AN INTERACTIVE MULTICRITERIA APPROACH TO FACILITY LOCATION-ALLOCATION MODELS UNDER STOCHASTIC DEMAND

By

MORTEZA ABTAHI

Bachelor of Science
Oklahoma State University
Stillwater, Oklahoma
1981

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1983

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
DOCTOR OF PHILOSOPHY
July, 1989

AN INTERACTIVE MULTICRITERIA APPROACH TO
FACILITY LOCATION-ALLOCATION MODELS
UNDER STOCHASTIC DEMAND

Thesis Approved:

M. P. Lenell
Thesis Adviser

M. H Branson

A. C. Schuerman

Joe H. Mize

Norman N. Durham
Dean of the Graduate College

PREFACE

This research focuses on the development of suitable models to support the strategic planning of facilities location-allocation in the presence of multiple conflicting objectives and stochastic demands.

Two mathematical models based on chance-constrained and stochastic programming are developed. Both models implement zero-one integer goal programming methodology for the analysis of multiple objectives. A solution algorithm based on the chance-constrained goal programming is proposed for the former model. And a two stage algorithm is suggested for dealing with the nonlinear structure of the stochastic programming model. Two types of demand distributions, normal and uniform are considered. An integrated interactive computer program is designed and implemented to experiment with the proposed models on microcomputers.

I wish to express my sincere gratitude to my major adviser and chairman of my Ph.D. Committee, Dr. M. Palmer Terrell, for his guidance and encouragement during this study and throughout my graduate program. I also wish to express my thanks and appreciation to my committee members, Dr. Michael H. Branson, Dr. Joe H. Mize, Dr. Allen Schuermann, and Dr. William D. Warde for their assistance and suggestions.

A note of appreciation is extended to Dr. Donald W. Grace and Dr. Gary R. Stevens who initially served on my committee before leaving Oklahoma State University.

I am greatly thankful to the School of Industrial Engineering and Management at Oklahoma State University for the financial assistance throughout my graduate study.

A note of thanks goes to Camille Deyong for her friendship.

To my sisters, Fahimeh and Felor, whom I have not seen for eleven years, I feel guilty for living in comfort while they had to experience war.

I am forever grateful to my wife, Marzieh, for her love, patience, and sacrifices. I am very fortunate to share my life with her.

Finally, I wish to dedicate this dissertation to my parents Nahid and Hossein Abtahi, who have devoted their life to their children. I missed them very much during my studying years.

TABLE OF CONTENTS

Chapter		Page
I.	INTRODUCTION	1
	The General Problem	1
	Background	1
	Location Models	3
	Location Problems on a Plane	4
	Location Problems on a Network	5
	Distance Metric	6
	Location-Allocation Models	7
	Definition	7
	Costs in LAP	8
	LAP Classification	9
	Multiple Objectives in LAP Models	11
	Stochastic Demand in LAP Models	14
	Research Objectives	15
	Primary Objectives	16
	Secondary Objectives	16
	Research Plan	16
	Phase 1 - Investigation and System Design	17
	Phase 2 - Program Development	18
	Phase 3 - System Validation and Sensitivity Analysis	20
	Summary	21
II.	LITERATURE REVIEW	22
	Location Problems on a Plane	22
	The Location-Allocation Problem	25
	The Basic Problem	25
	Solution Techniques for the LAPs	27
	Heuristic Procedures	28
	Exact Procedures	33
	Simulation Techniques	49
	Multi-Objective LAPs	50
	Conclusion	52
III.	MULTIPLE OBJECTIVE DECISION MAKING	56
	Introduction	56
	An Overview of MCDM Methods	57
	MCDM Classification	59

Chapter Page

Goal Programming Methods..... 63
GP Computational Algorithms..... 67
Integer GP Techniques..... 69
Interactive GP and Sensitivity Analysis. 70
Summary and Conclusions..... 72

IV. MODEL DEVELOPMENT AND SOLUTION METHODOLOGY..... 73

Introduction..... 73
Model Assumptions..... 74
Notations..... 76
Chance-Constrained Programming..... 79
Normally Distributed Demands..... 81
Uniformly Distributed Demands..... 82
Stochastic Programming Model..... 83
Normally Distributed Demands..... 86
Uniformly Distributed Demands..... 89
Mathematical Formulations..... 90
Model A - Chance-Constrained Goal Programming Formulation..... 91
Model B - Stochastic Goal Programming Formulation..... 98
Solution Algorithms..... 101
Model A Solution Procedure..... 101
Model B Solution Procedure..... 104
Summary..... 111

V. VALIDATION, COMPUTATIONAL EXPERIENCE, AND SENSITIVITY ANALYSIS..... 112

Introduction..... 112
Validating the Algorithms and Computer Programs..... 112
Test Problem 1 - A Multicriteria Warehouse Location Model..... 113
Test Problem 2 - Location-Allocation Model I..... 115
Test Problem 3 - Location-Allocation Model II..... 117
An Illustrative Example for Model A..... 121
System Description..... 121
System Formulation..... 124
An Illustrative Example for Model B..... 128
Sensitivity Analysis..... 134
Type 1 Sensitivity Analysis..... 135
Type 2 Sensitivity Analysis..... 137
Type 3 Sensitivity Analysis..... 140
Computational Difficulties..... 145
Summary..... 147

Chapter Page

VI. INTERACTIVE COMPUTER PROGRAM..... 148

Introduction..... 148
General Structure of the Program..... 148
Data Base Management Module..... 150
Solution Algorithms Module..... 153
Sensitivity Analysis Module..... 157
Summary..... 160

VII. SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS..... 161

Summary..... 161
Conclusions..... 162
Recommendations for Future Research..... 163

BIBLIOGRAPHY..... 166

APPENDIXES..... 177

APPENDIX A - TEST MODELS DATA AND RESULTS..... 178

APPENDIX B - APPROXIMATION TO THE CDF AND INVERSE CDF OF STANDARD NORMAL DISTRIBUTION..... 185

APPENDIX C - PASCAL PROGRAM SOURCE CODES..... 188

LIST OF TABLES

Table		Page
1.1.	Location-Allocation Problem Classification.....	9
2.1.	Summary of Single Objective LAP Procedures.....	54
2.2.	Summary of Multiple Objective LAP Procedures.....	55
5.1.	Comparison of Results for Test Problem 1.....	114
5.2.	Test Problem 1 - Algorithm Performance.....	115
5.3.	Comparison of Results for Test Problem 2.....	116
5.4.	Test Problem 2 - Algorithm Performance.....	117
5.5.	Comparison of Results for Test Problem 3.....	119
5.6.	Test Problem 3 - Algorithm Performance.....	120
5.7.	Allocation Costs, Stochastic Demands, Fixed Costs, and Capacities for the Example Problem.....	123
5.8.	Summary of the Results of the Model A Example Problem for Normal and Uniform Distribution of Demands.....	126
5.9.	Per Unit Oversupplying and Undersupplying Costs of Demand Centers for the Example Problem.....	129
5.10.	Summary of the Results of the Model B Example Problem for Normal and Uniform Distribution of Demands.....	134
5.11.	Type 1 Sensitivity Analysis of the Model A Example Problem for Normal Distribution of Demands.....	136
5.12.	Type 2 Sensitivity Analysis of the Model A Example Problem for Normal Distribution of Demands.....	138
5.13.	Trade-off Analysis of the Model A Example Problem for Normal Distribution of Demands.....	139

Table Page

5.14. Solution of the Model A Example Problem for Normal Distribution of Demands and Modified Budget Goal..... 140

5.15. The Design and Partial Solutions of the Test Problems for Type 3 Sensitivity Analysis..... 142

A.1. Test Problem 1 Input Data..... 179

A.2. Test Problem 2 Input Data..... 180

A.3. Test Problem 3 Input Data..... 182

C.1. Index to Program Units and Procedures..... 189

LIST OF FIGURES

Figure		Page
1.1.	Facilities Planning Hierarchy.....	2
1.2.	A Typical Cost Trade-off Curve Between Transportation and Fixed Costs.....	12
1.3.	Interactive System Components and Flow.....	19
4.1.	Flowchart of the Algorithm for the SMOLAP of Model A.....	103
4.2.	Flowchart of the Stage 1 Algorithm for the SMOLAP of Model B.....	109
5.1.	Graphical Representation of Potential Plant Sites and Existing Demand Centers.....	122
5.2.	Expected Cost of Penalties at Each Destination for the Normally Distributed Demands.....	130
5.3.	Expected Cost of Penalties at Each Destination for the Uniformly Distributed Demands.....	132
6.1.	General Structure of the Computer Program.....	149
6.2.	Display of the Main Menu.....	149
6.3.	Structure of the Data Base Management Module.....	151
6.4.	Display of the Deterministic Input Data Screen....	151
6.5.	Structure of the Solution Algorithms Module.....	153
6.6.	Sample Output Screen for Continuous Solution.....	155
6.7.	Structure of the Sensitivity Analysis Module.....	158
6.8.	Display of the Sensitivity Analysis Menu.....	159

CHAPTER I

INTRODUCTION

The General Problem

Background

The strategic issue of facility location in a given system has been and continues to be of significant interest to practitioners and researchers alike. The research interest in this area stems from both its potential economic return and applicability to problems in many diverse fields. Historically, Alfred Weber pioneered the analytical approach to location theory in the early 1900's. He considered the problem of locating an industry between two resources and a single market to minimize the transportation cost. In general, facility location problems are concerned with the selection of sites for new facilities in relation to some existing demand centers to optimize some measure of effectiveness.

In general, the problem of facilities location is a part of facilities planning. Figure 1.1 illustrates the hierarchy of facilities planning [Tompkins and White (1984)]. Because of the nature of this problem and its breadth of application, an interdisciplinary interest has been developed in this

area. In particular, the problem has been studied by technical geographers, urban planners, operation researchers, regional scientists, engineers, architects, economists, logisticians, management scientists, applied mathematicians and system analysts, [White and Case (1974)]. The facility location problems occur in many settings both in private and public sectors of the economy, [ReVelle, Marks *at al.* (1970)].

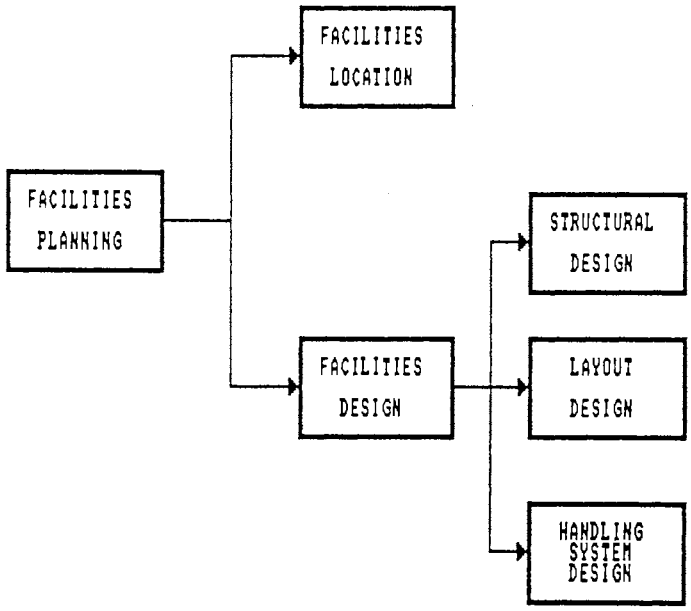


Figure 1.1. Facilities Planning Hierarchy
[Tompkins & White (1984)]

Finally, among the examples of facility location problems are the determination and location of warehouses,

distribution centers, production plants, machine tools, waste-disposal facilities, hospitals, fire stations, computers, missile batteries, and communication centers. In addition, Sule (1981) presented an application of facility location-allocation problems to production planning and fleet management problems. Cornuejols *et al.* (1977) further extended the application of this problem into financial planning.

Location Models

Despite the large number of approaches to the site selection problem, it is possible to distinguish between two basic structural categories [Scott (1970), ReVelle, Marks *et al.* (1970)];

1. Location on a plane.
2. Location on a network.

In addition, based upon criteria and constraints used in formulating locational problems, ReVelle, Marks *et al.* (1970) have also distinguished between private and public sector location models. In short, private sector models emphasize quantitative measures such as minimization of cost or maximization of profit while public sector models are concerned with qualitative factors which are not usually measurable in monetary terms. In general, the structure, criteria, and constraints of a given problem will determine the appropriate methodology to employ.

Location Problems on a Plane

Location on a plane, also referred to as the infinite set method, considers that a site may be selected anywhere on the coordinate plane. Therefore, an infinite number of potential locations are available for selection. Eilon *et al.* (1971) has identified the main features of this approach as follows:

a. Locations which are selected are not required to be a priori attractive.

b. Alternative selections are available in multi-facility location problems.

c. The solution obtained may involve non-feasible locations.

d. Transport costs are a monotonic function of distance.

These models are based on a single objective and explicitly incorporate a distance metric, l_p, into their formulation. As item (d) indicates they also assume transportation costs to be proportional to the distance travelled. They seek to minimize the total cost by minimizing the total sum of distances travelled between source(s) and destination(s). As Lee and Franz (1979) suggest, these models, the location of facilities as points on the plane, do not treat many of today's realities and even may not be feasible. For instance, the location(s) indicated may be in conflict with many corporate policies or legislated regulations or may be geographically infeasible. Also, as indicated by Geoffrion (1975), treating transportation cost

as an explicit well-behaved function of distance (no look-ups) does not represent a realistic cost structure for the transportation flows.

Location Problems on a Network

This class of problems is characterized by a solution space which consists of points on a network. The network of interest may be a road network, a rail network, an air transport network, a river network, or a network of shipping lanes. These models enumerate previously determined alternative facility locations (as contrasted to location on a plane) and sites of demands as nodes on a network, Lee and Franz (1979). Network problems can be further classified into two categories; points only on the nodes of the network and points on the nodes and/or the arcs joining the nodes. Eilon *et al.* (1971) has identified the following main features of the network problems:

a. They incorporate costs which are related to specific geographical locations.

b. Transportation costs are not required to be any single specific function of distance.

c. They require a set of sites which are known to be feasible and for which all cost data are available.

d. The number of locations must be finite and sufficiently small for computational efficiency.

Plant location-allocation problems are typical of this category since in practice plant locations are usually selected from a set of predetermined sites.

Distance Metric

The criterion used often for evaluation of locational problems is minimization of some distance measure. Such distance measures in relation to locations on planes and locations on networks will be discussed next.

In the case of location on planes distances between facilities are measured in various functional forms called norms. In general, the distance between points q and s using the l_p metric is represented as follows:

$$l_p\ (q,s) = \|q-s\|_P = \left[\sum_{i=1}^{n} |q_i - s_i|^P \right]^{1/P} \qquad (1.1)$$

where n is the dimension of the solution space. The two most common distance measures in locational analysis are rectilinear and Euclidean distances.

When p=1 the distance is called rectilinear, rectangular, Manhattan, metropolitan, or l_1 metric. The rectilinear distance in two dimensional space is as follows:

$$l_1(q,s) = (|x_1 - x_2| + |y_1 - y_2|) . \qquad (1.2)$$

Rectilinear distances are typically used to measure travel distances between points via rectilinear aisles or street networks.

When p=2 the distance is called Euclidean, radial, straight-line or l_2 metric. An example of Euclidean distance in two dimensional space is given below:

$$l_2(q,s) = \left[(x_1 - x_2)^2 + (y_1 - y_2)^2\right]^{1/2}. \qquad (1.3)$$

Euclidean distances are used whenever travel between points (sites) occur along a straight line, such as air or conveyor travel. In cases where cost is not a linear function of distance traveled (eg. emergency cases), squared-Euclidean distances are frequently used.

When $0<p<1$ the l_p metric is called hyper-rectangular distance. Generally, such distances occur whenever travel distances exceed rectilinear.

In the case of location on networks, distances are determined as the length (time) of the shortest path between the nodes. As the result, the expression for the distance may not appear explicitly in the formulation of network problems.

Location-Allocation Models

Definition

The location-allocation problem (LAP) was first introduced by Leon Cooper (1963). Since then, many modifications to the problem parameters have been made, and a variety of techniques have been proposed for its solution. The location-allocation problem may be generically stated as follows: Given the location or distribution of a set of customers/destinations and their associated demands, simultaneously determine the number and location of

supplies/sources and the allocation of their products or services to customers/destinations to optimize some measure of effectiveness.

The area of facility location-allocation determination covers a wide range of problems. Among others, applications occur frequently in service systems, manufacturing systems, and distribution systems. Although suppliers or sources may refer to a variety of facilities and machines, the intent of this research is specific to plant location-allocation problems. Also, as mentioned previously, in practice, the selection of plant locations is usually from a set of pre-specified sites. As such, locations on networks is the most appropriate structure to be used for modeling of these problems. Throughout this research the term facilities will be used generically to refer to plants, warehouses, or distribution centers.

Costs in LAP

There are two important cost elements in the LAPs:

1. Transportation costs between plants and customers;
2. Production costs at each plant location;
    a. Fixed costs of construction and operations;
    b. Unit production costs.

Transportation and unit production costs are usually assumed to be a linear function of the quantities distributed and produced, respectively. And, construction/operation

costs are usually assumed constant, representing annual fixed charges. However, some researchers have considered unit production costs and/or construction and operation costs to reflect economies-of-scale. Because, more realistically, the marginal cost of supplying a customer usually decreases as facility throughput (capacity) increases. This results in a concave cost function which often is approximated with a continuous, piece-wise linear, and concave function.

LAP Classification

Plant location-allocation problems may be classified according to several characteristics. The major factors considered in the literature are shown in Table 1.1:

TABLE 1.1

LOCATION-ALLOCATION PROBLEM CLASSIFICATION

Item	Factor	Factor Levels
A	Objective	1. Single objective 2. Multiple objectives
B	Solution Space	1. Discrete (finite set) 2. Continuous (infinite set)
C	Nature of Demand	1. Deterministic 2. Stochastic
D	Types of Plants	1. Uncapacitated 2. Capacitated
E	Hierarchy	1. Zero echelon (transportation) 2. Single echelon (transshipment) 3. Multiple echelon (transship.)

TABLE 1.1 (continued)

Item	Factor	Factor Levels
F	Planning Horizon	1. Static (single period) 2. Dynamic (multiple periods)
G	Product	1. Single Product 2. Multiple Products
H	Costs (Transportation, Production, Fixed plant cost)	1. Fixed 2. Linear 3. Nonlinear
I	Elasticity of demand	1. Demand is price/distance insensitive 2. Demand is price/distance sensitive
J	Solution Procedure	1. Heuristics 2. Optimizers 3. Simulators
K	Other (problem-dependent) constraints	1. Single sourcing 2. Mutually exclusive plants 3. Etc.

The complexity of a model varies with the selection of different characteristics from Table 1.1. For example, under this system, problem A2, B1, C2, D2, E3, F2, G2, H3, I2, J2 is substantially complex while problem A1, B2, C1, D1, E1, F1, G1, H1, I1, J2 is relatively simple. However in practice, whenever modeling a system, it is desirable to achieve a compromise between simplicity and reality. As such, typically, based upon the availability of data and the real problem encountered, a particular combination of the

above characteristics will be selected for modeling and analysis.

Multiple Objectives in LAP Models

Location-allocation analysis like most other strategic decision making problems is multi-objective in nature. The multiple objective aspect of LAPs have gained considerable attention from researchers in recent years. Traditionally, the objective function for location-allocation models has been based upon monetary criteria: minimization of total costs or maximization of profit. Profit maximization models incorporate revenues generated from sales into the formulation and are generally used whenever demand is not constant or when it can be influenced by other decision variables. On the other hand, the basic cost minimization models minimize transportation costs or a combination of fixed costs and transportation costs. In the latter case, as the number of facilities increase, fixed costs increase while the shipping costs decrease. On the contrary, as the number of facilities decrease, shipping costs increase while the fixed costs of establishing and operating facilities decrease. Thus the problem becomes a search for the optimal trade-off between the cost of building and operating facilities and the cost of transportation. A typical cost trade-off curve representing this trade-off is depicted in Figure 1.2.

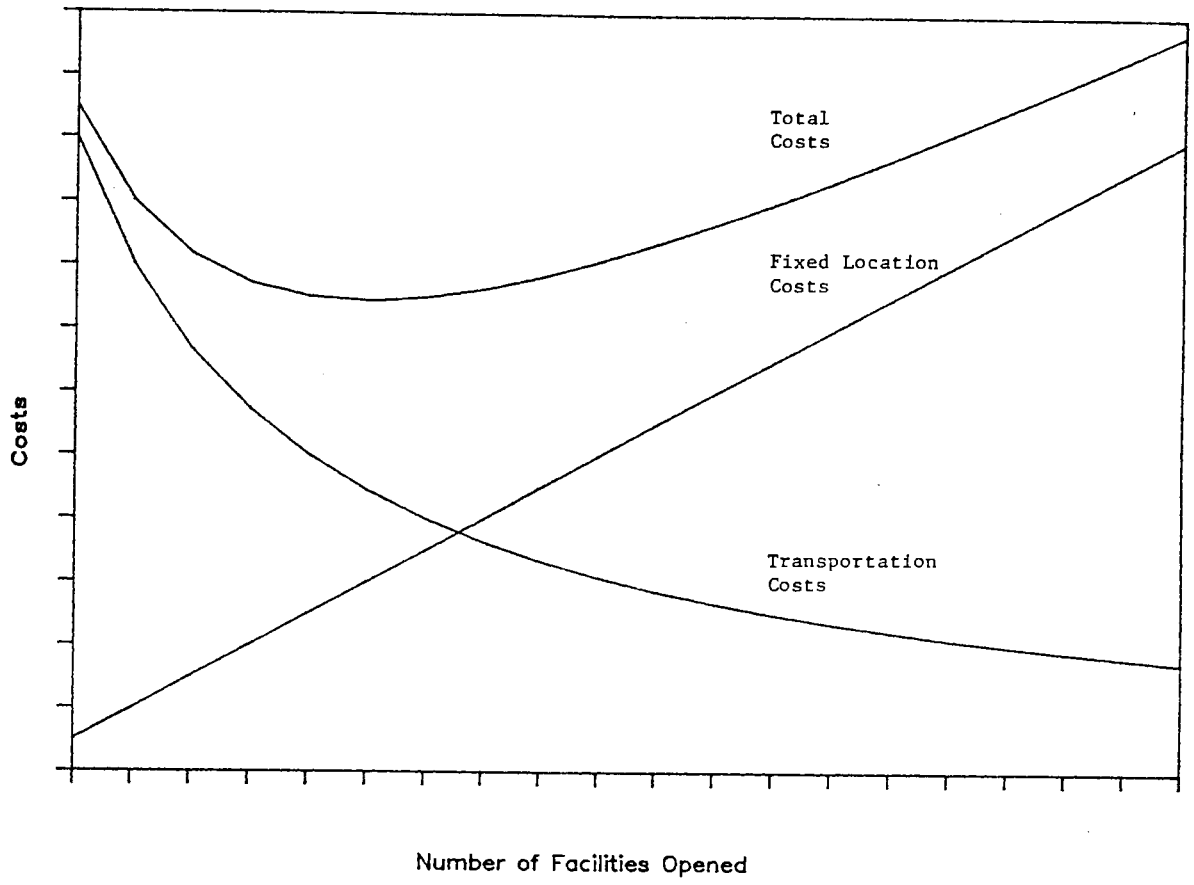


Figure 1.2. A Typical Cost Trade-off Curve Between Transportation and Fixed Costs

Nevertheless, LAPs are complex and like most other real world problems depend upon a number of tangible and intangible factors which are unique to each problem. According to Lee *et al.* (1981), although cost trade-off remains an essential consideration, the trend of the 1970s and the outcome of the future would include social, psychological, safety and public oriented non-economic considerations in the facility location determination.

Furthermore, location-allocation decisions involve a substantial capital investment and result in long-term constraints on production and distribution of products. In view of these issues, and the significant benefits derived from implementing a realistic model, it is appropriate to study LAPs in their natural environment of multiple and conflicting objectives. The need for multiple criteria models are further emphasized by a survey of industrial development activities, Lynch (1973). On the bases of this survey the top ten factors important in locating new facilities are as followings:

1. environmental considerations.
2. labor factors, emphasis on quality and supply.
3. availability of utilities.
4. transportation, primarily highways.
5. social factors, emphasis on trend to rural areas and suburbs.
6. community attitude toward industry.
7. low cost financing.
8. supply and cost of available land.
9. markets.
10. taxes.

In addition, ReVelle, Marks *et al.* (1970) point out that concentrating only on economic terms produces solutions which are non-optimal with respect to governmental rules and regulations. Fulton (1971) and Student (1976) have also emphasized the growing significance of environmental and

social factors in the facility location decisions.

In real life, it is evident that other criteria beside costs play a significant role in determination of locations. Therefore, clearly, single objective, pure cost minimization models are no longer adequate to represent locational problems in the presence of social, energy, and environmental considerations. As such, a multiple criteria approach is the most appropriate strategy to be used in modeling and analyzing the location-allocation problems.

Stochastic Demand in LAP Models

Often, in real-life situations demands are not known with certainty and only estimates are available. Whenever demand at destinations is not known with certainty, it should be treated as a random variable. Among the sources of variations in demand are changes in market share (competition), population movements, fluctuating costs, and seasonal demand patterns. In general, in view of these uncertainties, it is advantageous to incorporate the assumption of stochastic demand into LAPs models. This results in more realistic and comprehensive models and increases validity and credibility of the solutions obtained.

From an economic perspective, inclusion of stochastic demand is justified since in the presence of market uncertainties it is likely to oversupply or undersupply the demand centers which in turn would result in inventory "carry

overs" or "stock outs" costs. ReVelle, Marks *et al.* (1970) have also emphasized the importance of considering the stochastic nature of the demand and supply with respect to seasonal or periodic fluctuations, as well as changes in economic conditions and population patterns for the facility location problems.

Therefore, it is believed that introduction of stochastic demand into multi-criteria LAPs will provide a greater element of reality into the formulation and analysis of this class of complex problems.

Research Objectives

Multiple objectives and stochastic demand are two important elements of LAPs. Although studies are conducted incorporating these factors separately, both facets have not been considered simultaneously. This study is to explore the effects of random demands explicitly in the modeling and solution of multi-criteria location-allocation problems. The objectives of this research are divided into two sets: The primary objectives and the secondary objectives. The primary objectives focus on the development of suitable models for the multiple objective location-allocation problem in the presence of stochastic demand and the determination of appropriate solution methodologies. The secondary objectives are to develop an interactive computer program based on the solution algorithms developed earlier and to conduct a sensitivity analysis by varying some appropriate parameters.

Specifically, the primary and secondary objectives are stated below:

Primary Objectives

1) Development of mathematical models for the multi-objective location-allocation problem with stochastic demands.

2) Development of appropriate solution algorithms for these models.

Secondary Objectives

1) Development of an interactive multiple objective computer program based on the algorithms developed above.

2) Testing and validating the models by relaxing the assumption of stochastic demand or multiple objectives and comparing the results with the earlier work in multi-criteria and single objective facility location-allocation problems, respectively.

3) Demonstrating the sensitivity analysis of the models by varying parameter(s) of the demand distribution and performing what-if analysis.

Research Plan

In order to accomplish the above objectives the research will be divided into three phases: 1) investigation and system design, 2) program development, 3) system validation and sensitivity analysis. A general outline of the tasks to

be performed in each phase follows:

Phase 1 - Investigation and System Design

In this stage a review of existing algorithms for single and multiple objective programming will be performed and an appropriate solution methodology will be selected specifically suitable for interactive implementation. Next in this stage, the mathematical model of the multi-criteria location-allocation problem with stochastic demand will be developed. Based on the above formulation a solution algorithm will bedetermined. The design of the model will include the following characteristics:

- o multiple objectives;
- o stochastic demand;
- o capacitated/uncapacitated plants;
- o single (aggregated, homogeneous) product;
- o static planning horizon;
- o zero echelon (no transshipment).

Potential objectives to be included are:

- o minimize total costs (fixed costs plus transportation costs);
- o minimize transportation costs;
- o maintain production capacity within prespecified limits (e.g. for compliance with pollution control standards within state regulations);
- o locate where the quality of life is satisfactory;
- o satisfy product demand goal;
- o achieve any desired configuration constraints (e.g. set upper and/or lower limits on the

number of open plants, specify minimum and/or maximum number of locations to be selected from a subset of locations, mutually exclusive or mutually dependent locations, etc.);

- o satisfy an upper limit on total fixed cost.

In addition, the following distribution of demands will be considered:

- o normal distribution;
- o uniform distribution.

Phase 2 - Program Development

Given the solution algorithm developed previously, a computer code will be written. The program will provide data management facilities and will operate in an interactive mode. The interactive routine will be designed such that the decision maker (planner) can iteratively provide information regarding various target values and preference data concerning different objectives, in order to achieve satisfactory trade-offs among various objectives. Figure 1.3 illustrates the components of the proposed interactive system. The inputs/outputs expected from the computer system are given below:

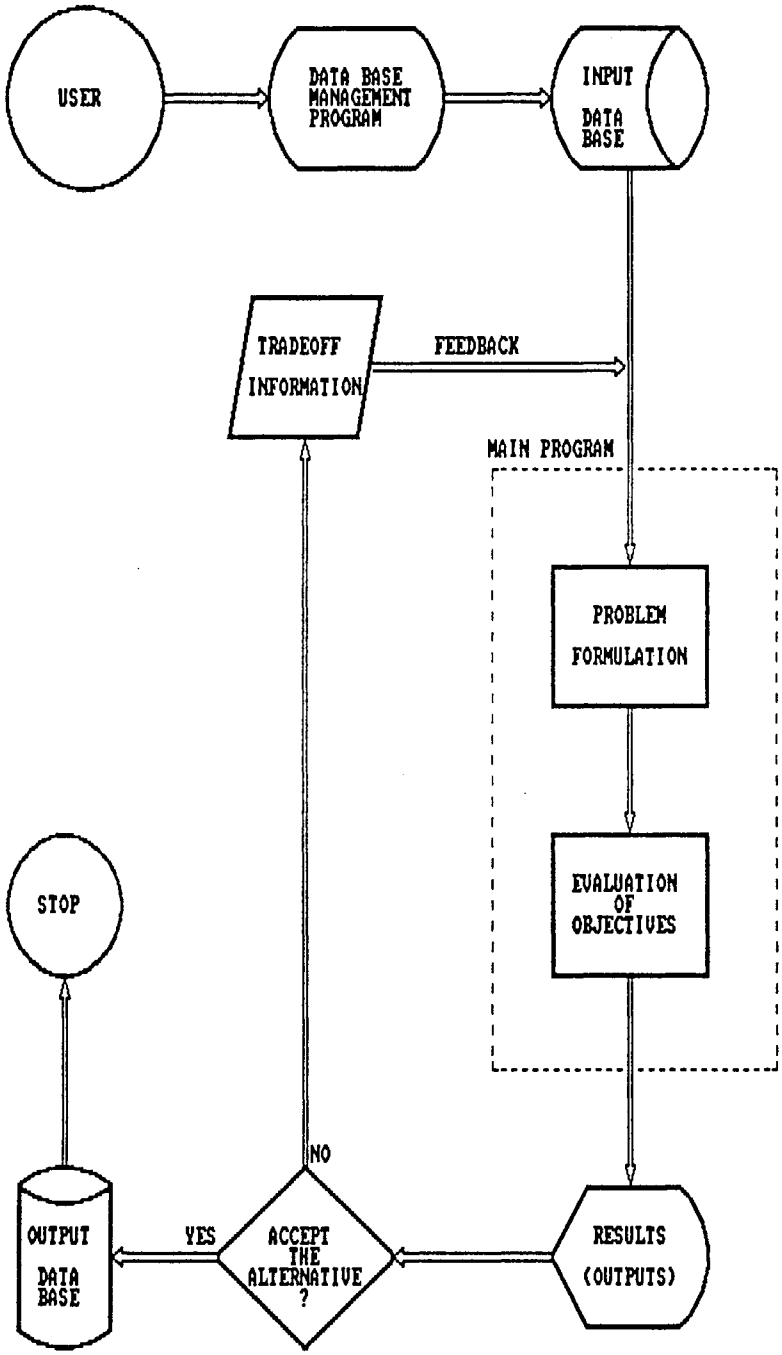


Figure 1.3. Interactive System Components and Flow

INPUTS:

- o multiple objectives;
- o demand pattern for each destination;
- o location of destinations;
- o potential sites;
- o capacity of sources (plants);
- o cost data.

OUTPUTS:

- o status of objectives;
- o number of sources;
- o location of each source;
- o size of sources at each site;
- o assignment of destinations to new sources;
- o allocation of products from sources to destinations

Phase 3 - System Validation and Sensitivity Analysis

This step consists of testing and validating the integrated system and performing sensitivity analysis on the parameter(s) of random demands. It includes debugging the program and relaxing the stochastic demand so that its results can be compared with the results available from earlier work in the multiple objectives analysis of LAPs. Furthermore, the assumption of multiple objectives will be relaxed so that the results could be compared with the results from single objective methods. The sensitivity analysis will be conducted on the distribution parameter(s)

to provide insights into the behavior of the model and its tolerance for estimation error in parameters. Also, changes in constraints and criteria and their effect on the solution will be investigated.

Summary

This study is about the facility location and product allocation problem in the presence of multiple, conflicting objectives and stochastic demand. The motivation behind this research is to formulate and analyze mathematical models which would better portray the real-life problems in the area of LAPs. Besides integrating the two important factors, multiple goals and stochastic demand, another advantage of the proposed models is their ability for sensitivity analysis. The latter will be accomplished by developing an interactive program based on the proposed solution methodologies. The interactive feature of the program will be a great asset in understanding the sensitivity of the solutions to changes in parameters, constraints, and/or criteria and hence, in helping the decision maker achieve better solutions. Finally, the application and sensitivity analysis of the proposed models will be demonstrated through some example problems.

CHAPTER II

LITERATURE REVIEW

This chapter contains a review of literature in the area of location-allocation problem (LAP). The basic LAP is to determine the location of m facilities and their allocation of a product to n existing demand centers to minimize the distribution cost. In an even more general form, LAP also involves the determination of the optimal number of new facilities. The LAP was first formulated by Cooper (1963). Since then, many researchers have contributed to the modeling and the solution methodology of this problem. Since, location-allocation problems are a class of general facility location problems, this chapter begins with a brief review of location models on a plane, followed by a review of location-allocation literature. Finally, the research in the area of multiple objective LAP is reviewed.

Location Problems on a Plane

The modern location theory has been credited to Alfred Weber, who published the book, "Uber den Standort der Industrien" (Theory of location of Industries) in 1909. He was the first to perform a quantitative analysis of a location problem. Weber examined the location on a plane of

a factory in relation to two raw material sources and a market place, with the objective of minimizing distribution cost of a single product. The mathematical formulation of the generalized Weber problem with Euclidean distances is given below:

$$\text{Minimize} \quad Z = \sum_{i=1}^{n} w_i \left[(x_i - x_o)^2 + (y_i - y_o)^2 \right]^{1/2} \qquad (2.1)$$

where

w_i = the weight assigned to point i (based on demand, population, etc.);

x_i, y_i = the coordinate of point i;

x_o, y_o = the unknown coordinate of central facility;

n = the number of existing points.

Therefore, the objective is to find a single point which minimizes the sum of weighted Euclidean distances from the given points. Kuhn and Kuenne (1962) and Cooper (1963) both have described an iterative process to solve this problem.

As discussed earlier in chapter I, an important element in the formulation of analytical models for the facility location problems on a plane is the inclusion of a distance measure. Furthermore, locational problems, based on their objectives, could be classified as follows:

1. p-median problems (minisum, maxisum);
2. p-center problems (minimax, maximin);
3. Covering problems.

In general, the median problem seeks to minimize (maximize) the average distance (time) travelled by all

customers to a facility. The p-median problem on a network consists of locating p facilities among n (>=p) locations on a network, so that the sum of shortest distances from each of the nodes of the network to its nearest facility is minimized.

Next, the center problem is concerned with minimizing (maximizing) the distance of the farthest (nearest) customer from a facility. A p-center problem on a plane is to find p new facilities on the plane that minimizes the maximum weighted Euclidean distance between each demand point and its closest new facility given n demand points on the plane and a weight associated with each point. Among examples of center problems are locating emergency or obnoxious facilities such as hospitals and waste-disposal facilities, respectively.

To motivate the covering problems in the facility location models, assume a customer is covered if a facility is within its certain distance or time. Then, the objective of covering problems is to find the number and the location of new facilities to cover all the customers at minimum cost. Some examples of covering problems are locating police stations, hospitals, radar installations, and libraries.

As is evident from the examples discussed in this section, these problems arise frequently in conjunction with public-sector location modeling. An extensive review of this class of problems is provided by Tansel *et al.* (1983a, 1983b).

