (NSRCS+NSNKS.GT.NODES) CALL ERRO(2)
      IF (NTSRC+G.T.NSRC) CALL ERRO(3)
      IF (NTSNKS.GT.NSNKS) CALL ERRO(4)
      IF (MINCST.GT.MAXCST) CALL ERRO(5)
      IF (KMINCAP.GT.KMAXCAP) CALL ERRO(6)
      IF (NSPDM.LE.NSRCS) CALL ERRO(7)
      IF (NSPDM.LE.NSNKS) CALL ERRO(7)
      IF (ISEED.LT.1) CALL ERRO(8)
      IF (ARCS.GT.(NODES*(NODES-1))) CALL ERRO(9)
      IF (ARCS.LT.(NODES-1)) CALL ERRO(10)
      KCNARCS = 0
      N S KLS = 0
      NLTPT = NODES-NSNKS
      NLTSPTP = NLTPTP+NTSNKS
      NTR = NLTPTP-NSRC
      NFSPTR = NLTPTP+1
      NSSRC = NODES+1
      NNSNK = NODES+2
      NNPDSR = NODES-NSRCS+NTSRC
      NPSNKS = NSNKS-NTSNKS
      NDSLFT = NODES-NSRCS-NTSRC
      NSTP = NSRCS+1
      NFTSPTP = NSRCS-NTSRC+1
      NPSRC = NSRCS-NTSRC
      MULT = 16807
      MODUL = 2147483647
      I15 = 2**15
      I16 = 2**16
      J RAN = ISEED
      ARCN = 0
      DO 400 I = 1, ARCS
        CALL GENPSR
      ARCNUM(I) = I
      400 DO 410 I = 1, NODES
        NDNUM(I) = I
CONTINUE
DO 100 I=1,NSRCS
   IPRED(I) = I
100 CONTINUE
IF (NTR.EQ.0) GO TO 1170
IST=NFTPTR
IPRED(NLTPTR)=0
K=NLTPTR-1
DO 110 I = NFTRPTR,K
   IPRED(I) = I + 1
110 CONTINUE
NTRAVL = 6 * NTR / 10
NTRREM = NTR - NTRAVL
1140 LSORC = 1
1150 IF (NTRAVL.EQ.0) GOTO 1160
   LPICK = INTRAN(1,NTRAVL+NTRREM)
   NTRAVL=NTRAVL-1
   CALL CHAIN(LPICK,LSORC)
   IF (LSORC.EQ.NSRCS) GOTO 1140
   LSORC = LSORC + 1
   GOTO 1150
1160 IF (NTRREM.EQ.0) GO TO 1170
   LPICK=INTRAN(1,NTRREM)
   NTRREM=NTRREM-1
   LSORC = INTRAN(1,NSRCS)
   CALL CHAIN(LPICK,LSORC)
   GOTO 1160
1170 DO 120 I = NFSPTR,NODES
   IPRED(I) = 0
120 CONTINUE
DO 130 LSORC =1,NSRCS
   CALL CHNARC(LSORC)
   DO 140 I = NFSPTR,NODES
      IFLAG(I) = 0
140 CONTINUE
IF (NTR.EQ.0) GO TO 1200
NSKSR=(NSRT*2*NSNKS)/NTR
GOTO 1210
1200 NSKSR=NSNKS/NSRCS+1
1210 IF (NSKSR.LT.2) NSKSR = 2
   IF (NSKSR.GT.NSNKS) NSKSR = NSNKS
   NSRCHN = NSRT
   KNTL = NSNKS
   DO 150 J=1,NSKSR
      ITEM=INTRAN(1,KNTL)
      KNTL=KNTL-1
      DO 160 L = NFSPTR,NODES
         IF (IFLAG(L).EQ.1) GO TO 160
         ITEM = ITEM - 1
         IF (ITEM.EQ.0) GO TO 1230
160 CONTINUE
GOTO 1250
1230 LSNKS(J) = L
   IFLAG(L) = 1
150 CONTINUE
1250 IF (LSORC.NE.NSRCS) GO TO 1270
   DO 170 J = NFSPTR,NODES
      IF (IPRED(J).NE.0) GO TO 170
      IF (IFLAG(J).EQ.1) GO TO 170
      NSKSR=NSKSR+1
170 CONTINUE
LSNKS(NSKSR)=J
IFLAG(J)=1
170 CONTINUE
1270 KS = SUPPLY(LSORC) / NSKSR
K = IPRED(LSORC)
DO 180 I = 1,NSKSR
NSRT=NSRT+1
KSP=INTRAN(1,KS)
J=INTRAN(1,NSKSR)
IFROM(NSRT)=K
LI=LSNKS(I)
ITO(NSRT)=LI
IPRED(LI)=IPRED(LI)+KSP
SUPPLY(LI)=-IPRED(LI)
LI=LSNKS(J)
IPRED(LI)=IPRED(LI)+KS-KSP
SUPPLY(LI)=-IPRED(LI)
N=INTRAN(1,NSRCHN)
K=LSORC
DO 190 II = 1,N
K = IPRED(K)
190 CONTINUE
180 CONTINUE
LI = LSNKS(1)
IPRED(LI)=IPRED(LI)+SUPPLY(LSORC)-(KS*NSKSR)
SUPPLY(LI)=-IPRED(LI)
NSKLS=NSKLS+NSRT
CALL SORT
I = 1
IFROM(NSRT+1)=0
1300 DO 20 J = NFTSPTR,NODES
IFLAG(J)=0
200 CONTINUE
KNTL=NNPSR-1
IT=IFROM(I)
IFLAG(IT)=1
1320 IH=ITO(I)
IFLAG(IH)=1
 KCNARCS=KCNARCS+1
KNTL=KNTL-1
ICAP=NSPDM
JCAP=INTRAN(1,100)
IF (JCAP.GT.KPCAP) GO TO 1330
ICAP=SUPPLY(LSORC)
IF (KMINCAP.GT.ICAP) ICAP = KMINCAP
1330 DO 210 JKL = 1,OBJS
ICOST(JKL+3)=MAXCST
JCAP=INTRAN(1,100)
IF (JCAP.LE.MPMAX) GO TO 210
ICOST(JKL+3)=INTRAN(MINCST,MAXCST)
210 CONTINUE
ICOST(1)=IT
ICOST(2)=IH
ICOST(3)=ICAP
MUP=OBJS+3
ARCN=ARCN+1
FROM(ARCN)=IT
TO(ARCN)=IH
CAP(ARCN)=ICAP
DO 220 JKL = 1,OBJS
COSTS(ARCN, JKM) = ICOST(JKM+3)

CONTINUE
I=I+1
IF (IFROM(I).EQ.IT) GO TO 1320
CALL PICKJ(IT)
IF (I.LE.NSRT) GO TO 1300

CONTINUE
IF (NTSNKS.EQ.0) GO TO 1390
NZ = 0
DO 230 I = NFSPTR,NLTS PTR
   DO 240 J = NFTSPTR,NODES
      IFLAG(J) = 0
   240 CONTINUE
KNTL = NNPSR - 1
IFLAG(I) = 1
CALL PICKJ(I)

CONTINUE
DO 500 I = 1,ARCS - 1
   CURFROM = FROM(I)
   CURTO = TO(I)
   PTR = I
   DO 510 J = I + 1,ARCS
      IF (CURFROM.GT.FROM(J)) THEN
         PTR = J
         CURFROM = FROM(PTR)
         CURTO = TO(PTR)
      ELSE
         IF (CURFROM.EQ.FROM(J)) THEN
            IF (CURTO.GT.TO(J)) THEN
               PTR = J
               CURTO = TO(PTR)
            END IF
         END IF
      END IF
   510 CONTINUE
   TEMP(1) = FROM(I)
   FROM(I) = FROM(PTR)
   FROM(PTR) = TEMP(1)
   TEMP(2) = TO(I)
   TO(I) = TO(PTR)
   TO(PTR) = TEMP(2)
   TEMP(3) = CAP(I)
   CAP(I) = CAP(PTR)
   CAP(PTR) = TEMP(3)
   DO 520 J = 1,OBJS
      TEMP(J) = COSTS(I, J)
      COSTS(I, J) = COSTS(PTR, J)
      COSTS(PTR, J) = TEMP(J)
   520 CONTINUE
500 CONTINUE
RETURN
END