The Location-Allocation Problem

The Basic Problem

The location-allocation problem (LAP) was first proposed by Cooper (1963). Originally, Cooper studied the problem in a continuous solution space. However, today, LAP in discrete solution space is used frequently when locating industrial plants or warehouses. The simplest version of the problem known as "Simple Plant Location" problem is as follows: Given a set of locations where plants (warehouses) may be built, a known demand from a given set of customers which must be satisfied, and unlimited plant capacities, determine the numbers and locations of plants to be established and the allocation of products to the customers in order to minimize total annual distribution and fixed costs. Assuming m potential plant sites and n customers, this problem may be represented by the following mixed integer programing formulation:

$$\text{Minimize} \quad Z = \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} x_{ij} + \sum_{i=1}^{m} F_i y_i \tag{2.2}$$

Subject to:

$$\sum_{i=1}^{m} x_{ij} = 1 , \qquad (j=1, \ldots,n) \tag{2.3}$$

$$y_i - x_{ij} \geq 0 , \qquad (i=1,\ldots,m \; ; \; j=1,\ldots,n) \tag{2.4}$$

$$1 \geq x_{ij} \geq 0 , \qquad (i=1,\ldots,m \; ; \; j=1,\ldots,n) \tag{2.5}$$

$$y_i = 0 \ , \ 1 \ , \qquad (i=1,\ldots,m) \tag{2.6}$$

where

x_{ij} = proportion of customer j's demand satisfied by plant i;

C_{ij} = total production and distribution costs for supplying all of customer j's demand from plant i;

y_i = 1 if plant i is established, 0 otherwise;

F_i = fixed cost of opening a facility at site i;

i = indices associated with the plants;

j = indices associated with the customers;

m = number of possible plant (warehouse) sites;

n = number of demand centers (areas).

In this formulation the objective function represents the minimization of total production, distribution, and fixed costs. Constraints in (2.3) ensure that each customer's demand is fully satisfied. Constraints in (2.4) state that assignments are made only from open facilities. And, constraints in (2.5) and (2.6) are non-negativity and integrality constraints respectively. An alternative formulation which has also been used is to define x_{ij} as the number of units supplied from plant i to demand center j and to define C_{ij} as the per unit cost of supplying customer j's demand from plant i. In this formulation constraint sets (2.3), (2.4), and (2.5) will change as follows:

$$\sum_{i=1}^{m} x_{ij} = D_j \ , \qquad (j=1,\ldots,n) \tag{2.7}$$

$$\sum_{j=1}^{n} x_{ij} - M y_i \leq 0, \quad (i=1,\ldots,m) \qquad (2.8)$$

$$x_{ij} \geq 0, \quad (i=1,\ldots,m;\ j=1,\ldots,n) \qquad (2.9)$$

where

D_j = demand for customer j;
M = some large positive number.

While the objective function and the integrality constraints are represented by Equations (2.2) and (2.6) respectively, as in the previous formulation.

Solution Techniques for the LAPs

Location-allocation models can take many forms, but based upon solution approaches, they may be classified into the following three distinct types:

1. Heuristics;
2. Optimizers (exact);
3. Simulators.

Aside from these basic approaches, based on the formulation of the problem, a variety of techniques have been utilized to solve LAPs. Among these methodologies are standard transportation/assignment, linear programming, integer and mixed integer programming, stochastic programming, decomposition, Lagrangian relaxations, and dynamic programming. Application of any specific procedure or method listed above is determined by the formulation and assumptions of a given problem.

The principal focus of this chapter is on the

mathematical formulation and solution of LAPs using heuristic and optimizer methods, and the multiple objective LAPs. As such, simulation techniques as the main analytical tool will not be reviewed extensively.

Heuristic Procedures

A heuristic algorithm involves procedures based on the "rules of thumb" (common-sense principals) and/or mathematical methods which produce "good" (acceptable) results. The solution obtained from a heuristic procedure may be optimal, but optimality is not guaranteed in general. It is worth noting that this fact could limit suitability of heuristics for exact sensitivity analysis. Heuristic procedures are used whenever size and complexity of a problem make exact optimizing algorithms impossible, or resources for finding an optimal solution, such as computer time and memory storage, are not available. Still, a heuristic algorithm is proven to be an effective method whenever it can be shown that the solution space near the optimal point is flat (shallow), that is, there are many good near optimal solutions.

The principle difficulty in solving location-allocation problems is in their combinatorial structure. To illustrate this point, first consider the location aspect of the problem. Assume m potential sites are available, then there are 2^m possible combinations (including the infeasible solution of all facilities being closed) for selecting sites. Second,

take into account all the possible allocations when there are n demand centers. Also assume that each demand center may be supplied only from one supplier (single sourcing). Then, the total number of location-allocation combinations is given by:

$$C_m^m \cdot (m)^n + C_{m-1}^m \cdot (m-1)^n + \ldots + C_1^m \cdot (1)^n \qquad (2.10)$$

Therefore, it is evident that even for moderate values of m and n (eg. 40 and 50) the possible combinations of location and allocation patterns will be significantly large. This is the combinatorial structure of LAPs which makes them candidate for heuristic solution methods.

A large portion of the developed heuristics employ the concept of largest marginal saving for the solution of location-allocation problems. According to this procedure, after starting from some arbitrary starting point, the solution is driven toward an improved point gradually via an iteration process. In each iteration, the value of one of the components of the location vector is changed. That is an open facility is set closed or vice versa. This could be compared to moving on the lattice points of a unit hypercube in one dimension. Using this approach, the choice of a component is directed by the marginal saving that could result from the change. The heuristic terminates if no further change is possible. This procedure does not guarantee optimality since the final solution depends upon the specific starting point.

Several heuristic procedures which produce good results

are proposed by; Kuehn and Hamburger (1963), Manne (1964), Feldman, Lehrer, & Ray (1966), Sa (1969), Walker (1976), Sule 1981), and Klincewicz and Luss (1986).

One of the earliest and best known heuristics for solving the simple (single commodity), uncapacitated warehouse location model is the "add" or "construction" heuristic by Kuehn and Hamburger (1963). Their heuristic program consists of two stages: First, the main program or construction stage and second, the "bump and shift" routine or improvement stage. The main program locates facilities one at a time until no additional facilities can be opened without increasing the total cost, then the second routine attempts to improve the solution obtained earlier, by evaluating the profit implications of closing or relocating open facilities. The following three heuristics are employed in the Kuehn and Hamburger algorithm:

1. Potential locations will be at or near demand concentrations.

2. Near optimum systems can be achieved by adding facilities one at a time, proceeding at each stage to add that facility which produces the greatest cost savings for the whole system.

3. At each stage, only a small subset of all possible facility locations needs to be evaluated in detail in order to determine the next facility site to open. The size of the subset depends on the size and the variance in the demands at all possible sites. The larger the variance in the market demands, the smaller the subset of possible locations.

Manne (1964) investigated the use of SAOPMA (Steepest

Ascent One Point Move Algorithm) for solving simple plant location problems. This method starts at an arbitrary lattice point of the unit hypercube and then proceeds to examine other alternative adjacent points. Alternative adjacent points are formed by adding a new plant to or dropping an existing plant from the subset under consideration. If an improvement can be realized in terms of total location and allocation costs, the new lattice point will be selected as the best solution and the search will continue from this point. Otherwise, the iterative process terminates. Moreover, at each iteration, in the absence of plant capacities, the total cost of any configuration is readily found by assigning each demand center to a plant with minimum sum of variable and fixed costs.

Feldman, Lehrer, and Ray (1966) in their heuristic procedure, considered economies of scale to be continuous and concave over the entire range of warehouse sizes and proposed a "drop" or "elimination" heuristic as opposed to the "add" heuristic by Kuehn and Hamburger (1963). The "drop" heuristic assumes all the facilities are opened initially and then drops facilities one at a time until no further savings are realized.

Sa (1969) proposed a two phase heuristic procedure for solving the capacitated facility location problem. The first phase employs a combination of "add" and "drop" heuristics to find a solution. Then the second phase performs single exchanges to improve the solution obtained in phase one.

Walker (1976) proposed a two phase heuristic procedure called SWIFT (Simplex WIth Forcing Trials). The main thrust of this algorithm is that it complements the variable selection rule of the standard simplex method with the fixed charge of entering and leaving vectors. The first phase uses the standard simplex method with modified variable selection rule to find a local optimum (this is a nonconvex program). The second phase tries to improve the solution obtained previously by exploring the extreme points non-adjacent to the current point. In phase two, forcing the solution to a new non-adjacent extreme point may initially increase the objective value, but iterating from this point could lead to an improved solution.

Sule (1981) investigated three simple heuristic procedures for solving uncapacitated facility location problems. In addition, a simple procedure to deal with multiperiod problems has also been discussed.

Klincewicz and Luss (1986) presented a Lagrangian relaxation heuristic algorithm for capacitated problems in which each customer is served by a single facility. The Lagrangian relaxation technique incorporates the capacity constraints into the objective function, leading to an uncapacitated facility location subproblem. An iterative procedure updates the Lagrangian multipliers between successive solutions of the uncapacitated subproblems. The dual ascent procedure of Erlenkotter (1978) (without branch

and bound) is used to generate feasible solutions to the uncapacitated subproblems. The algorithm is also complemented by an "add" heuristics which finds an initial upper bound and feasible solution to the problem. Finally, an adjustment heuristic is employed which attempts to improve the best feasible solution obtained from the relaxation, by adjusting the customer assignments.

Exact Procedures

Exact procedures yield an optimal solution, given there is one, in a finite number of steps. However, since LAPs are NP-complete, the computational requirements of optimal seeking procedures grow exponentially with the problem size.

The formulation of LAP is one of mixed integer programming. The integer portion of the formulation results from the variables associated with fixed charges. Fixed charges or fixed costs correspond with the building and operating expenses of facilities. Whenever a facility is established (opened) it incurs a fixed cost, and this cost is zero when the facility is closed. It is the nonlinearity of this cost function (discontinuity occurs when the facility is closed), which makes the standard linear programming techniques ineffective in solving this class of problems. Furthermore, nonlinearities occur as the result of economies of scale in transportation, production, and construction/ operation costs. However, in the absence of fixed charges and economies of scale, or for a given location vector, the

facility location problem can be simply reduced to a transportation problem and procedures such as Out-of-Kilter algorithm may be used effectively to solve the problem.

Many algorithms developed to date for optimal solution of LAPs employ the branch and bound procedure of integer linear programming. The branch and bound procedure is an implicit enumeration technique which is guided by an upper and a lower bound on the value of the objective function. The method is based on solving a series of linear programming problems with the integer requirements relaxed. The procedure progressively improves the bounds for the optimal solution of the original mixed integer problem. A major advantage of this technique is that it continually recomputes the bounds on the objective value, which enables the decision maker to stop the calculation whenever the solution is within a prespecified tolerance of optimal value. For this method, the lower bound could simply be established by solving the original mixed integer (or integer) problem without considering the integrality constraints. And, the upper bound may be obtained by arbitrarily assigning values of 0 and 1 to the binary variables.

Nevertheless, better lower bounds are established through applying the Lagrangian relaxation technique. This technique is based on multiplying some of the constraints by a penalty factor and then adding them to the objective function. It is shown that the resulting subproblem is

usually easier to solve than the original problem and provides a better lower bound than the linear relaxation method mentioned earlier. As will be evident throughout the literature, the direction of research has been to improve computational efficiency of the branch and bound procedures by improving lower bounds, upper bounds, and the node selection and branching rules.

Studies of exact methods in LAPs may be further classified according to main characteristics of the problem formulation, these are:

1) Simple (uncapacitated) problems;
2) Capacitated problems;
3) Dynamic problems;
4) Multi-commodity problems;
5) Stochastic problems.

Simple (Uncapacitated) Problems. In simple LAPs a number of facilities with unlimited capacities are selected from among a set of predetermined sites and then demand centers are assigned to them. The assumption of uncapacitated plants is usually justified whenever considering establishing new plants. This assumption greatly simplifies the allocation part of the problem. That is, for this case, the optimal allocations for a given location vector are found simply by assigning the demand for each demand center from a single plant which has the lowest unit cost (i.e. combination of the unit production and

distribution costs).

Among the exact methods proposed for solving the simple, uncapacitated facility location problem, algorithms by Efroymson and Ray, Spielberg, Khumawala, and Erlenkotter are particularly well known.

An early attempt to optimize the simple, uncapacitated facility location problem is a branch and bound procedure proposed by Efroymson and Ray (1966). By reformulating the problem, Efroymson and Ray were able to simplify the solution of the linear programming problems at each node. Additionally, they presented certain simplifications at each node which reduced the number of evaluations required in solving the original problem.

Spielberg (1969)a employed an implicit enumeration to solve the simple plant location with side constraints. In another paper Spielberg (1969)b reported computational efficiency in solving the simple plant location problem by relocating the search origin from a "natural" search origin (where all facilities are initially opened or closed) to a generalized search origin. This paper also suggests a series of tests for pruning the branches of the branch and bound tree.

Curry and Skeith (1969) utilized dynamic programming to solve simple facility location problems.

Khumawala (1972) significantly improved the branch and bound algorithm of Efroymson and Ray by proposing a set of

branching decision rules in conjunction with a more efficient method for solving the linear programming problem at each node. This algorithm partitions the set of feasible locations into three sets: (1) the set of locations with closed warehouses, K_0 ; (2) the set of locations with open warehouses, K_1 ; and (3) the set of locations at which the status of warehouses are undecided (free warehouses), K_2. The branching decision rules determine which of the warehouses should be opened or closed at each node. Among the set of four proposed branching rules (Delta, Omega, Y, Demand), the largest Omega rule was shown to perform the best. Omega is the symbol used to denote the minimum savings of opening a free (not yet assigned open or closed) warehouse in the presence of all open warehouses. Delta is a measure similar to Omega except for the comparisons which are made with respect to all non-closed (open and free) warehouses. The Y branching rule selects a free warehouse with largest or smallest Y_i value at each node, and fixes it open or closed respectively. Finally, the Demand rule selects a free warehouse among the set of free warehouses which can supply the greatest or smallest total demand, and fixes it open or closed respectively.

Kaufman, Eede, and Hansen (1977) extended the work of Efroymson and Ray by considering a single echelon facility location problem. They applied the branch and bound procedure to simultaneously solve for the location of plants

and warehouses in a distribution system. In this system customer demands may be satisfied directly from plants or through warehouses.

Erlenkotter (1978), proposed a dual-based solution for the simple, uncapacitated facility location problem. He applied a simple ascent and adjustment method to the condensed dual formulation of the problem. The procedure begins with an initial dual solution and adjusts the multipliers (dual variables) incrementally in a way that reduces complementary slackness violations. The procedure continues until either complementary slackness is satisfied or dual feasibility is violated. Moreover, if the optimal dual solution does not correspond to the optimal integer primal solution, then a branch and bound procedure is employed to complete the solution. Erlenkotter demonstrated computational efficiency of this algorithm through some example problems.

Tcha and Lee (1984), generalized the work of Kaufman *et al*. (1977) by studying the multi-echelon facility location problems. Their algorithm, based on the modified dual ascent procedure of Erlenkotter, is shown to be superior to the algorithm of Kaufman *et al* (1977).

Capacitated Problems. In capacitated LAPs, it is assumed that there exist an upper and/or lower bounds on the production (capacity) of the potential facilities. Among the exact procedures for solving capacitated facility location

problems are algorithms by Davis and Ray (1969), Sa (1969), Ellwein and Gray (1971), Truscott (1975), Akinc and Khumawala (1977), Geoffrion and McBride (1978), Nauss (1978), Christofides and Beasley (1983), and Van Roy (1986).

Davis and Ray (1969) incorporated the capacity constraints into facility location problems. Their method employs a branch and bound procedure and uses Benders decomposition technique to solve the dual of the associated continuous linear problem at each node of the branch and bound tree. The decomposition technique at each iteration produces a "master problem" and a single "sub-problem". The dual of the "sub-problem" represents a capacitated transportation problem, and is solved effectively at each iteration by an Out-of-Kilter algorithm.

Sa (1969) proposed a branch and bound procedure similar to Davis and Ray's method. However, his method added a dominance test and a feasible total fixed cost test which are performed before solving any subproblem.

Ellwein and Gray (1971) studied capacitated facility location problems with configuration constraints. They employed an enumerative search technique where the enumeration of the solution vectors is carried out by generating a sequence of partial assignments. The partial assignments constituted assignment of "zero", "one", and "free" to the integer variables. Ellwein and Gray achieved computational efficiency in solving the problem by reducing the feasible solution set and therefore the size of the

search, through utilizing adaptive bounds on the fixed costs and constraints based on the dual variables.

Truscott (1975) also investigated facility location problems with capacity and configuration constraints. He added the dimension of revenue generation to his model. Because, the choice of facilities can effect the price realized and/or the quantities demanded. Therefore, the problem was formulated and solved as a zero-one integer programming problem with an objective of maximizing profit.

Akinc and Khumawala (1977) presented a procedure based upon the branch and bound algorithm for the capacitated warehouse location problems. They increased the efficiency of the branch and bound procedure by developing powerful lower and upper bounds along with a different set of rules for selecting nodes and branches. For example, they proposed a hybrid node selection rule. This rule employs both least lower bound and LIFO to select a node. The algorithm switches between these two rules based on the value of the two parameters. They indicated that the least lower bound rule results in a large number of terminal nodes, therefore it requires relatively large storage but has the advantage of minimizing computational time. On the other hand, the LIFO rule requires relatively smaller storage, but results in longer computational time. Hence, they proposed the hybrid node selection rule in an attempt to compromise between these two rules.

Geoffrion and McBride (1978) applied Lagrangian relaxation to capacitated facility location problems with lower bounds on the capacity of each facility and an arbitrary set of linear constraints. The Lagrangian problem decomposes into m continuous Knapsack problems, one for each facility. The linear side constraints are used to control distribution flows as well as opening and closing of facilities. Geoffrion and McBride (1978) also have shown, in applying the branch and bound technique, that lower bounds generated via Lagrangian relaxation is superior to the ones obtained by traditional linear relaxation.

Nauss (1978) improved the branch and bound procedure of Akins and Khumawala (1977) by deriving tighter lower bounds through employing Lagrangian relaxation of demand constraints. The tighter lower bounds facilitate fixing certain facilities open or closed thus reducing the amount of branching required.

Christofides and Beasley (1983) developed a similar approach to that of Nauss (1978) and obtained slightly better results.

Van Roy (1986) presented a different approach based on the Cross Decomposition (CD) method developed by Van Roy (1983) to solve the capacitated facility location problem. The method is designed to exploit simultaneously the primal and dual structure of the problem. This method unifies Benders decomposition and Lagrangian relaxation into a single framework that involves successive solutions to a Benders

(primal) subproblem and a Lagrangian (dual) subproblem. The primal and dual subproblems are transportation and simple plant location problems respectively.

Furthermore, capacitated location models with nonlinear economies of scale are studied by Soland (1974), and Kelly and Khumawala (1982).

Dynamic Problems. Whenever demands and/or costs change from period to period, it is appropriate to incorporate the time dimension into the formulation of the location-allocation problem. Relocation costs, possible expansion, and changes in customer locations over time are other factors that require dynamic location considerations, Green *et al.* (1981). In short, multi-period or dynamic warehouse location problem considers the locational decisions over a specified planning horizon such that the total discounted costs of meeting demands are minimized. Of course, the profit maximization aspect could easily replace the objective of cost minimization. Some of the studies which consider dynamic characteristic of the problem are by Wesolowsky and Truscott (1975), Khumawala and Whybark (1976), Karanicolas (1979), Van Roy and Erlenkotter (1982).

Wesolowsky and Truscott (1975) presented two methods for solving dynamic (multi-period) location problems. In the first method, they discounted all costs to their present values and then used a mixed integer programing formulation to find the optimal solution. In the second method, they

applied dynamic programming solution methodology to solve the problem. The dynamic programming formulation includes costs of vacating and entering sites and defines stages, states, and decision variables to be periods, facility configurations, and choices of location changes respectively.

Khumawala and Whybark (1976) proposed a solution procedure based upon the implicit enumeration, for solving warehouse location problems with changing markets and costs from period to period. The algorithm is comprised of three steps. Steps one and two are applied iteratively to determine if any free warehouses can be opened or closed. The third step, a branch and bound procedure, is entered only if there is at least one free warehouse following the application of the previous cycle.

Karanicolas (1979) presented an algorithm for the solution of multiperiod capacitated and uncapacitated plant location problems. The proposed algorithm employs the Lagrangian relaxation technique to decompose a multiperiod problem into T single period mixed integer subproblems and an integer master problem.

Van Roy and Erlenkotter (1982) developed a branch and bound solution procedure incorporating an extension of the dual ascent procedure of Erlenkotter (1978) with a primal-dual adjustment procedure to solve the dynamic, uncapacitated facility location problem.

Dynamic facility location-allocation problems have also

been studied by Ballou (1968), Tapiero (1971), and Sweeney and Tatham (1976).

Multi-Commodity Problems. Another generalization of the facility location problems has been to incorporate the aspect of multi-commodity into the formulation. A simple method to deal with this problem is to replicate each demand center as many times as there are products and to assign an appropriate demand to each one. Among the research in the area of multi-commodity problems is the study by Elson (1972), Warszawski (1973), Geoffrion and Graves (1974), Khumawala and Neebe (1978), and Karkazis and Boffey (1981).

Elson (1972) was among the first to study single echelon, capacitated, multi-commodity facility location problem. Elson presented a mixed-integer formulation of the problem and applied existing mixed integer programming codes to solve the problem. Among the characteristics of Elson's model is the use of different set of variables to represent product flows from and to the warehouses.

Warszawski (1973) also investigated the dimension of multi-commodity. His study was motivated from the need to locate different supply sources at a construction site. Warszawski proposed a branch and bound procedure along with a heuristic method to solve this problem.

Warszawski and Peer (1973) presented a formulation for the multi-commodity, multi-period location problem, but did not attempt to solve it.

Geoffrion and Graves (1974) also studied the single echelon, capacitated, multi-commodity distribution system. They implemented a solution technique based on the Benders decomposition to decompose the full multi-commodity problem into a series of simpler single commodity problems. Benders method proceeds by alternatively solving an integer programming master problem and then several transportation subproblems. The master problem involves the 0-1 variables of the location vector while the subproblems are a simple transportation problem for each commodity. The solution procedure starts by solving the master problem ignoring the 0-1 requirements followed by solving the transportation subproblems. With each iteration, one or more additional constraints are added to the master integer problem setting fractional Y's to 0 or 1. These new constraints are called Benders Cuts. Contrary to Elson's model, Geoffrion and Graves (1974) considered the product flow in such a way as to preserve the identity of plants in the final assignment of products from warehouses to customers. Moreover, Geoffrion and Graves (1974) presented details of the application of their model to a real world system.

Khumawala and Neebe (1978) and Neebe and Khumawala (1981) improved the branch and bound procedure of Warszawski by incorporating stronger lower bounds. Their algorithm employed LIFO and the least lower bound as the node selection rules and used the largest delta rule of Khumawala's

algorithm for the branching decision rule.

Karkazis and Boffey (1981) also proposed two dual based algorithms for solving multi-commodity facility location problems.

Loh (1983) studied the multiple commodity and multiple stage transshipment location problem. He proposed an algorithm based upon the integration of branch and bound and dynamic programming as the fundamental solution methodology.

Stochastic Problems. Uncertainty considerations in the forecast of demands for the LAPs are important and have been accounted for by several researchers. In general, whenever demand is not known with certainty, it is appropriate to treat it as a random variable. A simple technique suggested for dealing with the assumption of stochastic demands is to compute an expected or most likely value for the demands. Then the problem can be solved by applying the conventional methodologies for the deterministic problems. Price sensitive demand is another characteristic of LAPs which has been considered by the researchers. In this situation prices received at a demand center varies depending upon the location of supplying facility because of transportation costs, local utility costs, competition, etc. And, the price of a product will determine its demand at a demand center.

Gonzalez-Valenzuela (1975) was the first to incorporate stochastic demands into the simple and capacitated warehouse location problems. This study investigated two different

approaches for dealing with the uncertainty in demands, producing two different formulations: a chance-constrained formulation and a stochastic or two-stage programming formulation. The chance-constrained model is transformed to an equivalent deterministic model and is solved by means of one of the existing methods for deterministic warehouse location problem. The stochastic programming model requires an explicit assumption as when the actual values of the demands will become known. Hence, an analysis is made of the differences in the formulation of the problems that arise as the result of this assumption. Then the stochastic programming problem is transformed into a deterministic model and the resultant problem is solved by an existing method (Khumawala 1972) for deterministic warehouse location problems.

Balachandran and Jain (1976) studied the facility location problem with random demand and general cost structure. They assumed cost of operating a plant to be a piece wise linear function of the production level.

Jucker and Carlson (1976) extended the simple plant location problem by permitting uncertainty in either the price or the demand. They stated the problem as one of maximizing total profit and presented a mean-variance formulation for the objective function. Furthermore, Jucker and Carlson assumed that there is no relationship between price and quantity demanded and given the identification of a firm's risk taking behavior, decomposed the problem into two

simpler problems which were solved by existing methods.

Erlenkotter (1977) formulated a simple plant location problem such that demands are related to the prices established at the various locations. Erlenkotter presented a profit maximization model for private, public, and quasi-public facilities. In the latter case, the problem is to maximize the net social benefits subject to generating sufficient revenues to cover costs. In all the above cases pricing and location decisions are determined simultaneously. Erlenkotter reformulated the above problems into an equivalent fixed demand models and then applied the existing solution techniques for their solutions.

Hansen and Thisse (1977) also investigated the dimension of price sensitivity and presented a profit maximization model for the problem. Their solution technique is to reformulate the problem into the simple plant location problem and then to apply the existing well known methods for its solution.

Harrison (1979) presented a stochastic programming model, which through the expectation function, can deal with uncertainties in demand.

Sicsu (1979) proposed a model to analyze capacitated location-allocation problems with price-sensitive demands. He considered a profit maximization objective and formulated the problem as a mixed-integer, nonlinear optimization problem.

Rasaratnam (1984) combined the dimensions of price sensitivity and stochastic demand for the capacitated facility location problems. He assumed parameter(s) of the demand distribution vary with variations in price without altering the distribution of demands. The results were also extended to include uncapacitated facility location problems.

Logendran and Terrell (1988) reported on the solution and results of the uncapacitated plant location-allocation problems with price sensitive stochastic demands.

Simulation Techniques

Simulation is a flexible modeling and design tool. It allows for the investigation of alternative system designs and strategies without the expenses of actually building and operating them.

The basic process of simulation methods in facility location problems is to vary the facilities location-allocation pattern and compare the resultant effects on distribution and total costs. According to Geoffrion (1975), "simulators can take detailed account of policies and activities relating to inventory replenishment, individual buying patterns of customers, order filling, redistribution, transportation, and so on, and produce a simulated daily history of such activities for a period of one year or more". Clearly, this depth of analysis is not easily possible by other methods. Another great benefit of simulation

procedures is that extremely rich and precise cost parameters may be included in the analysis. However, although simulation allows for representation of complex and large distribution systems, but it does not guarantee optimality of these models.

Among the most widely cited simulation models in the literature are the ones developed by Shycon and Maffei (1960), and Cerson and Maffei (1963). These models deal with two real distribution systems. For these models, gathering and employing the data base was reported to be the most difficult task in designing the logistical system.

Additionally, more complex simulation models are reported by Bowersox (1972), Connors, Coray *et al.* (1972), Camp (1973), and Markland (1973).

Multi-Objective LAPs

Multiple objectives location allocation models have gained considerable attention from researchers in recent years. LAPs traditionally have been studied as single objective optimization problems. Nevertheless, in almost all real world applications of LAPs, decisions must be made in the presence of a number of conflicting objectives. As such, a multiple criteria approach is the most appropriate strategy to be used for analyzing the effect of various multiple and often competing objectives in LAPs.

Although, literature on LAPs is considerably large, but

there are relatively a few papers in the area of multi-criteria LAPs. Among the research in this area are the work by Lee and Franz (1979), Ross and Soland (1980), Green, Kim, and Lee (1981), Eilon (1982), Fortenberry and Mitra (1986), Lee and Luebbe (1987), and Sinha and Sastry (1987).

Lee and Franz (1979), and Lee, Green, Kim (1981) applied the branch and bound method of integer goal programming for the solution of facility location-allocation problems with multiple objectives.

Ross and Soland (1980) conducted a multicriteria analysis of the location of public facilities. The problem is formulated as a generalized assignment problem (GAP) with a set of additional constraints. The efficient solution of the multicriteria location problem is generated by solving a finite sequence of GAP-type problems. Furthermore, an interactive approach is presented with which the decision maker can efficiently arrive at an acceptable compromise solution among the various criteria. The extension of this model to private sector problems is also discussed.

Green, Kim, and Lee (1981) applied the integer goal programming to study a multi-criteria warehouse location problem in the presence of a single supply source. Their model incorporates both qualitative and quantitative factors in solving the problem.

Eilon (1982) presented an alternative approach based upon the heuristic algorithm for the loading problem, Eilon *et al.* (1971), to solve the problem presented by Green, Kim

et al. (1981).

Fortenberry and Mitra (1986) proposed a model based on the weighted objective function to solve multi-criteria location-allocation problems. The weights are established based on the relative importance assigned to qualitative factors and are applied to the transportation costs from each location. The proposed model combines both qualitative and quantitative techniques for the solution of the problem. In particular, in the absence of fixed costs, it uses the traditional transportation algorithm to obtain a solution for the location-allocation problem.

Lee and Luebbe (1987) demonstrated the capability and flexibility of the model developed originally by Green, Kim *et al.* (1981) through conducting a sensitivity analysis of this model.

Sinha and Sastry (1987) presented a zero-one linear goal programming model and demonstrated its application for a real world multi-objective facility location problem.

Conclusion

A summary of the literature in the area of single objective and multiple objective LAPs is provided in Tables 2.1 and 2.2, respectively. Although single objective LAPs have received a substantial amount of attention in the literature, the multi-criteria formulations of the problem have not been studied extensively. Specifically, to the best

of the author's knowledge, incorporation of stochastic demand into these models is nonexistent. This research will combine multi-criteria and uncertainty of demands into an interactive model to better represent actual decision making environment of this class of problems.

TABLE 2.1

SUMMARY OF SINGLE OBJECTIVE LAP PROCEDURES

Heuristic	Simulation
Kuehn and Hamburger (1963)	Shycon and Maffei (1960)
Manne (1964)	Cerson and Maffei (1963)
Feldman, Lehrer, & Ray (1966)	Bowersox (1972)
Sa (1969)	Connors *et al.* (1972)
Walker (1976)	Camp (1973)
Sule (1981)	Markland (1973)
Klincewicz and Luss (1986)	

Optimization (Exact)

Uncapacitated
- Efroymson and Ray (1966)
- Spielberg (1969)
- Curry and Skeith (1969)
- Khumawala (1972)
- Kaufman *et al.* (1977)
- Erlenkotter (1978)
- Tcha and Lee (1984)

Capacitated
- Davis and Ray (1969)
- Sa (1969)
- Ellwein and Gray (1971)
- Soland (1974)
- Truscott (1975)
- Akinc *et al.* (1977)
- Geoffrion *et al.* (1978)
- Nauss (1978)
- Kelly and Khumawala (1982)
- Christofides *et al.* (1983)
- Van Roy (1986)

Dynamic Problems
- Ballou (1968)
- Tapiero (1971)
- Wesolowsky and Truscott (1975)
- Khumawala and Whybark (1976)
- Sweeney and Tatham (1976)
- Karanicolas (1979)
- Van Roy and Erlenkotter (1982)

Multi-commodity
- Elson (1972)
- Warszawski (1973)
- Warszawski and Peer (1973)
- Geoffrion and Graves (1974)
- Khumawala and Neebe (1978)
- Neebe and Khumawala (1981)
- Karkazis and Boffey (1981)
- Loh (1983)

Stochastic/Price Sensitive
- Gonzalez-Valenzuela (1975)
- Balachandran and Jain (1976)
- Jucker and Carlson (1976)
- Erlenkotter (1977)
- Hansen and Thisse (1977)
- Harrison (1979)
- Sicsu (1979)
- Rasaratnam (1984)
- Logendran and Terrell (1988)

TABLE 2.2

SUMMARY OF MULTIPLE OBJECTIVE LAP PROCEDURES

Author	Solution Strategy	Stochastic Demand
Lee and Franz (1979)	GP	NO
Ross and Soland (1980)	Weighting Technique	NO
Green, Kim, & Lee (1981)	GP	NO
Eilon (1982)	Weighting Technique	NO
Fortenberry *et al.* (1986)	Weighting Technique	NO
Lee and Luebbe (1987)	GP	NO
Sinha and Sastry (1987)	GP	NO

CHAPTER III

MULTIPLE OBJECTIVE DECISION MAKING

Introduction

Multiple objective decision making (MODM) is a dynamic process of making decisions in the presence of multiple and frequently conflicting objectives and often subject to satisfying the rigid constraints of the system. Today, minimizing cost or maximizing profit is no longer recognized as the sole objective of most organizations. The strategic problems faced by today's managers and decision makers, require the achievement of a balance between multiple and often incommensurate objectives. These objectives or criteria usually include issues such as economic, environmental, political, public relations, labor relations, and social responsibilities.

This chapter contains two main sections. First, an overview of multiple criteria decision making techniques, including basic terminology, and a classification scheme is discussed. Second, a review of goal programming and in particular its formulation, solution procedures, integer and interactive goal programming is presented.

An Overview of MCDM Methods

This section presents some of the important terminologies in multiple criteria decision making (MCDM) and describes a classification of MCDM techniques.

Multiple criteria decision making involves any or all of the following criteria: attributes, objectives, and goals. An attribute describes an objective reality such as weight, height, profit, cost, etc. An objective represents direction of improvement or preference for attributes (Zeleny 1982); for example improving quality is an objective. A goal represents a specific value or level of an objective or attribute; for instance achieving a profit of at least two million dollars is a goal. Another important concept in MCDM is the nondominated or noninferior solution. According to Zeleny [1982, p. 72] "a nondominated solution is a feasible solution for which an increase in value of any one criterion can be achieved only at the expense of a decrease in value of at least one other criterion". The set of nondominated solutions are usually referred to as the "efficient set", "admissible set", "noninferior set", or "Pareto optimal set". Since in MCDM objectives are often conflicting, there is usually no single solution which optimizes all objectives simultaneously. As the result, the decision maker (DM) will select the best compromise solution from the nondominated set of solutions through a value trade-off analysis.

Some researchers have attempted to deal with multi-

objective problems through applying existing single objective algorithms. Examples of these approaches include weighting techniques and constraint techniques. Weighting techniques aggregate all objectives into a single one by assigning the same utility measure, such as monetary values, to all objectives. The trade-off analysis of this approach is accomplished by varying the specified weights and then resolving the problem. On the other hand, constraint techniques deal with multiple objectives by assigning to one of the objectives the role of primary objective and treating the remaining objectives (secondary objectives) as a system of constraints to be satisfied. In this technique trade-off analysis is performed by selecting a different primary constraint and/or specifying different requirements for the secondary objectives.

Nevertheless, these techniques are not adequate for handling problems with multiple objectives. More specifically, weighting techniques are not appropriate since in practice some of the objectives are non-commensurable. In this situation it is extremely difficult if not impossible to assign the same utility measure across all objectives. Furthermore, the use of a utility function requires that the DM to specify his/her preferences quite accurately. On the other hand, constraint techniques may fix the relative importance of objectives improperly, and thus not allow for compromise solutions. Additionally, when performing trade-off analysis, the computational requirement of both

techniques increases exponentially with an increase in the number of objectives. Therefore, it is advantageous to use algorithms developed specifically for MCDM to analyze this class of problems.

MCDM Classification

An early attempt to classify MCDM methods was made by MacCrimmon (1973). He described 19 methods and grouped them into four main categories: 1) weighting methods, 2) sequential elimination methods, 3) mathematical programming methods, and 4) spatial proximity methods. However, since then various classification schemes have been proposed; Cohon (1978), Hwang *et al.* (1979, 1980), and Goicoechea *et al.* (1982). A major component of all these classifications is the point in time at which the decision maker incorporated his/her preferences into the decision making process in order to generate or rank the various alternative solutions. Perhaps the most complete MCDM classification is presented by Hwang *et al.* (1980). They proposed a hierarchical structure based upon first, the stage at which the preference information is needed, and second, the type of information needed. With respect to the DM's articulation of preference, they proposed four categories. These are:

1) no articulation of preference;
2) a priori articulation of preference;
3) progressive articulation of preference;

4) a posteriori articulation of preference.

Next, considering the type of information, they distinguished between four categories: a) cardinal information, b) ordinal information, c) explicit tradeoff, and d) implicit tradeoff. A brief description of these classifications along with advantages and drawbacks of each class is given next.

No Articulation of Preference Methods. These methods include mainly global criteria methods. They do not require the DM to input his/her subjective preferences once the problem is formulated. They provide a single alternative solution for the DM. An advantage of these methods is that the DM is not required to work with an analyst/computer during the solution process. A disadvantage is that it requires the analyst to make many assumptions about the DM's preferences which is often very difficult or impossible. An example illustrating this technique is to find a solution vector which minimizes the sum of squares of the relative deviation of the objective functions at this point from their respective ideal points, where the latter is defined as the value of each objective if it was the only one being optimized.

A Priori Articulation of Preference Methods. This class of methods relies on the decision maker to specify his/her preference information about objective levels and/or their ranks prior to analysis. The preference information may be

either cardinal or a combination of ordinal and cardinal information. The advantages of these techniques are that the DM is not required to participate during the solution process, and that their computational speed is usually higher than other classes, because only a small portion of the nondominated solutions will be investigated. On the other hand, a major disadvantage of this class is that in most situations, particularly when the DM is not familiar with the available alternatives, he/she is unable to provide accurate preference information prior to the analysis. In summary, the underlying assumption of the algorithms in this category is that the DM can provide accurate preference information prior to analysis, and that his/her preference structure remains relatively fixed and consistent throughout the solution process. Among methods in this category are utility function methods, lexicographic methods, and goal programming.