C
SUBROUTINE ERRO(INUMB)
C PRINTS ERROR MESSAGES
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
COMMON /LAB1/ IQFI,IIFI,IWFI,ISFI,ICFI,INFI,IZFI
10 FORMAT(/,1X,'ERROR: PROBLEM IS TO LARGE FOR SOLVER.')
20 FORMAT(/,1X,'ERROR: TOO MANY SOURCES OR SINKS.')
30 FORMAT(/,1X,'ERROR: MORE T-SOURCES THAN SOURCES.')
40 FORMAT(1X,'ERROR: MORE T-SINKS THAN SINKS.')
50 FORMAT(1X,'ERROR: MIN-COST > MAX-COST.')
60 FORMAT(1X,'ERROR: MIN-CAP > MAX-CAP.')
70 FORMAT(1X,'ERROR: TOTAL SUPPLY < NUM. OF SOURCES OR SINKS.')
80 FORMAT(1X,'ERROR: SEED TOO LOW.')
90 FORMAT(1X,'ERROR: TOO MANY ARCS FOR NUMBER OF NODES.')
100 FORMAT(1X,'ERROR: NOT ENOUGH MANY ARCS FOR NUMBER OF NODES.')
110 FORMAT(1X,'LOOP ERROR:.EXTREME POINTS BEYOND POSSIBLE')
120 FORMAT(1X,'ERROR:.INFEASIBLE NETWORK')

IF (INUMB.EQ.1) WRITE(ISFI,10)
IF (INUMB.EQ.2) WRITE(ISFI,20)
IF (INUMB.EQ.3) WRITE(ISFI,30)
IF (INUMB.EQ.4) WRITE(ISFI,40)
IF (INUMB.EQ.5) WRITE(ISFI,50)
IF (INUMB.EQ.6) WRITE(ISFI,60)
IF (INUMB.EQ.7) WRITE(ISFI,70)
IF (INUMB.EQ.8) WRITE(ISFI,80)
IF (INUMB.EQ.9) WRITE(ISFI,90)
IF (INUMB.EQ.10) WRITE(ISFI,100)
IF (INUMB.EQ.11) WRITE(ISFI,110)
IF (INUMB.EQ.12) WRITE(ISFI,120)
STOP
RETURN
END

C
SUBROUTINE GENSUPS
C CALCULATES SUPPLIES FOR SOURCE NODES FOR RANDOM NETWORKS
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER I,J,KS,KSP
INTEGER NODES,ARCS,OBJ
INTEGER NSRCS,NSNKS,NTSRCS,NTSNKS,NSPDM
INTEGER NDNUM(1000),SUPPLY(1000)
COMMON /NET1/ NODES,ARCS,OBJ
COMMON /RAND1/ NSRCS,NSNKS,NTSRCS,NTSNKS,NSPDM
COMMON /NET2/ NDNUM,SUPPLY
DO 100 I = 1,NODES
  SUPPLY(I) = 0
100 CONTINUE
KS = NSPDM/NSRCS
DO 110 I = 1,NSRCS
  KSP = INTRAN(1, KS)
  J = INTRAN(1, NSRCS)
  SUPPLY(I) = SUPPLY(I) + KSP
  SUPPLY(J) = SUPPLY(J) + KS - KSP
110 CONTINUE
J = INTRAN(1, NSRCS)
SUPPLY(J) = SUPPLY(J) + NSPDM - (KS * NSRCS)
RETURN
END

C
SUBROUTINE CHAIN(LPICK,LSORC)
C ADDS NODE LPICK TO THE END OF THE CHAIN WHICH C STARTS AT LSORC.
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
COMMON /RNET1/ IPRED,IST
INTEGER IPRED(1000),IST
INTEGER K,J,L,I,M
K = 0
M = IST
IF (LPICK.EQ.1) IST = IPRED(IST)
DO 100 I = 1,LPICK
  L = K
  K = M
  M = IPRED(K)
100  CONTINUE
IPRED(L) = M
J = IPRED(LSORC)
IPRED(K) = J
IPRED(LSORC) = K
RETURN
END

C SUBROUTINE CHNARC(LSORC)
C PUTS THE ARCS IN THE CHAIN FROM SOURCE LSORC
C INTO THE ITO AND IFROM ARRAYS FOR SORTING
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER IPRED(1000)
INTEGER IFROM(1000),ITO(1000),ICOST(1000),IFLAG(1000),LSNKS(1000)
COMMON /RNET1/ IPRED,IST
COMMON /RNET2/ NSRT
COMMON /RNET3/ IFROM,ITO,ICOST,IFLAG,LSNKS
NSRT=0
KTO=IPRED(LSORC)
110 IF (KTO.EQ.LSORC) GO TO 120
NSRT=NSRT+1
KFROM=IPRED(KTO)
ITO(NSRT)=KTO
IFROM(NSRT)=KFROM
KTO=KFROM
GOTO 110
120 RETURN
END

C SUBROUTINE SORT
C SORTS THE NSRT OF RANDOM NETWORK
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER IFROM(1000),ITO(1000)
COMMON /RNET2/ NSRT
COMMON /RNET3/ IFROM,ITO,ICOST,IFLAG,LSNKS
N=NSRT
M=N
110 M=M/2
IF (M.EQ.0) GO TO 150
K=N-M
J=1
120 I=J
130 L=I+M
IF(IFROM(I)-IFROM(L).LE.0) GO TO 140
IT=IFROM(I)
IFROM(I)=IFROM(L)
IFROM(L)=IT
IT=ITO(I)
ITO(I)=ITO(L)
ITO(L)=IT
I=I-M
IF (I.GE.1) GO TO 130
140 J=J+1
IF (J.GE.1) GO TO 120
GO TO 110
150 RETURN
SUBROUTINE PICKJ(IT)
C CREATES A RANDOM NUMBER OF ARCS OUT OF NODE IT
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER IFROM(1000), ITO(1000), ICOST(1000), IFLAG(1000)
INTEGER NDNUM(1000), SUPPLY(1000)
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER COSTS(1000, 10), ARCNUM
INTEGER NODES, ARCS, OBJS
COMMON /NET1/ NODES, ARCS, OBJS
COMMON /NET2/ NDNUM, SUPPLY
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /RNET3/ IFROM, ITO, ICOST, IFLAG, LSNKS
COMMON /RNET4/ NDSLFT, KCNARCS, NNPSR, KNTL, ARCN
COMMON /RAND1/ NSRCS, NSNKS, NTSRCs, NTSNKS, NSPDM
COMMON /RAND2/ KMINCAP, KMAXCAP, KPCAP
COMMON /RAND3/ MINCST, MAXCST, MPMAX
COMMON /PTR1/ NFTSPTR
IF ((NDSLFT-1)*2.LE.ARCS-KCNARCS-1) GO TO 110
NDSLFT = NDSLFT-1
GOTO 1111
110 IF ((ARCS-KCNARCS+NNPSR-KNTL-1)/NDSLFT-NNPSR+1.LT.0) GO TO 130
120 K=NNPSR
GO TO 150
130 NUPBND=(ARCS-KCNARCS-NDSLFT)/NDSLFT*2
140 K=INTRAN(1, NUPBND)
IF (NDSLFT.EQ.1) K=ARCS-KCNARCS
IF ((NDSLFT-1) *(NNPSR-1).LT.ARCS-KCNARCS-K) GO TO 140
150 NDSLFT=NDSLFT-1
DO 200 J = 1, K
NN=INTRAN(1, KNTL)
KNTL=KNTL-1
DO 210 L = NFTSPTR, NODES
   IF (IFLAG(L).EQ.1) GO TO 210
   NN=NN-1
   IF (NN.EQ.0) GO TO 170
210 CONTINUE
GOTO 1111
170 IFLAG(L)=1
ICAP=NSPDM
JCAP=INTRAN(1, 100)
IF (JCAP.GT.KPCAP) GO TO 180
ICAP=INTRAN(KMINCAP, KMAXCAP)
180 DO 220 JKL = 1, OBJS
   ICOST(JKL+3) = INTRAN(MINCST, MAXCST)
220 CONTINUE
ICOST(1)=IT
ICOST(2)=L
ICOST(3)=ICAP
MUP=OBJ+3
ARCNUM=ARCNUM+1
FROM(ARCNUM)=IT
TO(ARCNUM)=L
CAP(ARCNUM)=ICAP
DO 230 JKM = 1, OBJS
   COSTS(ARCNUM, JKM)=ICOST(JKM+3)
230 CONTINUE
KCNARCS=KCNARCS+1
200 CONTINUE
FUNCTION INTRAN(ILOW,IHIGH)
C GENERATES AN INTEGER RANDOM NUMBER BETWEEN ILOW AND IHIGH
C IF ILOW.GT.IHIGH THEN INTRAN RETURNS IRAN =IHIGH
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
COMMON /RAN1/ MULT,MODUL,I15,I16,JRAN
IXHI=JRAN/I16
IXLO=JRAN-IXHI*I16
IXALO=IXLO*MULT
LEFTLO=IXALO/I16
IXAHI=IXHI*MULT
IFULHI=IXAHI+LEFTLO
IRTLO=IXALO-LEFTLO*I16
IOVER=IFULHI/I15
IRTHI=IFULHI-IOVER*I15
JRAN=((IRTLO-MODUL)+IRTHI*I16)+IOVER
IF (JRAN.LT.0) JRAN=JRAN+MODUL
J=IHIGH-ILOW+1
IF (J.LE.0) GO TO 110
INTRAN = ABS(MOD(JRAN,J))+ILOW
RETURN
110 INTRAN = IHIGH
RETURN
END