Progressive Articulation of Preference Methods. These methods, commonly referred to as interactive procedures, are based upon interaction of the DM with the analyst or computer during the solution process. At each iteration, given the current solution(s), the DM is asked to provide some tradeoff or preference information in order to generate the next solution. This process continues until either the DM is satisfied with a set of achievement levels for the objectives or decides that there is no satisfactory solution for the

current system. Advantages of these methods according to Hwang *et al.* (1980) are: 1) a priori preference information is not required, 2) the DM will explore the criterion space through a learning process, 3) only local preference information is required, and 4) the solution has a better chance of being accepted, since the DM is involved in the solution process. To the contrary, the disadvantages are: 1) the solution is highly dependent upon the ability of the DM to indicate accurate local preferences, 2) a preferred solution may not be obtained within a reasonable time, and 3) effort on the part of the DM may be excessive. The underlying assumption for the algorithms in this category is that the DM's preferences form and evolve as the result of a learning process, from one iteration to the next. Also because of the complexity of the system, the DM is only able to provide his/her preference information on a local level for a particular solution. Interactive procedures such as the methods of Zionts-Wallenius, STEM, interactive goal programming, and interactive MOLP are a few examples in this category.

A Posteriori Articulation of Preference Methods. These methods, also referred to as generating techniques, are designed to generate a subset or the complete set of nondominated solutions. Then, the DM selects the best solution among available alternatives based upon his/her preference structure. The underlying assumption in this

class of algorithms is that the DM can not identify his/her preferences prior to a knowledge of available alternative solutions. The advantages of these methods are that they do not require any information regarding the DM's utility function before or during the computational phase. Also, once the set of nondominated points are generated, they may be used by different DMs to reach a solution without resolving the problem. However, there are at least two disadvantages to these methods. First, they are very resource consuming. That is, given a large problem, it may not be feasible to generate the whole set of nondominated solutions due to constraints on computer time and/or storage requirements. Second, as the number of nondominated solutions grows, it becomes very difficult, if not impossible, for the DM to select the most satisfactory solution among the alternatives. Some of the techniques in this category are: constraint method, multiple objective linear programming, and parametric (weighting) methods such as compromise programming.

A comprehensive review of methods and techniques in MCDM is provided by Cohon (1978), Hwang *et al.* (1979), Zeleny (1982), Goicoechea *et al.* (1982), and Steuer (1986).

Goal Programming Methods

Goal programming (GP) is one of the popular methods of multiple objective analysis. It is particularly valuable

whenever achieving target values of objectives are important and when preferences of the DM regarding the goals and their priorities can be specified properly. The concept of goal programming was originally introduced by Charnes and Cooper (1961). Later, this technique was refined and extended by Ijiri (1965), Lee (1972), and Ignizio (1976). For instance, Ignizio (1976) extended the formulation of the original continuous linear GP to include integer and nonlinear models. Today, research and application of goal programming continues to grow significantly. In fact, studies by Petty and Bowlin (1976), and Green *et al.* (1977) have identified goal programming as the major multiple objective tool in use by practitioners.

For the general goal programming formulation, the DM specifies goals, targets, or aspiration levels for multiple objectives and provides an ordinal ranking of these objectives. Next, each goal is written as an equality constraint including a positive and a negative deviational variable, d^+, d^-. Then, the preferred solution is obtained by minimizing the weighted set of these deviations which represent the differences between actual objective achievements and their prespecified desired goals, subject to satisfying the technological (system) constraints. In general, this is equivalent to finding a feasible solution which satisfies the goals as closely as possible, based upon some specified measure.

Regarding to the assignment of weights to the

deviational variables in the objective function, two basic GP models are distinguished: Archimedian (weighted or minisum) GP, and preemptive (lexicographic) GP. Both of these models rely on setting the goals a priori. With the Archimedian GP, all goals are considered simultaneously. That is, the objective function consists of minimizing a weighted sum of all goal deviations at the same time. The deviations are measured using l_p metric, with p usually set at 1, 2, or ∞. Alternatively, preemptive GP considers the goals separately based on a specified priority structure. For these models, goals at a higher priority levels are considered to be infinitely more important than the goals at a lower priority levels. Considering these two approaches to GP, preemptive GP has gained more attention in the literature. The general mathematical formulation of the preemptive goal programming problem can be stated as follows:

$$\text{MINIMIZE} \quad D = \sum_{k=1}^{K} P_k \left[\sum_{i=1}^{m} \left(w_{ik}^{+} d_i^{+} + w_{ik}^{-} d_i^{-} \right) \right] \tag{3.1}$$

SUBJECT TO:

$$\sum_{j=1}^{n} a_{ij} x_j - d_i^{+} + d_i^{-} = G_i \,, \qquad i=1,2,\ldots,m \tag{3.2}$$

$$G_i \,,\ d_i^{+} \,,\ d_i^{-} \geq 0 \,, \qquad i=1,2,\ldots,m \tag{3.3}$$

$$x_j \geq 0 \,, \qquad j=1,2,\ldots,n. \tag{3.4}$$

where

P_k = the k*th* preemptive priority level; k = 1,2,...,K.

w_{ik}^{+} = the weight factor for d_i^{+} at priority P_k.

w_{ik}^{-} = the weight factor for d_i^{-} at priority P_k.

d_i^{+} = over-achievement of goal i.

d_i^{-} = under-achievement of goal i.

a_{ij} = the coefficient of the j*th* decision variable in the i*th* goal constraint.

x_j = j*th* decision variable; j = 1,2, ... ,n.

G_i = the i*th* goal (target) level; i = 1,2, ... ,m.

The constraints in Equation (3.2) may also include the system (technological) constraints, in which case the goals, G_is, represent the rigid constraining values and d_i^{-}, d_i^{+} represent the slack and surplus variables respectively as appropriate. In this situation, slacks or surplus variables, will be represented in the objective function at the highest priority level. Then the minimization of variables at this priority level must be fully achieved for the problem to have a feasible solution.

If over-achievement or under-achievement is allowed for goal i, then the negative or positive deviations about goal i (d_i^{-} or d_i^{+}), must be minimized respectively. Alternatively, if goal i is to be achieved exactly then both deviational variables must be minimized.

In Equation (3.1) $P_k >> P_{k+1}$ which means that goals at priority level P_{k+1} are considered only after goals at priority level P_k are fully achieved or reach a point beyond which further improvement is not possible. Furthermore, in considering goals at lower priorities, provisions are made to

prevent diminishing objective achievements for higher priority goals. More specifically, in preemptive GP, one first obtains all alternative solutions which minimize the sum of deviations of all priority one objectives from their corresponding goal values. Then, from those alternatives we select the ones which minimize the sum of deviations of priority two objectives from their corresponding goal levels. This iterative process continues until all priorities are considered or no more alternative solutions are available. Thus, the solution method is a dynamic process in which the information from the previous stage is used to solve the subsequent stage.

GP Computational Algorithms

Generally, as discussed earlier, the solution procedure of preemptive GP follows a sequential optimization process, where successive optimizations are performed on the available alternatives. At each iteration, a LP problem is solved for each priority, from higher to lower priorities, with the restriction of not deteriorating the previously established goal attainments. However, since the development of the GP technique, different algorithms have been proposed for its solution.

Originally, Lee (1972) developed the "modified simplex method" or "multiphase simplex" to solve the preemptive linear GP. This technique is based on the conventional LP

simplex algorithm. The multiphase technique, along with its extensions to integer and nonlinear GP, is discussed in detail by Ignizio (1976). Other primal simplex variations such as revised simplex and product form of the revised simplex, Olson (1984), have also been used to solve preemptive GP. The computational advantage of these latter methods are in their storage requirements and solution accuracy.

Another approach for solving linear GP problems is denoted as Sequential Linear Goal Programming or SLGP method, Ignizio and Perlis (1979). In this technique the existing linear programming computer codes, such as MPSX, are applied sequentially to solve the linear GP problems. The above computer codes are capable of solving very large linear programming problems, thus this approach is particularly advantageous whenever large-scale GP problems are involved.

Arthur and Ravindran (1978) developed an efficient "Partitioning Algorithm", (PAGP), by taking advantage of the hierarchical structure of preemptive GP. This algorithm is based on solving a series of linear programming subproblems. At each iteration, the solution of the higher priority subproblem is used as the starting solution of the lower priority subproblem. The algorithm iterations continue until either no alternative solution is present for one of the subproblems or all subproblems (priorities) are considered. If the algorithm terminates before reaching the lowest priority goal, achievement levels of the lower priority goals are calculated by substituting values of the current optimal

solution into their corresponding equations. Computational efficiency is gained by considering only the variables and constraints affecting the current most important unsatisfied goal (priority level).

Schniederjans and Kwak (1982) presented the dual simplex goal programming algorithm based on the dual simplex method of linear programming. The algorithm starts from an infeasible basic solution formed by the positive deviational variables and applies the dual iterations to find the optimal solution. According to Olson (1984), in comparing various GP algorithms, this method gains computational efficiency by eliminating up to half of the deviational variable columns from the simplex tableau. Stated differently, this algorithm becomes more efficient as the number of positive deviations in the problem increases.

Integer Goal Programming Techniques

Since the solution vector for multiobjective LAPs is discrete or integer, it is necessary to employ integer techniques for their solutions. According to Lee (1979), given there is no conflict among multiple objectives of an integer problem, the conventional integer linear programming algorithms may be used to solve the problem. However, in the presence of conflicting objectives and preemptive priority weights, it requires an integer GP procedure.

There are three basic approaches to integer GP. They include modifications of the Gomery's (1958) cutting plane technique, Land and Doig's (1960) branch-and-bound method, and a combination of the Balas' additive algorithm (1965), and Glover's backtracking procedure (1965) for solving the zero-one GP problems. A complete description of these methods along with several solution examples are provided by Lee and Morris (1977) and Lee (1979).

Interactive GP and Sensitivity Analysis

Interactive procedures are the most effective methods of searching the tradeoff space for the most satisfactory solution and are gaining wide acceptance for implementation. They enable the DM to find the best solution through a systematic process. The reasons for employing an interactive procedure are: 1) to allow the DM the ability to explore the criterion space through the objective tradeoff analysis, 2) to perform sensitivity analysis, 3) to reduce computational burden of producing and then selecting from the whole set of nondominated solutions, and 4) to exclude the requirement of exact data from the DM prior to analysis.

Zeleny (1982) presents some of the assumptions and limitations of preemptive GP. According to Zeleny there are two main limitations or drawbacks of using the preemptive GP. First, improper setting of goals may result in a dominated solution. Second, no trade-off is allowed among achievement levels of various goals, which means small improvement in

higher priority goals are preferred and achieved regardless of the cost to the lower priority goals. In other words, higher priority goals are infinitely more important than lower priority goals. The application of interactive GP overcomes these difficulties by allowing the DM to perform value tradeoff analysis of the achievement levels of various objectives. In interactive GP, at each iteration, the DM is asked to express his/her preferences regarding the goal priority and/or the target value of the goal constraints, based upon the previous solution(s), to generate a new solution. The algorithm terminates whenever the DM is satisfied with a solution or decides there is no satisfying solution under current constraints and resources.

The changes in the priority structure and/or levels of the goals also constitute sensitivity analysis of the model. It enables the DM to explore the feasible region to determine his/her preferred solution.

Beside the application of the general interactive multiobjective procedures to the GP, other algorithms have been developed specifically for the GP technique. Among these methods are the interactive sequential goal programming procedure of Masud and Hwang (1981) and the augmented goal programming method of Ignizio (1981). Both of these methods produce nondominated solutions.

Summary and Conclusions

This chapter presented some of the basic terminologies of multiple criteria decision making. Then, a classification scheme of MCDM models along with their descriptions were discussed. Finally, a brief review of goal programming models including formulation, various solution procedures, integer goal programming, and interactive GP was presented.

This research will incorporate an interactive integer/ zero-one goal programming procedure for the analysis of stochastic, multiple objectives location-allocation problems. The flexibility of GP in solving a variety of real world applications, its ease of use and understanding, the ability to analyze the performance of the system under different goal levels and/or goal priority structure, and the presence of relatively efficient algorithms are the criteria used for the selection of this technique.

The next chapter presents the development of two models along with their solution procedures for the stochastic multiobjective location-allocation problems.

CHAPTER IV

MODEL DEVELOPMENT AND SOLUTION METHODOLOGY

Introduction

This chapter contains the formulations and the solution procedures of the stochastic multiobjective facility location-allocation problem (SMOLAP). There are two main characteristics associated with this research problem, multiple objectives and stochastic demand. From previous chapter, application of interactive integer/zero-one goal programming appears to be an appropriate approach to deal with conflicting multiple objectives. On the other hand, to account for probabilistic uncertainty in the demand, we will explore two different techniques: chance-constrained programming and stochastic or two-stage programming. Both these methods deal with the stochastic nature of the problem by converting the probabilistic model into an equivalent deterministic case. The chance-constrained approach is based on satisfying an a priori "service level" while the latter method, stochastic programming, is based on expected value analysis.

This chapter begins with a presentation of the assumptions and notations employed throughout the development

of this research problem. Then a discussion of chance-constrained and stochastic programming is presented. Mathematical derivations of two distributions, normal and uniform, for the case of the chance-constrained and the stochastic programming formulations are demonstrated next. Finally, the mathematical models for the research problem, and their solution algorithms are presented.

Model Assumptions

The following assumptions are made and used in developing the mathematical models.

1. The probability distribution function of demand for each customer is known and is assumed to be normally or uniformly distributed. These distribution functions are shown to be appropriate representation of variability of demands in reality; Gonzalez-Valenzuela (1975), Jucker and Carlson (1976), Rasaratnam (1984).
2. The probabilistic demands are independent and are the only source of randomness introduced into the problem.
3. For the stochastic programming model, all decisions including the allocations are made prior to the time when actual demands becoming known.
4. Production costs and variable plant operating costs are a linear function of the amount produced at each plant.

5. The variable transportation cost is a linear function of the amount transferred between a plant and a demand center.
6. All costs are deterministic and do not include any economies-of-scale effect.
7. The potential plant sites are a priori known. Often, these locations are selected from a larger set of possible sites by a multi-criteria decision making technique.
8. The locations of demand centers (markets) are known. Also, demand from a region is assumed to be concentrated at a point representing concentration of demands.
9. There is no elasticity of demand. Specifically, we assume demand is not significantly influenced by any planning decisions such as plant configurations (distance), product flow, and price. Therefore, assuming there is no relationship between these factors and quantity demanded, it is not necessary to consider the revenue generation aspect of the problem in our models.
10. There are no interactions among new plants. That is no transfer of products are allowed between plants.
11. The products or services are homogeneous. Therefore, a single product model is appropriate.
12. The cost matrix (distance) representing the cost

of assigning demand centers to potential plant sites is known.

13. There is a fixed cost associated with establishing a plant at a potential site. This cost is assumed to be independent of the plant throughput. For this study, this cost is the sum of the amortized construction cost and fixed operating cost over the life of the plant.
14. Each demand center may be supplied from more than one source.
15. Assuming uncertainties in demands will affect the decision procedures for each of the two models differently. For the chance-constrained model we assume that the service level for each market is given a priori. This will establish the minimum probability of achieving each demand constraint. For the stochastic programming model, instead of including the uncertainties in the constraints, we will let the uncertainties appear as an objective function. In this case we will assume a linear overage and underage cost corresponding with oversupplying and undersupplying of each demand center respectively.

Notations

The following variables and definitions are employed to describe the mathematical formulations of the proposed models

throughout this research:

i = plant index, $i=1,2,\ldots,m$.

m = number of potential plant sites in the system.

j = demand center index, $j=1,2,\ldots,n$.

n = number of demand centers in the system.

X_{ij} = units of product transported from plant i to demand center j.

b_j = total units of product received at demand center j.

c_{ij} = total variable cost of production, distribution and operation for supplying one unit of product from plant i to demand center j.

Y_i = a 0-1 binary variable; $y_i=1$ if plant i is established, $y_i=0$ otherwise.

F_i = fixed cost per time period of opening and operating a plant at site i.

q_j = random demand at demand center j.

$f_q(q_j)$ = probability density function for the demand at destination j.

$F_q(q_j)$ = cumulative distribution function for the demand at destination j. It is the probability that total demand at center j takes on a value lass than or equal to q_j.

$f_U(.)$ = probability density function of the unit normal distribution.

$F_U(.)$ = probability distribution function of the standard normal distribution.

$P(.)$ = probability distribution function of a given distribution.

μ_{qj} = mean of random demand at destination j for a normally distributed demand.

σ_{qj} = standard deviation of random demand at destination j for a normally distributed demand.

$A_{i,max}$ = maximum allowed capacity of a plant at site i.

$A_{i,min}$ = minimum allowed capacity of a plant at site i.

UB_j = upper bound for demand at demand center j for a uniformly distributed demand.

LB_j = lower bound for demand at demand center j for a uniformly distributed demand.

α_j = probability or risk of shortage at demand center j.

$1-\alpha_j$ = minimum service level required at demand center j. This is the probability of not undersupplying demand center j in the chance-constrained model.

d_k^- = under-achievement of goals or constraints associated with the *kth* equation.

d_k^+ = over-achievement of goals or constraints associated with the *kth* equation.

K = number of priority levels in the achievement function.

$\bar{a}_L$ = the vector of goal achievements at various priority levels (P_1, P_2,..., P_k) at iteration L.

o_j = unit cost of oversupplying demand center j, for stochastic programming model.

u_j = unit cost of undersupplying demand center j, for stochastic programming model.

M = a sufficiently large positive number.

B = maximum budget allowed for opening new plants.

δ = the minimum specified resolution between current and previous objective value.

α = acceleration factor used for extending the step size, $\alpha \geq 1.0$.

β = contraction factor used for reducing the step size, $0.0 \leq \beta \leq 1.0$.

$\bar{q}$ = best allocation pattern at previous solution.

$\bar{q}_T$ = the allocation pattern after exploratory moves.

$\bar{d}$ = current best allocation pattern.

step[i] = step size along coordinate direction i, i=1,...,(m)(n).

ψ = maximum number of times allowed for evaluating the objective function.

ϕ = maximum number of times allowed for reducing the step size.

Z = an objective function.

z = a random variable.

Chance-Constrained Programming

Chance-constrained programming is a technique designed to deal with stochastic problems. It was first introduced by Charnes and Cooper (1963). This technique requires that for each stochastic constraint we meet a specified "service level". For instance, for this research problem we require demands to be satisfied with some minimum probability. In order to introduce the chance-constrained programming model, first consider the following deterministic linear programming problem model.

$$\text{Minimize} \quad Z = \sum_{i=1}^{m} c_i x_i \tag{4.1}$$

subject to:

$$\sum_{i=1}^{m} a_{ij} x_i \geq q_j \quad ; \quad j = 1,2,\ldots,n. \tag{4.2}$$

Equation (4.2) also includes the nonnegativity constraints. In the general case, the uncertainties are present in all of the parameters, a's, q's, and c's. However, for this presentation, we assume all a's and c's are

fixed and that q's are the only source of random variables present. Furthermore, assume that the q's are independently distributed with the following distribution function:

$$F_q(z) = P\ (q_j \leq z). \tag{4.3}$$

Then the chance-constrained programming model of (4.1-4.2) is defined as:

$$\text{Minimize} \quad Z = \sum_{i=1}^{m} c_i x_i \tag{4.4}$$

subject to:

$$P\left[\sum_{i=1}^{m} a_{ij} x_i \geq q_j\right] \geq 1-\alpha_j \ ; \quad j=1,\ldots,n. \tag{4.5}$$

The probability statements in Equation (4.5) are denoted as chance-constrained inequalities. Where, $0 < \alpha_j \leq 1$ is the probability or risk of not achieving the *jth* constraint. And $1-\alpha_j$ is the service level or the minimum probability of realizing the *jth* constraint. Next, recalling the definition of the cumulative distribution function and rewriting one of the constraints in Equation (4.5) we obtain:

$$\begin{aligned} P\left[q_j \leq \sum_{i=1}^{m} a_{ij} x_i\right] &= \int_{-\infty}^{b_j} f(q_j)\,dq_j \\ &= F_q\left(\sum_{i=1}^{m} a_{ij} x_i\right) \geq 1-\alpha_j. \end{aligned} \tag{4.6}$$

where $b_j = \sum_{i=1}^{m} a_{ij} x_i$. Inverting the probability distribution function $F_q(.)$, constraint (4.6) becomes:

$$\sum_{i=1}^{m} a_{ij} x_i \geq F_q^{-1}(1-\alpha_j). \tag{4.7}$$

Constraint (4.7) is the deterministic equivalent of a probabilistic constraint in Equation (4.5). As a result, the deterministic equivalent of the original chance-constrained programming problem defined by (4.4-4.5) can be written as follows:

$$\text{Minimize} \quad Z = \sum_{i=1}^{m} c_i x_i \tag{4.8}$$

subject to:

$$\sum_{i=1}^{m} a_{ij} x_i \geq F_q^{-1}(1-\alpha_j) \; ; \quad j=1,2,\ldots,n. \tag{4.9}$$

Other classes of the chance-constrained programming models, specifically, those with variations in the form of the objective function, or the ones containing randomness in other parameters, a's and c's, have been examined by Charnes and Cooper (1963).

Normally Distributed Demands

Derivations of an equivalent deterministic for a chance-constrained model when the right hand side is normally distributed is presented by Taha (1982). Using the notations defined previously, the chance-constrained equations for satisfying the random demand constraints can be stated as:

$$P\left[\sum_{i=1}^{m} x_{ij} \geq q_j\right] \geq 1-\alpha_j \; ; \quad j=1,2,\ldots,n. \tag{4.10}$$

Where demand q_j is normally distributed with mean μ_{qj} and variance σ^2_{qj}. Following the procedures for determining

deterministic equivalents, inequality (4.10) can be transformed to:

$$\sum_{i=1}^{m} x_{ij} = b_j \geq F_q^{-1}(1-\alpha_j) \; ; \quad j=1,2,\ldots,n. \qquad (4.11)$$

Now, the inverse cumulative function, Z_j, for a unit normal distribution with a given probability value, $(1-\alpha_j)$, can be obtained from standard normal tables or by approximation formulas. An approximation formula for calculating Z_j by Hastings (1955) is provided in Appendix B. Next, given the Z_j values and the expression for standardizing a normal distribution, the right hand side of inequality (4.11) can be given as follows:

$$F_q^{-1}(1-\alpha_j) = Z_j \, \sigma_{qj} + \mu_{qj} \; ; \quad j=1,2,\ldots,n. \qquad (4.12)$$

Therefore equation (4.11) can be written as:

$$\sum_{i=1}^{m} x_{ij} \geq Z_j \, \sigma_{qj} + \mu_{qj} \; ; \quad j=1,2,\ldots,n. \qquad (4.13)$$

where all the variables on the right hand side are known. Inequality (4.13) is the deterministic equivalent of Equation (4.10) for a normally distributed demand.

Uniformly Distributed Demands

Given the demand at demand center j, q_j, is uniformly distributed between LB_j and UB_j, its distribution function can be represented as:

$$1-\alpha_j = F_q(q_j=x) = \int_{-\infty}^{x} \frac{1}{UB_j-LB_j}\, dq_j = \frac{x-LB_j}{UB_j-LB_j}. \qquad (4.14)$$

Thus, its inverse cumulative function is given by:

$$x = F_q^{-1}(1-\alpha_j) = (UB_j-LB_j)(1-\alpha_j) + LB_j. \qquad (4.15)$$

Substituting (4.15) into Equation (4.11) we get:

$$\sum_{i=1}^{m} x_{ij} \geq (UB_j-LB_j)(1-\alpha_j) + LB_j \; ; \; j=1,2,\ldots,n. \qquad (4.16)$$

Inequality (4.16) is the deterministic equivalent of Equation (4.10) for a uniformly distributed demand.

Stochastic Programming Model

In this section an alternative method which deals with randomness by incorporating it into the objective function is considered. In this approach we will consider a penalty cost whenever the supply to a destination does not match the actual demand at that center. As stated previously, we assume actual demands are realized after allocation decisions are made. Also, we assume o_j and u_j are per unit cost of oversupplying and undersupplying of market j, respectively. The above overage and underage costs have an opposite effect on the supply of products to each demand center. The shortage or underage cost tends to increase the product allocation to a demand center while on the contrary the surplus or overage cost tends to decrease this allocation. As such, it is necessary to establish a balance between these two costs. Given that these costs depend on random demands,

our objective is to minimize their expected value. All the derivations in this section follows the inventory models for style goods and perishable items as described by Silver and Peterson (1985). Now, assume b_j units of a product are allocated to demand center j and a demand of q_j units occur at this center, then the cost realized is:

$$C(b_j, q_j) = \begin{cases} o_j(b_j - q_j) & \text{if } q_j \leq b_j \quad (4.17) \\ u_j(q_j - b_j) & \text{if } q_j > b_j \quad (4.18) \end{cases}$$

where

$$b_j = \sum_{i=1}^{m} X_{ij} \ . \qquad (4.19)$$

Equations (4.17) and (4.18) represent cost of oversupplying and undersupplying, respectively. Now, the expected value of the cost, as a function of demand at demand center j, is given by:

$$E\left[C(b_j,q_j)\right] = \int_{-\infty}^{\infty} C\ (b_j,q_j)\ f_q(q_j)\ dq_j. \qquad (4.20)$$

Next, substituting from Equations (4.17) and (4.18), we obtain:

$$E\left[C(b_j,q_j)\right] = o_j \int_{-\infty}^{b_j} (b_j - q_j)\ f_q(q_j)\ dq_j + \\ + \ u_j \int_{b_j}^{\infty} (q_j - b_j)\ f_q(q_j)\ dq_j. \qquad (4.21)$$

Now given that demand cannot be less than zero, Equation (4.21) can be written as:

$$E\left[C(b_j,q_j)\right] = o_j b_j \int_0^{b_j} f_q(q_j)\, dq_j - o_j \int_0^{b_j} q_j\, f_q(q_j)\, dq_j + \\ + u_j \int_{b_j}^{\infty} q_j\, f_q(q_j)\, dq_j - u_j b_j \int_{b_j}^{\infty} f_q(q_j)\, dq_j. \tag{4.22}$$

The Equation (4.22) represents a nonlinear cost function in the supply quantities, b_j's, and therefore in the allocation variables, x_{ij}'s. To show that this function is convex and therefore has a global minimum we have to prove that

$$d^2\, E\left[C(b_j,q_j)\right] / db_j^2 > 0. \tag{4.23}$$

Recalling Leibniz's rule, given the following function:

$$G(x) = \int_{h_1(x)}^{h_2(x)} F(x,y)\, dy. \tag{4.24}$$

Its derivative is given by:

$$dG(x)/dx = \int_{h_1(x)}^{h_2(x)} \partial F(x,y)/\partial x\, dy + F(x,h_2)\, dh_2(x)/dx - \\ - F(x,h_1)\, dh_1(x)/dx. \tag{4.25}$$

Therefore, applying (4.25) to (4.22) we get:

$$d\, E\left[C(b_j,q_j)\right] / db_j = o_j \int_0^{b_j} f_q(q_j)\, dq_j + o_j b_j \left[0+f_q(b_j)-0\right] - \\ - o_j \left[0+b_j f_q(b_j)-0\right] + u_j \left[0+0-b_j f_q(b_j)\right] - \\ - u_j \int_{b_j}^{\infty} f_q(q_j)\, dq_j - u_j b_j \left[0+0-f_q(b_j)\right]. \tag{4.26}$$

After some simplification we get:

$$d\,E\left[C(b_j,q_j)\right] / db_j = o_j\int_0^{b_j} f_q(q_j)\,dq_j - u_j\int_{b_j}^{\infty} f_q(q_j)\,dq_j \qquad (4.27)$$

or

$$= o_j P_q < (b_j) - u_j\,[1 - P_q < (b_j)]. \qquad (4.28)$$

Now setting the first derivative to zero results in the unconstrained minimum of b_j:

$$P_q < (b_j) = \frac{u_j}{o_j + u_j}. \qquad (4.29)$$

Applying the Equation (4.25) one more time, the second derivative is as follows:

$$d^2E\left[C(b_j,q_j)\right] / db_j^2 = o_j\left[0+f_q(b_j)-0\right] - u_j\left[0+0-f_q(b_j)\right] =$$

$$= (o_j+u_j)\,f_q(b_j) > 0. \qquad (4.30)$$

This proves that the cost function in Equation (4.22) is convex and therefore has a global minimum. The result of Equation (4.22) is applicable for any distribution of demands. However in the following sections we will look at the simplifications for special cases of normally and uniformly distributed demands.

Normally Distributed Demands

The simplification of Equation (4.22) for the case of normally distributed demand is shown next. Suppose the demand at demand center j, q_j, is normally distributed with a mean of μ_{qj} and variance of σ^2_{qj}. Now, define

$$k_j = \frac{b_j - \mu_{qj}}{\sigma_{qj}} \tag{4.31}$$

and

$$U_j = \frac{q_j - \mu_{qj}}{\sigma_{qj}}. \tag{4.32}$$

From Equation (4.31), we also get:

$$b_j = \mu_{qj} + k_j \sigma_{qj}. \tag{4.33}$$

Recalling the transformation of a normal distribution to standard (unit) normal form, we can write:

$$\text{Prob } (q_j \geq b_j) = \text{Prob } (U_j \geq k_j) = P_{U\geq}(k_j) \tag{4.34}$$

or

$$\int_{b_j}^{\infty} f_q(q_j)\, dq_j = \int_{k_j}^{\infty} f_U(U_j)\, dU_j = 1 - F_U(k_j) = P_{U\geq}(k_j). \tag{4.35}$$

where U_j is a normally distributed variable with mean of zero and variance one; $U_j \sim N(0,1)$. And $P_{U\geq}(k_j)$ is the probability that a unit normal variable takes on a value of k_j or larger. Now, since the chance of a negative value for q_j is zero, that is no negative demand is allowed, we can write:

$$\begin{aligned}\int_0^{b_j} f_q(q_j)\, dq_j &= \int_{-\infty}^{\infty} f_q(q_j)\, dq_j - \int_{-\infty}^{0} f_q(q_j)\, dq_j - \int_{b_j}^{\infty} f_q(q_j)\, dq_j \\ &= 1 - 0 - P(q_j \geq b_j) \\ &= 1 - P_{U\geq}(k_j). \end{aligned} \tag{4.36}$$

Similarly;

$$\int_0^{b_j} q_j f_q(q_j)\, dq_j = \int_{-\infty}^{\infty} q_j f_q(q_j)\, dq_j - \int_{-\infty}^{0} q_j f_q(q_j)\, dq_j - \int_{b_j}^{\infty} q_j f_q(q_j)\, dq_j$$

$$= \mu_{qj} - \int_{b_j}^{\infty} q_j f_q(q_j)\, dq_j. \tag{4.37}$$

Next, let

$$A = \int_{b_j}^{\infty} q_j f_q(q_j)\, dq_j = \int_{b_j}^{\infty} (q_j - b_j) f_q(q_j)\, dq_j + \int_{b_j}^{\infty} b_j f_q(q_j)\, dq_j. \tag{4.38}$$

Now, substituting from Equations (4.33) and (4.35) and writing the expression for $f_q(q_j)$, Equation (4.38) becomes:

$$A = \int_{q_j=\mu_{qj}+k_j\sigma_{qj}}^{q_j=\infty} (q_j - \mu_{qj} - k_j\sigma_{qj}) \frac{1}{\sigma_{qj}\sqrt{2\pi}} \text{Exp}\left[-(q_j-\mu_{qj})^2/2\sigma_{qj}^2\right] dq_j + b_j P_{U\geq}(k_j). \tag{4.39}$$

Next, from Equation (4.32) we get:

$$(q_j - \mu_{qj}) = U_j\sigma_{qj} \tag{4.40}$$

and $$\frac{dU_j}{dq_j} = \frac{1}{\sigma_{qj}} \quad \text{or} \quad dq_j = \sigma_{qj} dU_j. \tag{4.41}$$

Also when the lower limit of the integral in (4.39) is $q_j = \mu_{qj} + k_j\sigma_{qj}$, from (4.40) we get $U_j = K_j$, and when the upper limit is $q_j = \infty$ we obtain $U_j = \infty$. Now, substituting from Equations (4.40) and (4.41) and these limits into Equation (4.39) and after some simplification we get:

$$A = \sigma_{qj} \int_{U_j=k_j}^{U_j=\infty} (U_j - k_j) \frac{1}{\sqrt{2\pi}} \text{Exp}\left[-U_j^2/2\right] dU_j + b_j P_{U\geq}(k_j) \tag{4.42}$$

or

$$A = \sigma_{qj}\int_{k_j}^{\infty}(U_j - k_j)\ f_U(U_j)\ dU_j + b_j\ P_U{\geq}(k_j). \qquad (4.43)$$

However, a special property of the unit normal distribution is that;

$$\int_{k_j}^{\infty} U_j\ f_U(U_j)\ dU_j = f_U(k_j). \qquad (4.44)$$

Thus, after separating the terms in the integral of Equation (4.43) and substituting from (4.44) we can write:

$$A = \sigma_{qj}[f_U(k_j) - k_j\ P_U{\geq}\ (k_j)] + b_j\ P_U{\geq}(k_j). \qquad (4.45)$$

Finally, substituting from Equations (4.35), (4.36), (4.37), and (4.45) into Equation (4.22) and after some simplification we obtain:

$$E\left[C(b_j,q_j)\right] = \sigma_{qj}(o_j + u_j)\left[f_U(k_j) - k_j\ P_U{\geq}\ (k_j)\right] + o_j(b_j - \mu_{qj}). \qquad (4.46)$$

For the equation above, values of $f_U(k_j)$ and $P_U{\geq}(k_j)$ for a given value of k_j may be found from the unit normal distribution tables. Also, since the other parameters and cost variables are known, therefore the expected cost can be easily calculated for a given value of k_j or supply quantity, b_j.

Uniformly Distributed Demands

Next we demonstrate the computational simplification of Equation (4.22) for the case of uniformly distributed demand.

Suppose the demand q_j at demand center j is uniformly distributed between LB_j and UB_j. Also, define $r_j = UB_j - LB_j$, then Equation (4.22) can be written as:

$$E\left[C(b_j,q_j)\right] = o_j b_j \int_{LB_j}^{b_j} \frac{1}{r_j}\, dq_j - o_j \int_{LB_j}^{b_j} q_j \frac{1}{r_j}\, dq_j + u_j \int_{b_j}^{UB_j} q_j \frac{1}{r_j}\, dq_j - u_j b_j \int_{b_j}^{UB_j} \frac{1}{r_j}\, dq_j. \quad (4.47)$$

Simplifying Equation (4.47) yields:

$$= \frac{o_j b_j}{r_j}(b_j - LB_j) - \frac{o_j}{2r_j}(b_j^2 - LB_j^2) + \frac{u_j}{2r_j}(UB_j^2 - b_j^2) - \frac{u_j b_j}{r_j}(UB_j - b_j) \quad (4.48)$$

or

$$= \frac{1}{2r_j}\left[(o_j + u_j)\, b_j^2 + (o_j LB_j^2 + u_j UB_j^2) - 2(o_j LB_j + u_j UB_j) b_j\right]. \quad (4.49)$$

As described in Equation (4.22) and is evident from Equation (4.49) this is nonlinear cost function in terms of supply quantities, b_j's.

Mathematical Formulations

Based upon the procedures described for handling random demands two models are presented. Model A and model B refer to formulations employing chance-constrained and stochastic programming respectively. Furthermore, assume the following objectives are to be considered: 1A) meet the random demands with some minimum probability or 1B) minimize the total

expected cost of allocation, 2) maintain production capacity within prespecified limits, 3) satisfy the upper bound on fixed cost, 4) minimize transportation cost, 5) minimize total cost, and 6) satisfy configuration constraints. As discussed previously, given the assumption on elasticity of demands, the revenue generation aspect of the problem has not been included in these objectives. The above objectives are only representative of some of the possible multiple objectives in stochastic LAP which are often in conflict. Nevertheless, additional objectives/constraints may be included, or existing objectives may be removed from the models depending on the actual system under study.

In constructing the mathematical models of the SMOLAP, special care must be taken to prevent trivial solutions. For example to ensure that certain levels of demands are satisfied, the solution space must be bounded by demand constraints appearing as rigid system constraints and/or goal constraints at priority one. This point will be further discussed under the sensitivity analysis section of the succeeding chapter.

The mathematical formulation of model A is presented next.

Model A - Chance-Constrained Goal Programming Formulation

In this section we present the chance-constrained goal programming model of SMOLAP. This model deals with stochastic demand through the chance-constrained programming

technique. The model consists of goal constraints, system constraints, and the achievement function. The mathematical formulation of model A is given next:

Goal Constraints. The goal constraints, as opposed to rigid system constraints, are soft in that they do not restrict the feasible region. The multiple objectives involved in this study can be represented by the following goal constraints. For these goal constraints it is desired to achieve the specified target values as closely as possible.

1. Probabilistic Constraints of Meeting Demand at Demand Centers - using the chance-constrained concept these equations may be stated as:

$$P\left[\sum_{i=1}^{m} X_{ij} \geq q_j\right] \geq 1 - \alpha_j \quad , \quad j = 1,2,\ldots,n. \tag{4.50}$$

Then, the deterministic equivalent of these constraints are:

$$\sum_{i=1}^{m} X_{ij} \geq F_q^{-1}(1 - \alpha_j) \quad , \quad j = 1,2,\ldots,n. \tag{4.51}$$

Converting these to goal constraints form, we have:

$$\sum_{i=1}^{m} X_{ij} + d_k^- - d_k^+ = F_q^{-1}(1 - \alpha_j) \quad , \quad j = 1,2,\ldots,n. \tag{4.52}$$

where k=1,2,...,n for j=1,2,...,n respectively. The negative deviational variables in (4.52) must be minimized in order to satisfy demand with a specified minimum probability bound.

2. Maintain Production Capacity within Prescribed Limits -

The lower and upper bounds on production capacity may be set to represent the efficient operating range of each plant. The upper bound on capacity may be determined based on a number of factors such as environmental considerations (pollution), availability of skilled labor, raw materials, etc. Mathematically, these can be expressed as:

$$A_{i,min} \leq \sum_{j=1}^{n} X_{ij} \leq A_{i,max} \quad , \quad i= 1,2,...,m. \tag{4.53}$$

Once again, separating (4.53) and converting these to goal constraints, we get Equations (4.54) and (4.55).

$$\sum_{j=1}^{n} X_{ij} + d_k^- - d_k^+ = A_{i,max} \quad , \quad i= 1,2, \ldots ,m \tag{4.54}$$

where k=n+1,n+2, ... ,n+m for i=1,2, ... ,m respectively. Also d_k^+ represents the degree of over production at site i which must be minimized:

$$\sum_{j=1}^{n} X_{ij} + d_k^- - d_k^+ = A_{i,min} \quad , \quad i= 1,2, \ldots ,m \tag{4.55}$$

where K=n+m+1,n+m+2,...,n+2m for i=1,2,...,m respectively. And, d_k^- denotes the under-achievement of capacity at site i which must be minimized.

3. Satisfy an Upper Limit on Total Fixed Cost - Another important consideration for the LAPs is the total fixed cost goal. In actual practice there is a limitation on the capital allocated to a project. Let B represents the limit on the total investment budget. Then this can be mathematically expressed as:

$$\sum_{i=1}^{m} F_i Y_i \leq B. \tag{4.56}$$

Converting this to a goal constrained form, we have:

$$\sum_{i=1}^{m} F_i Y_i + d_k^- - d_k^+ = B. \tag{4.57}$$

where k=n+2m+1. So, d_k^- represents the amount of expenditure below B, while d_k^+ indicates this amount above B. Therefore, this goal constraint can be achieved by minimizing its positive deviational variable.

4. Transportation Cost Objective - An important consideration in traditional LAPs is to minimize the total transportation cost of allocating products from plants to demand centers. Mathematically, this can be expressed as:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} X_{ij} \leq 0. \tag{4.58}$$

Converting this to a goal constraint form, we have

$$\sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} X_{ij} + d_k^- - d_k^+ = 0. \tag{4.59}$$

where k=n+2m+2. This goal can be achieved by minimizing the positive deviational variable.

5. Total Cost Objective - A primary objective of the LAPs is the minimization of the total cost. The total cost consists of the total transportation cost between plants and demand centers and the fixed cost of establishing and operating new plants. Mathematically, this objective can be

expressed as:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} X_{ij} + \sum_{i=1}^{m} F_i Y_i \leq 0. \qquad (4.60)$$

Converting this to a goal constraint form, we have:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} X_{ij} + \sum_{i=1}^{m} F_i Y_i + d_k^- - d_k^+ = 0. \qquad (4.61)$$

where k=n+2m+3. Therefore, minimization of total cost can be achieved by setting the target value to zero and minimizing the positive deviational variable.

6. Configuration Constraints - A possible configuration constraint in locational analysis is mutually exclusive sites. Whenever potential sites are geographically close to each other it may be desirable to establish facilities, if any, in only one of these locations. This is justified, for example, if we want to eliminate service overlaps between these locations or to evenly distribute the facilities. Assume locations s and t are mutually exclusive sites, then this relationship can be mathematically expressed as:

$$Y_s + Y_t \leq 1 \quad , \qquad (s \ , \ t) \in m. \qquad (4.62)$$

Converting Equation (4.62) to a goal constraint, we have:

$$Y_s + Y_t + d_k^- - d_k^+ = 1 \ , \quad (s \ , \ t) \in m. \qquad (4.63)$$

where k=2m+n+4. In order to satisfy the above goal constraint we need to minimize the positive deviational variable.

System Constraints. The system constraints are those which must be strictly satisfied before an optimal solution can be realized. Mathematically, for this model, these constraints are stated as follows:

$$\sum_{j=1}^{n} X_{ij} - M\, Y_i \leq 0 \quad , \quad i=1,2,\ldots,m. \tag{4.64}$$

These constraints insure that the shipments to demand centers are made only from open facilities. In choosing a value for M, special care must be taken such that its magnitude does not affect the computational accuracy of the problem. For the case of deterministic models this value must be equal to or greater than the sum of all demands. Next, non-negativity and integrality requirements are given by:

$$X_{ij} \geq 0 \quad , \qquad i = 1,2,\ldots,m \; ; \; j = 1,2,\ldots,n \tag{4.65}$$

$$d_k^- \; , \; d_k^+ \geq 0 \quad , \quad \text{for all } k \tag{4.66}$$

$$Y_i \leq 1 \text{ and } Y_i = 0 \; , \; 1 \quad , \qquad i = 1,2,\ldots,m \tag{4.67}$$

$$d_k^- \; . \; d_k^+ = 0 \quad , \quad \text{for all } k. \tag{4.68}$$

The inequality constraints in Equation (4.67) are used to obtain zero-one solutions through application of a branch and bound routine. From Equation (4.68) the product of the positive and negative deviational variables must be zero, indicating that one can either be above or below the desired goal targets. However, these constraints are automatically satisfied in a linear programming type solution and need not

be considered explicitly.

Model A Achievement Function. The achievement function for this model involves the minimization of appropriate deviational variables according to their preemptive priority weights. Assuming priorities are assigned to the goal constraints in the order they are presented, the model A achievement function is as follows:

$$\begin{aligned}\text{Minimize } Z = {} & P_1(d_1^- + d_2^- + \ldots + d_n^-) + P_2(d_{n+1}^+ + d_{n+2}^+ + \ldots + \\ & d_{n+m}^+ + d_{n+m+1}^- + d_{n+m+2}^- + \ldots + d_{n+2m}^-) + \\ & P_3(d_{n+2m+1}^+) + P_4(d_{n+2m+2}^+) + P_5(d_{n+2m+3}^+) + \\ & P_6(d_{2m+n+4}^+).\end{aligned} \tag{4.69}$$

Another assumption in Equation (4.69) is that the weights for all the deviational variables are equal to one. Priorities P_1, P_2, ..., and P_6 in Equation (4.69) are called preemptive priorities or priority weights. They determine the hierarchy of goals. For this model we also assume that goals with lower indexed priority factors always take priority over goals with higher indexed priority factors. For example, the relationship between P_1 and P_2 is as follow:

$$P_1 > > > P_2. \tag{4.70}$$

This means that lower priority goals are considered only after higher priority goals are either fully achieved or reached to a point beyond which no further improvement is

possible. However the system constraints are given the highest priority ranking, P_0, and must be fully satisfied before any of the above priorities can be considered.

Model B - Stochastic Goal Programming Formulation

This section presents an alternative model for the formulation of SMOLAP. This model uses the stochastic programming method to deal with random demands. The formulation of this model is similar to model A except for the probabilistic goal constraints in (1). For this model, since demand or "service level" for meeting random demands at each destination is not given beforehand, it is not possible to include any explicit restriction to satisfy demands. Instead, goal constraint (1) will be altered to include penalties for oversupplying and undersupplying of the markets. As shown previously, this will result in a nonlinear goal constraint. The mathematical formulation of model B is as follows:

Goal Constraints. Similar to model A the following goal constraints can be formulated for model B:

1. Minimize the Total Expected Cost of Allocation - The expected cost of allocating products from plants to demand centers is the penalty costs of oversupplying and undersupplying the demand centers. Recalling Equation (4.22), this goal can be mathematically expressed as:

$$\sum_{j=1}^{n} \left[o_j b_j \int_0^{b_j} f_q(q_j)\, dq_j - o_j \int_0^{b_j} q_j\, f_q(q_j)\, dq_j + \right.$$
$$\left. + u_j \int_{b_j}^{\infty} q_j\, f_q(q_j)\, dq_j - u_j b_j \int_{b_j}^{\infty} f_q(q_j)\, dq_j \right] \leq 0 \qquad (4.71)$$

where by definition $b_j = \sum_{i=1}^{m} X_{ij}$. Next, converting Equation (4.71) to a goal constraint, we get:

$$\sum_{j=1}^{n} \left[o_j b_j \int_0^{b_j} f_q(q_j)\, dq_j - o_j \int_0^{b_j} q_j\, f_q(q_j)\, dq_j + \right.$$
$$\left. + u_j \int_{b_j}^{\infty} q_j\, f_q(q_j)\, dq_j - u_j b_j \int_{b_j}^{\infty} f_q(q_j)\, dq_j \right] + d_k^- - d_k^+ = 0 \qquad (4.72)$$

where k=1. As an alternative formulation, Equation (4.72) may be divided into n goal constraints with n deviational variables.

2. Maintain Production Capacity within Prescribed Limits -

$$\sum_{j=1}^{n} X_{ij} + d_k^- - d_k^+ = A_{i,max} \quad , \qquad i= 1,2,\ldots,m \qquad (4.73)$$

where k=2,3, ... ,m+1 for i=1,2,...,m respectively.

$$\sum_{j=1}^{n} X_{ij} + d_k^- - d_k^+ = A_{i,min} \quad , \qquad i= 1,2,\ldots,m \qquad (4.74)$$

where K=m+2,m+3, ... ,2m+1 for i=1,2,...,m respectively.

3. Satisfy an Upper Limit on Total Fixed Cost -

$$\sum_{i=1}^{m} F_i Y_i + d_k^- - d_k^+ = B \qquad (4.75)$$

where k=2m+2.

4. Transportation Cost Objective -

$$\sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} X_{ij} + d_k^- - d_k^+ = 0 \qquad (4.76)$$

where k=2m+3.

5. Total Cost Objective -

$$\sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} X_{ij} + \sum_{i=1}^{m} F_i Y_i + d_k^- - d_k^+ = 0 \qquad (4.77)$$

where k=2m+4.

6. Mutually Exclusive Locations -

$$Y_s + Y_t + d_k^- - d_k^+ = 1 \ , \quad (s \ , \ t) \in m \qquad (4.78)$$

where k=2m+5.

System Constraints.

$$\sum_{j=1}^{n} X_{ij} - M Y_i \leq 0 \ , \qquad i=1,2,\ldots,m \qquad (4.79)$$

$$X_{ij} \geq 0 \ , \qquad i=1,2,\ldots,m \ ; \ j=1,2,\ldots,n \qquad (4.80)$$

$$d_k^- \ , \ d_k^+ \geq 0 \ , \quad \text{for all } k \qquad (4.81)$$

$$Y_i \leq 1 \text{ and } Y_i = 0 \ , \ 1 \ , \qquad i=1,2,\ldots,m \qquad (4.82)$$

$$d_k^- \ . \ d_k^+ = 0 \ , \quad \text{for all } k. \qquad (4.83)$$

Model B Achievement Function.

$$\text{Minimize } Z = P_1(d_1^-) + P_2(d_2^+ + d_3^+ + \ldots + d_{m+1}^+ + d_{m+2}^- + d_{m+3}^- + \ldots + d_{2m+1}^-) + P_3(d_{2m+2}^+) +$$

$$+ P_4(d^+_{2m+3}) + P_5(d^+_{2m+4}) + P_6(d^+_{2m+5}) \tag{4.84}$$

The solution procedures for the models developed in this chapter are presented next.

Solution Algorithms

In the preceding sections we proposed two models for the stochastic multiobjective location-allocation problems. As was demonstrated, model A, a multiobjective chance-constrained model, can be easily transformed into an equivalent deterministic model. Therefore, standard multiobjective techniques may be applied to its solution. However, transformation of model B, a multiobjective stochastic model, to its deterministic equivalent is considerably more difficult. As such, we develop different solution algorithms for each model. The following sections present the proposed solution algorithms for solving these models.

Model A Solution Procedure

Once the problem is formulated utilizing the method explained for model A and converted into its equivalent deterministic model, it may be solved through interactive preemptive goal programming. At each solution step the analyst is given the opportunity to change his/her priority structure and/or target goals in order to obtain a better

solution. Additionally, to aid the decision maker toward better solutions, at each iteration, aside from the current achievements, a list of conflicting objectives and their trade-offs will be provided. The steps for this algorithm are outlined below. Also, Figure 4.1 presents the logic flowchart for this algorithm.

Step 0: Formulate the problem as explained for model A. This model is a collection of deterministic goals, probabilistic goals, deterministic system constraints, and the achievement function.

Step 1: Establish the appropriate priority structure, service levels, and target values.

Step 2: Convert the probabilistic goal constraints to their deterministic equivalents.

Step 3: Solve the problem by the modified simplex method of preemptive goal programming.

Step 4: Examine the optimal solution. If it satisfies the integer requirements go to step 5. Otherwise, apply the branch-and-bound method of integer programming.

Step 5: Examine the integer solution. If achievements are satisfactory, go to step 7. Otherwise proceed to step 6.

Step 6: Perform trade-off analysis. Then solicit new priority structure and/or target values from the DM. Reformulate the problem and go to step 3.

Step 7: Terminate.

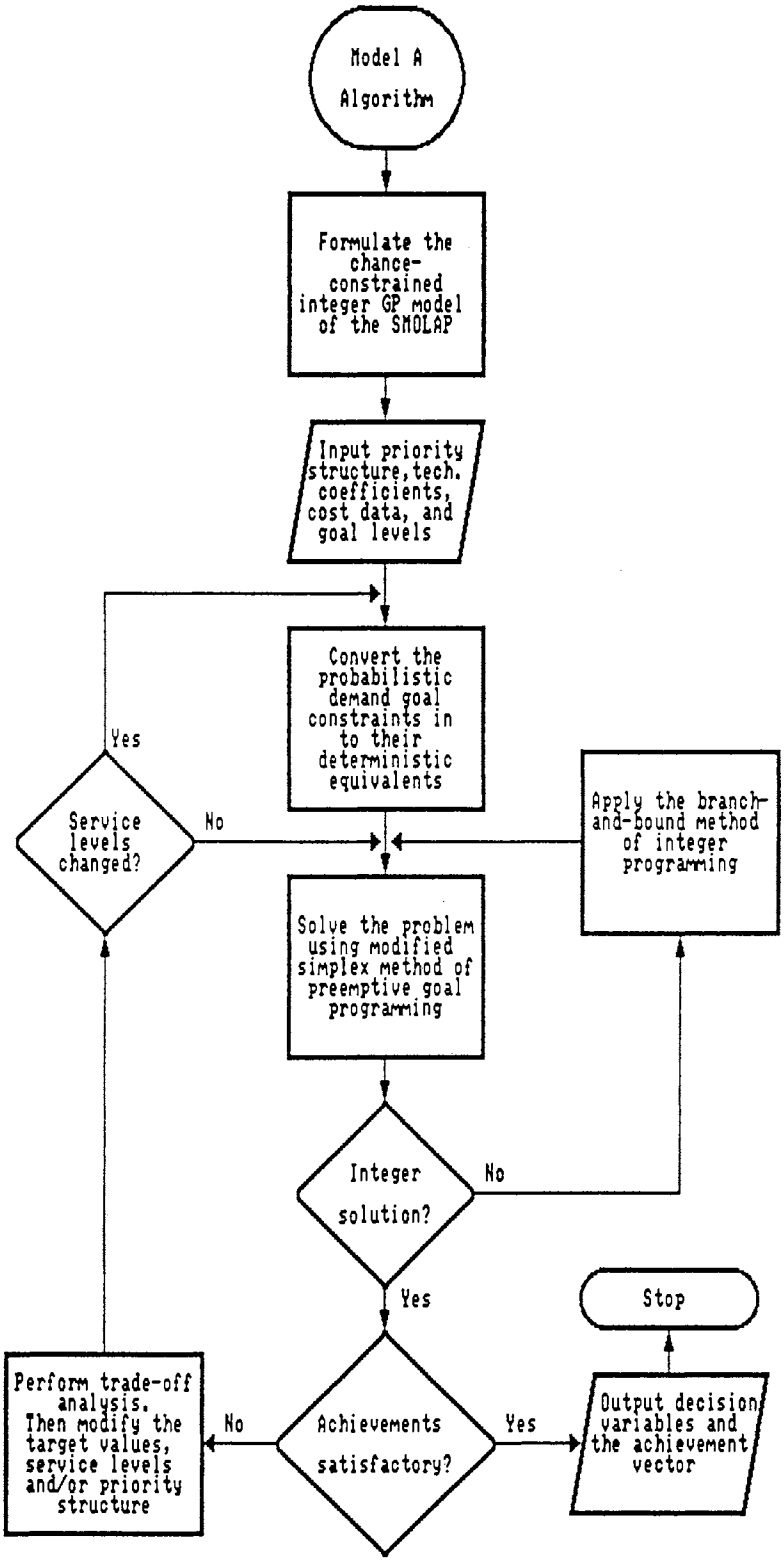


Figure 4.1. Flowchart of the Algorithm for the SMOLAP of Model A

Model B Solution Procedure

The nonlinear goal constraint in model B is the convex quadratic penalty cost function. The two basic approaches for dealing with this nonlinearity are either to replace it with an approximate linear cost function or to treat it using nonlinear techniques. For a fixed location vector and a single objective function, for example cost minimization, this problem reduces to a transportation problem with stochastic demand. Wilson (1972), proposed a linear approximation technique for this nonlinear cost function by establishing upper and lower bounds on the supply quantities to each destination. Then, the nonlinear cost function was replaced with a linear function between these two bounds. Of course the quality of the solution can improve as tighter bounds can be established. However, it is believed that for the case of multiple objectives in this research problem such bounds can not easily be determined.

The formulation of model B indicates the presence of an integer nonlinear goal programming model. By its structure, this is an extremely difficult problem to solve. For a given location vector, this problem reduces to a nonlinear goal programming problem. Ignizio (1976) has discussed an extension of Hooke and Jeeves pattern search technique for the solution of nonlinear goal programming problems. This procedure consists of two major steps: exploratory moves, and pattern move. The algorithm starts from a base point and

does a complete cycle through coordinate directions (exploratory moves). If the new solution vector dominates the solution at the base point, it takes a step in the direction resulting from the net change in the initial point (pattern move). Otherwise, the search continues from the previous point with a contracted step size.

An initial experimentation with the above modified pattern search technique did not prove satisfactory for solving the allocation subproblems. The difficulty arise after fully achieving the first priority goal (nonlinear total expected penalty cost goal). For instance, although after achieving the first priority goal, there are many allocation patterns in a subproblem which result in the minimum expected penalty cost, the modified pattern search algorithm can not distinguish among them. And therefore, for this problem, it always resulted in a dominated solution vector. In general, once the first priority goal is fully achieved, the algorithm, in its present form, is unable to distinguish among available alternatives. Therefore, the lower priority goals are ignored which can lead to dominated solution. This problem may be solved if the solution vector is allowed to go to a dominated point in the process of obtaining a nondominated solution. But, a further modification of different search parameters, such as vector of step sizes and vector of resolutions (errors) between achievement vectors did not overcome this difficulty. Thus,

an alternative method for the solution of model B was investigated.

Goal one in model B, Equation (4.72), is the sum of n expected penalty costs, one for each demand center. Given that the expected penalty cost function at each destination is convex, application of any nonlinear programming technique to the n component of goal one will result in the determination of optimum supply quantities to each demand center. With this introduction, we propose a two-stage algorithm for the solution of model B. In stage one we apply the direct search method of Hooke and Jeeves to find the optimum supply quantities at each demand center. Next, in stage two, we use the results from stage one to construct the deterministic demand goal constraints. These goal constraints along with goals at priorities 2 through 6 represent a deterministic linear integer goal programming problem which is then solved by the modified simplex method. In addition, the branch-and-bound routine along with the appropriate system constraints are used to satisfy the zero-one integer requirements. Figure 4.2 depicts the logic flowchart for the stage 1. The logic flowchart for the stage 2 will be similar to Figure 4.1.

Stage 1 - Pattern Search. In the following steps "point" refers to a set of allocations (X_{ij}'s). And, the objective function is the minimization of total expected penalty cost function.

Step 0: Formulate the total expected penalty cost goal for normal or uniform distribution of demands. This is the objective function to be minimized. Specify convergence criteria (ψ, ϕ), α, β, δ, step sizes, open plants, and an initial set of allocations, $\bar{d}$. Let m_1 represent the number of open plants.

Step 1: Evaluate the total expected penalty cost value at the starting (current) point, $\bar{d}$. Initialize the optimal (upper bound) solution. Also, let the previous point $\bar{q}$ be equal to $\bar{d}$.

Step 2: Search along each coordinate direction by moving step[i] along the *ith* direction for i=1,...,$(m_1)(n)$. Let the new point be $\bar{q}_T$. Determine the objective value at this new point.

Step 3: If the total expected penalty cost at the new point plus δ dominates the upper bound, then update the upper bound. Otherwise, go to step 7.

Step 4: Examine the convergence criteria. If the number of times the total cost is evaluated is greater than or equal to ψ, or if the number of times the step size is reduced is grater than or equal to ϕ, go to step 8. Otherwise perform a pattern move as follows:

$$\bar{d} = \bar{q}_T + \alpha(\bar{q}_T - \bar{q})$$

Set the previous point, $\bar{q}$, equal to $\bar{q}_T$.

Step 5: Determine the total expected penalty cost at the new point, $\bar{d}$. If this solution dominates the upper

bound, update the upper bound and go to step 2. Otherwise, proceed to step 6.

Step 6: Turn on the flag for unsuccessful pattern move (UPM). Go to step 2.

Step 7: The exploratory moves are unsuccessful. Examine the convergence criteria. If the number of times the total cost is evaluated is greater than or equal to ψ, or if the number of times the step size is reduced is grater than or equal to ϕ, go to step 8. Otherwise, reduce the step size by β. If the flag for unsuccessful pattern move is on, then set the new (current) point be equal to the previous point (i.e. $\bar{d}=\bar{q}$). Turn off the flag for unsuccessful pattern move (UPM). Go to step 2.

Step 8: The current upper bound contains the optimal solution (a set of allocations which minimizes the total expected penalty cost function). Use this solution to calculate the optimal supply quantities to each demand center. Terminate.

Application of stage one determines the optimum supply quantities to each demand center. For example, the optimum supply quantities to demand center one, (OSQ1), can be calculated as shown below:

$$OSQ1 = X_{11} + X_{21} + \ldots + X_{m1}.$$

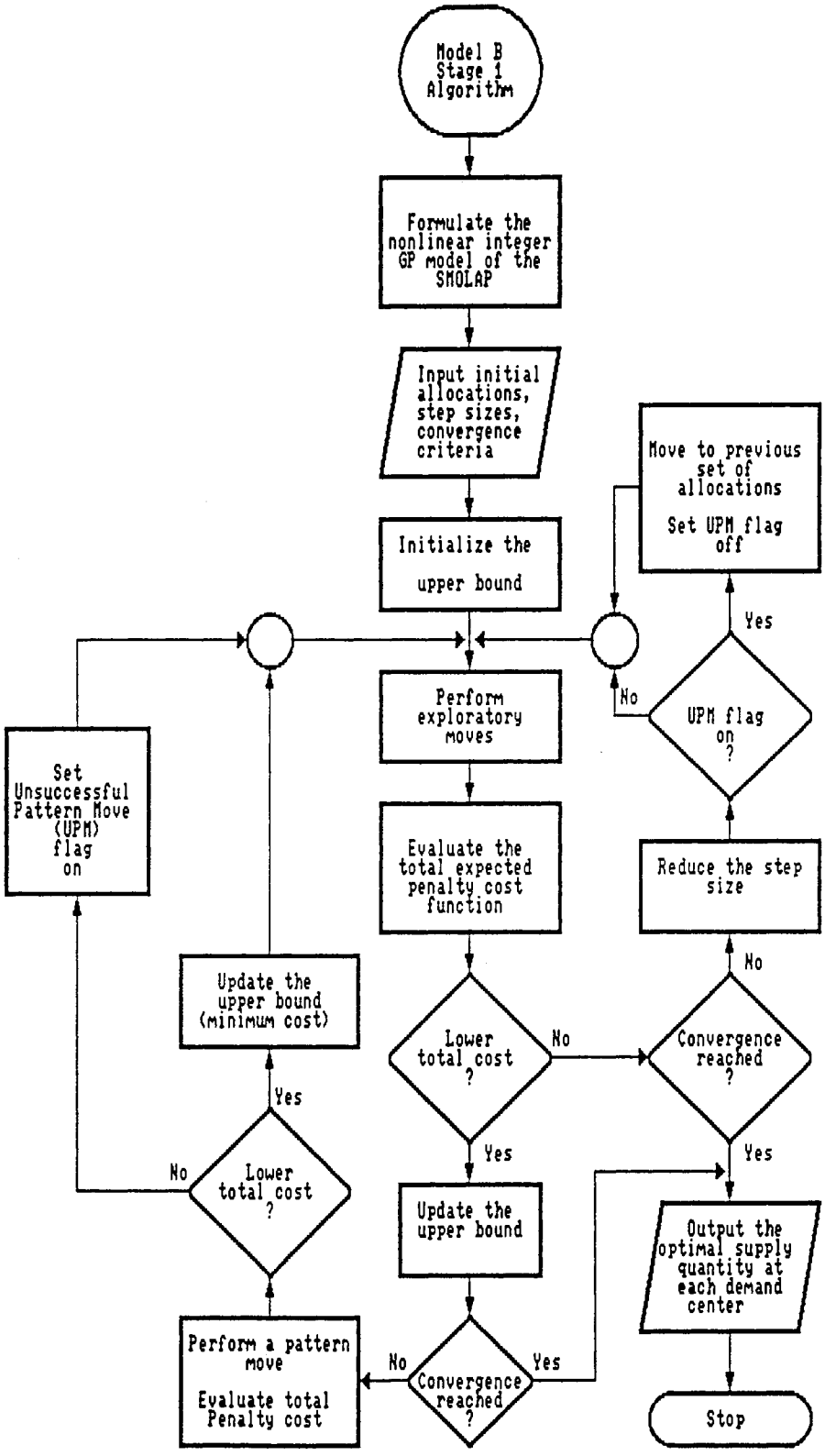


Figure 4.2. Flowchart of the Stage 1 Algorithm for the SMOLAP of Model B

Stage 2 - Optimal Solution. Once the optimal supply quantities to each destination is determined in stage 1, the problem is solved as follows:

Step 0: Formulate the n demand goal constraints as in Equation (4.52). However, set the target values of these goals equal to the deterministic values obtained in stage 1. Formulate other goals and system constraints as presented for model B.

Step 1: Establish the appropriate priority structure and target levels. Note that the target values (optimum supply quantities) for the demand goal constraints are obtained from stage 1 of the algorithm.

Step 2: Solve the problem by the modified simplex method of preemptive goal programming.

Step 3: Examine the optimal solution. If it satisfies the integer requirements go to step 4. Otherwise, apply the branch-and-bound routine of integer programming for an integer solution.

Step 4: Examine the integer solution. If achievements are satisfactory, go to step 6. Otherwise proceed to step 5

Step 5: Perform trade-off analysis. Then solicit new priority structure and/or target values from the DM. Reformulate the problem and go to step 2.

Step 6: Terminate.

Summary

This chapter presented the development of two models along with their solution algorithms for the stochastic multiobjective facility location-allocation problem (SMOLAP). Chance-constrained and stochastic programming were used as alternative procedures to deal with stochastic demand. Also an integer/zero-one goal programming algorithm was used to deal with the integer multiple objective aspect of the problem.

In the next chapter, the above proposed models and their solution procedures will be illustrated through different example problems.

CHAPTER V

VALIDATION, COMPUTATIONAL EXPERIENCE, AND SENSITIVITY ANALYSIS

Introduction

This Chapter addresses the validity, test problems, computational results, and sensitivity analysis of the models developed in this research. To accomplish this task and according to the objectives established previously, an interactive computer program based on the solution procedures of Chapter IV is developed. The program is written in TURBO PASCAL 5.0 for IBM compatible PC's with at least 640 KB of memory.

Next, as a part of validation, descriptions of three problems from the literature and their solutions are discussed.

Validating the Algorithms and Computer Programs

The algorithms and programs developed in this research are extensively tested and validated through variety of sample problems from the literature. In this section three test problems used to validate the algorithms and computer

programs are presented. The test problems reported here include one location model and two location-allocation models. All of three models are of multiple objective nature. However, test problem three has also been solved as a pure (single objective) cost minimization problem.

Test Problem 1 - A Multicriteria Warehouse Location Model

This location problem is presented by Green, Kim, and Lee (1981). The proposed problem is to determine the location of potential warehouses to be served from a single existing source. As such, contrary to multi-source facility location problems, it does not consider the allocation or demand aspect of the problem. The problem formulation contains 12 variables, 10 constraints, and 7 priority levels. The priorities, ranked based on their importance are:

P_1: locate new warehouses where competition saturation is low.

P_2: meet the upper bound on the fixed cost of new warehouses.

P_3: avoid service overlap among warehouse locations by locating them a minimum distance apart.

P_4: satisfy mutual dependency between two specified locations.

P_5: satisfy favored customer service.

P_6: avoid decentralization by locating warehouses within a specified distance from supply source.

P_7: minimize transportation costs from supply source to warehouses.

The input data for this problem is given in Table A.1 in

Appendix A. And Table 5.1 presents a comparison of the solution results for this problem. The solution for this problem was obtained by the integer routine of the program. The results were consistent with the ones reported by the authors and are $X_2 = X_4 = X_{10} = 1$, with all other X_j equal to zero. In addition, all goals were completely achieved except for minimization of transportation cost. Of course this was expected since this goal was set at zero level.

TABLE 5.1

COMPARISON OF RESULTS FOR TEST PROBLEM 1

Green, Kim, Lee (1981)	Abtahi, M.
Locations:	
Y2=1, Y4=1, Y10=1	Y2=1, Y4=1, Y10=1
Goal Under-Achievement:	
P1=0	P1=0
P2=0	P2=0
P3=0	P3=0
P4=0	P4=0
P5=0	P5=0
P6=0	P6=0
P7=790	P7=790

Furthermore, the statistics reported by the program for this problem are listed in Table 5.2. Although execution time is included in this table, its value is greatly influenced by the hardware and software being used and should

not be employed for comparison with other algorithms. This time statistic is mainly valuable for demonstrating the relative time requirements among different problems. All time statistics reported in this research are obtained on a 12 MHZ microcomputer with no math-coprocessor.

TABLE 5.2

TEST PROBLEM 1 - ALGORITHM PERFORMANCE

Total Iterations	Total Nodes Evaluated	Upper Bounds Updated	Execution Time (Sec.)
54	5	1	21.58

Test Problem 2 - Location-Allocation Model I

This problem is presented by Lee and Franz (1979). They studied a location-allocation problem for five potential sites of manufacturing facilities and four distribution centers. The problem contains 6 priority levels, 23 constraints and 25 variables. The variables include both zero-one location variables and the integer assignment variables. The constraints are divided into 13 goal constraints and 10 system constraints. The priorities considered in this problem are given below:

P_1 : meet the product demand of all distribution centers.

P_2: do not exceed the fixed budget for establishing facilities.

P_3: keep an upper limit on the production level at each site.

P_4: satisfy at least 50 units of the demand at distribution center 3 from site 1 or site 2.

P_5: minimize total fixed costs and transportation costs.

P_6: minimize transportation costs.

Table A.2 in Appendix A presents the input data for this problem. A comparison of solution results for this problem is reported in Table 5.3.

TABLE 5.3

COMPARISON OF RESULTS FOR TEST PROBLEM 2

Lee, Franz (1979)	Abtahi, M.
Locations:	
Y4=1, Y5=1	Y4=1, Y5=1
Allocations:	
X41=400	X41=400
X52=300	X52=300
X53=200	X53=200
X54=100	X44=80
	X54=20
Goal Under-Achievement:	
P1=0	P1=0
P2=0	P2=0
P3=0	P3=0
P4=50	P4=50
P5=1,304,500	P5=1,303,300
P6=54,500	P6=53,300

The solution generated by the proposed algorithms gives the same location pattern as the one reported by the authors [Lee and Franz (1979)] with sites 4 and 5 selected for establishing new manufacturing plants. However, the proposed algorithms produce a different pattern for allocation of products among manufacturing plants and distribution centers. The new distribution assignments are: X_{41}=400, X_{52}=300, X_{53}=200, X_{44}=80, and X_{54}=20. This solution dominates the solution presented by the authors which matches priorities 1 through 4 as reported, but results in a higher achievement level for priorities 5 and 6. More precisely, this new solution reduces the transportation and total cost by $1200. The computational results for this problem is given in Table 5.4.

TABLE 5.4

TEST PROBLEM 2 - ALGORITHM PERFORMANCE

Total Iterations	Total Nodes Evaluated	Upper Bounds Updated	Execution Time (Sec.)
892	39	2	1006.89

Test Problem 3 - Location-Allocation Model II

This problem is presented by Lee, Green, and Kim (1981).

They solved a multiple criteria location-allocation problem for 6 potential plant location sites and 4 different distribution centers. The model consists of 8 priority levels, 28 constraints, and 30 variables. 6 of these variables are zero-one integer variables representing the location vector. The constraint set consist of 16 goal and 12 system constraints. The priorities considered are:

P_1: satisfy the demand for all distribution centers.

P_2: insure favored customer service for distribution center 1.

P_3: meet the goal on budget ceiling.

P_4: locate where quality of life is satisfactory.

P_5: maintain a policy of desired expansion by establishing a minimum of three plants.

P_6: keep the production level below the upper bound set by state regulations for air pollution control.

P_7: minimize total costs of opening plants and distribution costs.

P_8: minimize transportation costs.

The input data for this problem is given in Table A.3 in Appendix A. The solution generated by the proposed algorithms is superior to the solution reported by the authors. The optimum solution indicates opening plants at sites 2, 4, and 6 with the following assignments: $X_{21}=50$, $X_{41}=500$, $X_{61}=30$, $X_{22}=420$, $X_{23}=130$, $X_{63}=130$, $X_{64}=150$. This solution, contrary to the solution presented by the authors, satisfies priority 2 completely. It also produces a lower transportation cost. However, total cost is higher due to

selection of site 2 instead of site 3. Table 5.5 presents the comparison of the solution results for this problem. In addition, Table 5.6 gives the computational results for this problem.

TABLE 5.5

COMPARISON OF RESULTS FOR TEST PROBLEM 3

Lee, Green, Kim (1979)	Abtahi, M.[1]	Abtahi, M.[2,3]
Locations:		
Y3=1, Y4=1, Y6=1	Y2=1, Y4=1, Y6=1	Y2=1, Y4=1, Y6=1
Allocations:		
X31=80	X21=80	X21=50
X41=500	X41=500	X41=500
X62=420	X22=420	X61=30
X63=260	X23=100	X22=420
X34=150	X63=160	X23=130
	X64=150	X63=130
		X64=150
Goal Under-Achievement:		
P1=0	P1=0	P1=0
P2=50	P2=0	P2=0
P3=0	P3=0	P3=0
P4=785	P4=775	P4=775
P5=0	P5=0	P5=0
P6=0	P6=0	P6=0
P7=1,841,250	P7=1,990,800	P7=1,990,200
P8=91,250	P8=90,800	P8=90,200

[1] The formulation includes the total cost goal.
[2] The formulation excludes the total cost goal.
[3] This is the nondominated solution.

TABLE 5.6

TEST PROBLEM 3 - ALGORITHM PERFORMANCE

Total Iterations	Total Nodes Evaluated	Upper Bounds Updated	Execution Time (Sec.)
666	27	4	1059.52

It may be noted that the execution time for this problem to obtain the optimal solution on an IBM 370-158 is reported by the authors to be 439.26 seconds.

In experimenting with this problem it was discovered that the solution can be very sensitive to the value of M in the system constraints. While this value needs to be sufficiently large to enforce the constraints of type (4.64), selection of excessively large values for M can significantly affect the computational accuracy of the algorithms.

Another observation made was the need to increase the precision of the real type variables. This reduces the round off errors whenever the number of iterations becomes too large. However, the drawbacks are an increase in CPU time and larger storage (virtual memory) requirements.

Further analysis identified another characteristic of this problem which can affect the computational accuracy and therefore the final decision set produced by the model. As shown in Table A.3, the coefficients of the location

variables for the total cost goal (row 15) are considerably larger than the coefficients of the allocation variables in the problem. This relatively large gap among the technological coefficients will influence the accuracy of the simplex based calculations. To eliminate this potential source of error, the problem was formulated and solved without the total cost goal. Then this goal was evaluated at the current solution. This method resulted in a different allocation pattern and reduced the transportation cost by $600.00, an improvement of 0.66%.

Finally, this problem was formulated and solved as a single objective cost minimization problem as explained by Lee, *et al.* (1981). The solution obtained was similar to the one reported by the authors. The transportation cost is minimized at $50,350. However, the program identified an alternative solution. This new solution is to select location I instead of location II and to replace X_{23}=260 with X_{13}=260.All other location and allocation assignments remain the same as reported in the paper.

An Illustrative Example for Model A

Next, the solution algorithms and sensitivity analysis of SMOLAP for model A is illustrated by a sample problem.