FUNCTION REALRAN(RLOW, RHIGH)
C GENERATES AN REAL RANDOM NUMBER BETWEEN RLOW AND RHIGH
C IF RLOW.GT.RHIGH THEN REALRAN RETURNS REALRAN =RHIGH
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
RRANGE=RHIGH-RLOW
IF (RRANGE.GT.0) GO TO 110
REALRAN=RHIGH
RETURN
110 ISCALE=10000
IVALUE=INTRAN(0, ISCALE)
RSCALE=ISCALE
RVALUE=IVALUE
REALRAN=RLOW+(RVALUE/RSCALE)*RRANGE
RETURN
END

SUBROUTINE CODING(A,FLGS,CDWORD)
C CALCULATES THE CODE FOR CURRENT BASIS.
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER A,FLGS(A),CDWORD(30)
INTEGER I,J
DO 100 I = 1,30
   CDWORD(I) = 0
100 CONTINUE
DO 200 I = 1,30
   DO 210 J = 1,30
      K = (I-1)*30+J
      IF (K.GT.A) GOTO 220
      IF (FLGS(K).EQ.1) THEN
         CDWORD(I) = CDWORD(I)+2**J
      END IF
210 CONTINUE
200 CONTINUE
SUBROUTINE CLIST
C Initializes and creates the basis list
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER BCODE(30), NEXT, BL(400000, 33), TOTBAS
COMMON /LIST1/ BCODE, NEXT, BL, TOTBAS
INTEGER I, J
DO 100 I = 1, 33
   BL(1, I) = 0
100 CONTINUE
DO 110 I = 1, 30
   BL(1, I) = BCODE(I)
110 CONTINUE
NEXT = 1
RETURN
END

SUBROUTINE ZMAKE
C Initializes and creates the non-dominated vectors list
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
COMMON /NET1/ NODES, ARCS, OBJS
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /NET5/ ROOT, FLOW, TPLG
COMMON /LIST2/ ZLIST, NEXTZ, TOTALZ
INTEGER NODES, ARCS, OBJS
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER COSTS(1000, 10)
INTEGER ROOT, FLOW(1000), TPLG(1000)
INTEGER ZLIST(400000, 12), NEXTZ, TOTALZ
INTEGER I, J
DO 100 I = 1, 12
   ZLIST(1, I) = 0
100 CONTINUE
DO 200 I = 1, OBJS
   DO 250 J = 1, ARCS
      ZLIST(1, I) = ZLIST(1, I) + FLOW(J) * COSTS(J, I)
   250 CONTINUE
200 CONTINUE
NEXTZ = 2
TOTALZ = 1
RETURN
END

SUBROUTINE EFFSRCH(FS)
C Sets up and solves for adjacent efficient points
C from current basis
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES, ARCS, OBJS
INTEGER MODE, DEGFLG
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER PRED(1000), THD(1000), TPL(1000), DUAL(1000), M(1000)
INTEGER ROOT, FLOW(1000), TPLG(1000)
INTEGER BCODE(30), NEXT, BL(400000, 33), TOTBAS
INTEGER COSTS(1000, 10)
COMMON /QPARM2/ MODE, DEGFLG, CRSFLG, INCR, NUP
INTEGER*8 NUP
INTEGER CRSFLG, INCR
COMMON /NET1/ NODES, ARCS, OBJS
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /NET4/ PRED, THD, TFL, DUAL, M
COMMON /NET5/ ROOT, FLOW, TFLG
COMMON /LIST1/ BCODE, NEXT, BL, TOTBAS
INTEGER CCODE(30), CL(400000, 30)
COMMON /LIST3/ CCODE, CL
INTEGER PRICES(1000, 10), STAT, IFLAG(1000)
INTEGER NEW, G1, G2, BIGD, NFLG(1000), W, MU, BIGK, OLD
INTEGER SSTAT, KF, CAPFLAG(1000)
COMMON /EFF1/ PRICES, IFLAG
COMMON /EFF2/ NEW, G1, G2, BIGD, NFLG, W, MU, BIGK, OLD
COMMON /EFF3/ SSTAT
INTEGER I, J, JA, FS, CNCODE, E, PARENT
INTEGER COLS, K, L, II
DOUBLE PRECISION A(10, 1000), C(1000), RHS(10)
DOUBLE PRECISION BINVERSE(10, 10)
INTEGER BINDEX(100), BFLAG(1000)
INTEGER STAT, KF, OUT, NEWCAP
INTEGER CURRENT, PREVIOUS, OLDTHD, CURTEST, FLAG1, FLAG2
CURTEST = NEXT
CURRENT = NEXT
PREVIOUS = CURRENT
OLDTHD = BL(CURRENT, 33)
DO 100 I=1, ARCS
   IFLAG(I) = TFLG(I)
   IF ((TFLG(I).EQ.0).AND.(FLOW(I).EQ.CAP(I))) THEN
      CAPFLAG(I) = 1
   ELSE
      CAPFLAG(I) = 0
   ENDIF
100 CONTINUE
JA = 0
KF = -1
COLS = ARCS - NODES + 1
DO 150 I = 1, OBJS
   K = 0
   DO 160 L = 1, ARCS
      IF (TFLG(L).EQ.0) THEN
         K = K + 1
      ENDIF
   160 CONTINUE
   A(I, COLS + 1) = FLOAT(0)
   DO 170 J = 1, OBJS
      A(I, COLS + 1 + J) = FLOAT(0)
   170 CONTINUE
   A(I, COLS + 1) = FLOAT(0)
   DO 180 J = 1, ARCS
      IF (TFLG(J).EQ.0) THEN
         JA = JA + 1
         DO 210 I = 1, (COLS + 1)
BFLAG(I)=0
210 CONTINUE
FLAG1=0
FLAG2=0
DO 220 I=1,OBJS
   A(I,COLS+1)=-A(I,JA)
   IF (FLOW(L).EQ.0) THEN
      IF (A(I,COLS+1).LE.0) THEN
         FLAG1 = FLAG1+1
      END IF
      IF (A(I,COLS+1).EQ.0) THEN
         FLAG2 = FLAG2+1
      END IF
   ELSE
      IF (A(I,COLS+1).GE.0) THEN
         FLAG1 = FLAG1+1
      END IF
      IF (A(I,COLS+1).EQ.0) THEN
         FLAG2 = FLAG2+1
      END IF
   END IF
   DO 230 II=1,OBJS
      BINVERSE(I,II)=A(I,COLS+1+II)
230 CONTINUE
   BINDEX(I)=COLS+1+I
   BFLAG(COLS+1+I)=1
220 CONTINUE
   IF (FLAG1.EQ.FLAG2) GOTO 240
   IF (FLAG1.EQ.OBJS) GOTO 200
240 CALL SOLVE(A,C,RHS,BINVERSE,BINDEX,BFLAG,STAT)
   IF (STAT.NE.1) GOTO 200
   NEW=J
   CALL RATTEST(NEW,G1,G2,BIGD,NFLG,W,MU,BIGK,OLD)
300 CNODE=TO(NEW)
   E=-1
310 IF (CNODE.EQ.W) THEN
      IF (E.EQ.-1) THEN
         CNODE=FROM(NEW)
         E=1
         GOTO 310
      ELSE
         GOTO 200
      END IF
   END IF
   OUT=ABS(M(CNODE))
   IF (FLOW(NEW).EQ.0) THEN
      IF (F((E*M(CNODE)*G2)).LT.0) THEN
         IF((FLOW(OUT)-BIGD.EQ.0).AND.(BIGD.LT.CAP(NEW))) THEN
            NEWCAP=CAPFLAG(NEW)
            TFLG(NEW)=1
            CAPFLAG(NEW)=0
            TFLG(OUT)=0
            CAPFLAG(OUT)=0
            CALL CODING(ARCS,TFLG,BCODE)
            CALL CAPCODE(ARCS,CAPFLAG,CCODE)
            CALL SEARCH(BCODE,BL,CCODE,CL,SSTAT,PARENT)
            IF (SSTAT.NE.0) THEN
               CURRENT = TOTBAS+1
               TOTBAS=TOTBAS+1
            CALL INSERT(BCODE,BL,CCODE,CL,PARENT,SSTAT,TOTBAS,JA,NEXT)
BL(PREVIOUS,33) = CURRENT
BL(CURRENT,33) = OLDTHD
PREVIOUS = CURRENT

END IF
TFLG(NEW) = 0
CAPFLAG(NEW) = NEWCAP
TFLG(OUT) = 1
CAPFLAG(OUT) = 0
END IF

IF (E*M(CNODE)*G2) .GT. 0 THEN
IF ((FLOW(OUT) + BIGD.EQ.CAP(OUT)).AND.(BIGD.LT.CAP(NEW))) THEN
NEWCAP = CAPFLAG(NEW)
TFLG(NEW) = 1
CAPFLAG(NEW) = 0
TFLG(OUT) = 0
CAPFLAG(OUT) = 1
CALL CODING(ARCS, TFLG, BCODE)
CALL CAPCODE(ARCS, CAPFLAG, CCODE)
CALL SEARCH(BCODE, BL, CCODE, CL, SSTAT, PARENT)
IF (SSTAT .NE. 0) THEN
CURRENT = TOTBAS + 1
TOTBAS = TOTBAS + 1
CALL INSERT(BCODE, BL, CCODE, CL, PARENT, SSTAT, TOTBAS, JA, NEXT)
BL(PREVIOUS, 33) = CURRENT
BL(CURRENT, 33) = OLDTHD
PREVIOUS = CURRENT
END IF
TFLG(NEW) = 0
CAPFLAG(NEW) = NEWCAP
TFLG(OUT) = 1
CAPFLAG(OUT) = 0
END IF

END IF

IF (FLOW(NEW) + BIGD.EQ.CAP(NEW)) THEN
NEWCAP = CAPFLAG(NEW)
CAPFLAG(NEW) = 1
CALL CODING(ARCS, TFLG, BCODE)
CALL CAPCODE(ARCS, CAPFLAG, CCODE)
CALL SEARCH(BCODE, BL, CCODE, CL, SSTAT, PARENT)
IF (SSTAT .NE. 0) THEN
CURRENT = TOTBAS + 1
TOTBAS = TOTBAS + 1
CALL INSERT(BCODE, BL, CCODE, CL, PARENT, SSTAT, TOTBAS, JA, NEXT)
BL(PREVIOUS, 33) = CURRENT
BL(CURRENT, 33) = OLDTHD
PREVIOUS = CURRENT
END IF
CAPFLAG(NEW) = NEWCAP
END IF

END IF

IF (DEGFLG .EQ. 0) THEN
IF ((FLOW(OUT) .EQ. CAP(OUT)).OR. (FLOW(OUT) .EQ. 0)) THEN
NEWCAP = CAPFLAG(NEW)
CAPFLAG(NEW) = 0
TFLG(NEW) = 1
TFLG(OUT) = 0
IF (FLOW(OUT) .EQ. CAP(OUT)) THEN
CAPFLAG(OUT) = 1
ELSE
CAPFLAG(OUT) = 0
END IF
CALL CODING(ARCS,TFLG,BCODE)
CALL CAPCODE(ARCS,CAPFLAG,CCODE)
CALL SEARCH(BCODE,BL,CCODE,CL,SSTAT,PARENT)
IF (SSTAT.NE.0) THEN
  CURRENT = TOTBAS+1
  TOTBAS=TOTBAS+1
CALL INSERT(BCODE,BL,CCODE,CL,PARENT,SSTAT,TOTBAS,JA,NEXT)
  BL(PREVIOUS,33)=CURRENT
  BL(CURRENT,33)=OLDTHD
  PREVIOUS = CURRENT
END IF
  CAPFLAG(NEW)=NEWCAP
  TFLG(NEW)=0
  TFLG(OUT)=1
  CAPFLAG(OUT)=0
END IF
ENDIF
ENDIF
IF ((E*M(CNODE)*G2).EQ.CAP(NEW)) THEN
  IF ((E*M(CNODE)*G2).LT.0) THEN
    IF ((FLOW(OUT)-BIGD.EQ.0).AND.(BIGD.LT.CAP(NEW))) THEN
      NEWCAP=CAPFLAG(NEW)
      TFLG(NEW)=1
      CAPFLAG(NEW)=0
      TFLG(OUT)=0
      CALL CODING(ARCS,TFLG,BCODE)
      CALL CAPCODE(ARCS,CAPFLAG,CCODE)
      CALL SEARCH(BCODE,BL,CCODE,CL,SSTAT,PARENT)
      IF (SSTAT.NE.0) THEN
        CURRENT = TOTBAS+1
        TOTBAS=TOTBAS+1
        CALL INSERT(BCODE,BL,CCODE,CL,PARENT,SSTAT,TOTBAS,JA,NEXT)
        BL(PREVIOUS,33)=CURRENT
        BL(CURRENT,33)=OLDTHD
        PREVIOUS = CURRENT
      END IF
      TFLG(NEW)=0
      CAPFLAG(NEW)=NEWCAP
      TFLG(OUT)=1
      CAPFLAG(OUT)=0
    END IF
  END IF
  IF ((E*M(CNODE)*G2).GT.0) THEN
    IF ((FLOW(OUT)+BIGD.EQ.CAP(OUT)).AND.(BIGD.LT.CAP(NEW))) THEN
      NEWCAP=CAPFLAG(NEW)
      TFLG(NEW)=1
      CAPFLAG(NEW)=0
      TFLG(OUT)=0
      CALL CODING(ARCS,TFLG,BCODE)
      CALL CAPCODE(ARCS,CAPFLAG,CCODE)
      CALL SEARCH(BCODE,BL,CCODE,CL,SSTAT,PARENT)
      IF (SSTAT.NE.0) THEN
        CURRENT = TOTBAS+1
        TOTBAS=TOTBAS+1
        CALL INSERT(BCODE,BL,CCODE,CL,PARENT,SSTAT,TOTBAS,JA,NEXT)
        BL(PREVIOUS,33)=CURRENT
        BL(CURRENT,33)=OLDTHD
      END IF
    END IF
  END IF
PREVIOUS = CURRENT
END IF
TFLG(NEW)=0
CAPFLAG(NEW)=NEWCAP
TFLG(OUT)=1
CAPFLAG(OUT)=0
END IF
END IF
IF (FLOW(NEW)-BIGD.EQ.0) THEN
NEWCAP=CAPFLAG(NEW)
CAPFLAG(NEW)=0
CALL CODING(ARCS,TFLG,BCODE)
CALL CAPCODE(ARCS,CAPFLAG,CCODE)
CALL SEARCH(BCODE,BL,CCODE,CL,SSTAT,PARENT)
IF (SSTAT.NE.0) THEN
CURRENT = TOTBAS+1
TOTBAS=TOTBAS+1
CALL INSERT(BCODE,BL,CCODE,CL,PARENT,SSTAT,TOTBAS,JA,NEXT)
BL(PREVIOUS,33)=CURRENT
BL(CURRENT,33)=OLDTHD
PREVIOUS = CURRENT
END IF
CAPFLAG(NEW)=NEWCAP
END IF
END IF
IF (DEGFLG.EQ.0) THEN
IF ((FLOW(OUT).EQ.CAP(OUT)).OR.(FLOW(OUT).EQ.0)) THEN
NEWCAP=CAPFLAG(NEW)
TFLG(NEW)=1
CAPFLAG(NEW)=0
TFLG(OUT)=0
IF (FLOW(OUT).EQ.CAP(OUT)) THEN
CAPFLAG(OUT)=1
ELSE
CAPFLAG(OUT)=0
END IF
CALL CODING(ARCS,TFLG,BCODE)
CALL CAPCODE(ARCS,CAPFLAG,CCODE)
CALL SEARCH(BCODE,BL,CCODE,CL,SSTAT,PARENT)
IF (SSTAT.NE.0) THEN
CURRENT = TOTBAS+1
TOTBAS=TOTBAS+1
CALL INSERT(BCODE,BL,CCODE,CL,PARENT,SSTAT,TOTBAS,JA,NEXT)
BL(PREVIOUS,33)=CURRENT
BL(CURRENT,33)=OLDTHD
PREVIOUS = CURRENT
END IF
TFLG(NEW)=0
CAPFLAG(NEW)=NEWCAP
TFLG(OUT)=1
CAPFLAG(OUT)=0
END IF
ENDIF
CNODE = PRED(CNODE)
GOTO 310
END IF
200 CONTINUE
NEXT=BL(CURTEST,33)
IF (NEXT.EQ.0) THEN
FS=0
SUBROUTINE PRCCAL
C CALCULATES PRICES FOR NONBASIC ARCS
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES,ARCS,OBJS
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER PRED(1000), THD(1000), TFL(1000), DUAL(1000), M(1000)
INTEGER ROOT, FLOW(1000), TFLG(1000)
INTEGER COSTS(1000,10)
COMMON /NET1/ NODES, ARCS, OBJS
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /NET4/ PRED, THD, TFL, DUAL, M
COMMON /NET5/ ROOT, FLOW, TFLG
INTEGER PRICES(1000,10), STAT, IFLAG(1000)
COMMON /EFF1/ PRICES, IFLAG
INTEGER J, I, TEMP(1000), DLS(1000), PRS(1000)
DO 100 I = 1, OBJS
  DO 200 J = 1, ARCS
    TEMP(J) = COSTS(J, I)
  200 CONTINUE
CALL DUALCAL(ROOT, NODES, ARCS, THD, M, TEMP, PRED, DLS)
CALL PRICING(NODES, ARCS, FROM, TO, DLS, TEMP, TFLG, PRS)
DO 300 J = 1, ARCS
  PRICES(J, I) = PRS(J)
  300 CONTINUE
100 CONTINUE
RETURN
END