System Description

A manufacturing firm is considering the establishment of

a facility or facilities to service three major demand centers. Through initial analysis the firm has identified four potential sites for plant location which satisfy production requirements such as availability of skilled labor, closeness to suppliers, raw materials, access to transportation, etc. Figure 5.1 depicts a graphical configuration of the proposed plant sites and demand centers for this example problem.

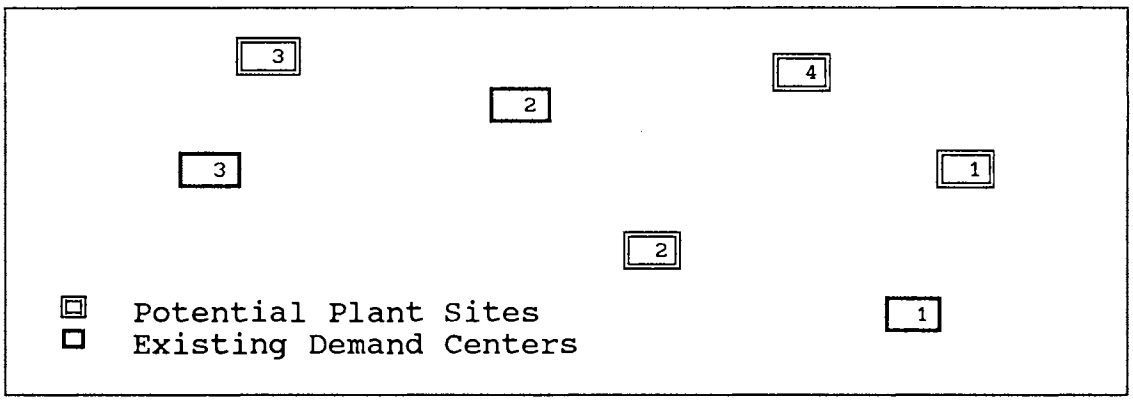


Figure 5.1. Graphical Representation of Potential Plant Sites and Existing Demand Centers

Furthermore, Table 5.7 presents unit distribution cost between potential plants and demand centers, distribution of demand at each destination, annual fixed costs of establishing a facility at each potential site, and the capacity limits at each proposed location.

TABLE 5.7

ALLOCATION COSTS, STOCHASTIC DEMANDS, FIXED COSTS, AND CAPACITIES FOR THE EXAMPLE PROBLEM

	Demand Center 1	Demand Center 2	Demand Center 3	Fixed Costs[1]	Capacity Max	Capacity Min
Site 1	80	90	200	650	500	0
Site 2	60	70	180	800	700	0
Site 3	250	40	30	725	400	0
Site 4	100	20	100	600	650	0
Mean[2]	350	400	500			
S.D.[2]	10	15	20			
Lower[3]	300	350	400			
Upper[3]	400	450	550			

[1] Multiply by 1000.
[2] For normally distributed demands.
[3] For uniformly distributed demands.

Next assume management is considering the following goals/priorities for selecting "ideal" location(s) among potential sites and allocating products to the demand centers.

Priority 1: Satisfy random demand at each destination with the minimum probability of 0.9 (service level=0.9).

Priority 2: Capacity of potential facilities (plants) should not exceed or fall below their planned upper and lower bounds. Note, specifying a minimum capacity for a given site at this priority level, can force a facility to be established at that site.

Priority 3: Limit the total annual fixed costs to $1,350,000.

Priority 4: Minimize transportation cost of allocating products.

Priority 5: Minimize the total cost of location and allocation.

Priority 6: Satisfy the forecasted future growth, by opening at least three facilities.

System Formulation

The chance-constrained goal programming formulation for this problem can be stated as follows:

Goal 1 (Demand):

$$X_{11} + X_{21} + X_{31} + X_{41} + d_1^- + d_1^+ = F_q^{-1}(0.9)$$
$$X_{12} + X_{22} + X_{32} + X_{42} + d_2^- + d_2^+ = F_q^{-1}(0.9)$$
$$X_{13} + X_{23} + X_{33} + X_{43} + d_3^- + d_3^+ = F_q^{-1}(0.9)$$

where distribution of demands are as given in Table 5.7.

Goal 2 (Capacity):

$$X_{11} + X_{12} + X_{13} + d_4^- - d_4^+ = 500$$
$$X_{21} + X_{22} + X_{23} + d_5^- - d_5^+ = 700$$
$$X_{31} + X_{32} + X_{33} + d_6^- - d_6^+ = 400$$
$$X_{41} + X_{42} + X_{43} + d_7^- - d_7^+ = 650$$

Goal 3 (Budget):

$$650\ Y_1 + 800\ Y_2 + 725\ Y_3 + 600\ Y_4 + d_8^- - d_8^+ = 1200$$

Goal 4 (Transportation):

$$80\ X_{11} + 60\ X_{21} + 250\ X_{31} + 100\ X_{41} + 90\ X_{12} +$$
$$70\ X_{22} + 40\ X_{32} + 20\ X_{42} + 200\ X_{13} + 180\ X_{23} +$$
$$30\ X_{33} + 100\ X_{43} + d_9^- - d_9^+ = 0$$

Goal 5 (Total Cost):

$$\sum_{i=}^{4}\sum_{j=1}^{3} c_{ij}x_{ij} + (650000)Y_1 + (800000)Y_2 + (725000)Y_3 + (600000)Y_4 + d_{10}^- - d_{10}^+ \leq 0$$

Goal 6 (Configuration):

$$Y_1 + Y_2 + Y_3 + Y_4 + d_{11}^- - d_{11}^+ = 3$$

and the system constraints are:

$$Y_i \leq 1, \qquad i = 1,2,3,4$$

$$X_{11} + X_{12} + X_{13} - 2500\ Y_1 \leq 0$$

$$X_{21} + X_{22} + X_{23} - 2500\ Y_2 \leq 0$$

$$X_{31} + X_{32} + X_{33} - 2500\ Y_3 \leq 0$$

$$X_{41} + X_{42} + X_{43} - 2500\ Y_4 \leq 0$$

Then, the achievement function can be written as:

$$\text{Minimize } Z = P_1(d_1^- + d_2^- + d_3^-) + P_2(d_4^+ + d_5^+ + d_6^+ + d_7^+) + P_3(d_8^+) + P_4(d_9^+) + P_5(d_{10}^+) + P_6(d_{11}^-).$$

Based on the distribution of demands, two problems are realized. Each problem contains 16 variables, 19 constraints, and 6 priorities. The processing time for each problem is about 230 seconds.

Table 5.8 presents the summary of the results for both normal and uniform distribution of demands.

TABLE 5.8

SUMMARY OF THE RESULTS OF THE MODEL A EXAMPLE PROBLEM FOR NORMAL AND UNIFORM DISTRIBUTION OF DEMANDS

	Normal[a]	Uniform[a]
Open Facilities:	Y2=1, Y4=1	Y1=1, Y3=1, Y4=1
Allocation Assignments:		
	X21=363	X11=390
	X22=296	X33=400
	X42=124	X42=440
	X43=526	X43=135
Priority Order	Underachievements	
1. Demand	0	0
2. Capacity	0	0
3. Budget	50[b]	625[b]
4. Trans.	97580	65500
5. Total	1497580	2040500
6. Config.	1	0

[a] Service Level=0.9.
[b] Multiply by 1000.

From Table 5.8, the optimal solution when the demand distribution is normal and service level is 0.9, is to establish facilities at sites 2 and 4. Also, the solution results in the following assignments: Assign facility at site 2 to demand center 1, assign facility at site 4 to demand center 3, and assign both open facilities to demand center 2. This solution along with specified allocations satisfy the goals at priorities 1 and 2 completely. However, the fixed cost budget goal exceeds by $50,000. Also the

goal on minimum number of facilities (goal 6) is under-achieved by 1. The under-achievement of transportation and total cost is expected since their initial goals were intentionally set very low at zero. This forces the minimum of these goals within there specified priority structure. The results in Table 5.8 can also be verified through inspection of the problem. For instance, converting the chance-constrained demand goals into their equivalent deterministic goals results in a total demand of 1309 units by all demand centers. Now, by inspection, facilities at sites 2 and 4 are the best combination of available facilities which have enough capacity to satisfy these demands and to result in minimum annual fixed costs.

Next, from Table 5.8, for the uniform distribution of demands, facilities are to be established at sites 1, 3, and 4. Once again obtaining the deterministic equivalence of demand goals, the total demand to be satisfied at the 90% service level is 1365 units. Recalling the available capacities in Table 5.7 and realizing the high priority of meeting demands, at least 3 facilities are required to meet this demand. Now, as can be seen in Table 5.8, the model has selected three sites with the lowest annual fixed costs, from the potential sites. This will satisfy priority 3, requiring the minimization of total fixed costs. Next, comparing the two problems, although total demand for uniformly distributed demands is higher than the one for normally distributed demands, the transportation cost is lower by near to 33%.

This can easily be justified from the fact that the extra facility provides more flexibility for distributing the products. As the result, lower transportation cost is expected.

Finally, the location-allocation problem with normal distribution of demands was solved as a single objective fixed charge problem. The problem was modified to minimize the total cost subject to meeting the random demands at the 90% service level. The resulting solution indicates a facility at site 4 with $697,300 total annual cost. This cost is only about 46.5% of the total cost obtained from the earlier multi-objective model in Table 5.8 ($1,497,580). The difference between these two costs ($800,280) can be explained as the amount the management is willing to spend in order to satisfy the multiple goals.

An Illustrative Example for Model B

Consider the example problem presented previously. The data for this problem is given in Table 5.7. To formulate the model B we require additional penalty costs associated with deviations between the supplies (quantities) assigned to the demand centers and the actual demands that occur at these centers. Assume that the per unit cost of undersupplying and oversupplying a demand center are as given in Table 5.9.

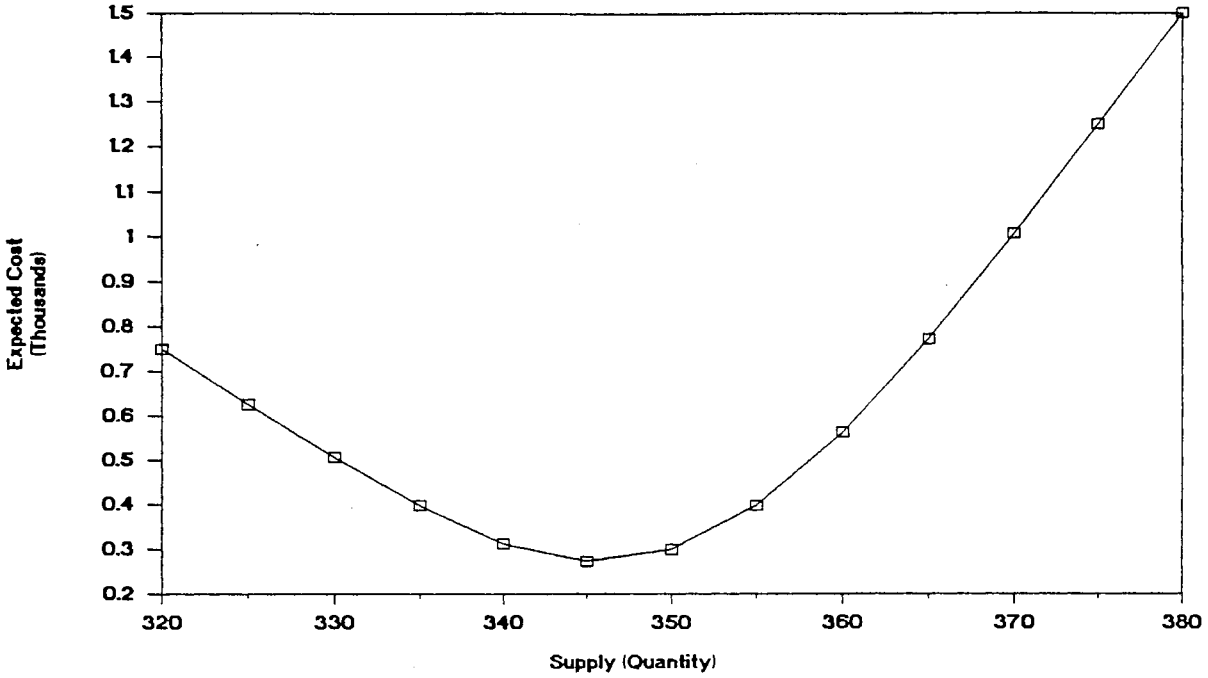
TABLE 5.9

PER UNIT OVERSUPPLY AND UNDERSUPPLY COSTS OF DEMAND CENTERS FOR THE EXAMPLE PROBLEM

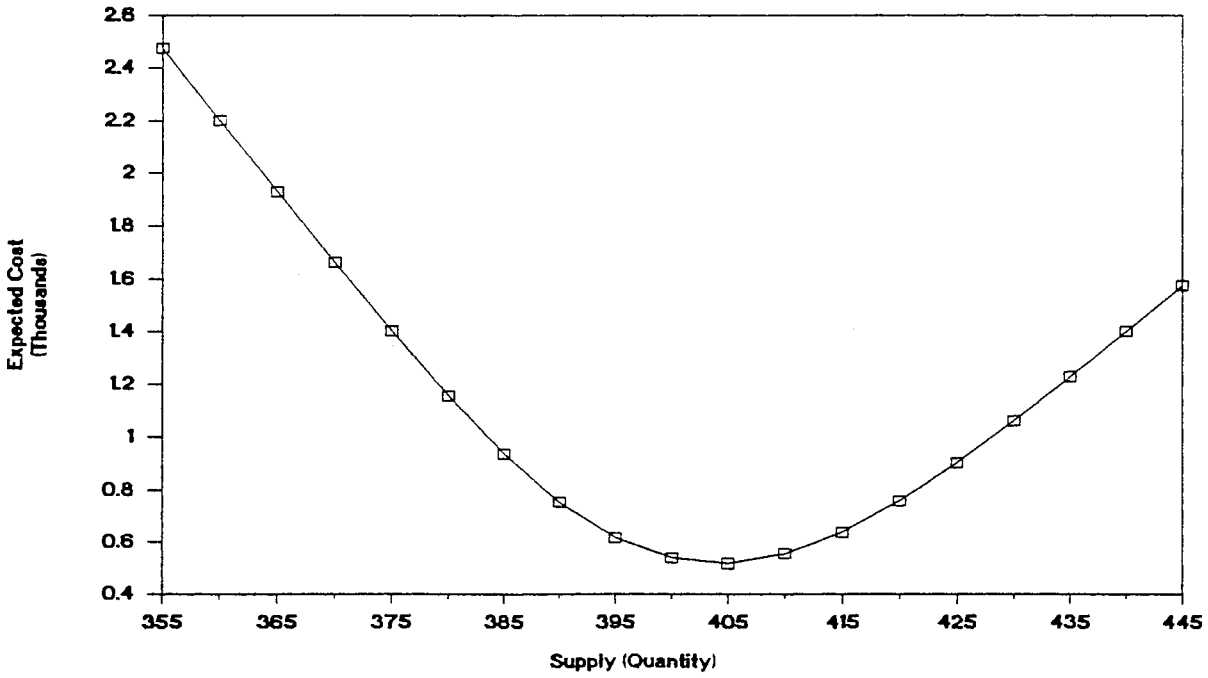
Cost	Demand Center 1	Demand Center 2	Demand Center 3
Oversupply	50	35	45
Undersupply	25	55	50

Figures 5.2 and 5.3 depict the effects of supply quantities on the expected penalty costs at each destination for the case of normally and uniformly distributed demands. As was shown previously and is evident from these graphs, the expected penalty cost at each destination represents a nonlinear convex function. Therefore, model B has a nonlinear goal programming structure.

The stochastic goal programming formulation of this problem is similar to the formulation presented earlier for model A with the exception of goal one. Goal one for model B is to minimize the expected penalty costs at each demand center and is expressed as in Equation (4.72). Breaking Equation (4.72) into n goal constraints and applying the pattern search to the resultant nonlinear goal constraints give the optimum supply quantities to each demand center.

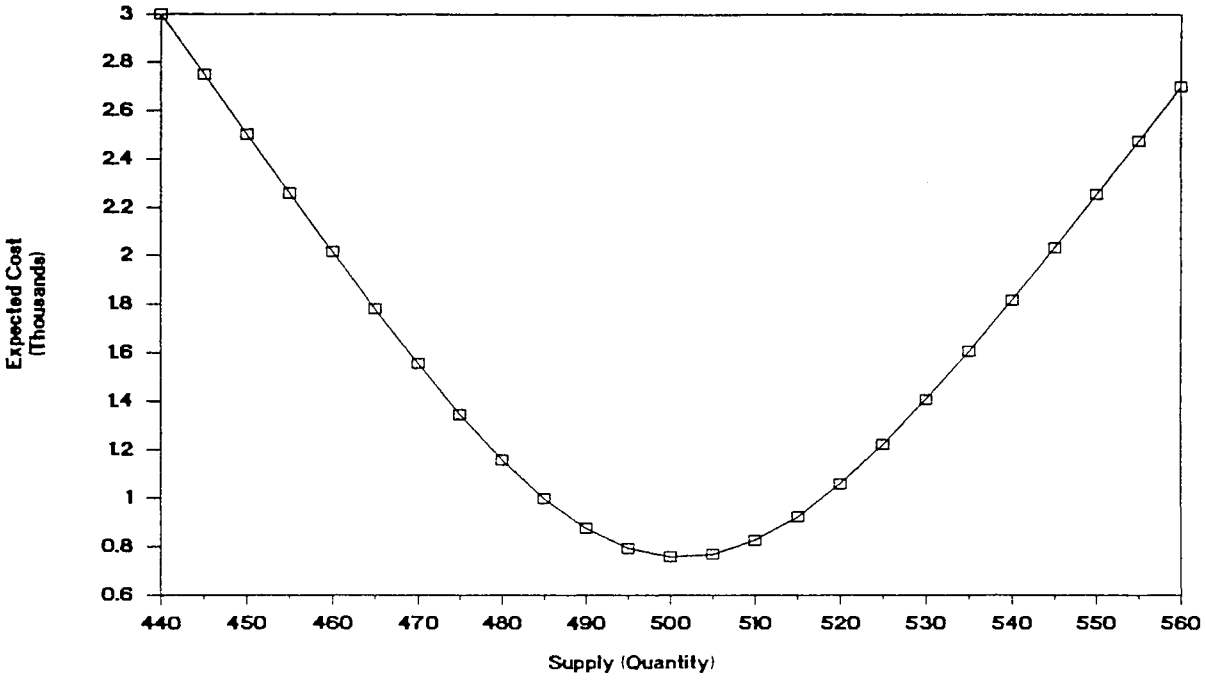


(A) Demand Center 1



(B) Demand Center 2

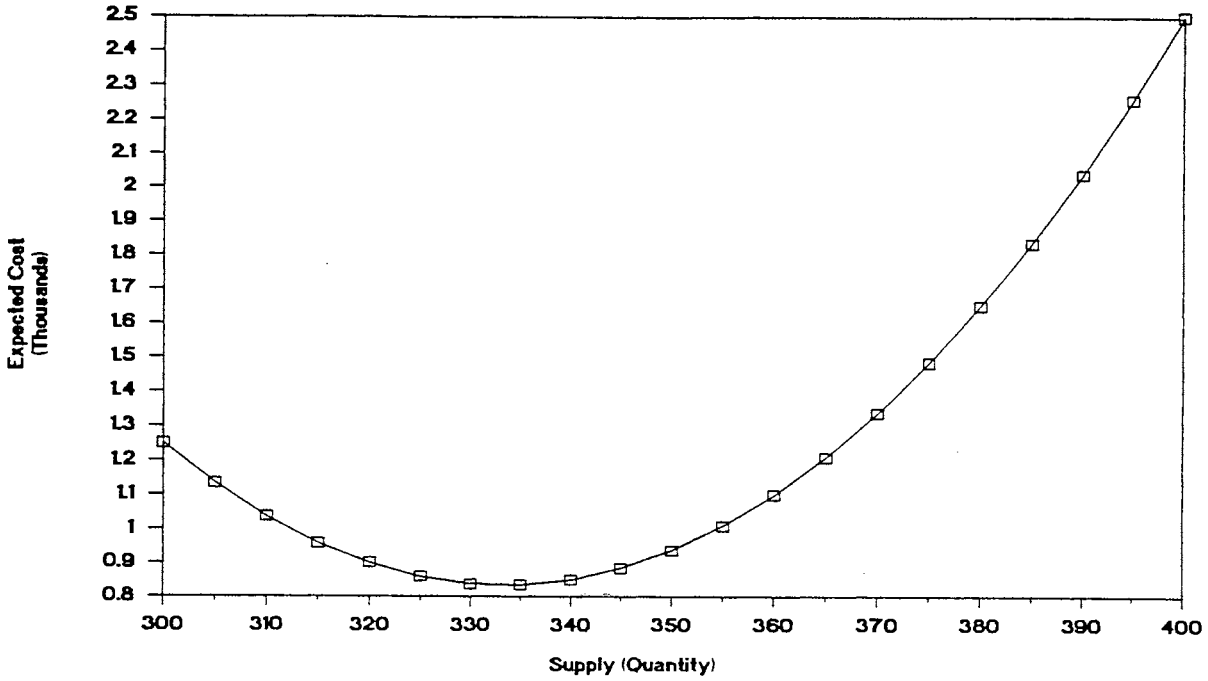
Figure 5.2. Expected Cost of Penalties at Each Destination For the Normally Distributed Demands



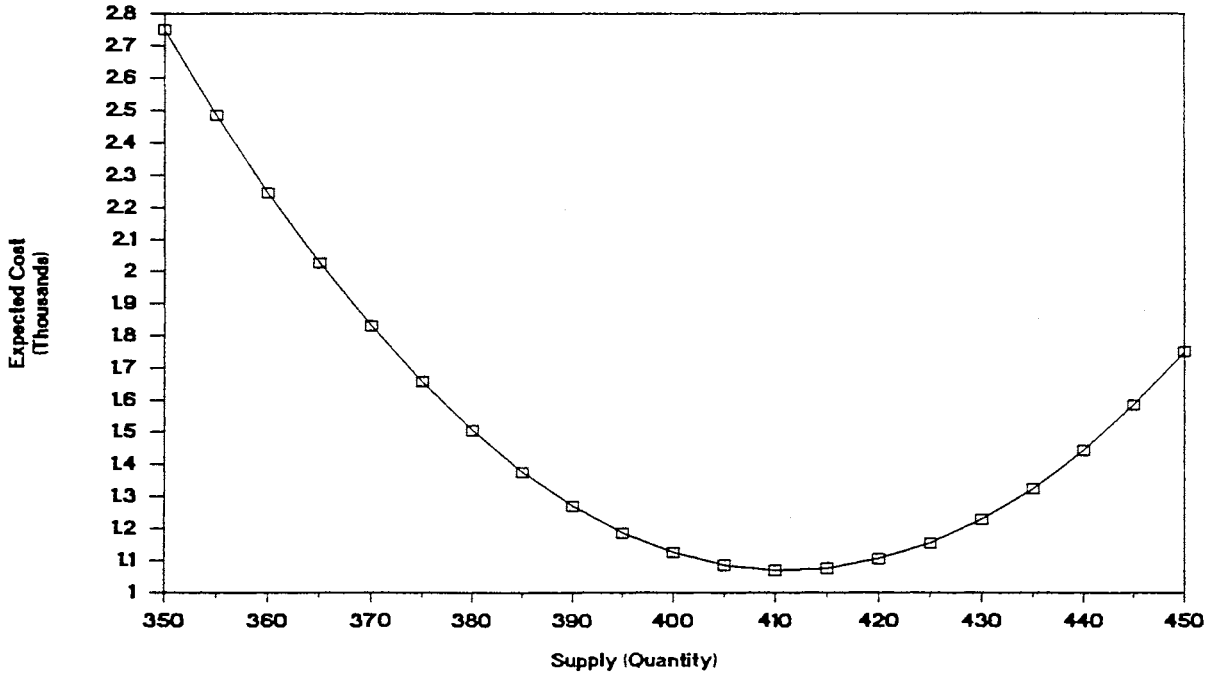
(C) Demand Center 3

Figure 5.2. (Continued)

For the case of normally distributed demands, the optimum supply quantities to demand centers 1, 2, and 3 are 345, 404, and 501 units respectively. And the total expected penalty cost is $1,547.78. These values for the case of uniformly distributed demands are 332, 410, and 477 units with the total expected penalty cost of $3,680.64.

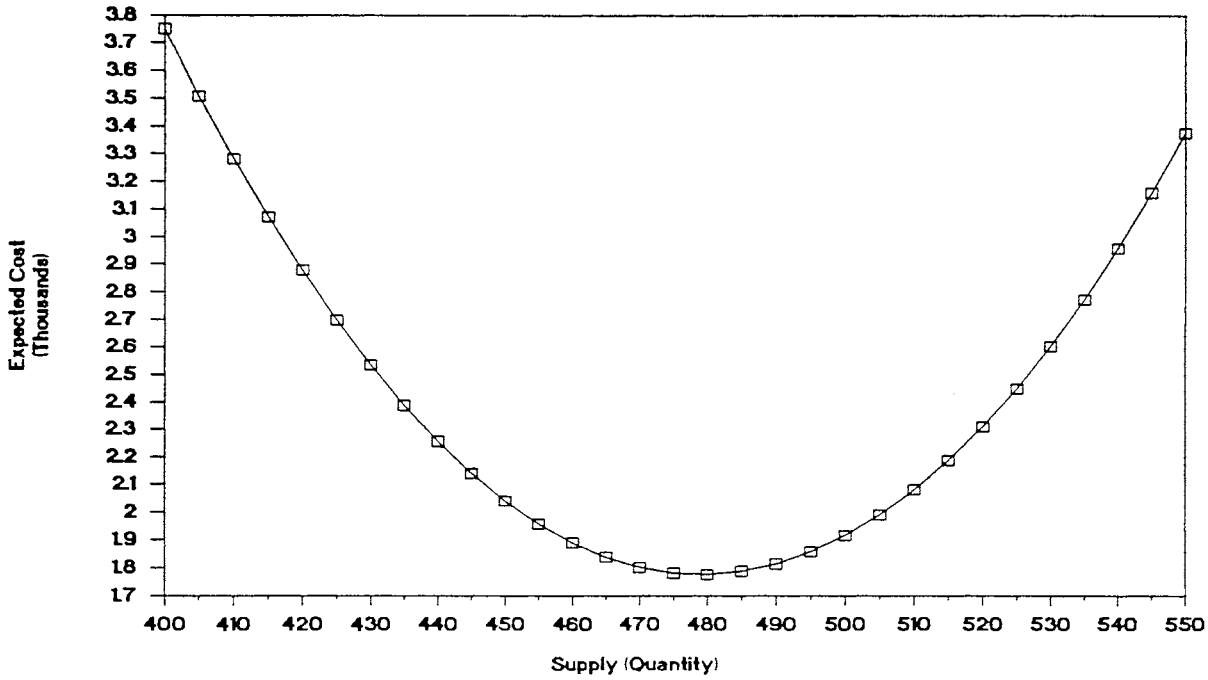


(A) Demand Center 1



(B) Demand Center 2

Figure 5.3. Expected Cost of Penalties at Each Destination For the Uniformly Distributed Demands



(C) Demand Center 3

Figure 5.3. (Continued)

Table 5.10 presents summary of the results for both normal and uniform distribution of demands. From this table, the zero underachievement for goal 1 indicate that the total penalty cost is fully minimized. In situations were this goal can not be completely satisfied, the underachievement for this goal depicts the difference between the resulting total expected penalty cost and its minimum value possible.

TABLE 5.10

SUMMARY OF THE RESULTS OF THE MODEL B EXAMPLE PROBLEM FOR NORMAL AND UNIFORM DISTRIBUTION OF DEMANDS

	Normal	Uniform
Open Facilities:	Y2=1, Y4=1	Y2=1, Y4=1
Allocation Assignments:		
	X21=345	X21=332
	X22=255	X22=237
	X42=149	X42=173
	X43=501	X43=477
Priority Order	Underachievements	
1. Penalty	0[a]	0[b]
2. Capacity	0	0
3. Budget	50[c]	50[c]
4. Trans.	91630	87670
5. Total	1491630	1487670
6. Config.	1	1

[a] Minimum expected penalty cost is 1547.78.
[b] Minimum expected penalty cost is 3680.64.
[c] Multiply by 1000.

Sensitivity Analysis

Sensitivity analysis is an integral part of the decision making process. It provides insights into the problem and facilitates the successful implementation of the model.

This section presents sensitivity analysis of the sample problem accomplished by changing its priority structure, goal levels, and parameters of the demands distributions. From this point the above changes are referred to as Type 1, Type 2, and Type 3 sensitivity analysis, respectively. Moreover,

as an illustration, the sensitivity analysis is only demonstrated for model A and for the case of normally distributed demands.

Type 1 Sensitivity Analysis

Type 1 sensitivity analysis for the example problem presented earlier, is to reorder the specified priority structure. However, in doing so the user must be cautious about the resulting model structure. For instance, identifying the total cost goal as the first priority goal without imposing any "hard" constraint(s) on the system will result in the trivial solution of do nothing (zero allocations) with zero total cost. This is similar to the traditional linear programming problems in which objectives must be optimized subject to satisfying system constraints. Because, in the absence of limiting constraints the objectives become unbounded.

Given the general purpose design of the computer programs and the variety of problem structures possible, the developed software will not check for trivial solutions. Therefore, it is the responsibility of the modeler to design a sound model structure which will not result in a trivial solution.

Table 5.11 presents the results of type 1 sensitivity analysis. In column 3 we assumed management decided that minimization of transportation (allocation) cost should take priority over the capacity goal. Also in column 4 we assumed

that management is interested in evaluating the alternative of assigning higher priority level to goal 3 (fixed annual budget) than to goal 2 (capacity).

TABLE 5.11

TYPE 1 SENSITIVITY ANALYSIS OF THE MODEL A EXAMPLE PROBLEM FOR NORMAL DISTRIBUTION OF DEMANDS

Priority Structure	1,2,3,4,5,6	1,4,3,2,5,6	1,3,2,4,5,6
Open Facilities:	Y2=1, Y4=1	Y2=1, Y3=1 Y4=1	Y1=1, Y4=1
Allocations:			
	X21=363	X21=363	X11=363
	X22=296	X33=526	X13=296
	X42=124	X42=420	X42=420
	X43=526		X43=230
Goals Identification		Underachievements	
1. Demand[a]	0	0	0
2. Capacity	0	126	159
3. Budget	50[b]	775[b]	0
4. Trans.	97580	45960	119640
5. Total	1497580	2170960	1369640
6. Config.	1	0	1

[a] Service Level=0.9.
[b] Multiply by 1000.

As a comparison, the solution to the original priority structure is also provided in column two. As shown in Table 5.11, the set of selected sites are different based upon the specified priority structure. However, a closer look reveals that the facility at site 4 is selected regardless of the

priority structure used. This can be contributed to the low fixed cost and relatively high capacity of a plant at this site. This analysis provides the management with a great insight into selecting among alternative sites for establishing new facilities.

Type 2 Sensitivity Analysis

Type 2 sensitivity analysis is used to analyze the changes in the decision variables and priority achievements which result from changing the goal levels. Assume, the decision maker desires to evaluate the effects of changes in service level on location and allocation decisions. Table 5.12 presents the results of type 2 sensitivity analysis. Once again column two of this table provides the solution for the original problem when service levels are set at 90%. Column 3 presents the results when service level at each demand center increases to 99%. And column 4 contains the results for 80% service level. Similar to the observation made in type 1 sensitivity analysis, site 4 is selected regardless of the service level specified.

From Table 5.12, when service level increases to 99 percent, the number of open facilities increases by one. This is because the higher service level at a demand center translates to higher supplies to that demand center, which in turn demands higher capacity. Moreover, although the extra facility increases the fixed costs, but the transportation

cost decreases by 33.1 percent. This is expected because the higher number of established plants offers more flexibility in distributing the products. Next, from column 4, when service level at all demand centers change to 80 percent, it reduces the transportation cost by 2.13% without affecting the location or allocation patterns. Finally, the lower service level also indicates more chance of undersupplying the demand centers.

TABLE 5.12

TYPE 2 SENSITIVITY ANALYSIS OF THE MODEL A EXAMPLE PROBLEM FOR NORMAL DISTRIBUTION OF DEMANDS

Service Level	0.90	0.99	0.80
Open Facilities:	Y2=1, Y4=1	Y1=1, Y3=1 Y4=1	Y2=1, Y4=1
Allocations:			
	X21=363	X11=374	X21=359
	X22=296	X33=400	X22=280
	X42=124	X42=433	X42=133
	X43=526	X43=147	X43=517
Priority Order		Underachievements	
1. Demand	0	0	0
2. Capacity	0	0	0
3. Budget	50[a]	625[a]	50[a]
4. Trans.	97580	65280	95500
5. Total	1497580	2040280	1495500
6. Config.	1	0	1

[a] Multiply by 1000.

Additional type 2 sensitivity analysis are performed by

employing the trade-off information provided by the sensitivity analysis algorithm. Table 5.13 presents the trade-off information for the example problem of model A when demands are normally distributed. This Table is obtained as a part of sensitivity analysis from the computer program described latter in Chapter VI. The trade-off information indicates how much higher priority goals must be relaxed such that lower priority goals can be increased by one unit. The small trade-off values in Table 5.13 are justified by relatively large allocation (transportation) costs.

TABLE 5.13

TRADE-OFF ANALYSIS OF THE MODEL A EXAMPLE PROBLEM FOR NORMAL DISTRIBUTION OF DEMANDS

Priority	<Conflicts with>	Priority	Trade-Offs
3	.....	System Constraints	
4	.....	2	0.02
4	.....	1	0.01
5	.....	2	0.02
5	.....	1	0.01
5	.....	System Constraints	
6	.....	5	650000.00
6	.....	3	650.00[a]

[a] Multiply by 1000.

From Table 5.13, in order to improve the priority 6 (configuration goal) achievement level by one unit, we need to relax priority 3 goal level (annual fixed cost budget

limitation) by 650 thousand units. This can be verified by increasing the goal level of priority 3 from 1,350,000 dollars to 2,000,000 dollars and resolving the problem. Table 5.14 contains the priority achievements along with the location pattern and allocation quantities for this modified problem.

TABLE 5.14

SOLUTION OF THE MODEL A EXAMPLE PROBLEM
FOR NORMAL DISTRIBUTION OF DEMANDS
AND MODIFIED BUDGET GOAL

Priority	Underachievement	Location	Allocation
1. Demand	0	Y1	X11=363
2. Capacity	0	Y3	X33=400
3. Budget[1]	0	Y4	X42=420
4. Trans.	62040		X43=126
5. Total	2037040		
6. Config.	0		

[1] Budget goal is 2000000.

Type 3 Sensitivity Analysis

The primary purpose of type 3 sensitivity analysis is to investigate changes in the location-allocation pattern and the achievement vector for different parameters of the demand distribution. This is extremely important whenever the above parameters can not be determined accurately or if only their estimates are available.

To explore the changes in the parameters of the demand distribution on the solution of SMOLAP, 24 test problems are designed and solved. Each problem is basically the same as the sample problem presented earlier for model A with normally distributed demands. However, with the exception that the parameters of the demand distributions for each demand center are systematically changed to produce the different test problems. Also, for each problem, it is assumed that the distribution parameters of all demand centers are equal. This along with equal service levels at all demand centers lead to equal equivalent deterministic demands at these centers.

Table 5.15 presents the different distribution parameters used for the test problems. As shown, the mean of the demands at the demand centers is changed from 100 to 600 in increments of 100, while the standard deviation is changed from 10 to 40 in steps of 10. These values are chosen such that they represent the extreme values of the demands for the current system. Additionally, service level at all demand centers is set to 95% level.

The average CPU time to solve these problems was 183 seconds with values ranging from 86 to 269 seconds. In all the problems, priorities 1 and 2 were fully achieved while priorities 3 through 6 were achieved at various levels. In order to perform site selection analysis, the location patterns for all test problems are shown in Table 5.15. From

examining these locational variables, four observations are made:

TABLE 5.15

THE DESIGN AND PARTIAL SOLUTIONS OF THE TEST PROBLEMS FOR TYPE 3 SENSITIVITY ANALYSIS

S.D. / MEAN	10	20	30	40
100	i Y_3, Y_4	ii Y_3, Y_4	iii Y_3, Y_4	iv Y_3, Y_4
200	v Y_3, Y_4	vi Y_3, Y_4	vii Y_3, Y_4	viii Y_3, Y_4
300	ix Y_3, Y_4	x Y_3, Y_4	xi Y_3, Y_4	xii Y_1, Y_4
400	xiii Y_2, Y_4	xiv Y_2, Y_4	xv Y_2, Y_4	xvi Y_1, Y_3, Y_4
500	xvii Y_1, Y_2, Y_4	xviii Y_1, Y_2, Y_4	xix Y_1, Y_2, Y_4	xx Y_1, Y_2, Y_4
600	xxi Y_1, Y_2, Y_4	xxii Y_1, Y_2, Y_3, Y_4	xxiii Y_1, Y_2, Y_3, Y_4	xxiv Y_1, Y_2, Y_3, Y_4

NOTE: for each problem, mean and standard deviation of all demand centers are assumed equal.

1) Site 4 is a candidate for establishing a new plant regardless of the variations in the demand parameters. This may be explained by the low fixed cost and relatively high capacity available for a plant at this site.