SUBROUTINE SOLVE(A,C,RHS,BINVERSE,BINDEX,BFLAG,STAT)
C SOLVES LP TO TEST FOR ADJACENT EFFICIENT BASES
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES, ARCS, OBJS
COMMON /NET1/ NODES, ARCS, OBJS
INTEGER PRICES(1000,10), STAT, IFLAG(1000)
COMMON /EFF1/ PRICES, IFLAG
INTEGER I,J,COLS,L, ROW
DOUBLE PRECISION BCS(10), CP, P
DOUBLE PRECISION Z, CBIA, CC, CBI(1000), ANEW, ACOL(10)
DOUBLE PRECISION A(10,1000), C(1000), RHS(10), PIVOT
DOUBLE PRECISION BINVERSE(10,10)
INTEGER BINDEX(10), BFLAG(1000)
INTEGER STAT, PINDEX, COLUMN
INTEGER K
INTEGER INUM
Z = 0.00001
STAT = 0
COLS = ARCS - NODES + 1
100 DO 200 L=1, (COLS+1+OBJS)
  IF (BFLAG(L).EQ.0) THEN
    CBIA = FLOAT(0)
    DO 150 I=1, OBJS
      CBI(I) = FLOAT(0)
      DO 160 J=1, OBJS
        CBI(I) = CBI(I) + C(BINDEX(J)) * BINVERSE(J, I)
      160 CONTINUE
    150 CONTINUE
  DO 140 I=1, OBJS
    CBI(I) = CBI(I) + CBIA
  140 CONTINUE
  200 CONTINUE
CONTINUE
CBI\text{A} = CBI\text{A} + CBI(I) \times A(I,L)

CONTINUE
CC = C(L) - CBI\text{A}
IF (CC.GT.Z) THEN
COLUMN = L
GOTO 300
ENDIF
ENDIF
CONTINUE
STAT = 1
GOTO 1000

ROW = 0
PIVOT = 0
PINDEX = 1000
DO 400 I = 1, OBJS
ANEW = 0
DO 450 J = 1, OBJS
ANEW = BINVERSE(I, J) \times A(J, COLUMN) + ANEW
450 CONTINUE
ACOL(I) = ANEW
IF (ANEW.GT.Z) THEN
IF (BINDEX(I).LT.PINDEX) THEN
PIVOT = ANEW
ROW = I
PINDEX = BINDEX(I)
ENDIF
ENDIF
CONTINUE
IF (PINDEX.EQ.1000) GOTO 1000
BFLAG(BINDEX(ROW)) = 0
BINDEX(ROW) = COLUMN
BFLAG(COLUMN) = 1
DO 500 I = 1, OBJS
IF (I.NE.ROW) THEN
DO 550 J = 1, OBJS
BINVERSE(I, J) = BINVERSE(I, J) - (ACOL(I)/PIVOT) \times BINVERSE(ROW, J)
550 CONTINUE
ENDIF
500 CONTINUE
DO 600 J = 1, OBJS
BINVERSE(ROW, J) = BINVERSE(ROW, J) / PIVOT
600 CONTINUE
GOTO 100
1000 RETURN
END

SUBROUTINE SEARCH(B, BL, C, CL, S, P)
SEARCHES THE BINARY TREE TO DETERMINE IF THE NEW BASIS SHOULD BE ADDED OR IS ALREADY THERE
IMPLICIT REAL*8 (A-H, O-Z), INTEGER*4 (I-N)
INTEGER B(30), BL(400000, 33)
INTEGER C(30), CL(400000, 30)
INTEGER S, P
INTEGER I, G, J
S = 0
I = 1
DO 200 J = 1, 30
IF (B(J).GT.BL(I, J)) THEN
G = 1
END IF
IF (B(J) .LT. BL(I,J)) THEN
G = -1
END IF
200 CONTINUE
IF (G.EQ.0) THEN
DO 250 J = 1,30
IF (C(J) .GT. CL(I,J)) THEN
G = 1
END IF
IF (C(J) .LT. CL(I,J)) THEN
G = -1
END IF
250 CONTINUE
END IF
IF (G.EQ.1) THEN
IF (BL(I,32).EQ.0) THEN
P = I
S = 1
GOTO 300
ELSE
I = BL(I,32)
GOTO 100
END IF
END IF
IF (G.EQ.-1) THEN
IF (BL(I,31).EQ.0) THEN
P = I
S = -1
GOTO 300
ELSE
I = BL(I,31)
GOTO 100
END IF
END IF
IF (G.EQ.0) THEN
P = 0
S = 0
END IF
300 RETURN
END
C
SUBROUTINE INSERT(B,BL,C,CL,P,S,T,J,N)
C THIS SUBROUTINE INSERTS THE NEW BASIS INTO THE
C BASIS LIST AND UPDATES POINTERS
C IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER B(30),BL(400000,33)
INTEGER C(30),CL(400000,30)
INTEGER P,S,T,J,N
INTEGER I
DO 100 I = 1,30
BL(T,I) = B(I)
CL(T,I) = C(I)
100 CONTINUE
BL(T,31) = 0
BL(T,32) = 0
BL(T,33) = 0
IF (S.LT.0) THEN
BL(P,31) = T
ELSE
   BL(P,32) = T
END IF
RETURN
END

C
SUBROUTINE ZSEARCH(S,P,ZS)
C THIS SUBROUTINE SEARCHES THE BINARY TREE TO
C DETERMINE IF THE NEW NONDOMINATED VECTOR SHOULD
C BE ADDED OR IS ALREADY THERE
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
COMMON /NET1/ NODES,ARCS,OBJ
COMMON /NET3/ ARCNUM,FRM,TO,CAP,CSTS
COMMON /NET5/ ROOT,FLOW,TFLG
COMMON /LIST2/ ZLIST,NEXTZ,TOTALZ
INTEGER NODES,ARCS,OBJ
INTEGER ARCN(1000),FRM(1000),TO(1000),CAP(1000)
INTEGER CSTS(1000,10)
INTEGER ROOT,FLOW(1000),TFLG(1000)
INTEGER ZLIST(400000,12),NEXTZ,TOTALZ
INTEGER I,G,J,K,ZS(10),S,P
S = 0
I = 1
DO 50 K=1,OBJ
   ZS(K)=0
   DO 70 J=1,ARCS
      ZS(K)=ZS(K)+CSTS(J,K)*FLOW(J)
    70 CONTINUE
  50 CONTINUE
  100 G = 0
  DO 200 J = 1,OBJ
     IF (ZS(J).GT.ZLIST(I,J)) THEN
        G = 1
     ELSE
        IF (ZS(J).LT.ZLIST(I,J)) THEN
           G = -1
        END IF
     END IF
  200 CONTINUE
  IF (G.EQ.1) THEN
     IF (ZLIST(I,12).EQ.0) THEN
        P = I
        S = 1
     GOTO 300
     ELSE
        I = ZLIST(I,12)
     GOTO 100
     END IF
  END IF
  IF (G.EQ.-1) THEN
     IF (ZLIST(I,11).EQ.0) THEN
        P = I
        S = -1
        GOTO 300
     ELSE
        I = ZLIST(I,11)
     GOTO 100
     END IF
  END IF
  IF (G.EQ.0) THEN
P = 0
S = 0
END IF
300 RETURN
END