2) Combinations of sites (1,4) and sites (1,3,4) are

selected only for a small range of variations in the parameters of the demand distribution (problems xii and xvi).

3) Site 3 is preferred to site 2 for low demand requirements while site 2 is preferred to site 3 for larger demands. This may be explained by the combination of high priority of capacity goal and the larger capacity available at site 2.

4) For relatively low demand requirements (eg. $[100,10^2]$), although the desired service level can be met by opening only one plant at any site, but in order to achieve the various priority goals, two site are selected by the algorithm.

Next considering the allocations, in test problems i through ix, the proposed plant at site 4 supplies both demand centers 1 and 2, while the proposed plant at site 3 only supplies the demand center 3. Furthermore, for test problems x through xxiv, demand center 1 is usually serviced by a single plant, while demand centers 2 and 3 are supplied by multiple plants.

Another observation is made regarding the capacity at site 4. With the exception of problem xvi, the upper capacity limit for the plant at this site is always reached for problems x through xxiv. This suggests the potential of increasing the plant capacity at this site to improve the solution. Therefore, further analysis may be performed by increasing the capacity at site 4.

Next, to examine the effects of unequal demands on the location-allocation decisions of the current system, and to compare the results with the earlier case of equal demands, 24 more test problems are designed. Each problem in the new set corresponds to a problem in the former set through a constraint on the total demand generated. That is, although the demands for the new problems are not equally distributed among the three demand centers, their total deterministic equivalent value is equal to the total equivalent deterministic demand obtained from the former set (case of equal demands). Therefore, the total demand for each test problem is divided into three unequal parts, one for each demand center. The unequal demands are selected such that they represent low, medium, and high demands. For instance, the equal demands of 450 units (this is the deterministic equivalent) at each demand center are arbitrary divided to demands of 220 units, 440 units and 690 units. The unequal demands are then arbitrary assigned to the three demand centers. A comparison of the location-allocation decisions between the cases of equal and unequal demands can then be accomplished.

Analyzing the locational decisions, all the new problems produced the same locational patterns as their counterparts, except for the problem xi which indicated selection of site 1 instead of site 3. This illustrates that the locational decisions for this problem are more sensitive to the total demand as opposed to an uneven allocation of the demand.

Regarding the allocations, half of the new problems produced allocation patterns which were different from the earlier cases of equal demands. The different allocation assignments occurred mainly for the problems in the high total demand category (Problems ix,xi,xiii,xiv,xvi,xvii,xviii,xix,xx,xxi, xxiii,xxiv). This may be explained by the large demand of one demand center in each of these problems. As stated previously in establishing these problems, the unequal demands were selected such that three levels of demands can be distinguished; low, medium, and high. Now, the different allocation patterns may be explained by the fact that the high demand at one of the demand centers is greater than the available capacity (total or remaining capacity) in the other supplying plant(s). As a result, new or alternative allocation assignments are required. Therefore, the modified allocation patterns are mainly due to the larger demands and capacity constraints.

Computational Difficulties

A common problem inherent in algorithms requiring a large number of iterations is the cumulative round-off error. This problem was encountered several times in the course of validating and performing computational analysis for the algorithms proposed in this research. The effects of round-off errors have varied from obtaining infeasible solutions to feasible but dominated solutions. To overcome

this difficulty and obtain optimal solutions, the accuracy of the real type variables was increased at the expense of computational speed and computer storage. Originally, the real type variables were changed to double precision variables. This increased the number of significant digits of real type variables from 11-12 to 14-16. However, this still proved to be inadequate for the illustrative example presented in this chapter. As a consequence, the definition of these variables was changed to extended variables. Extended type variables provide 19-20 significant digits for the real type variables.

Another difficulty encountered, as discussed previously in Test Problem 3, was the selection of M value for the system constraints. These constraints insure that allocation of products are made from open facilities. It was experienced that selection of relatively large values for M can lead to round-off errors in the simplex iterations.

Finally, the usually large coefficients of location variables in the total cost goal may contribute to computational errors in the simplex iterations. This point was discussed and illustrated earlier in analyzing the Test Problem 3. Thus, in order to insure the accuracy of the results, it is recommended that these models be solved both with and without the total cost goal. Then the results of these two formulations may be compared to determine if errors have occurred. Specifically, use the decision variables obtained from the latter formulation (formulation without

total cost goal) to evaluate the total cost goal. Then, compare the goal achievement levels of both formulations. If the solution from the latter formulation dominates the solution from the former one, errors have occurred, so we select the nondominated solution as the optimal solution.

Summary

This chapter presented the validation and computational analysis of the models developed in this research. To validate the algorithms and computer programs, various test problems from the literature were selected and solved. Results for three test problems were presented. In all cases the program performed well by reproducing the documented results. Furthermore, in two of the above three cases the developed algorithms performed better by dominating the reported solutions, and finding the true optimal solutions. Next, in order to demonstrate the formulation and solution procedures of the proposed models and to obtain computational experiments with them, a hypothetical problem was presented. The problem was solved using both models. Sensitivity analysis of the SMOLAP was demonstrated through analysis of model A. Finally the computational difficulties encountered were discussed.

CHAPTER VI

INTERACTIVE COMPUTER PROGRAM

Introduction

In order to experiment with the proposed models, an interactive computer program is developed. The program is written in TURBO PASCAL 5.0 and runs on IBM compatible microcomputers with at least 640 KB memory. The program consists of three main modules; data base management utilities, solution algorithms, and sensitivity analysis. The following sections present a description of the program structure and the main features of each module.

General Structure of the Program

Figure 6.1 depicts the general structure of the computer program. The program operates through two menu systems; the main menu and the sensitivity analysis menu. Figure 6.2 presents the display of the main menu system. The main menu presents options regarding data base management, solution algorithms, and the option to access the sensitivity analysis menu. The current model name (last model loaded or created) is displayed on the top right hand side of the menu screen. Also, displayed is the model type; deterministic, chance-

constrained, or stochastic.

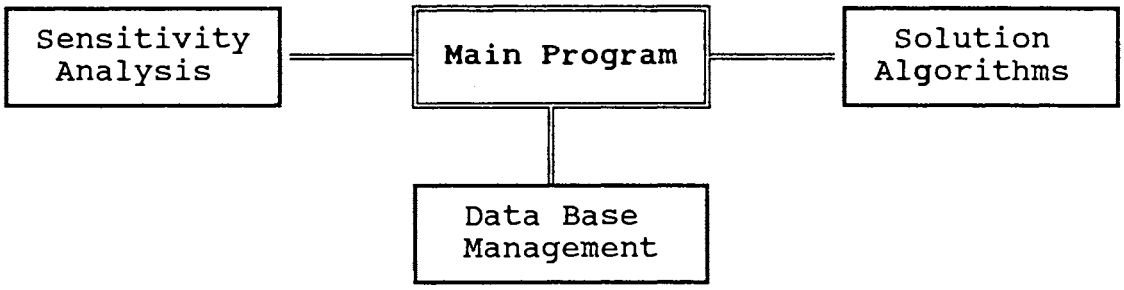


Figure 6.1. General Structure of the Computer Program

```
SMOLAP - Decision Support System

                                    Current Model > None
                                    Type: None
DATA BASE UTILITIES:

[A]  Create a New Model
[B]  Retrieve an Existing Model
[C]  Save Current Model
[D]  Display Current Model

SYSTEM ANALYSIS:

[E]  Continuous Solution
[F]  Integer Solution
[G]  Nonlinear Solution
[H]  Sensitivity Analysis

[I]  EXIT

Enter Option —▷
```

Figure 6.2. Display of the Main Menu

The program validates all the user inputs and checks for the out of sequence selection of menu items. For example, the sensitivity analysis option can not be selected before a model is created or loaded from storage and a solution is obtained for it. Furthermore, the last line of the screen is reserved mainly for soliciting inputs from the user and displaying various messages.

Data Base Management Module

This module presents to the user the capability of creating a new model, recalling an existing model, saving the current model,or displaying the data pertaining to the current model. The structure of this module is depicted in Figure 6.3.

To create a new model the user selects option A from the main menu. Then, the program requests the model type. There are three model types possible; Deterministic (D), Chance-Constrained (C), and Stochastic (S). Next, based on the user response, the program presents a data entry screen appropriate for the specified model type. In general, the user is required to provide three sets of information; information regarding priorities, information for non-zero technological coefficients, and information on the right hand side values. However, the information required for the last category differs based on the model type selected. Figure 6.4 presents a typical data entry screen for the deterministic models.

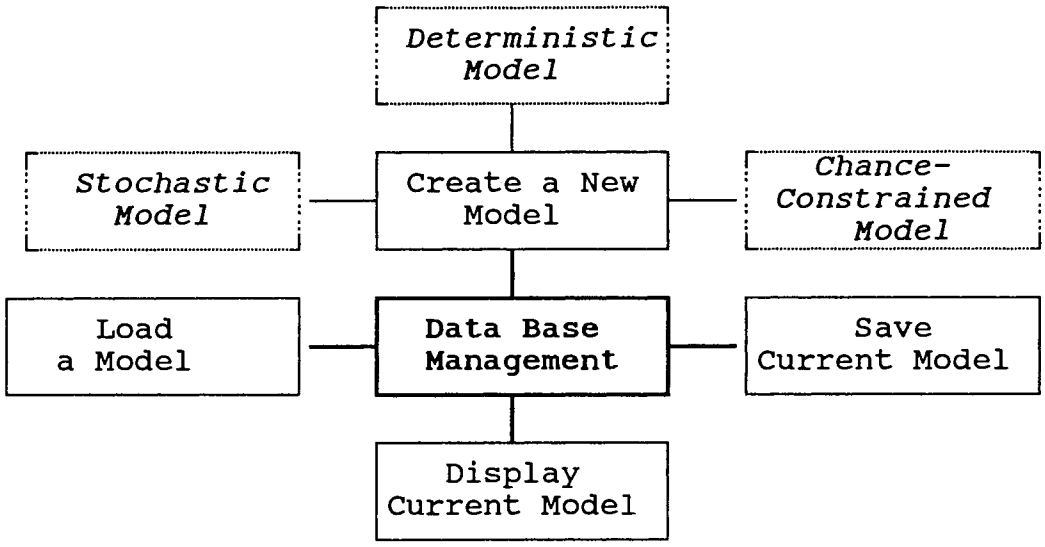


Figure 6.3. Structure of the Data Base Management Module

```
                    CREATE A NEW MODEL

                                                Type: Deterministic

SET 1 - PRIORITY STRUCTURE:                      ------ HELP ------
   Sign 'P' or 'N' ----->                        SET 1
   Row Number ---------->                        0 <    ROW    ≤ 30
   Priority ---------->                          0 < Priority ≤ 10
   Weight ---------->                            0 <  Weight
                                                 SET 2
                                                 0 <    ROW    ≤ 30
SET 2 - TECHNOLOGICAL COEFFICIENTS:              0 < Variable ≤ 30
                                                 0 < Coefficient
   Row Number ---------->                        SET 3
   Column Number ---------->                      Sign
   Coefficient ---------->                          E ..... =
                                                    G ..... ≥
SET 3 - CONST. SIGN AND RHS VALUES:                 L ..... ≤
                                                    B ..... GOAL
   Sign for Constraint 1 ---->                   RHS ≥ 0
   RHS  for Constraint 1 ---->

Save This Model? (Y/N) ->
```

Figure 6.4. Display of the Deterministic Input Data Screen

The window on the right hand side of the input screen provides help for data entries. Data for technological coefficients in SET 2 may be entered either row wise or column wise. In either case, the program always sorts this data columnwise for use by the solution algorithms.

After a complete set of information is entered for a given category the user can switch to the subsequent class just by pressing carriage return in response to the first question of the current category. Furthermore, the program checks the validity and the range of data for all entries. The program requires minimum input data from the user. This is accomplished by calculating some of the information such as number of rows, variables, and priorities from other input data. All inputs to the program are converted into an appropriate format for use by the solution algorithms. Finally, the present definition of array dimensions in the program allows a user to input and solve problems with up to 30 variables, 30 constraints (goals) and 10 priorities. To solve larger problems it is necessary to increase these dimensions. However, when modifying these dimension settings, special attention must be made to allow memory for dynamic variables. These variables are used by the branch and bound routine to obtain integer solutions. Insufficient memory allocation for dynamic variables can result in Out of Memory error in the course of obtaining an integer solution.

Solution Algorithms Module

This module is capable of obtaining a continuous or an integer (pure, mixed, zero-one) solution for a given model. In addition, the pattern search algorithm in this module finds solutions to single objective unconstrained problems. The structure of this module is depicted in Figure 6.5.

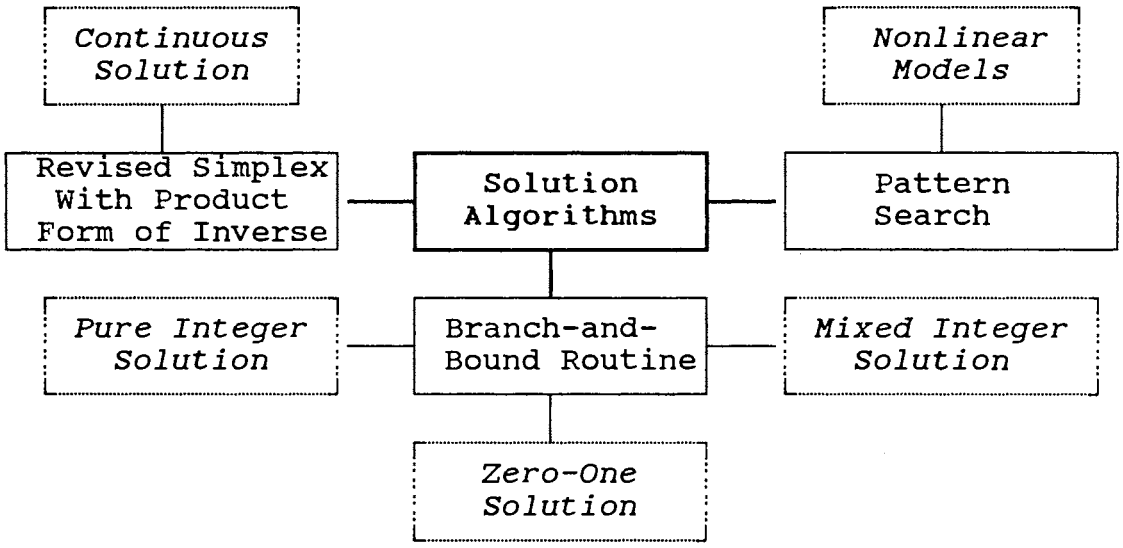


Figure 6.5. Structure of Solution Algorithm Module

The continuous solution algorithm is based on the modified simplex algorithm for preemptive GP problems by Lee (1972). However, for computer storage conservation and computational accuracy, the algorithm takes advantage of the revised simplex method and utilizing the product form of the inverse in finding the optimal solutions. Specifically, the

revised simplex method uses the original data to calculate the $Z_j - C_j$'s and updated columns, Y_k's, which tends to reduce the round-off errors. Figure 6.6 presents a sample output screen representing a continuous solution for a given problem. The output consists of three major parts: Analysis of multiple objectives which provides underachievement of all priorities, analysis of decision variables which reports the value for all decision variables, and analysis of deviational variables which provides positive and negative deviations for all goal and system constraints.

Furthermore, this module employs a branch-and-bound routine to drive pure or mixed integer solutions. The mixed integer solutions are made possible by allowing the DM to mark the variables with integer requirements through an interactive menu system. The primary data structure used in this routine is a binary tree. The zero-one requirements are handled through proper problem formulation and the branch-and-bound algorithm. In searching for an integer solution, the branch-and-bound routine employs a depth-first strategy. In this approach, the program attempts to go deeper and deeper into the tree before examining neighboring nodes. This strategy is employed in hope of establishing a tight upper bound early in the search for the optimal integer solution. A good upper bound can facilitate pruning the branches of the binary tree.

Finally, a modified pattern search based on the Hooke and Jeeves algorithm is used to solve the nonlinear models of

stochastic formulation.

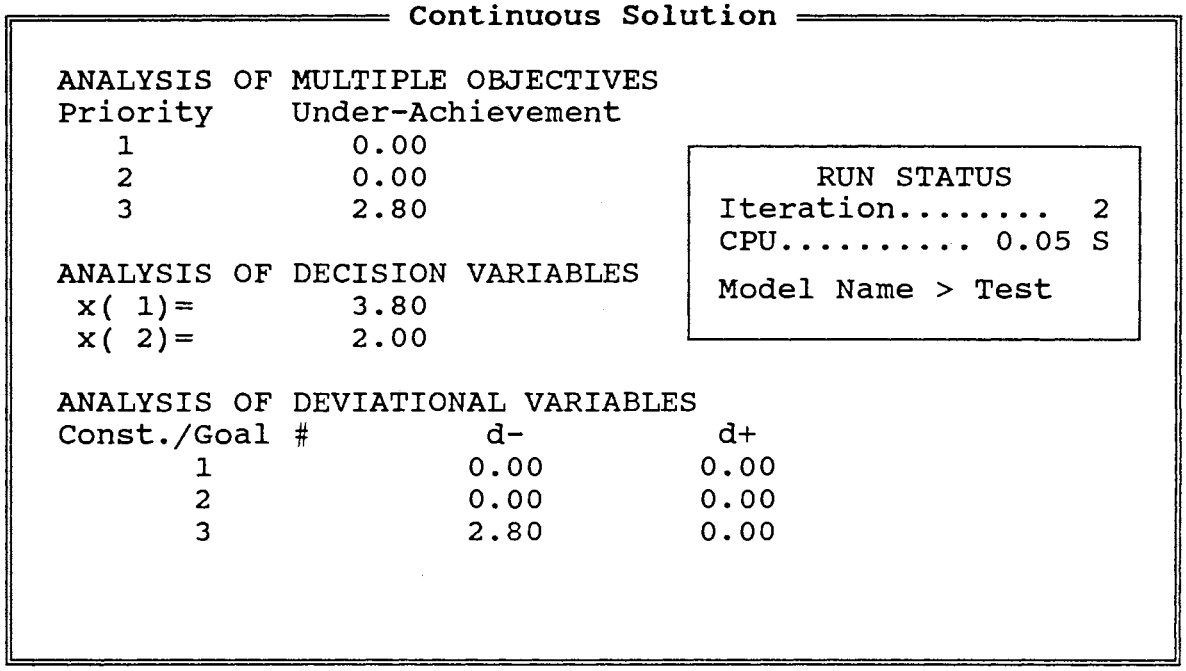


Figure 6.6. Sample Output Screen for Continuous Solution

Managing the computer storage and execution time for integer solutions is particularly important. In order to reduce the demand on virtual storage, the branch-and-bound procedure takes advantage of the dynamic variables. These variables allow for the nodes to be allocated and disposed as necessary in finding an integer solution. This will enables the program to handle larger integer problems. On the other hand, in order to speed the execution time and reduce computer storage, tighter upper bounds are established for

problems containing only goal constraints. This is accomplished by first solving the continuous problem and then rounding the variables with integer requirement to their nearest integer values. Next, the upper bound is determined by calculating the achievement function for these new variables. In course of validating and experimenting with the program, this was proved to be very effective in solving multiobjective integer problems with only goal constraints. Moreover, after solving a subproblem at a node and selecting the next node, a dominance test is performed at the new node before solving its subproblem. The branch at this new node is pruned if the objective vector at this node is dominated by the upper bound. This test compares the set of achievement levels at the current node with the upper bound. If the current solution dominates the upper bound the solution continues, otherwise the selected node is terminated (disposed) and the search continues by selecting a new node. The selection of nodes follows the LIFO rule.

Next, during the execution of this module a window on the right hand side of the screen will inform the user of the status of the program. In the case of continuous and nonlinear solutions this information includes current iteration number, execution time in seconds, and the name of the model under study. Additionally, for integer solutions, the program also displays total number of nodes generated, total number of nodes evaluated, and the number of times the

upper bound is updated. To be more specific, number of iterations refers to the number of pivots performed in the modified simplex tableau, number of nodes generated indicates how many nodes are created for the branch-and-bound tree, number of nodes evaluated means how many nodes from the latter set are currently being evaluated explicitly, and number of upper bound updated indicates the number of solutions obtained in course of finding the optimal solution which satisfies the integer requirements and dominate the existing upper bound. Of course, in case of such solutions the upper bound will be updated to reflect the new achievement vector.

Sensitivity Analysis Module

This module presents four options to assist the DM in making an intelligent trade-offs among various objectives. The structure of this module is shown in Figure 6.7. Figure 6.8 presents the display of the sensitivity analysis menu. There are four options available: list actual vs. desired goals, perform trade-off analysis, change priority structure, and change right hand side of the goals or rigid constraints. While the first two options assist in determination of the appropriate changes, the last two options are used to actually accomplish the necessary modifications in the model. More specifically, the trade-off analysis lists the conflicting objectives and displays the marginal substitution rates (MSR) for each pair. So, this value is calculated only

for conflicting objectives. The conflicting objectives in the modified simplex tableau are identified by the sign of their $Z_j - X_j$'s in the objective column. The $Z_j - C_j$ of the higher priority goal will be negative while this value will be positive for the lower priority goal.

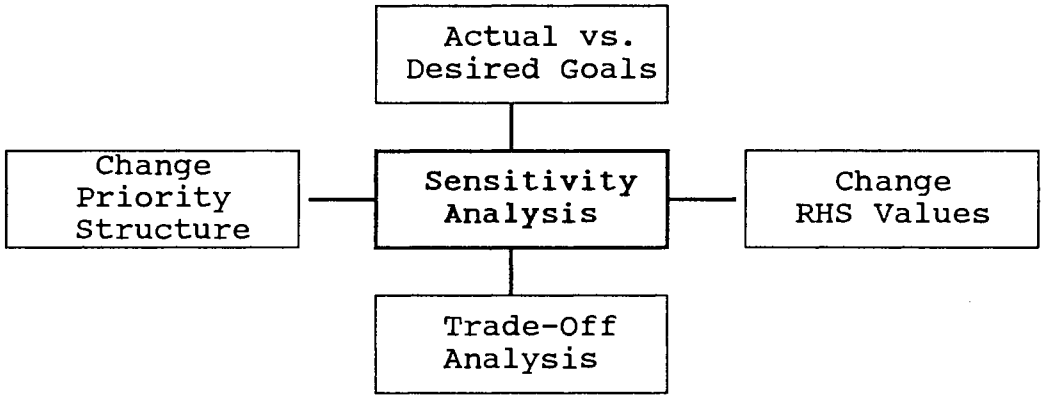


Figure 6.7. Structure of the Sensitivity Analysis Module

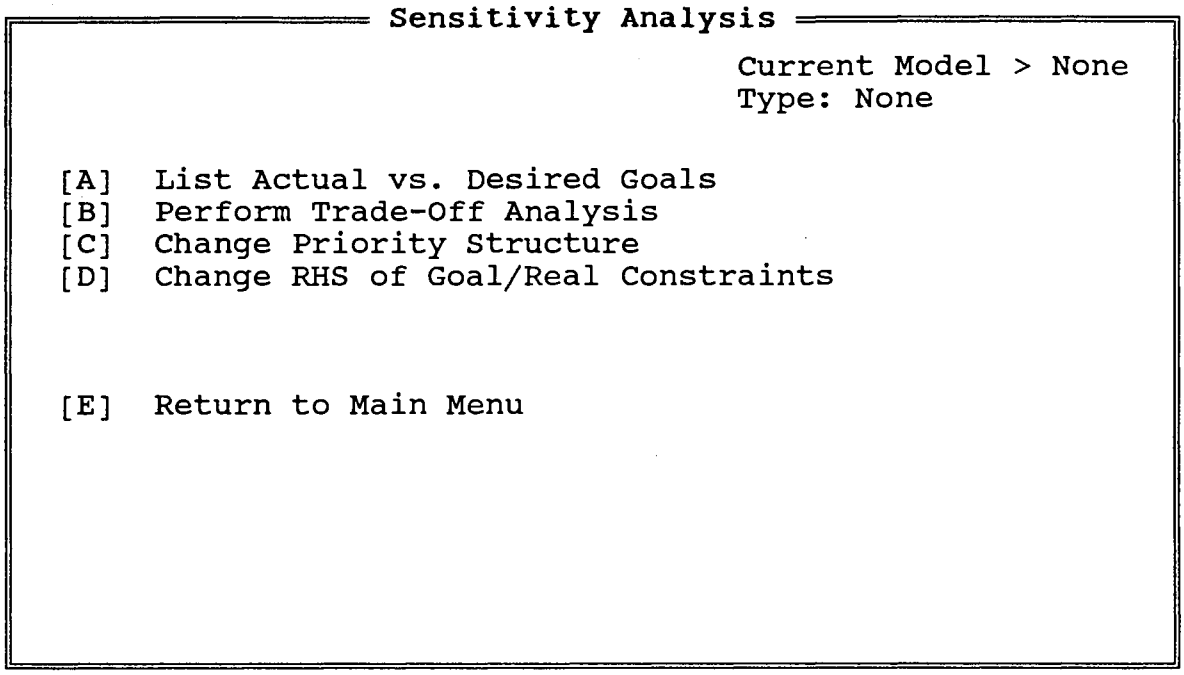


Figure 6.8. Display of the Sensitivity Analysis Menu

The marginal substitution rate implies how much achievement of a higher priority goal must be deteriorated so that the achievement level of the lower priority goal can be increased by one unit. Mathematically, this relationship can be stated as follows:

$$MSR = - (Z_j - C_j)_m / (Z_j - C_j)_n$$

Where m and n are conflicting goals and m>n (i.e. m indicates the higher priority goal). Obviously, if a goal conflicts with a system constraint, then its MSR is nonexistent.

Summary

In this chapter the general structure of the computer program along with some of its main features was presented. Development of this software for use on a microcomputer greatly enhances the flexibility and convenience of its use. The program is designed such that it can be easily applied to other applications requiring multiple objective analysis without and modifications to the existing source codes.

The source codes for the computer program, except for the procedure Update, are listed in Appendix C. In addition, Table C.1 provides the index to the units and procedures of the program. The next chapter presents a summary, conclusion, and future studies for this research problem.

CHAPTER VII

SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

Summary

This research has explored the effects of variability of demand in the multiple objective analysis of location-allocation models. As the result, two multiobjective models based on chance-constrained programming and stochastic programming were developed. A solution algorithm based on chance-constrained goal programming was proposed for the former model. A two-stage algorithm was suggested for dealing with the nonlinear goal programming structure of the latter model. Both algorithms produce an optimal solution for their respective models.

Next, in order to experiment with the proposed models, an interactive computer program was written. Development of this interactive computer program on a microcomputer adds to the convenience and ease of use of the proposed solution algorithms by a decision maker. Although the software developed in this research is used mainly to analyze the multi-objective LAPs with stochastic demands, its general structure allows for the solution and sensitivity analysis of other multiobjective models without requiring any

modifications to the existing codes. Finally, the solution algorithms and the different sensitivity analysis of the proposed models were demonstrated through an example problem.

Conclusions

From the analysis of the stochastic multiobjective location-allocation problem (SMOLAP) in Chapter V, optimal location of facilities and their optimal allocation of products to the demand centers are greatly influenced by the priority structure of the multiple objectives, their goal levels, and the demand distribution. Therefore, inclusion of stochastic demand into the analysis of multiobjective LAPs has provided for a more comprehensive treatment of these problems. Furthermore, it was shown that based on the data available and the decision maker's preference, different models may be established to analyze SMOLAP's.

The application of the nonlinear multiobjective pattern search as presented by Ignizio (1976) to allocation subproblems of model B was not successful. Also, initial experimentation with modifying step sizes, error levels (δ), initial starting point, and convergence criteria did not prove encouraging.

The models developed in this research can be easily extended to incorporate multiple products through reformulation of the problem. The approach, except for handling the random demand, is basically equivalent to the one already suggested for single objective deterministic

LAP's. This is accomplished by defining the N demand centers and the M potential plants appropriately. For instance, demand centers j and j+1 may be defined to refer to the same physical demand center but, indicating the requirement for two different commodities at that location. Similarly, we can introduce K artificial facilities at site i to represent the source of K different commodities (services) at site i. Now, the stochastic demand for each product can be specified separately at each destination and the problem can be formulated and solved using one of the methods presented earlier. However, the drawback of this technique is that the problem size increases significantly with an increase in the number of products.

Finally, the proposed models may be extended to employ other important objectives or different demand distributions. The next section will present some of the possible extensions to this research study.

Recommendations for Future Research

Several recommendations can be made with regard to the proposed models and further research in this area. But, first, there are two recommendations for improving the developed software.

It is recommended to enhance the existing interactive computer program with a graphic system. The graphic system can facilitate the process of multi-objective decision making

by conveying the trade-off information effectively to the decision maker.

The computational speed of the developed software may be improved by devising and implementing more efficient selection and branching rules for the branch-and-bound routine. Furthermore, larger problems may be solved by allowing the use of auxiliary storage devices to store the intermediate results.

It is recommended to include inventory carryovers and backorders into the proposed models. Inclusion of these dynamic aspects will enable the analyst to study the behavior of the system over some predetermined planning horizon.

Another possibility is to study the effect of randomness in other factors such as capacity (supply), transportation costs, and fixed costs for the models developed in this research. Additionally, the dimension of price sensitivity can be added to the stochastic demand.

Further analysis of the proposed models can be made by incorporating other characteristics of the LAP, such as the interaction among facilities at potential locations and the presence of existing facilities in the system.

The possibility of deriving a linear approximation for the nonlinear cost function of model B should be examined. This can result in simpler linear models. The goodness of this approximation can then be verified by comparing its results with the optimal solution.

Another area of further research is development of an

alternate optimal seeking algorithm to solve the nonlinear integer multi-objective programming problem of model B.

Application and effectiveness of other nonlinear techniques such as the Rosenbrock's method with variable search directions, Nelder and Mead's "simplex method", or a modification of the existing pattern search method for the solution of model B and in general for the solution of the nonlinear goal programming models should be explored.

Investigate other multicriteria approaches, such as compromise programming and linear multiobjective programming, for the solution of the SMOLAP.

Multiobjective formulation and analysis of distribution systems where location of warehouses are to be determined in relation to existing suppliers and demand centers is another potential area for future study.

Finally, further research could be conducted to develop heuristic procedures for the nonlinear integer programing and nonlinear integer multiobjective programming problems.

BIBLIOGRAPHY

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

Akinc, U. & Khumawala, B. M. "An Efficient Branch and Bound Algorithm for the Capacitated Warehouse Location Problem." Management Science, 23, 6 (1977), 585-594.

Alcouffe, A. & Muratet, G. "Optimal Location of Plants." Management Science, 23, 3 (1976), 267-274.

Arthur, J. L. & Ravindran, A. "An Efficient Goal Programming Algorithm using Constraint Partitioning and Variable Elimination." Management Science, 24, 8 (1978), 867-868.

Balachandran, V. & Jain, S. "Optimal Facility Location Under Random Demand with General Cost Structure." Naval Research Logistics Quarterly, 23 (1976), 421-436.

Balas, E. "An Additive Algorithm for Solving Linear Programs with Zero-One Variables." Operations Research, 13 (1965), 517-545.

Balinski, M. L. "Fixed-Cost Transportation Problems." Naval Research Logistics Quarterly, 8 (1961), 41-54.

Ballou, R. H. "Dynamic Warehouse Location Analysis." Journal of Marketing Research, 5, (1968), 271-276.

Banks, J., Spoerer, J. P., & Collins, R. L. IBM PC Applications for the Industrial Engineer and Manager, Englewood Cliffs, N.J.: Reston. (1986).

Baumol, W. J. & Wolfe, P. "A Warehouse Location Problem." Operations Research, 6, 2 (1958), 252-263.

Bellman, R. "An Application of Dynamic Programming to Location-Allocation Problems." SIAM Review, 7, 1 (1965), 126-128.

Bowersox, D. J. "Planning Physical Distribution Operations with Dynamic Simulation." Journal of Marketing, 36, 1 (1972), 17-25.

Camp, R. C. "The Effect of Variable Lead Times on Logistics Systems." (Ph.D. dissertation, Penn State University, 1973).

Cerson, M. L. & Maffei, R. B. "Technical Characteristics of Distribution Simulators." Management Science, 10, 1 (1963), 62-69.

Changchit, C. "A Multiobjective Approach to the Reservoir Operation Problem with Stochastic Inflows." (Ph.D. dissertation, Oklahoma State University, Stillwater, 1986).

Charnes, A., & Cooper, W. W. Management Models and Industrial Applications of Linear Programming, Vol. 1&2, Wiley, New York, 1961.

Charnes, A., & Cooper, W. W. "Deterministic Equivalents for Optimizing and Satisficing Under Chance Constraints." Operations Research, 11, 1 (1963), 18-39.

Chaudhry, S. S., McCormick, S. T., & Moon, I. D. "Locating Independent Facilities with Maximum Weight: Greedy Heuristics." OMEGA International of Management Science, 14, 5 (1986), 383-389.

Christofides, N. & Beasley, J. E. "Extensions to a Lagrangean Relaxation Approach for the Capacitated Warehouse Location Problem." European Journal of Operations Research, 12 (1983), 19-28.

Cohon, J. L. Multiobjective Programming and Planning, Academic Press, New York, 1978.

Connors, M. M., Coray, C., Cuccaro, C. J., Green, W. K., Low, D. W., & Markowitz, H. M. "The distribution System Simulator." Management Science, 18, 8 (1972), B-425 - B453.

Cooper, L. "Location-Allocation Problems." Operations Research, 11 (1963), 331-343.

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

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

Cooper, L. & Drebes, C. "An Approximate Solution Method for the Fixed Charge Problem." Naval Research Logistics Quarterly, 14, 1 (1967), 101-113.

Cooper, L. "The Transportation-Location Problem." Operations Research, 20 (1972), 94-108.

Cornuejols, G., Fisher, M., & Nemhauser, G. "Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms." Management Science, 23 (1977), 789-810.

Curry, G. L., & Skeith, R. W. "A Dynamic Programming Algorithm for Facility Location and Allocation." AIIE Transactions, 1, 2 (1969), 133-138.

Davis, P. S. & Ray T. L. "A Branch and Bound Algorithm for the Capacitated Facility Location Problem." Naval Research Logistics Quarterly, 16, 3 (1969), 331-343.

Efroymson, M. A. & Ray, T. L. "A Branch-Bound Algorithm for Plant Location." Operations Research, 14, 3 (1966), 361-369.

Eilon, S. "Multi-Criteria Warehouse Location." International Journal of Physical Distribution and Materials Management, 12, 1 (1982), 42-45.

Eilon, S., Watson-Gandy, C. D. T., & Christofides, N. (1971). Distribution Management: Mathematical Modelling and Practical Analysis. New York: Hafner Publishing, 1-94.

Ellwein, L. B., & Gray, P. "Solving Fixed Charge Location-Allocation Problems with Capacity and Configuration Constraints." AIIE Transactions, 3, 4 (1971), 290-298.

Elson, D. G. "Site Selection via Mixed-Integer Programming." Operational Research Quarterly, 23, 1 (1972), 31-43.

Erlenkotter, D. "Facility Location with Price-Sensitive Demands: Private, Public, and Quasi-Public." Management Science, 24 (1977), 378-386.

Erlenkotter, D. "A Dual-Based Procedure for Uncapacitated Facility Location." Operations Research, 26, 6 (1978), 992-1009.

Feldman, E., Lehrer, F. A., & Ray, T. L. "Warehouse Location Under Continuous Economies of Scale." Management Science, 12, 9 (1966), 670-684.

Fortenberry, J. C. & Mitra, A. "A Multiple Criteria Approach to the Location-Allocation Problem." Computers & Industrial Engineering, 10, 1 (1986), 77-87.

Francis, R. L., & Goldstein, J. M. "Location Theory: A

Selective Bibliography." Operations Research, 22 (1972), 400-410.

Francis, R. L., & White, J. A. (1974). Facility Layout and Location: An Analytical Approach. Englewood Cliffs, N.J: Prentice-Hall, pp. 230-232.

Fulton, M. "New Factors in Plant Location." Harvard Business Review, May-June (1971), 4-17.

Gelders, L. F., Pintelon, L. M., & Wassenhove, L. N. "A Location-Allocation Problem in a Large Belgian Brewery." European Journal of Operational Research, 28 (1987), 196-206.

Geoffrion, A. M. "A Guide to Computer-Assisted Methods for Distribution Systems Planning." Sloan Management Review, 16, 2 (1975), 17-41.

Geoffrion, A. M. & Graves, G. W. "Multicommodity Distribution System Design by Benders Decomposition." Management Science, 20, 5 (1974), 822-844.

Geoffrion, A. M. & Mc Bride, R. "Lagrangean Relaxation Applied to capacitated Facility Location Problems." AIIE Transactions, 10 (1978), 40-47.

Gerson, M. L. & Maffei, R. B. "Technical Characteristics of Distribution Simulator." Management Science, 10, 1 (1963), 62 - 69.

Glover, F. "Multi-Phase Dual Algorithm for the Zero-One Integer Programming Problems." Operations Research, 13, 6 (1965), 879-919.

Goicoechea, A., Hansen, D. R., & Duckstein, L. Multi-Objective Decision Analysis with Engineering and Business Applications, Wiley, New York, 1982.

Gomory, R. E. "An Algorithm for Integer Solutions to Linear Programs." Bulletin of the American Math Society, 64 (1958), 275-278.

Gonzalez-Valenzuela, F. "Simple and Capacitated Warehouse Location Problems with Stochastic Demands." (Ph.D. dissertation, Stanford University, 1975).