C

SUBROUTINE ZINSERT(S,P,ZS)
C INSERTS THE NEW BASIS INTO THE BASIS LIST AND
C UPDATES POINTERS
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
COMMON /NET1/ NODES, ARCS, OBJS
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /NET5/ ROOT, FLOW, TFLG
COMMON /LIST2/ ZLIST, NEXTZ, TOTALZ
INTEGER NODES, ARCS, OBJS
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER COSTS(1000, 10)
INTEGER ROOT, FLOW(1000), TFLG(1000)
INTEGER ZLIST(400000, 12), NEXTZ, TOTALZ
INTEGER I
INTEGER S, P, ZS(10)
DO 100 I = 1, OBJS
   ZLIST(NEXTZ, I) = ZS(I)
100 CONTINUE
ZLIST(NEXTZ, 11) = 0
ZLIST(NEXTZ, 12) = 0
IF (S.LT.0) THEN
   ZLIST(P, 11) = NEXTZ
ELSE
   ZLIST(P, 12) = NEXTZ
END IF
NEXTZ = NEXTZ + 1
TOTALZ = TOTALZ + 1
RETURN
END

C

SUBROUTINE CRASH
C CRASHES TO THE NEXT BASIS
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES, ARCS, OBJS
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER COSTS(1000, 10)
INTEGER ROOT, FLOW(1000), TFLG(1000)
INTEGER ZLIST(400000, 12), NEXTZ, TOTALZ
INTEGER BCODE(30), NEXT, BL(400000, 33), TOTBAS
INTEGER ZS(10), PRENT, ZSTAT
INTEGER CCODE(30), CL(400000, 30)
INTEGER IDPROB, INPFLG, WGTFLG, PRTFLG(7), ISEED, DEGFLG
COMMON /QPARM1/ IDPROB, INPFLG, WGTFLG, PRTFLG, ISEED
COMMON /QPARM2/ MODE, DEGFLG, CRSFLG, INCR, NUP
INTEGER*8 NUP
INTEGER CRSFLG, NCR1, INCR
REAL CTIME, INTIME
COMMON /LIST3/ CCODE, CL
COMMON /LIST2/ ZLIST, NEXTZ, TOTALZ
COMMON /NET1/ NODES, ARCS, OBJS
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /NET5/ ROOT, FLOW, TFLG
COMMON /LIST1/ BCODE, NEXT, BL, TOTBAS
COMMON /CDATA2/ NCR1, CTIME
COMMON /LAB1/ IQFI, IIIFI, IWFI, ISFI, ICFI, INFI, IZFI
INTEGER NEWT(1000), CAPFLAG(1000), DEL
INTEGER NEWARC, G1, G2, BIGD, NFLG(1000), W, MU, BIGK, OUT
INTEGER PTR1, PTR2, CCOUNT

INTIME = MCLOCK()
NCR1 = NCR1+1
DO 100 I = 1,30
   BCODE(I) = BL(NEXT, I)
   CCODE(I) = CL(NEXT, I)
100 CONTINUE

CALL UNCODING(ARCS, BCODE, NEWT)
CALL UNCODING(ARCS, CCODE, CAPFLAG)
IF (CRSFLG.EQ.1) GOTO 400
PTR1 = 1
PTR2 = 1
CCOUNT = 0
110 IF ((TFLG(PTR1).EQ.0).AND.(NEWT(PTR1).EQ.1)) THEN
   PTR2 = PTR1
   NEWARC = PTR1
   CALL RATTEST2(NEWARC, G1, G2, BIGD, NFLG, W, MU, BIGK, OUT, NEWT)
   CCOUNT = CCOUNT+1
   IF (CCOUNT.EQ.ARCS+1) GOTO 400
   IF (OUT.NE.0) THEN
      CALL UPDATE2(NEWARC, G1, G2, BIGD, NFLG, W, MU, BIGK, OUT)
   END IF
   PTR1 = PTR2+1
   IF (PTR1.GT.ARCS) THEN
      PTR1 = 1
   END IF
   PTR2 = PTR1
   GOTO 110
ELSE
   PTR1 = PTR1+1
   IF (PTR1.GT.ARCS) THEN
      PTR1 = 1
   END IF
   IF (PTR1.EQ.PTR2) THEN
      GOTO 220
   ELSE
      GOTO 110
   ENDIF
END IF
220 DO 300 I = 1, ARCS
   IF (CAPFLAG(I).EQ.1) THEN
      IF (FLOW(I).NE.CAP(I)) THEN
         GOTO 400
      ELSE
         IF ((FLOW(I).NE.0).AND.(TFLG(I).EQ.0)) THEN
            GOTO 400
         ELSE
            IF ((FLOW(I).NE.0).AND.(TFLG(I).EQ.0)) THEN
               GOTO 400
            ELSE
               GOTO 300
            END IF
         END IF
      END IF
   END IF
300 CONTINUE
GOTO 500
400 CALL CRASH2(NEWT, CAPFLAG)
500 CALL PRCCAL
INTIME = MCLOCK() - INTIME
CTIME = CTIME + INTIME
CALL ZSEARCH(ZSTAT, PRENT, ZS)
IF (ZSTAT.NE.0) THEN
   CALL ZINSERT(ZSTAT, PRENT, ZS)
C
SUBROUTINE RATTEST2(EDGE,G1,G2,BIGD,NFLG,W,MU,BIGK,OUT,NEWT)
C ASSUMES THAT 1) FOR EACH ARC THAT IS NOT IN NETWORK THERE IS
C AT LEAST ONE THAT OUGHTN'T BE AND THAT ONE SHOULD BE IN CYCLE
C 2) PIVOTS CAN BE MADE BETWEEN TWO ARCS IN SAME CIRCUIT WITHOUT
C VIOLATING CAPACITY CONSTRAINTS. NOTE: IF THERE ARE TWO ARCS
C TIED TO LEAVE THE LAST ONE FOUND WILL BE SELECTED.
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES,ARCS,OBJS
INTEGER ARCNUM(1000),FROM(1000),TO(1000),CAP(1000)
INTEGER COSTS(1000,10)
INTEGER PRED(1000),THD(1000),TFL(1000),DUAL(1000),M(1000)
INTEGER ROOT,FLOW(1000),TFLG(1000)
INTEGER G1,G2,BIGD,W,MU,BIGK,OUT
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /NET4/ PRED,THD,TFL,DUAL,M
COMMON /NET5/ ROOT,FLOW,TFLG
INTEGER NEWT(1000)
INTEGER NFLG(1000)
INTEGER I,CNODE,EDGE
BIGK = -1
MU= 0
DO 100 I = 1,NODES
   NFLG(I) = 0
100 CONTINUE
IF (FLOW(EDGE).EQ.0) THEN
   G1 = -1
   G2= 1
ELSE
   G1 = 1
   G2 = -1
END IF
BIGD = CAP(EDGE)
CNODE = FROM(EDGE)
110 NFLG(CNODE) = 1
IF (CNODE.NE.ROOT) THEN
   CNODE = PRED(CNODE)
   GOTO 110
END IF
CNODE = TO(EDGE)
200 IF (NFLG(CNODE).EQ.1) THEN
   W = CNODE
   GOTO 300
END IF
NFLG(CNODE)=1
IF ((M(CNODE)*G2).GT.0) GOTO 210
IF (((CAP(ABS(M(CNODE)))-TFL(CNODE)).GT.BIGD) THEN
   MARC = ABS(M(CNODE))
   IF (((FLOW(MARC).EQ.0).AND.(NEWT(MARC).EQ.0)) THEN
      BIGD = 0
      MU = 2
      BIGK = CNODE
   END IF
CNODE=PRED(CNODE)
GOTO 200
ELSE
   BIGD = CAP(ABS(M(CNODE)))-TFL(CNODE)
   IF (NEWT(ABS(M(CNODE))).EQ.0) THEN
      MU = 2
      BIGK = CNODE
   END IF
   CNODE = PRED(CNODE)
   GOTO 200
END IF
210 IF (TFL(CNODE).GT.BIGD) THEN
   MARC = ABS(M(CNODE))
   IF ((FLOW(MARC).EQ.CAP(MARC)).AND.(NEWT(MARC).EQ.0)) THEN
      BIGD = 0
      MU = 2
      BIGK = CNODE
   END IF
   CNODE = PRED(CNODE)
   GOTO 200
ELSE
   BIGD = TFL(CNODE)
   IF (NEWT(ABS(M(CNODE))).EQ.0) THEN
      MU = 2
      BIGK = CNODE
   END IF
   CNODE = PRED(CNODE)
   GOTO 200
END IF
300 CNODE = FROM(EDGE)
310 IF (CNODE.EQ.W) GOTO 400
   IF ((M(CNODE)*G1).GT.0) GOTO 320
   IF ((CAP(ABS(M(CNODE)))-TFL(CNODE)).GT.BIGD) THEN
      MARC = ABS(M(CNODE))
      IF ((FLOW(MARC).EQ.0).AND.(NEWT(MARC).EQ.0)) THEN
         BIGD = 0
         MU = 1
         BIGK = CNODE
      END IF
      CNODE = PRED(CNODE)
      GOTO 310
   ELSE
      BIGD = CAP(ABS(M(CNODE)))-TFL(CNODE)
      IF (NEWT(ABS(M(CNODE))).EQ.0) THEN
         MU = 1
         BIGK = CNODE
      END IF
      CNODE = PRED(CNODE)
      GOTO 310
   END IF
320 IF (TFL(CNODE).GT.BIGD) THEN
   MARC = ABS(M(CNODE))
   IF ((FLOW(MARC).EQ.CAP(MARC)).AND.(NEWT(MARC).EQ.0)) THEN
      BIGD = 0
      MU = 1
      BIGK = CNODE
   END IF
   CNODE = PRED(CNODE)
   GOTO 310
ELSE
BIGD = TFL(CNODE)
IF (NEWT(ABS(M(CNODE))).EQ.0) THEN
  MU = 1
  BIGK = CNODE
END IF
CNODE=PRED(CNODE)
GOTO 310
END IF
400 IF (BIGK.GT.0) THEN
  OUT=M(BIGK)
ELSE
  OUT = 0
END IF
IF (OUT.NE.0) THEN
  OOO = ABS(OUT)
  IF (((FLOW(OOO).EQ.0).AND.(BIGD.NE.CAP(OOO))) THEN
    BIGD=0
  END IF
  IF (((FLOW(OOO).EQ.CAP(OOO)).AND.(BIGD.NE.CAP(OOO))) THEN
    BIGD=0
  END IF
END IF
RETURN
END