Green, G. I., Kim, C. S., & Lee, S. M. "A Multicriteria Warehouse Location Model." International Journal of Physical Distribution and Material Management, 11, 1 (1981), 5-13.

Green, T. B., Newsom, W. B., & Jones, S. R. "A Survey of the

Application of Quantitative Techniques to Production/Operations." Academy of Management Journal, 20, 4 (1977), 669-676.

Hansen, P. & Thisse, J-F, "Multiplant Location for Profit Maximisation." Environment Planning A, 9 (1977), 63-73.

Harrison, H. "A Planning System for Facility and Resources in Distribution Networks." Interfaces, 9, 2 (1979), 6-22.

Hastings, C. Approximations for Digital Computers, Princeton University Press, Princeton, N.J., 1955.

Hwang, C. L., Masud, A. S. M., Paidy, S. R., & Yoon, K. Multiple Objective Decision Making-Methods and Applications: A State-of-the-Art Survey, Springer-Verlag, Berlin/Heidelberg/New York, 1979.

Hwang, C. L., Paidy, S. R., Yoon, K., & Masud, A. S. M. "Mathematical Programming with Multiple Objectives: A Tutorial." Computers and Operations Research, 7, 1 (1980), 5-31.

Ignizio, J. P. Goal Programming and Extensions, Lexington Books, Heath, Lexington, Mass., 1976.

Ignizio, J. P. "The Determination of a Subset of Efficient Solutions Via Goal Programming." Computers & Operations Research, 8 (1981), 9-16.

Ignizio, J. P., & Perlis, J. H. "Sequential Linear Goal Programming: Implementation Via MPSX." Computers & Operations Research, 6 (1979), 141-145.

Ijiri, Y. Management Goals and Accounting for Control, Amsterdam: North Holland, 1965.

Jucker, J. V., & Carlson, R. C. "The simple Plant-Location Problem under Uncertainty." Operations Research, 24, 6, (1976), 1045-1055.

Karanicolas, P. C. "Multiperiod Plant Location Problems." (Ph.D. dissertation, Syracuse University, 1979).

Karkazis, J., & Boffey, T. B. "The Multi-Commodity Facilities Location Problem." Journal of Operational Research Society, 32, 9 (1981), 803-814.

Kaufman, L., Eede, M. V., & Hansen, P. "A plant and Warehouse Location Problem." Operational Research Quarterly, 28, 3 (1977), 547-554.

Kelly, D. L., & Khumawala, B. M. "Capacitated Warehouse Location with Concave Costs." Journal of the Operational Research Society, 33 (1982), 817-826.

Khan, A. M. "Solid-Waste Disposal with Intermediate Transfer Stations: An Application of the Fixed-Charge Location Problem." Journal of the Operational Research Society, 38, 1 (1987), 31-37.

Khumawala, B. M. "An Efficient Branch and Bound Algorithm for the Warehouse Location Problem." Management Science, 18, 12 (1972), 718-731.

Khumawala, B. M., & Kelly, D. L. "Warehouse Location with Concave Costs." INFOR, 12, 1 (1974), 55-65.

Khumawala, B. M., & Whybark, D. C. "Solving the Dynamic Warehouse Location Problem." International Journal of Distribution, 6, 5 (1976), 238-251.

Khumawala, B. M., & Neebe, A. W. "A Note on Warszawski's Multi-Commodity Location Problem." Journal of Operational Research Society, 29, 2 (1978), 171-172.

Klein, M., & Klimpel, R. R. "Application of Linearly Constrained Nonlinear Optimization to Plant Location and Sizing." The Journal of Industrial Engineering, 18, 1 (1967), 90-95.

Klincewicz, J. G., & Luss, H. "A Lagrangian Relaxation Heuristic for Capacitated Facility Location with Single-Source Constraints." Journal of the Operational Research Society, 37, 5 (1986), 495-500.

Kuehn, A. A. & Hamburger, M. J. "A Heuristic Program for Locating Warehouses." Management Science, 9, 4 (1963), 643-666.

Kuhn, H. W. & Kuenne, R. E. "An Efficient Algorithm for the Numerical Solution of the Generalized Weber Problem in Spatial Economics." Journal of Regional Science, 4, 2 (1962), 21-33.

Land, A. H., & Doig, A. "An Automatic Method of Solving Discrete Programming Problems." Econometrica, 28 (1960), 497-520.

Lee, S. M. Goal Programming for Decision Analysis, Philadelphia: Auerbach, 1972.

Lee, S. M. "Goal Programming for Decision Analysis of Multiple Objectives." Sloan Management Review, 14, 2 (1973), 11-24.

Lee, S. M. Goal Programming Methods for Multiple Objective Integer Programs, OR Monograph Series, No. 2., American Institute of Industrial Engineers (1979).

Lee, S. M. "Goal Programming Methods for Multiple Objective Integer Programs." OR Monograph Series, No. 2. American Institute of Industrial Engineers (1979).

Lee, S. M., & Franz, L. S. "Optimising the Location-Allocation Problem with Multiple Objectives." International Journal of Physical Distribution and Materials Management, 9, 6 (1979), 245-255.

Lee, S., Green, G., & Kim, C. "A Multiple Criteria Model for the Location-Allocation Problem." Computers & Operations Research, 8 (1981), 1-8.

Lee, S. M., & Luebbe, R. L. "The Multi-Criteria Warehouse Location Problem Revisited." International Journal of Physical Distribution and Materials Management, 17, 3 (1987), 56-59.

Lee, S. M. & Morris, R. "Integer Goal Programming Methods." TIMS Studies in the Management Sciences, 6 (1977), 273-289.

Logendran, R. & Terrell, M. P. "Uncapacitated Plant Location-Allocation Problems with Price Sensitive Stochastic Demands." Computers & Operations Research, 15, 2 (1988), 189-198.

Loh, A. "Multiple Commodity and Multiple Stage Capacitated Dynamic Facility Location." (Ph.D. dissertation, University of Houston, 1983).

Lynch, A. A. "Environment and Labor Quality Take Top Priority in Site Selection." Industrial Development, 142, 2 (1973), 13-15.

MacCrimmon, K. R. "An Overview of Multiple Objective Decision Making." in Multiple Criteria Decision Making, Cochrane, J. L., and Zeleny, M. (Eds.), University of South Carolina Press (1973), 18-44.

Manne, A. S. "Plant Location Under Economies of Scale-Decentralization and Computation." Management Science, 11, 2 (1964), 213-235.

Markland, R. E. "Analyzing Geographically Discrete Warehousing Networks by Computer Simulation." Decision Science, 4, 2, (1973), 216-236.

Marks, D. H., ReVelle, C. S., & Liebman, J. C. "Mathematical Models of Location: A Review." Journal of Urban Planning and Development Division, 96 (1970), 81-93.

Masud, A. S., & Hwang, C. L. "Interactive Sequential Goal Programming." Journal of the Operational Research Society, 32 (1981), 391-400.

McGinnis, L. F., & White, J. A. "A Single Facility Rectilinear Location Problem with Multiple Criteria," Transportation Science, 12, 3 (1978), 217-231.

Narula, S. C. "Hierarchical Location Allocation Problems: A Classification Scheme." European Journal of Operational Research, 15 (1984), 93-99.

Nauss, R. M. "An Improved Algorithm for the Capacitated Facility Location Problem." Journal of Operational Research Society, 29, 12 (1978), 1195-1201.

Neebe, A. W., & Khumawala, B. M. "An Improved Algorithm for the Multi-Commodity Location Problem." Journal of Operational Research Society, 32, 2 (1981), 143-149.

Olson, D. L. "Comparison of Four Goal Programming Algorithms." Journal of Operational Research Society, 35, 4 (1984), 347-354.

Park, Y. B. "The Solution of Vehicle Routing Problems in a Multiple Objective Environment" (Ph.D. dissertation, Oklahoma State University, Stillwater, 1984).

Petty, J. W., & Bowlin, O. D. "The Financial Manager and Quantitative Decision Models." Financial Management, 5, 4 (1976), 32-41.

Rasaratnam, L. "Plant Location-Allocation Problems with Price Sensitive Demands." (Ph.D. dissertation, Oklahoma State University, Stillwater, 1984).

Ravindranath, K., Vrat, P., & Singh, N. "Bicriteria Single Facility Rectilinear Location Problems in the Presence of a Single Forbidden Region." The Journal of Operational Research Society of India, 22, 1 (1985), 1-16.

ReVelle, C. S., Marks, D., & Liebman, J. C. "An Analysis of Private and Public Sector Location Models." Management Science, 16 (1970), 692-707.

ReVelle, C. S., & Swain, R. W. "Central Facility Locations." Geographical Analysis, 2, 1 (1970), 30-42.

Ross, T. G., & Soland, R. M. "A Multicriteria Approach to the Location of Public Facilities." European Journal of Operations Research, 4, 5 (1980), 307-321.

Sa, G. "Branch-and-Bound and Approximate Solutions to the Capacitated Plant-Location Problem." Operations Research, 17 (1969), 1005-1016.

Schniederjans, M. J. & Kwak, N. K. "An Alternative Solution Method for Goal Programming Problems: A Tutorial." Journal of Operational Research Society, 33, (1982), 247-251.

Scott, A. J. "Location-Allocation Systems: A Review." Geographical Analysis, 2 (1970), 95-119.

Shannon, R. E. & Ignizio, J. P. "A Heuristic Programming Algorithm for Warehouse Location." AIIE Transactions, 2, 4 (1970), 334-339.

Shycon, H. N. & Maffei, R. B. "Simulation-Tool for Better Distribution." Harvard Business Review, Nov. Dec. (1960), 65-75.

Sicsu, A. L. "The Capacitated Location-Allocation Problem with Price-Sensitive Demands." (Ph.D. dissertation, Stanford University, 1979).

Silver, E. A. & Peterson, R. (1985). Decision Systems for Inventory Management and Production Planning (2nd ed.). Wiley, Chapter 10.

Sinha, S. B. & Sastry, S. V. C. "A Goal Programming Model for Facility Location Planning." Socio-economic Planning Sciences, 21, 4 (1987), 251-255.

Soland, R. M. "Optimal Facility Location with Concave Costs." Operations Research, 22 (1974), 373-385.

Spielberg, K. "Algorithms for the Simple Plant Location Problem With Some Side Conditions." Operations Research, 17 (1969)a, 85-111.

Spielberg, K. "Plant Location with Generalized Search Origin." Management Science, 16, 3 (1969)b, 165-178.

Steuer, R. E. Multiple Criteria Optimization: Theory, Computation, and Application, Wiley, New York, 1986.

Student, K. R. "Cost vs. Human Values in Plant Location." Business Horizons, 19, 2 (1976), 5-14.

Sule, D. R. "Simple Methods for Uncapacitated Facility

Location/Allocation Problems." Journal of Operations Management, 1, 4 (1981), 215-223.

Sweeney, D. J. & Tatham, R. L. "An Improved Long Run Model for Multiple Warehouse Locations." Management Science, 22, 7, (1976), 748-758.

Taha, H. A. (1982). Operations Research (3rd ed.). Macmillan, pp. 784-788.

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

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

Tapiero, C. S. "Transportation-Location-Allocation Problems Over Time." Journal of Regional Science, 11, 2 (1971), 377-384.

Tcha, D. & Lee, B. "A Branch-and-Bound Algorithm for the Multi-Level Uncapacitated Facility Location Problem." European Journal of Operational Research, 18 (1984), 35-43.

Tompkins, J. A. & White, J. A. (1984). Facility Planning. Wiley, pp. 487-526.

Truscott, W. G. "The Treatment of Revenue Generation Effects of Facility Location." AIIE Transactions, 7, 1 (1975), 63-69.

Van Roy, T. J. "Cross Decomposition for Mixed Integer Programming." Mathematical Programming, 25 (1983), 46-63.

Van Roy, T. J. "A Cross Decomposition Algorithm for Capacitated Facility Location." Operations Research, 34, 1 (1986), 145-163.

Van Roy, T. J. & Erlenkotter, D. "A Dual-Based Procedure for Dynamic Facility Location." Management Science, 28, 10 (1982), 1091-1105.

Walker, W. E. "A Heuristic Adjacent Extreme Point Algorithm for the Fixed Charge Problem." Management Science, 22, 5 (1976), 587-596.

Warszawski, A. "Multi-Dimensional Location Problems." Operational Research Quarterly, 24, 2 (1973), 165-179.

Warszawski, A. & Peer, S. "Optimizing the Location of Facilities on a Building Site." Operational Research Quarterly, 24 (1973), 35-44.

Weber, A. (1909). Uber den Standort der Industrien, Tubingen, Translated as Alfred Weber's Theory of Location of Industries by C.J. Friedrich, Chicago: University of Chicago Press.

Wesolowsky, G. O. "Location in Continuous Space." Geographical Analysis, 5, 2 (1973), 95-112.

Wesolowsky, G. O. & Truscott, W. G. "The Multiperiod Location-Allocation Problem with Relocation of Facilities." Management Science, 22, 1 (1975), 57-65.

White, J. A. & Case, K. E. "On Covering Problems and the Central Facilities Location Problem." Geographical Analysis, 6 (1974), 281-293.

Wilson, D. "An a Priori Bounded Model for Transportation Problems with Stochastic Demand and Integer Solutions." AIIE Transactions, 4, 3 (1972), 186-193.

Zeleny, M. Multiple Criteria Decision Making, McGraw-Hill, New York, NY, 1982.

APPENDIXES

APPENDIX A

TEST MODELS DATA AND RESULTS

TABLE A.1

TEST PROBLEM 1 INPUT DATA

(Green, Kim, and Lee 1981)

NO[1]	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	RHS	TYPE
1	85	438	165	275	63	155	50	77	298	90	120	74	750	B[2]
2	126	210	363	240	122	340	320	203	210	135	388	177	900	B
3		1	1										1	B
4	1	1											1	B
5								1	1				1	B
6				1							1		1	B
7					1						-1		0	B
8	1	1	1							1			2	B
9						1	1	1		1			1	B
10	1.8	1.6	2.1	1.9	2.1	4.8	4.1	4.1	3.5	4.4	3.3	1.2	0	B

[1]The constraint numbers correspond to the subscript of the deviational variables.
[2]Type B refers to goal constraints.

Achievement Function:

$$\text{Min } Z = P_1 d_1^- + P_2 d_2^+ + P_3 \sum_{i=3}^{6} d_i^+ + P_4 d_7^+ + P_5 d_8^- + P_6 d_9^+ + P_7 d_{10}^+ .$$

TABLE A.2

TEST PROBLEM 2 INPUT DATA

(Lee and Franz 1979)

NO	X_{11}	X_{21}	X_{31}	X_{41}	X_{51}	X_{12}	X_{22}	X_{32}	X_{42}	X_{52}	X_{13}	X_{23}	X_{33}	X_{43}
1	1	1	1	1	1									
2						1	1	1	1	1				
3											1	1	1	1
4														
5														
6	1					1					1			
7		1					1					1		
8			1					1					1	
9				1					1					1
10					1					1				
11											1	1		
12	200	180	50	35	210	110	90	200	160	35	40	40	225	250
13	200	280	50	35	210	110	90	200	160	35	40	40	225	250
14	1					1					1			
15		1					1					1		
16			1					1					1	
17				1					1					1
18					1					1				
19														
20														
21														
22														
23														

TABLE A.2 (Continued)

NO	X_{53}	X_{14}	X_{24}	X_{34}	X_{44}	X_{54}	Y_1°	Y_2°	Y_3°	Y_4°	Y_5°	RHS	TYPE
1												400	B
2												300	B
3	1											200	B
4		1	1	1	1	1						100	B
5							825	750	600	600	650	1300	B
6		1										600	B
7			1									600	B
8				1								480	B
9					1							480	B
10	1					1						800	B
11												50	B
12	125	90	80	25	35	50	825[a]	750[a]	600[a]	600[a]	650[a]	0	B
13	125	90	80	25	35	50						0	B
14		1					-3000					0	L[b]
15			1					-3000				0	L
16				1					-3000			0	L
17					1					-3000		0	L
18	1					1					-3000	0	L
19							1					1	L
20								1				1	L
21									1			1	L
22										1		1	L
23											1	1	L

[o]These variables are specified to be integers (0 or 1).
[a]These numbers must be multiplied by 1000.
[b]Type L refers to less than or equal to constraints.

Achievement Function:

$$\text{Min } Z = P_1 \sum_{i=1}^{4} d_i^- + P_2 d_5^+ + P_3 \sum_{i=6}^{10} d_i^+ + P_4 d_{11}^- + P_5 d_{12}^+ + P_6 d_{13}^+ \ .$$

TABLE A.3

TEST PROBLEM 3 INPUT DATA

(Lee, Green, and Kim 1981)

NO	X_{11}	X_{21}	X_{31}	X_{41}	X_{51}	X_{61}	X_{12}	X_{22}	X_{32}	X_{42}	X_{52}	X_{62}	X_{13}	X_{23}
1	1	1	1	1	1	1								
2							1	1	1	1	1	1		
3													1	1
4														
5	1	1												
6														
7														
8														
9	1						1						1	
10		1						1						1
11			1						1					
12				1						1				
13					1						1			
14						1						1		
15	200	180	50	35	210	180	110	90	200	160	35	120	40	40
16	200	280	50	35	210	180	110	90	200	160	35	120	40	40
17														
18														
19														
20														
21														
22														
23	1						1						1	
24		1						1						1
25			1						1					
26				1						1				
27					1						1			
28						1						1		

TABLE A.3 (Continued)

NO	X_{33}	X_{43}	X_{53}	X_{63}	X_{14}	X_{24}	X_{34}	X_{44}	X_{54}	X_{64}	Y_1^o	Y_2^o	Y_3^o
1													
2													
3	1	1	1	1									
4					1	1	1	1	1	1			
5													
6											825	750	600
7											70	75	65
8											1	1	1
9					1								
10						1							
11	1						1						
12		1						1					
13			1						1				
14				1						1			
15	225	250	125	60	90	80	25	35	50	50	825*	750*	600*
16	225	250	125	60	90	80	25	35	50	50			
17											1		
18												1	
19													1
20													
21													
22													
23					1						-3000		
24						1						-3000	
25	1						1						-3000
26		1						1					
27			1						1				
28				1						1			

TABLE A.3 (Continued)

NO	Y_4°	Y_5°	Y_6°	RHS	TYPE
1				580	B[1]
2				420	B
3				260	B
4				150	B
5				50	B
6	600	650	550	2000	B
7	80	50	70	600	B
8	1	1	1	3	B
9				600	B
10				600	B
11				500	B
12				500	B
13				800	B
14				800	B
15	600*	650*	550*	0	B
16				0	B
17				1	L[2]
18				1	L
19				1	L
20	1			1	L
21		1		1	L
22			1	1	L
23				0	L
24				0	L
25				0	L
26	-3000			0	L
27		-3000		0	L
28			-3000	0	L

[1]Type B refers to goal constraints.
[2]Type L refers to less than or equal to constraints.
°These variables are specified to be integers (0 or 1).
*These number must be multiplied by 1000.

Achievement Function:

$$\text{Min } Z = P_1 \sum_{i=1}^{4} d_i^- + P_2 d_5^- + P_3 d_6^+ + P_4 d_7^- + P_5 d_8^- + P_6 \sum_{i=9}^{14} d_i^+ + P_7 d_{15}^+ + P_8 d_{16}^+ .$$

APPENDIX B

APPROXIMATION TO THE CDF AND INVERSE CDF OF STANDARD NORMAL DISTRIBUTION

APPROXIMATION TO THE CDF AND INVERSE CDF OF STANDARD NORMAL DISTRIBUTION

Assume random variable X is normally distributed with mean μ and variance σ^2, then its density function can be written as follows:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right], \quad -\infty < x < \infty$$

where $-\infty < \mu < \infty$ and $\sigma^2 > 0$. A substitution of variables

$$Z = (x-\mu)/\sigma \qquad -\infty < Z < \infty$$

results in Z's normally distributed with mean zero and standard deviation 1, i.e. standard normal distribution. Next, the cumulative distribution function of random variable Z is:

$$Y = P(z) = \text{Prob}\ (Z \le z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} \exp\left(-\frac{u^2}{2}\right) du$$

and the inverse cumulative distribution function is:

$$Z = F^{-1}(y).$$

The value of the above inverse CDF for various y values or the value of the above probability distribution function for different z values are available from normal tables. However two formulas to approximate the inverse CDF and CDF is given by Hastings (1955) and are as follows:

$$Z = w - \sum_{i=0}^{2} a_i w^i \;/\; \sum_{i=0}^{3} b_i w^i$$

where:

$$w = \sqrt{\ln\ (1/p^2)} \qquad \begin{cases} p = p(z) & \text{for } 0 < p(z) \le 0.5 \\ p = 1-p(z) & \text{for } p(z) > 0.5 \end{cases}$$

$a_0 = 2.515517$, $b_0 = 1.0$

$a_1 = 0.802853$, $b_1 = 1.432788$

$a_2 = 0.010328$, $b_2 = 0.189269$, $b_3 = 0.001308$.

and the maximum, error is 0.00045.

Next, the approximation for CDF is:

$$P(z) = 1 - f(z) \sum_{i=1}^{5} a_i w^i \quad , \quad z \geq 0$$

where:

$$f(z) = \exp\left(-\frac{z^2}{2}\right) / \sqrt{2\pi}$$

$$w = [1+(0.2316419)\ z\ ]^{-1}$$

$a_1 = 0.3193815$

$a_2 = -0.3565638$

$a_3 = 1.781478$

$a_4 = -1.821256$

$a_5 = 1.330274$

and the maximum error is 0.0000007.

APPENDIX C

PASCAL PROGRAM SOURCE CODES

TABLE C.1

INDEX TO PROGRAM UNITS AND PROCEDURES

Units and Procedures Name	Page
PROGRAM SMLAP	193
EXECUTEPROGRAM	193
Setup	193
FindPivotColumn	196
FindPivotRow	198
Update	198
ComputeResult	199
ExecuteProgram	200
BRANCHBOUND	202
IndexFrac	202
UpdateUB	203
InitialUB	204
CheckPriority	205
GetLeaf	206
BranchBound	206
Smlap	209
UNIT smlautil	219
PATTERNSEARCH	220
Normal CDF	221
MultiObjective	221
Compare	221
PatternSearch	223
Uppercase	226
Cursor	227
LineDraw	227
Blank	227
DrawBox	228
Message	228
InputInteger	229
InputReal	230
InputChar	231
LineCount	231
SaveInput	232
SortCoef	232
Unit dbasutil	233
NInvCDF	233
UInvCDF	233
CREATEDATABASE	234
Stochastic	234
ChanceConst	237
InputData	240
CreateDataBase	244

TABLE C.1 (continued)