SUBROUTINE UPDATE2(EDGE,G1,G2,BIGD,NFLG,W,MU,BIGK,OUT)
C UPDATES FOR NEW BASIS
C
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER PRED(1000),THD(1000),TFL(1000),DUAL(1000),M(1000)
INTEGER ARCNUM(1000),FROM(1000),TO(1000),CAP(1000)
INTEGER COSTS(1000,10)
INTEGER ROOT,FLOW(1000),TFLG(1000)
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /NET4/ PRED,THD,TFL,DUAL,M
COMMON /NET5/ ROOT,FLOW,TFLG
COMMON /NET6/ DUALS,PRICES
COMMON /LAB1/ IQFI,IIFI,IWFI,ISFI,ICFI,INFI,IZFI
INTEGER PRICES(1000,10),DUALS(1000,10)
INTEGER NODES,ARCS,OBJS
INTEGER EDGE,G1,G2,BIGD,W,MU,BIGK,OUT
INTEGER NFLG(1000),UCOST(1000),DEL
INTEGER D1,D2,K,D,G,III
INTEGER BETA,SIG(10),DPRIME,Q,QPRIME
INTEGER LAMBDA,I,J,KPRIME,OLDFLOW
INTEGER MS,Z,X,R,O
DO 50 I=1,OBJS
  SIG(I) = PRICES(EDGE,I)
50 CONTINUE
DEL=G2
TFLG(EDGE) = 1
TFLG(ABS(OUT)) = 0
O = OUT*DEL
IF (((O.GT.0).AND.(MU.EQ.1)) .OR. ((O.LT.0).AND.(MU.EQ.2))) THEN
  FLOW(ABS(OUT)) = FLOW(ABS(OUT))+BIGD
ELSE
  FLOW(ABS(OUT)) = FLOW(ABS(OUT))-BIGD
END IF
IF (FLOW(EDGE).EQ.CAP(EDGE)) THEN
  DPRIME = CAP(EDGE)-BIGD
ELSE
  DPRIME = BIGD
END IF
IF (MU.EQ.1) THEN
  Q = FROM(EDGE)
  QPRIME = TO(EDGE)
  LAMBDA = -EDGE
  DO 65 II = 1,OBJS
    SIG(II) = -SIG(II)
  CONTINUE
G = G1*BIGD
ELSE
  Q = TO(EDGE)
  QPRIME = FROM(EDGE)
  LAMBDA = EDGE
  G = G2*BIGD
END IF
I = Q
J = PRED(Q)
KPRIME = PRED(BIGK)
410 OLDFLOW = TFL(I)
DO 465 III=1,OBJS
  DUALS(I,III)=DUALS(I,III)+SIG(III)
465 CONTINUE
TFL(I) = DPRIME
IF (M(I).GT.0) THEN
  BETA = 1
ELSE
  BETA = -1
END IF
MS = ABS(M(I))
M(I) = LAMBDA
FLOW(ABS(M(I))) = TFL(I)
NFLG(I) = 0
Z = I
X = THD(I)
520 IF ((X.EQ.ROOT).OR.(NFLG(PRED(X)).EQ.1)) GOTO 610
DO 565 III=1,OBJS
  DUALS(X,III)=DUALS(X,III)+SIG(III)
565 CONTINUE
Z = X
X = THD(X)
GOTO 520
610 R = J
620 IF (THD(R).EQ.I) GOTO 710
R = THD(R)
GOTO 620
710 IF (I.EQ.BIGK) GOTO 790
DPRIME = OLDFLOW-G*BETA
LAMBDA = -BETA*MS
THD(R)=X
THD(Z)=J
R=I
I=J
J=PRED(J)
PRED(I)=R
GOTO 410
790 THD(R)=X
THD(Z)=THD(QPRIME)
THD(QPRIME)=Q
PRED(Q)=QPRIME
IF (MU.EQ.1) THEN
  D1 = KPRIME
  D2 = QPRIME
  K = 1
ELSE
  D1 = QPRIME
  D2 = KPRIME
  K = 1
END IF
D = D1
G = G1*BIGD
830 IF (D.EQ.W) GOTO 880
IF (M(D).GT.0) THEN
  BETA = 1
ELSE
  BETA = -1
END IF
TFL(D) = TFL(D)-G*BETA
FLOW(ABS(M(D)))=TFL(D)
NFLG(D)=0
D = PRED(D)
GOTO 830
880 IF (K.NE.1) GOTO 910
  K=2
D = D2
G = G2*BIGD
GOTO 830
910 I=W
920 NFLG(I) = 0
IF (I.EQ.ROOT) GOTO 970
I = PRED(I)
GOTO 920
970 RETURN
END
C
SUBROUTINE UNCODING(A,CD,FLS)
C DETERMINES CODE TO BE STORED
C IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER A
INTEGER CD(30),FLS(1000)
INTEGER I,K,R,L
DO 100 I = 1,A
  FLS(I) = 0
100 CONTINUE
DO 200 I = 1,30
  IF (CD(I).NE.0) THEN
    K = CD(I)
    K = INT(K/2)
    DO 150 J = 1,30
      L = (I-1)*30+J
      IF (L.GT.A) GOTO 250
      R = MOD(K,2)
      IF (R.EQ.1) THEN
        FLS(L) = 1
      END IF
      K = INT(K/2)
    150 CONTINUE
  END IF
200 CONTINUE
SUBROUTINE UPDATE3(EDGE, DEL, G1, G2)
C CALLED TO ADJUST FLOWS WHICH SHOULD BE AT 0 OR AT CAPACITY AND C ARE NOT
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER PRED(1000), THD(1000), TFL(1000), DUAL(1000), M(1000)
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER COSTS(1000,10)
INTEGER ROOT, FLOW(1000), TFLG(1000)
INTEGER NODES, ARCS, OBJS
COMMON /NET1/ NODES, ARCS, OBJS
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /NET4/ PRED, THD, TFL, DUAL, M
COMMON /NET5/ ROOT, FLOW, TFLG
INTEGER EDGE, G1, G2, BIGD, W, MU, OUT
INTEGER NFLG(1000), UCOST(1000), DEL
INTEGER D1, D2, K, D, G
INTEGER BETA, SIG, DPRIME, Q, QPRIME
INTEGER LAMBDA, I, J, KPRIME, OLFLOW
INTEGER MS, Z, X, R, O, CNODE
DO 10 I = 1, NODES
   NFLG(I) = 0
10 CONTINUE
CNODE = FROM(EDGE)
110 NFLG(CNODE) = 1
IF (CNODE.NE.ROOT) THEN
   CNODE = PRED(CNODE)
   GOTO 110
END IF
CNODE = TO(EDGE)
200 IF (NFLG(CNODE).EQ.1) THEN
   W = CNODE
   GOTO 300
END IF
NFLG(CNODE) = 1
CNODE = PRED(CNODE)
GOTO 200
300 BIGD = CAP(EDGE)
IF (DEL.LT.0) THEN
   FLOW(EDGE) = 0
ELSE
   FLOW(EDGE) = CAP(EDGE)
END IF
D1 = FROM(EDGE)
D2 = TO(EDGE)
K = 1
D = D1
G = G1*BIGD
230 IF (D.EQ.W) GOTO 290
IF (M(D).GT.0) THEN
   BETA = 1
ELSE
   BETA = -1
END IF
TFL(D) = TFL(D) - G*BETA
FLOW(ABS(M(D))) = TFL(D)
D = PRED(D)
GOTO 230
290 IF (K.NE.1) GOTO 910
    K = 2
    D = D2
    G = G2*BIGD
    GOTO 230
910 RETURN
END

C SUBROUTINE INARC(F,E,D)
C DETERMINES FLOW STATUS OF INCOMING EDGE
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES,ARCS,OBJS
INTEGER ARCNUM(1000),FROM(1000),TO(1000),CAP(1000)
INTEGER ROOT,FLOW(1000),TFLG(1000)
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /NET3/ ARCNUM,FROM,TO,CAP,COSTS
COMMON /NET5/ ROOT,FLOW,TFLG
INTEGER F(1000),E,D
INTEGER I
E = 0
D = 0
DO 100 I = 1,ARCS
  IF (F(I).NE.TFLG(I)) THEN
    IF (F(I).EQ.1) THEN
      E = I
      IF (FLOW(I).EQ.0) THEN
        D = 1
      ELSE
        IF (FLOW(I).EQ.CAP(I)) THEN
          D = -1
        END IF
      END IF
    END IF
  END IF
100 CONTINUE
RETURN
END

C SUBROUTINE CAPLIST(CAPFLAG)
C CREATES AND INITIALIZES THE DATA STRUCTURE
C FOR CODING THE NONBASIC ARCS AT CAPACITY
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES,ARCS,OBJS
INTEGER CCODE(30),CL(400000,30)
INTEGER CAPFLAG(1000)
COMMON /NET1/ NODES,ARCS,OBJS
COMMON /LIST3/ CCODE,CL
CALL CAPCODE(ARCS,CAPFLAG,CCODE)
DO 100 I=1,30
  CL(I) = CCODE(I)
100 CONTINUE
RETURN
END

C SUBROUTINE CRASH2(NEWT,CAPFLAG)
C CRASHES TO THE NEXT BASIS USING TRIANGULARIZATION WHEN
C CRASH ROUTINE FAILS
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER NODES,ARCS,OBJS
INTEGER NDNUM(1000),SUPPLY(1000)
INTEGER ARCNUM(1000), FROM(1000), TO(1000), CAP(1000)
INTEGER COSTS(1000, 10)
INTEGER PRED(1000), THD(1000), TFL(1000), DUAL(1000), M(1000)
INTEGER ROOT, FLOW(1000), TFLG(1000)
INTEGER BCODE(30), NEXT, BL(400000, 33), TOTBAS
INTEGER ZS(10), PRENT, ZSTAT
INTEGER CCODE(30), CL(400000, 30)
INTEGER IDPROB, INPFLG, WGTFLG, PRTFLG(7), ISEED, DEGFLG
INTEGER PRBAS, NCRASH, NCR1
REAL STIME, CTIME
COMMON /CDATA1/ PRBAS, STIME, NCRASH
COMMON /CDATA2/ NCR1, CTIME
COMMON /QPARM1/ IDPROB, INPFLG, WGTFLG, PRTFLG, ISEED
COMMON /LIST3/ CCODE, CL
COMMON /LIST2/ ZLIST, NEXTZ, TOTALZ
COMMON /NET1/ NODES, ARCS, OBJS
COMMON /NET2/ NDNUM, SUPPLY
COMMON /NET3/ ARCNUM, FROM, TO, CAP, COSTS
COMMON /NET4/ PRED, THD, TFL, DUAL, M
COMMON /NET5/ ROOT, FLOW, TFLG
COMMON /LIST1/ BCODE, NEXT, BL, TOTBAS
INTEGER PTR, PARENT, PNODE
INTEGER I, FLAG(1000), CAPFLAG(1000), RTHD(1000), B(1000)
INTEGER NEWT(1000)
NCRASH = NCRASH+1
DO 200 I=1, ARCS
  IF (NEWT(I).EQ.1) THEN
    FLOW(I) = 0
  ELSE
    IF (CAPFLAG(I).EQ.0) THEN
      FLOW(I) = 0
    ELSE
      FLOW(I) = CAP(I)
    END IF
  END IF
END IF
FLAG(I) = 0
TFL(I) = 0
200 CONTINUE
M(1) = 0
PRED(1) = 0
FLAG(1) = 1
K = 0
190 DO 300 I=1, ARCS
  TFLG(I) = NEWT(I)
  IF (NEWT(I).EQ.1) THEN
    IF ((FLAG(FROM(I)).EQ.1).AND.(FLAG(TO(I)).EQ.0)) THEN
      PRED(TO(I)) = FROM(I)
      M(TO(I)) = I
      FLAG(TO(I)) = 1
      K = K + 1
    ELSE
      IF ((FLAG(TO(I)).EQ.1).AND.(FLAG(FROM(I)).EQ.0)) THEN
        PRED(FROM(I)) = TO(I)
        M(FROM(I)) = -I
        FLAG(FROM(I)) = 1
        K = K + 1
      END IF
    END IF
  END IF
END IF
300 CONTINUE
310 IF (K.LT.NODES-1) GOTO 190
320 DO 350 I=0,NODES
   THD(I)=0
   RTHD(I)=0
350 CONTINUE
PNODE=ROOT
PTR=ROOT
DO 400 I=1,NODES-1
   DO 450 J=1,NODES
      IF (PRED(J).EQ.PNODE) THEN
         THD(J)=THD(PTR)
         RTHD(THD(PTR))=J
         THD(PTR)=J
         RTHD(J)=PTR
         PTR=J
      END IF
450 CONTINUE
PNODE=THD(PNODE)
PTR=PNODE
400 CONTINUE
DO 470 I = 1,NODES
   IF (THD(I).EQ.0) THEN
      THD(I)=ROOT
      RTHD(ROOT)=I
      GOTo 490
   END IF
470 CONTINUE
490 DO 500 I=1,NODES
   B(I)=SUPPLY(I)
500 CONTINUE
DO 510 I=1,ARCS
   IF (CAPFLAG(I).EQ.1) THEN
      B(FROM(I))=B(FROM(I))-CAP(I)
      B(TO(I))=B(TO(I))+CAP(I)
   END IF
510 CONTINUE
J=RTHD(1)
520 I=PRED(J)
   IF (M(J).GT.0) THEN
      FLOW(ABS(M(J)))=-B(J)
   ELSE
      FLOW(ABS(M(J)))=B(J)
   END IF
   B(I)=B(I)+B(J)
   J=RTHD(J)
   IF (J.NE.1) GOTO 520
DO 530 I=2,NODES
   TFL(I)=FLOW(ABS(M(I)))
530 CONTINUE
RETURN
END

C
SUBROUTINE CAPCODE(A,FLGS,CDWORD)
IMPLICIT REAL*8 (A-H,O-Z), INTEGER*4 (I-N)
INTEGER I,J,K
INTEGER A,FLGS(1000), CDWORD(30)
DO 100 I=1,30
   CDWORD(I)=0
100 CONTINUE
DO 200 I=1,30
   CDWORD(I)=0
200 CONTINUE
DO 210 J=1,30
   K=(I-1)*30+J
   IF (K.GT.A) GOTO 220
   IF (FLGS(K).EQ.1) THEN
      CDWORD(I)=CDWORD(I)+2**J
   END IF
210 CONTINUE
200 CONTINUE
220 RETURN
END

C
SUBROUTINE XTERNL
IMPLICIT REAL*8 (A-H,O-Z),INTEGER*4 (I-N)
COMMON /LAB1/ IQFI,IIFI,IWFI,ISFI,ICFI,INFI,IZFI
IQFI = 25
IIFI = 26
IWFI = 27
ISFI = 30
INFI = 31
ICFI = 32
IZFI = 33
C OPEN(IQFI,FILE='L12YZ.QFI')
C OPEN(IIFI,FILE='L12YZ.IFI')
C OPEN(IWFI,FILE='L12YZ.WFI')
C OPEN(ISFI,FILE='L12YZ.SFI')
C OPEN(INFI,FILE='L12YZ.NFI')
C OPEN(IZFI,FILE='L12YZ.ZFI')
RETURN
END