Units and Procedures Name	Page
LoadDataBase	246
SaveDataBase	247
DisplayDataBase	249
OutputResult	253
FinalZj_Cj	254
SENSIANALY	255
ListAchievmt	256
TradeoffAnly	256
ChangePri	258
ChangeRhs	259
SensiAnaly	262

```
{------------------------------------------------------------------------}
{                                                                        }
{                              S M O L A P                               }
{                                                                        }
{ This interactive, menu driven program allows for the solution and      }
{ sensitivity analysis of the integer, stochastic, multicriterion        }
{ optimization problems.  Additionally, a nonlinear routine is provided  }
{ for the solution of the model B facility location-allocation problem   }
{ developed in this research.  The program is written in Borland's       }
{ Turbo Pascal 5.0.                                                      }
{                                                                        }
{------------------------------------------------------------------------}
{                              Written By                                }
{                                                                        }
{                            MORTEZA ABTAHI                              }
{                                                                        }
{           School of Industrial Engineering and Management              }
{                      Oklahoma State University                         }
{                    Stillwater, Oklahoma  74078                         }
{                              July 1989                                 }
{------------------------------------------------------------------------}
{                                                                        }
{ Units and Procedures                          Descriptions             }
{ --------------------                          ------------             }
{ PROGRAM smlap                                                          }
{   ExecuteProgram.......Finds a continuous solution.                    }
{      Setup.............Prepares data for preemptive goal programming.  }
{      FindPivotColumn...Finds the pivot column.                         }
{      FindPivotRow......Finds the Pivot row.                            }
{      Update............Updates the modified simplex tableau.           }
{      ComputeResult.....Calculates the output variables.                }
{   BranchBound                                                          }
{      IndexFrac.........Reports the index of non-integer variable.      }
{      UpdateUB..........Updates the upper bound.                        }
{      InitialUB.........Calculates the initial upper bound.             }
{      CheckPriority.....Compares current priorities with upper bound.   }
{      GetLeaf...........Selects a node on the branch-and-bound tree.    }
{                                                                        }
{ UNIT smlautil                                                          }
{   PatternSearch........Modified Hooke and Jeeves pattern search.       }
{      NormalCDF.........Computes 1-CDF and density of the normal dist.  }
{      MultiObjective....Calculates multiple objectives for a point.     }
{      Compare...........Tests if current solution is better than UB    }
{   Uppercase............Returns upper case of the input string.         }
{   Cursor...............Turns the cursor On or Off.                     }
{   LineDraw.............Draws a line.                                   }
{   Blank................Blanks a specified entry.                       }
{   DrawBox..............Draws a box on the screen.                      }
{   Message..............Displays a message on the last line of screen.  }
{   InputInteger.........Accepts a valid integer number.                 }
{   InputReal............Accepts a valid real number.                    }
{   InputChar............Accepts a valid character.                      }
{   LineCount............Counts number of lines displayed on screen.     }
```

```
{   SaveInput.....Saves the necessary input information.        }
{   SortCoef.....Sorts the input coefficients columnwise.      }
{                                                              }
{ UNIT dbasutil                                                }
{   NInvCDF.....Inverse cumulative density function of normal.}
{   UInvCDF.....Inverse cumulative density function of uniform}
{   CreateDataBase.....Allows the user to input a new model.   }
{      Stochastic.....Accepts inputs for stochastic model      }
{      ChanceConst.....Accepts inputs for chance-constrained model }
{      InputData.....Inquires input data from the user.        }
{   LoadDataBase.....Loads an existing model from disk.        }
{   SaveDataBase.....Saves the current model to disk.          }
{   DisplayDataBase.....Displays the current model.            }
{   OutputResult.....Sends results to screen or printer.       }
{   FinalZj_Cj.....Calculates and stores the optimum Zj-Cj matrix}
{   SensiAnaly.....Performs various sensitivity analysis.      }
{      ListAchievmt.....Lists the achievements.                }
{      TradeoffAnly.....Performs trade-off analysis.           }
{      ChangePri.....Changes the priority structure.           }
{      ChangeRhs.....Changes deterministic or probabilistic RHSs.  }
{--------------------------------------------------------------}
{                                                              }
{ Definition of Variables                                      }
{ ----------------------                                       }
{ coef       - array of coefficient.                           }
{ coefficient - record containing the row number, column number, and  }
{                 value of technological coefficients.         }
{ csign      - array containing the sign (B, E, G, L) of constraints.  }
{ elapsed    - CPU time in second to find a solution.          }
{ filename   - string representing the models name.            }
{ obj        - array of objective record.                      }
{ objective - record containing sign, row number, priority, and weight }
{               of deviations in the achievement function.     }
{ ncols      - number of columns (negative deviations, positive }
{               deviations, and decision variables).           }
{ nelemty    - number of elementary matrices.                  }
{ nIteration- number of iterations.                            }
{ npdvs      - number of positive deviations.                  }
{ nprt       - number of priorities.                           }
{ nrows      - number of rows.                                 }
{ ntc        - number of technological variables.              }
{ nvars      - number of decision variables.                   }
{ opyZjCj    - matrix of optimum Zj-Cj values.                 }
{ pcol       - index for the pivot column.                     }
{ prow       - index for the pivot row.                        }
{ pw         - array of value.  contains priority number and weight for }
{               all the variables.                             }
{ pwBasis    - array of value.  Contains priority number and weight for }
{               variables in the basis.                        }
{ rhs        - array of right hand side values                 }
{ rhsF       - array of values for optimal solution.           }
{ tprt       - number of deviations in achievement function.   }
{ value      - record containing priority number and weight of }
```

```
{                deviational variables                                    }
{ ubUpdate  - number of times upper bound is updated in integer routine}
{ zmax      - maximum Zi-Cj value.                                        }
{-------------------------------------------------------------------------}

PROGRAM smlap;

   USES CRT,DOS,PRINTER,smlaUtil,dbasUtil;
   LABEL a;
   VAR
      tempFilename:                    STRING[10];
      x1,y1:                           BYTE;

{*************************************************************************}
{*                          ExecuteProgram                               *}
{*************************************************************************}

PROCEDURE ExecuteProgram;
   LABEL a,b;
   VAR
      store:        BYTE;
      priority1:    EXTENDED;

{*************************************************************************}
{*                              Setup                                    *}
{*************************************************************************}

PROCEDURE setup;
   BEGIN

{ Calculate number of positive deviational variables (surplus) }

     npdvs:= 0;
     FOR i:= 1 TO nrows DO
           IF((csign[i] = 'G')OR(csign[i] = 'B')) THEN INC(npdvs,1);

     { Calculate number of columns }

     ncols := nrows + npdvs + nvars;

     { Initialization phase }

     FOR j:= 1 TO ncols DO
        BEGIN
           pw[j].priority:= 0;
           pw[j].weight:= 0.0;
           currentBasic[j]:= 0;{Will contain the index of basic columns}
        END;

     FOR i:= 1 TO nrows DO
              pdevc[i] := 0;

     { Set up the initial tableau.  A negative deviational variable
```

```
      (artificial slack) will be added to "G" and "E" type constraints
      to form the initial basis.  These variables will be placed at
      priority 1 for minimization. In this case all other priorities
      will be shifted down by 1.  priority 1 must be completely
      satisfied (minimized to zero) for a feasible solution to exist. }

    flg1:= FALSE; {Indicates if artificial slacks are added to problem}
    npdvs:= 0;
    FOR i:= 1 TO nrows DO
       BEGIN
          basicCol[i] := i;                        { Array of size nrows }
          CASE csign[i] OF
             'E':
{ System constraint is of strict equality type.  No deviations present }
                BEGIN
                   pw[i].priority := 1;
                   pw[i].weight   := 1.0;
                   flg1 := TRUE;
                END;
            'G':
{ System constraint is of >= type.  Only positive deviation is present }
                BEGIN
                   pw[i].priority := 1;
                   pw[i].weight   := 1.0;
                   INC(npdvs,1);
                   pdevc[i] := nrows + npdvs;
                   flg1 := TRUE;
                END;

           'B':
{ Goal constraint.  Both positive and negative deviations are present }
                BEGIN
                   INC(npdvs,1);
                   pdevc[i]:=nrows + npdvs;
                END;

{ Otherwise it is a system constraint of 'L' (<= ) type.  Only negative
  deviation is present.  No action is required. }

          END                                    { End of case statement }
       END;

    { If we have to include negative deviational variables
      (artificials) in case of 'E' and 'G' type constraints to form the
      initial basis, then we need to minimize these variables to zero
      at priority 1.  So, we need to shift other priorities down by 1.}

    FOR i:= 1 TO tprt DO
       BEGIN
          rown := obj[i].row;
          CASE obj[i].sign OF
             'N':
                 BEGIN
```

```
                IF (flg1) THEN
                   pw[rown].priority:= obj[i].priority + 1
                ELSE
                   pw[rown].priority:= obj[i].priority;
                pw[rown].weight:= obj[i].weight;
             END;
          'P':
             BEGIN
                IF (flg1) THEN
                   pw[pdevc[rown]].priority:= obj[i].priority + 1
                ELSE
                   pw[pdevc[rown]].priority:= obj[i].priority;
                pw[pdevc[rown]].weight:= obj[i].weight;
             END;
       END                                          { End of case }
   END;

IF(flg1)THEN INC(nprt,1);    { Adjust for the additional priority }

{ Set the priorities and weights of the initial basis }

FOR i:= 1 TO nrows DO
         pwBasis[i]:= pw[i];   { Assigns both priority and weight }

{ Information for negative deviations }

FOR i:=1 TO nrows DO
   BEGIN
      currentBasic[i]:= i;
      avalue[i]:= 1.0;
      arow[i]:=i;
      n[i]:= 1;
   END;

{ Information for positive deviations }

FOR j:=1 TO npdvs DO
   BEGIN
      FOR i:=1 TO nrows DO
         BEGIN
            IF(pdevc[i] = nrows+j) THEN
               BEGIN
                  avalue[nrows+j]:=-1.0;
                  arow[nrows+j]:=i;
                  n[nrows+j]:=1;
               END;
         END;
   END;

{ Information for decision variables }

c:=nrows+npdvs;
FOR i:=1 TO nvars DO
```

```
            n[c+i]:=num[i];

      FOR j:=1 TO ntc DO
         BEGIN
            INC(c,1);
            avalue[c]:=coef[j].value;
            arow[c]:=coef[j].row;
         END;

      { Find starting position of each tableau column in 'avalue' array }

      start[1]:=1;
      FOR i:=2 TO ncols DO
              start[i]:=n[i-1]+start[i-1];

      { Set the right hand side values }

      FOR i:=1 TO nrows DO
              rhsF[i]:= rhs[i];
   END;                                                  { End of setup }

{*************************************************************************}
{*                          FindPivotColumn                              *}
{*************************************************************************}

PROCEDURE FindPivotColumn;
   LABEL s;
   VAR
      zjcj,tempZmax:                    EXTENDED;

   BEGIN
     zmax:=0.0;
     pcol:=0;

     FOR k:=1 TO ncols DO
        BEGIN

        {Do not consider the column if its variable is already in basis}

           IF(currentBasic[k] <> 0) THEN GOTO s;
           WITH pw[k] DO
              BEGIN
                 IF(((priority > 0)AND(priority < p ))OR
                    ((priority = p)AND( weight > lw )))THEN GOTO s;
              END;

           { Priority index of the current potential entering variable
             is either zero, higher than current priority being
             satisfied, or equal to with lower weight }

           { Initialize the potential new basic column }

           FOR i:=1 TO nrows DO
```

```
            y[i]:=0.0;

         { Construct the original a column }

         FOR i:=start[k] TO start[k]+n[k]-1 DO
            y[arow[i]]:=avalue[i];

         { Update the 'a' column }

         IF (nelemty <> 0)THEN
            BEGIN
               FOR i:=1 TO nelemty DO
                  BEGIN
                     ar:=y[position[i]];
                     y[position[i]]:=0.0;          { This is the a-hat }
                     IF(ABS(ar) > 1.0E-10) THEN
                        BEGIN
                           indx1:=ElCount[i];
                           indx2:=ElCount[i+1]-1;
                           FOR j:=indx1 TO indx2 DO
                              BEGIN
                                 ij:=ElRow[j];
                                 y[ij]:=y[ij]+ar*ElValue[j];
                              END;
                        END;
                  END;
            END;

         tempzmax:=0.0;

         { Calculate zj-cj for the current variable and priority }

         FOR i:=1 TO nrows DO
               IF(pwBasis[i].priority = p) THEN
                  tempzmax:=tempzmax+pwBasis[i].weight * y[i];

      { If p equal to priority of variable at column k, 'Cj' is nonzero.
        Therefore, we have to subtract Cj to find Zj-Cj }

         IF(pw[k].priority = p)THEN tempzmax:=tempzmax-pw[k].weight;
         IF((tempzmax <= 1.0E-10)OR(tempzmax <= zmax))THEN GOTO s;

      { Check If the entering variable deteriorate higher priority goals}

         IF(p-1 > 0)THEN                    { For priority 2 or higher }
            BEGIN
               { Consider all higher priorities up to p }
               FOR i:=1 TO p-1 DO
                  BEGIN
                     zjcj:=0.0;
                     FOR j:=1 TO nrows DO
                           IF(pwBasis[j].priority =i)THEN
                              zjcj:=zjcj+pwBasis[j].weight*y[j];
```

```
                    IF(zjcj < 0.0)THEN GOTO s;
                 END;
              END;

      { Update the maximum zjcj and its corresponding column and index }

           zmax:=tempzmax;
           FOR i:=1 TO nrows DO
                  x[i]:=y[i];
           pcol:=k;                                        { Pivot column }
s:      END;                                          { End of column loop }
   END;                                                   { End of column }

{*************************************************************************}
{*                            FindPivotRow                               *}
{*************************************************************************}

PROCEDURE FindPivotRow;
   VAR
      mRatio, ratio, mWeight:    EXTENDED;
      mPriority:                 BYTE;

   BEGIN

     { Initialization }

     mRatio:=1.0e20;
     mPriority:=0;
     mWeight:=0.0;
     prow:=0;

     FOR i:=1 TO nrows DO
        BEGIN
           IF(x[i] >1.0E-10)THEN
              BEGIN
                 ratio:=rhsF[i]/x[i];
                 IF((ratio<mRatio)OR
                 ((ratio=mRatio)AND(pwBasis[i].priority<mPriority))OR
                 ((ratio=mRatio)AND(pwBasis[i].priority=mPriority)AND
                 (pwBasis[i].weight>mWeight)))THEN
                    BEGIN
                       mRatio:=ratio;
                       prow:=i;                          { Pivoting row }
                       mPriority:=pwBasis[i].priority;
                       mWeight:=pwBasis[i].weight;
                    END;
              END;
        END;
   END;                                          { End of procedure row }

{*************************************************************************}
{*                              Update                                   *}
{*************************************************************************}
```

```
PROCEDURE update;
   VAR
      yrk, d:          EXTENDED;
      count:           INTEGER;

   BEGIN

      { This procedure updates the right hand side values and
        generates a new elementary matrix.  The complete source code
        for this procedure is not provided. }

   END;                                          { End of procedure update }

{*************************************************************************}
{*                              ComputeResult                            *}
{*************************************************************************}

PROCEDURE ComputeResult;
   VAR
      tpos:                 ARRAY[1..50] OF EXTENDED;

   BEGIN

      { If artificial slacks were added, priority 1 represents them and
        its value is zero at this stage. So, our original priorities
        start from index 2. }

      c:=1;
      IF(flg1)THEN c:=0;

      FOR p:=2-c TO nprt DO
         BEGIN
            prty[p]:=0.0;
            FOR i:=1 TO nrows DO
                IF(pwBasis[i].priority = p)THEN
                   prty[p]:=prty[p]+pwBasis[i].weight*rhsF[i];
         END;

      IF(flg1)THEN
         BEGIN
            DEC(nprt,1);                  { Original number of priority }
            FOR i:=1 TO nprt DO
               prty[i]:=prty[i+1];
         END;

      { Decision variables }

      j:=0;
      FOR i:=nrows+npdvs+1 TO ncols DO
         BEGIN
            INC(j,1);
            IF(currentBasic[i] = 0)THEN
```

```
            decn[j]:=0.0
         ELSE
            decn[j]:=rhsF[currentBasic[i]];
      END;

   { Negative deviations }

   FOR i:=1 TO nrows DO
      BEGIN
         IF(currentBasic[i] = 0)THEN
            neg[i]:=0.0
         ELSE
            neg[i]:=rhsF[currentBasic[i]];
      END;

    { Positive Deviations }

    j:=0;
    FOR i:=nrows+1 TO nrows+npdvs DO
       BEGIN
          INC(j,1);
          IF(currentBasic[i] = 0)THEN
             tpos[j]:=0.0
          ELSE
             tpos[j]:=rhsF[currentBasic[i]];
       END;
    j:=0;
    FOR i:=1 TO nrows DO
       BEGIN
          IF((csign[i] = 'G')OR(csign[i] = 'B'))THEN
             BEGIN
                INC(j,1);
                pos[i]:=tpos[j];
             END
          ELSE
             pos[i]:=0;
       END;
END;                                { End of procedure ComputeResult }

{.....................................................................}
{.                          ExecuteProgram                           .}
{.....................................................................}

BEGIN
  setup;                                        { Call Procedure setup }

  { Initialize index variables }

  ElCount[1]:=1;
  nelemty:=0;                           { Number of elementary matrices }
  feasible:=TRUE;    { Indicates if the current solution is feasible }

  FOR p:= 1 TO nprt DO
```

```
        BEGIN

      { If artificial slacks were added for initial basis, then the
        added priority 1 must be zero For a feasible solution to exist }

          IF((flg1)AND(p=2))THEN
             BEGIN

                { Calculate priority 1 }

                priority1:=0.0;
                FOR i:=1 TO nrows DO
                   IF(pwBasis[i].priority=1)THEN
                      priority1:=priority1+pwBasis[i].weight*rhsF[i];

                IF(priority1<>0)THEN
                   BEGIN
                      feasible:=FALSE;
                      nprt:=orig_nprt;
                      EXIT;
                   END;
             END;

{ Find the largest weight associated with highest priority in the basis}

          lw:=0.0;
          {flg2 indicates a match between current priority & basis pr.}
b:        flg2:=FALSE;

          FOR i:= 1 TO nrows DO
             BEGIN
                IF(pwBasis[i].priority = p) THEN
                   BEGIN
                      flg2:=TRUE;
                      IF(pwBasis[i].weight > lw) THEN
                         lw:=pwBasis[i].weight;
                   END;
             END;

          IF(NOT flg2) THEN GOTO a;        { Examine the next priority }

          { Find the pivot column }

          FindPivotColumn;                      { Call procedure column }
          IF(pcol=0)THEN GOTO a;

          { Find the pivot row }

          FindPivotRow;                            { Call procedure row }
          IF(prow=0)THEN GOTO a;

          INC(nIteration,1);              { Update number of iterations }
          GOTOXY(1,3);
```

```
            WRITE(' Iteration........',nIteration:4);

            { Update the basis by introducing the new variable }

            store:=basicCol[prow];
            basicCol[prow]:=pcol;
            currentBasic[pcol]:=prow;
            currentBasic[store]:=0;
            pwBasis[prow]:=pw[pcol];

            { Update the tableau }

            update;                                  { Call procedure update }

            GOTO b;                  { Next iteration for current priority }
a:        END;                                    { End of the priority loop }
      ComputeResult;
    END;                                          { End of ExecuteProgram }

{*************************************************************************}
{*                              BranchBound                              *}
{*************************************************************************}

PROCEDURE BranchBound;
    LABEL a, b;
    TYPE
       nodePtr =                          ^leafNode;

       leafNode = RECORD
          index:                          ARRAY[1..20] OF BYTE;
          sign:                           ARRAY[1..20] OF CHAR;
          rhs:                            ARRAY[1..20] OF EXTENDED;
          nac:                            BYTE;
          prty:                           ARRAY[1..20] OF EXTENDED;
       END;

    VAR
       index,leafCount,pdx:               BYTE;
       rhsL,rhsR,dif,lhs:                 EXTENDED;
       decnUB,pd,nd:                      ARRAY[1..50] OF EXTENDED;
       ptrArray:                          ARRAY[1..200] OF nodePtr;
       tempPtr,leftPtr,rightPtr:          nodePtr;
       potlLeaf, flg4:                    BOOLEAN;

{*************************************************************************}
{*                              indexFrac                                *}
{*************************************************************************}

FUNCTION indexFrac:BYTE;

    { This function returns index of the variable with largest fraction }

    VAR
```

```
      ndx:                              BYTE;
      maxFrac,tFrac:                    EXTENDED;

   BEGIN
      maxFrac:=0.0;
      ndx:=0;

      { Check to see if any of basic variables are fractional }

      FOR i:=1 TO nvars DO
        BEGIN
          IF(decnType[i]='I')THEN
            BEGIN
              tFrac:=FRAC(decn[i]);
              IF((tFrac >= 0.001)AND(tFrac <= 0.999)) THEN
                BEGIN
                  IF(tFrac > maxFrac) THEN
                    BEGIN
                      maxFrac:= tFrac;
                      ndx:=i;
                    END;
                END;
            END;
        END;
      indexFrac:=ndx;
   END;

{*************************************************************************}
{*                              updateUB                                 *}
{*************************************************************************}

PROCEDURE updateUB;

{ This updates the upper bound and saves its corresponding integer
  solution. }

   BEGIN
      FOR i:=1 TO nprt DO
         prtyUB[i]:=prty[i];
      FOR i:=1 TO nvars DO
         intDecn[i]:=decn[i];
      FOR i:=1 TO nrows DO
         BEGIN
            intNeg[i]:=neg[i];
            intPos[i]:=pos[i];
         END;
      keepFlg1:=flg1;
      FinalZjCj;
      INC(ubUpdate,1);
      GOTOXY(1,6);
      WRITELN(' U.B. Updates.....',ubUpdate:4);
   END;
```

```
{*************************************************************************}
{*                              initialUB                                *}
{*************************************************************************}

PROCEDURE initialUB;

{ This procedure calculates an initial upper bound for the branch and
  bound.  It is possible to establish a tighter bound for pure goal
  constraints because constraints remain feasible by selecting arbitrary
  integer variables. }

   BEGIN
      { Find out if all constraints are of goal type }

      flg4:=TRUE;      { Indicator for pure goal constraints, no system }
      FOR i:=1 TO nrows DO
         IF(csign[i] <> 'B')THEN flg4:=FALSE;

      { Set initial upper bound }

      IF(NOT flg4)THEN
         FOR i:=1 TO nprt DO
            prtyUB[i]:=1.0E20
      ELSE
         BEGIN
            { Round off all decision variables to the nearest integer }

            FOR i:=1 TO nvars DO
               IF(FRAC(decn[i]) <= 0.5)THEN
                  decnUB[i]:=INT(decn[i])
               ELSE
                  decnUB[i]:=INT(decn[i])+1.0;

            { Upper bound for pure goal constraints }

            { Calculate deviational variables for each goal constraint }

            FOR i:=1 TO nrows DO
               BEGIN
                  lhs:=0.0;
                  FOR j:=1 TO ntc DO
                     IF(coef[j].row = i)THEN
                        lhs:=lhs+coef[j].value*decnUB[coef[j].column];
                  dif:=rhs[i]-lhs;
                  IF(dif >= 0.0)THEN
                     BEGIN
                        nd[i]:=dif;
                        pd[i]:=0.0;
                     END
                  ELSE
                     BEGIN
                        nd[i]:=0.0;
                        pd[i]:=-dif;
```

```
                    END;
                END;

            { Upper bound }

            FOR i:=1 TO nprt DO
              BEGIN
                prtyUB[i]:=0.0;
                FOR j:= 1 TO tprt DO
                  BEGIN
                   IF(obj[j].priority = i)THEN
                    IF(obj[j].sign = 'P')THEN
                      prtyUB[i]:=prtyUB[i]+pd[obj[j].row]*obj[j].weight
                    ELSE
                      prtyUB[i]:=prtyUB[i]+nd[obj[j].row]*obj[j].weight;
                  END;
              END;
            IntegerSoln:=TRUE;
            FOR i:=1 TO nvars DO
              intDecn[i]:=decnUB[i];
            FOR i:=1 TO nrows DO
              BEGIN
              intNeg[i]:=nd[i];
              intPos[i]:=pd[i];
            END;
            keepFlg1:=flg1;
            FinalZjCj;
            INC(ubUpdate,1);
            GOTOXY(1,6);
            WRITELN(' U.B. Updates.....',ubUpdate:4);
          END;
    END;

{*************************************************************************}
{*                          CheckPriority                                *}
{*************************************************************************}

FUNCTION CheckPriority:BOOLEAN;

{ This function checks the current priority against the upper bound.    }

    VAR
       equal,better:                BOOLEAN;
    BEGIN

       equal:=TRUE;
       better:=FALSE;
       i:=0;
       REPEAT
          INC(i);
          IF(prty[i] <> prtyUB[i])THEN
             equal:=FALSE;
       UNTIL((NOT equal)OR(i=nprt));
```

```
      IF((NOT equal)AND(prty[i] < prtyUB[i]))THEN
         better:=TRUE;

      { If all priorities are equal, set better to true.  This will make
        the program to update the upper bound for the case the initial
        upper bound determined in 'initialUB' is the optimal solution }

      IF(equal)THEN better:=TRUE;
      CheckPriority:=better;
   END;

{*************************************************************************}
{*                              getLeaf                                  *}
{*************************************************************************}

FUNCTION getLeaf:BOOLEAN;

{ This function finds index of the most recent non nil node in ptrArray}

   VAR
      found:          BOOLEAN;
      i:              BYTE;
   BEGIN
      found:=FALSE;
      i:=leafCount;
      REPEAT
         IF(ptrArray[i] <> NIL) THEN
            BEGIN
               found:=TRUE;
               pdx:=i;
            END
         ELSE
            DEC(i);
      UNTIL((found)OR(i=0));
      getLeaf:=found;
   END;

{.........................................................................}
{.                            BranchBound                                .}
{.........................................................................}

   BEGIN
      integerSoln:=FALSE;
      index:=indexFrac; { Find index of the fractional variable if any }
      IF(index=0)THEN
         BEGIN
            integerSoln:=TRUE;          { Solution is already integer }
            updateUB;
            EXIT;
         END;

      { Initialization }
```

```
        initialUB;
        leafCount:=0;

        NEW(leftPtr);                    { Initialize the left leaf node }
        leftPtr^.nac:=0;
        NEW(rightPtr);                  { Initialize the right leaf node }
        rightPtr^.nac:=0;

        { Set up the left leaf node }

a:      INC(leftPtr^.nac,1);
        leftPtr^.index[leftPtr^.nac]:=index;
        leftPtr^.sign[leftPtr^.nac]:='L';
        leftPtr^.rhs[leftPtr^.nac]:=INT(decn[index]);
        FOR i:=1 TO nprt DO
           leftPtr^.prty[i]:=prty[i];

        { Set up the right leaf node }

        INC(rightPtr^.nac,1);
        rightPtr^.index[rightPtr^.nac]:=index;
        rightPtr^.sign[rightPtr^.nac]:='G';
        rightPtr^.rhs[rightPtr^.nac]:=INT(decn[index])+1.0;
        FOR i:=1 TO nprt DO
           rightPtr^.prty[i]:=prty[i];

        { Add the two new nodes to the ptrArray }

        INC(nNodGe,2);
        GOTOXY(1,4);
        WRITELN(' Nodes Generated..',nNodGe:4);
        INC(leafCount,2);
        ptrArray[leafCount-1]:=leftPtr;
        ptrArray[leafCount]:=rightPtr;

        { Get the most recent non nil node from ptrArray }

b:      potlLeaf:=FALSE;
        REPEAT
           IF(NOT getLeaf)THEN
              EXIT;                     { All nodes have been considered }

           tempPtr:=ptrArray[pdx];

           { Compare priorities of current node against the upper bound  }

           FOR i:=1 TO nprt DO
              prty[i]:=tempPtr^.prty[i];

           { Continue with this node only if its priorities are better
             than UB }
```

```
    IF(NOT CheckPriority)THEN
       BEGIN
          ptrArray[pdx]:=NIL;
          DISPOSE(tempPtr);
       END
    ELSE
       potlLeaf:=TRUE;
UNTIL(potlLeaf);

{ Prepare to solve the current node }

nrows:=orig_nrows+tempPtr^.nac;
FOR i:=1 TO tempPtr^.nac DO
   BEGIN
      ntc:=orig_ntc+i;
      orig_coef[ntc].row:=orig_nrows+i;
      orig_coef[ntc].column:=tempPtr^.index[i];
      orig_coef[ntc].value:=1.0;
      csign[orig_nrows+i]:=tempPtr^.sign[i];
      rhs[orig_nrows+i]:=tempPtr^.rhs[i];
   END;

{ Place new coefficient(s) in appropriate place in 'coef' array }

INC(nNodEv,1);
GOTOXY(1,5);
WRITELN(' Nodes Evaluated..',nNodEv:4);
sortCoef(orig_coef);
ExecuteProgram;
IF(NOT feasible)OR(NOT CheckPriority)THEN
   BEGIN
      ptrArray[pdx]:=NIL;
      DISPOSE(tempPtr);
      GOTO b;
   END;

{ Priority of the new solution is better or equal to the upper
  bound. If the solution is also integer, update the upper bound.}

index:=indexFrac;
IF(index=0)THEN
   BEGIN
      updateUB;
      integerSoln:=TRUE;
      ptrArray[pdx]:=NIL;
      DISPOSE(tempPtr);
      GOTO b;
   END;

NEW(leftPtr);
leftPtr^.nac:=tempPtr^.nac;
FOR i:=1 TO tempPtr^.nac DO
   BEGIN
```

```
            leftPtr^.index[i]:=tempPtr^.index[i];
            leftPtr^.sign[i]:=tempPtr^.sign[i];
            leftPtr^.rhs[i]:=tempPtr^.rhs[i];
         END;

      NEW(rightPtr);
      rightPtr^.nac:=tempPtr^.nac;
      FOR i:=1 TO tempPtr^.nac DO
         BEGIN
            rightPtr^.index[i]:=tempPtr^.index[i];
            rightPtr^.sign[i]:=tempPtr^.sign[i];
            rightPtr^.rhs[i]:=tempPtr^.rhs[i];
         END;

      ptrArray[pdx]:=NIL;
      DISPOSE(tempPtr);
      GOTO a;                                  { Setup the two new nodes }
   END;

{.....................................................................}
{.                              smlap                                  .}
{.....................................................................}

BEGIN                                                    { Program SMLAP }
     flg3:=FALSE;       { Indicates if a data base is created or loaded }
     filename:='None';
     modelType:='N';
     { flg4 indicates if a continuous or integer solution is obtained }
     flg4:=FALSE;
     REPEAT
        TEXTCOLOR(11);                { Selects light cyan characters }
        TEXTBACKGROUND(1);                  { Selects blue background }
        CLRSCR;
        drawBox(1,1,80,24);
        WINDOW(1,25,80,25);                 { Last line of the screen }
        TEXTBACKGROUND(7);                                { Light Gray }
        CLRSCR;
        WINDOW(1,1,80,25);
        TEXTBACKGROUND(11);
        TEXTCOLOR(1);
        GOTOXY(22,1);
        WRITELN (' SMOLAP - Decision Support System ');
        TEXTBACKGROUND(1);
        TEXTCOLOR(15);                       { Select white characters }
        GOTOXY(51,3);
        WRITELN('Current Model '+CHR(26),' ',filename);
        GOTOXY(51,4);
        CASE modelType OF
          'D':  WRITELN('Type: Deterministic');
          'C':  WRITELN('Type: Chance-Constrained');
          'S':  WRITELN('Type: Stochastic');
          'N':  WRITELN('Type: None');
        END;
```

```
          TEXTCOLOR(14);                          { Select yellow characters }
          GOTOXY(4,4);
          TEXTCOLOR(15);
          WRITELN('DATA BASE UTILITIES:');
          TEXTCOLOR(14);
          GOTOXY(4,6);
          WRITELN('[A]  Create a New Model');
          GOTOXY(4,7);
          WRITELN('[B]  Retrieve an Existing Model');
          GOTOXY(4,8);
          WRITELN('[C]  Save Current Model');
          GOTOXY(4,9);
          WRITELN('[D]  Display Current Model');
          GOTOXY(4,11);
          TEXTCOLOR(15);
          WRITELN('SYSTEM ANALYSIS:');
          TEXTCOLOR(14);
          GOTOXY(4,13);
          WRITELN('[E]  Continuous Solution');
          GOTOXY(4,14);
          WRITELN('[F]  Integer Solution');
          GOTOXY(4,15);
          WRITELN('[G]  Nonlinear Solution');
          GOTOXY(4,16);
          WRITELN('[H]  Sensitivity Analysis');
          GOTOXY(4,19);
          WRITELN('[I]  EXIT');
a:        message('Enter Option -'+CHR(16)+' ','1',validSet3,option);

          CASE option OF
            'A':                                  { Create Input Data Base }
              BEGIN
                message('Deterministic/Chance-Constrained/Stochastic '+
                        '(D/C/S)? -'+CHR(16)+' ','1',validSet6,modelType);
                WINDOW(1,1,80,24); { Do not reset color of the last line }
                CLRSCR;
                WINDOW(1,1,80,25);
                CreateDataBase;
                { Initialize priority order }
                FOR i:=1 TO nprt DO
                  prtyOrder[i]:=i;
                flg4:=FALSE;
              END;

            'B':                                  { Load Input Data Base }
              BEGIN
                tempFilename:=filename;
                REPEAT
                  GOTOXY(3,23);
                  WRITE ('Enter the Input File Name -'+CHR(16)+' ');
                  READLN (filename);
                UNTIL (filename <> '');       { Do not accept return only }
                LoadDataBase;
```

```
    IF(NOT flg5)THEN
      filename:=tempFilename
    ELSE                                { A new model is loaded }
      BEGIN
        { Initialize priority order }
        FOR i:=1 TO nprt DO
          prtyOrder[i]:=i;
        flg4:=FALSE;
      END;
  END;

'C':
  BEGIN
   IF(flg3=FALSE)THEN
     BEGIN
       message('No Output File is Present...','O',validSet4,
               inCh);
       GOTO a;
     END;
   SaveDataBase;
  END;

'D':                                { Display Current Data Base }
  BEGIN
    IF(flg3=FALSE)THEN
      BEGIN
        message('No Output File is Present...','O',validSet4,
                inCh);
        GOTO a;
      END;

      WINDOW(1,1,80,24);  { Do not reset color of last line }
      CLRSCR;
      WINDOW(1,1,80,25);
      TEXTCOLOR(11);
      drawbox(1,1,80,24);
      GOTOXY(29,1);
      WRITELN(' DISPLAY CURRENT MODEL ');
      TEXTCOLOR(14);
      DisplayDataBase;
    END;

'E':                                { Continuous Solution }
  BEGIN
    IF(flg3=FALSE)THEN
      BEGIN
        message('No Output File is Present...','O',validSet4,
                inCh);
        GOTO a;
      END;
    IF(modelType='S')THEN
      BEGIN
        message('This is a Nonlinear Model - Select [G] ...',
```

```
            '0',validSet4,inCh);
    GOTO a;
  END;
WINDOW(1,1,80,24);          { Do not reset the last line }
CLRSCR;
WINDOW(1,1,80,25);

{ Restore the original values in case continuous solution
were selected after the integer solution in the main menu}

ntc:=orig_ntc;
nprt:=orig_nprt;
nrows:=orig_nrows;
FOR i:=1 TO ntc DO
   coef[i]:=orig_coef[i];
FOR i:=1 TO nvars DO
   num[i]:=orig_num[i];

TEXTCOLOR(11);                                    { Cyan }
drawbox(1,1,80,24);
GOTOXY(30,1);
TEXTBACKGROUND(11);
TEXTCOLOR(1);
WRITELN(' Continuous Solution ');
TEXTBACKGROUND(1);
TEXTCOLOR(14);
nIteration:=0;          { Initialize number of iterations }
TEXTBACKGROUND(11);
drawBox(49,5,73,16);
WINDOW(50,6,72,15);
CLRSCR;
TEXTCOLOR(0);
WRITELN('      RUN STATUS');
WRITELN;
WRITELN(' Iteration........',nIteration:4);
WRITELN(' CPU..........');
WRITELN;
WRITELN(' Model Name '+CHR(26)+' ',filename);
cursor(FALSE);                    { Turn off the cursor }
GETTIME(hr1,min1,sec1,hsec1);
ExecuteProgram;
GETTIME(hr2,min2,sec2,hsec2);
elapsed:=(hr2*3600.0+min2*60.0+sec2+hsec2*0.01)-
          (hr1*3600.0+min1*60.0+sec1+hsec1*0.01);
GOTOXY(1,4);
WRITE(' CPU..........',elapsed:6:2,' S');
cursor(TRUE);                     { Turn on the cursor }

IF(feasible=FALSE)THEN
  message('No feasible solution exist','0',validSet4,inCh)
ELSE
  BEGIN
   OutputResult;
```

```
        firstTime:=TRUE;{Indicates a new model has been solved}
        solution:='c';          { Indicates continuous solution }
      END;
  END;

'F':                                          { Integer Solution }
  BEGIN
    IF(flg3=FALSE)THEN
      BEGIN
        message('No Output File is Present ....','0',
                validSet4,inCh);
        GOTO a;
      END;
    IF(modelType='S')THEN
      BEGIN
        message('This is a Nonlinear Model - Select [G] ...',
                '0',validSet4,inCh);
        GOTO a;
      END;
    message('All Variables Integer? (Y/N) -'+CHR(16)+' ','1',
            validset4,answer);
    IF(answer='N')THEN
      BEGIN
        flg6:=TRUE;{Indicates we have a mixed integer problem}
        FOR i:=1 TO nvars DO
          decnType[i]:='C';
        GOTOXY(3,23);
        WRITE('Enter Index of Integer Variable -'+CHR(16)+' ');
        x1:=WHEREX;y1:=WHEREY;
        REPEAT
          i:=inputInteger(x1,y1,1,nvars);
          decnType[i]:='I';
          message('More Integer Variables? (Y/N) -'+CHR(16)+
                  ' ','1',validSet4,answer);
          IF(answer='Y')THEN blank(x1,y1,2);
        UNTIL(answer='N');
      END
    ELSE
      BEGIN
        flg6:=FALSE;          { This is a pure integer problem }
        FOR i:=1 TO nvars DO
          decnType[i]:='I';
      END;

    WINDOW(1,1,80,24);              { Do not reset the last line }
    CLRSCR;
    WINDOW(1,1,80,25);

    {Restore the original values in case integer solution were
     selected two times in a row. }

    ntc:=orig_ntc;
    nprt:=orig_nprt;
```

```
nrows:=orig_nrows;
FOR i:=1 TO ntc DO
   coef[i]:=orig_coef[i];
FOR i:=1 TO nvars DO
   num[i]:=orig_num[i];

TEXTCOLOR(11);
drawbox(1,1,80,24);
GOTOXY(31,1);
TEXTBACKGROUND(11);
TEXTCOLOR(1);
WRITELN(' Integer Solution ');
TEXTBACKGROUND(1);
TEXTCOLOR(14);

nIteration:=0;          { Initialize number of iterations }
TEXTBACKGROUND(11);
drawBox(49,5,73,16);
WINDOW(50,6,72,15);
CLRSCR;
nNodEv:=1;
nNodGe:=1;
ubUpdate:=0;
TEXTCOLOR(0);
WRITELN('      RUN STATUS');
WRITELN;
WRITELN(' Iteration.......',nIteration:4);
WRITELN(' Nodes Generated..',nNodGe:4);
WRITELN(' Nodes Evaluated..',nNodEv:4);
WRITELN(' U.B. Updates.....',ubUpdate:4);
WRITELN(' CPU.........');
WRITELN;
WRITELN(' Model Name '+CHR(26)+' ',filename);
cursor(FALSE);                    { Turn off the cursor }
GETTIME(hr1,min1,sec1,hsec1);
ExecuteProgram;
IF(feasible=FALSE)THEN
  BEGIN
     message('No feasible solution exist','0',validSet4,
              inCh);
     cursor(TRUE);
  END
ELSE
  BEGIN
    BranchBound;
    GETTIME(hr2,min2,sec2,hsec2);
    elapsed:=(hr2*3600.0+min2*60.0+sec2+hsec2*0.01)-
             (hr1*3600.0+min1*60.0+sec1+hsec1*0.01);
    GOTOXY(1,7);
    WRITE(' CPU.........',elapsed:6:2,' S');
    cursor(TRUE);                  { Turn on the cursor }
    IF(NOT integerSoln)THEN
      message('No Integer Solution Exist','0',validSet4,
```

```
                    inCh)
        ELSE
          BEGIN

            { Get upper bound values }

            FOR i:=1 TO nprt DO
              prty[i]:=prtyUB[i];
            FOR i:=1 TO nvars DO
              decn[i]:=intDecn[i];
            FOR i:=1 TO orig_nrows DO
              BEGIN
                neg[i]:=intNeg[i];
                pos[i]:=intPos[i];
              END;
            nrows:=orig_nrows;

            OutputResult;
            firstTime:=TRUE;          { Indicates a new model }
            solution:='i';       { Indicates integer solution }
          END;
      END;
  END;

'G':                                        { Nonlinear Solution }
  BEGIN
    IF(flg3=FALSE)THEN
      BEGIN
        message('No Output File is Present ....','O',
                validSet4,inCh);
        GOTO a;
      END;
    IF(modelType<>'S')THEN
      BEGIN
        message('This is a Linear Model - Select [E] or [F] '+
                '...','O',validSet4,inCh);
        GOTO a;
      END;
    WINDOW(1,1,80,24);          { Do not reset the last line }
    CLRSCR;
    WINDOW(1,1,80,25);
    TEXTCOLOR(11);                                    { Cyan }
    drawbox(1,1,80,24);
    GOTOXY(30,1);
    TEXTBACKGROUND(11);
    TEXTCOLOR(1);
    WRITELN(' Nonlinear Solution ');
    TEXTBACKGROUND(1);
    TEXTCOLOR(14);
    nIteration:=0;          { Initialize number of iterations }
    TEXTBACKGROUND(11);
    drawBox(49,5,73,16);
    WINDOW(50,6,72,15);
```

```
CLRSCR;
TEXTCOLOR(0);
WRITELN('      RUN STATUS');
WRITELN;
WRITELN(' Iteration........',nIteration:4);
WRITELN(' Step Size....... 1.00');
WRITELN(' CPU..........');
WRITELN;
WRITELN(' Model Name '+CHR(26)+' ',filename);
WINDOW(1,1,80,24);
message('Input a new Starting Point (Y/N)? -'+CHR(16)+' ',
        '1',validSet4,answer);
IF(answer='Y')THEN
  BEGIN
    { Initialize all decision variables to zero }
    FOR i:=1 TO (nsources*ndestns)+nsources DO
      BEGIN
        decn[i]:=0.0;
        varChange[i]:='F';
      END;

    { Input the index of open sources }
    GOTOXY(6,21);
    WRITE('Index for an Open Source -'+CHR(16)+' ');
    x1:=WHEREX;y1:=WHEREY;
    REPEAT
      i:=inputInteger(x1,y1,1,nsources);
      decn[(nsources*ndestns)+i]:=1.0;
      message('More Open Sources? (Y/N) -'+CHR(16)+' ',
              '1',validSet4,answer);
      IF(answer='Y')THEN blank(x1,y1,2);
    UNTIL(answer='N');

    { Input the allocation variables }
    FOR i:=1 TO nsources DO
      BEGIN
        FOR j:=1 TO ndestns DO
          BEGIN
            IF(decn[(nsources*ndestns)+i]=1.0) THEN
              BEGIN
                varChange[(j-1)*nsources+i]:='V';
                GOTOXY(16,22);
                WRITE('ENTER X(',i:2,j:2,') -'+CHR(16)+
                      ' ');
                x1:=WHEREX;y1:=WHEREY;
                REPEAT
                  temp:=inputReal(x1,y1);
                  IF(temp<0.0)THEN
                    BEGIN
                      message('Allocations Must be Nonne'+
                      'gative ... ','2',validSet4,answer);
                      STR(temp,s);
                      blank(x1,y1,LENGTH(s));
```

```
                    END;
                  UNTIL(temp>=0.0);
                  decn[(j-1)*nsources+i]:=temp;
                  STR(temp,s);
                  blank(x1,y1,LENGTH(s));
                END;
            END;
        END;
    END;

    { Initialize the step size }

    FOR i:=1 TO nvars DO
      step[i]:=1.0;                          { default step size }
    message('Current Step Size is 1 - Change (Y/N)? -'+
            CHR(16)+' ','1',validSet4,answer);
    IF(answer='Y')THEN
      BEGIN
        GOTOXY(6,23);
        WRITE('Enter the New Step Size -'+CHR(16),' ');
        x1:=WHEREX;y1:=WHEREY;
        temp:=inputReal(x1,y1);
        FOR i:=1 TO nvars DO
          step[i]:=temp;
      END;
    WINDOW(5,21,79,23);CLRSCR;WINDOW(50,6,72,15);
    TEXTCOLOR(0);
    cursor(FALSE);                         { Turn off the cursor }
    { Rewrite the step size if it was changed }
    GOTOXY(1,4);
    WRITE(' Step Size.......',step[1]:5:2);
    GETTIME(hr1,min1,sec1,hsec1);
    PatternSearch;
    GETTIME(hr2,min2,sec2,hsec2);
    elapsed:=(hr2*3600.0+min2*60.0+sec2+hsec2*0.01)-
             (hr1*3600.0+min1*60.0+sec1+hsec1*0.01);
    GOTOXY(1,5);
    WRITE(' CPU.........',elapsed:6:2,' S');
    cursor(TRUE);                          { Turn on the cursor }
    { Get the optimum solution }

    FOR i:=1 TO nprt DO
       prty[i]:=prtyUB[i];
    FOR i:=1 TO nvars DO
       decn[i]:=intDecn[i];
    FOR i:=1 TO nrows DO
       BEGIN
         neg[i]:=intNeg[i];
         pos[i]:=intPos[i];
       END;
    OutputResult;
END;
```

```
          'H':                                    { Sensitivity Analysis }
            BEGIN
              IF(flg3=FALSE)THEN
                BEGIN
                  message('No Output File is Present...','0',validSet4,
                          inCh);
                  GOTO a;
                END;
              IF(NOT flg4)THEN
                BEGIN
                  message('No Solution Exist...','0',validSet4,inCh);
                  GOTO a;
                END;
              WINDOW(1,1,80,24);          { Do not reset the last line }
              CLRSCR;
              WINDOW(1,1,80,25);
              TEXTCOLOR(11);
              drawbox(1,1,80,24);
              GOTOXY(27,1);
              TEXTCOLOR(1);
              TEXTBACKGROUND(11);
              TEXTCOLOR(1);
              WRITELN(' Sensitivity Analysis ');
              TEXTBACKGROUND(1);
              TEXTCOLOR(14);
              SensiAnaly;
            END;
        END                                   { End of the case statement }
    UNTIL (option = 'I');
    CLRSCR;
END.                                                  { SMLAP Program }
```

```
UNIT smlaUtil;

   INTERFACE

      USES CRT,DOS,PRINTER;

      CONST
         enter = #13;                          {ASCII character for enter}
         bell = #7;                            {ASCII character for bell}
         validSet1: SET OF CHAR = ['P','N',enter];
         validSet2: SET OF CHAR = ['E','G','L','B'];
         validSet3: SET OF CHAR = ['A','B','C','D','E','F','G','H','I'];
         validSet4: SET OF CHAR = ['Y','N'];
         validSet5: SET OF CHAR = ['A','B','C','D','E'];
         validSet6: SET OF CHAR = ['D','C','S'];
         validSet7: SET OF CHAR = ['N','U'];

      TYPE
         setType = SET OF CHAR;

         objective = RECORD
            sign:                              CHAR;
            row:                               BYTE;
            priority:                          BYTE;
            weight:                            EXTENDED;
         END;

         coefficient = RECORD
            row:                               BYTE;
            column:                            BYTE;
            value:                             EXTENDED;
         END;

         value = RECORD
            priority:                          BYTE;
            weight:                            EXTENDED;
         END;

         coefArray =                           ARRAY[1..500] OF coefficient;

      VAR
         i, j, k, p, ij, c, tprt, nprt,
         orig_nprt,ntc,orig_ntc,nrows,
         orig_nrows,ncols,nIteration,
         nNodEv,nNodGe,nprt1,indx1,indx2:      INTEGER;
         nvars, npdvs, rown, pcol, prow,
         nelemty, lcount, ubUpdate,nsources,
         ndestns:                              BYTE;
         hr1,hr2,min1,min2,sec1,sec2,
         hsec1,hsec2:                          WORD;
         zmax,lw,ar,elapsed:                   EXTENDED;
         temp:                                 REAL;
         coef, orig_coef:                      coefArray;
```

```
        obj:                                    ARRAY[0..100] OF objective;
        pw:                                     ARRAY[1..130] OF value;
        pwBasis:                                ARRAY[1..70] OF value;
        rhs,rhsF,x,y,decn,prty,neg,pos,
        prtyUB,intDecn,intNeg,intPos:           ARRAY[1..70] OF EXTENDED;
        avalue, ElValue:                        ARRAY[1..500]OF EXTENDED;
        OptZjCj:                                ARRAY[1..11,1..130]OF EXTENDED;
        mu,sigma,Lbound,Ubound,Ocost,Ucost: ARRAY[1..29] OF REAL;
        step:                                   ARRAY[1..30] OF REAL;
        arow, ElRow:                            ARRAY[1..500]OF BYTE;
        prtyOrder:                              ARRAY[1..10] OF BYTE;
        pdevc,basicCol,num,orig_num:            ARRAY[1..70] OF BYTE;
        currentBasic,n,start,position,
        ElCount:                                ARRAY[1..400]OF INTEGER;
        csign,decnType,constType:               ARRAY[1..70] OF CHAR;
        varChange:                              ARRAY[1..30] OF CHAR;
        s:                                      STRING;
        filename:                               STRING[10];
        device:                                 STRING[3];
        datafile:                               TEXT; {Sequential file}
        option, answer, inCh, solution,
        modelType,demand:                       CHAR;
        flg1,flg2,flg3,flg4,flg5,flg6,
        feasible,integerSoln,keepFlg1,
        firstTime:                              BOOLEAN;

    PROCEDURE PatternSearch;
    FUNCTION UpperCase(inString:STRING):STRING;
    PROCEDURE Cursor(OnOff:BOOLEAN);
    PROCEDURE lineDraw(lsize:BYTE;lchar:CHAR);
    PROCEDURE blank(x,y,l:BYTE);
    PROCEDURE drawBox(xupL,yupL,xloR,yloR:INTEGER);
    PROCEDURE message(prompt:STRING;action:CHAR;insett:setType;
                      VAR inCh:CHAR);
    FUNCTION inputInteger(xpos,ypos:    BYTE;
                          LoLmt,UpLmt:  INTEGER):INTEGER;
    FUNCTION inputReal(xpos,ypos:BYTE):EXTENDED;
    FUNCTION inputChar(inSet:setType):CHAR;
    PROCEDURE lineCount;
    PROCEDURE saveInput;
    PROCEDURE sortCoef(inTemp:coefArray);

IMPLEMENTATION

{*************************************************************************}
{*                            PatternSearch                              *}
{*************************************************************************}

PROCEDURE PatternSearch;

   {This procedure applies a modified Hooke and Jeeves procedure to
    solve a nonlinear SMLAP.  All variables with the prefix of
    **** int **** or suffix of **** UB **** refer to the optimum values}
```

```
   VAR
      n,maxob,maxss,stoxRow,icount,jcount,ncount:   INTEGER;
      wPrev,curntValue,delta,q,qq,decnR:            ARRAY[1..30] OF REAL;
      TempPoint,alpha,beta:                                      REAL;
      kflag,terminate,better,different,nonLinearRow:           BOOLEAN;
      acoef:                                  ARRAY[1..31,1..30]OF REAL;

{************************************************************************}
{*                              NormalCDF                               *}
{************************************************************************}

PROCEDURE NormalCDF(k:REAL; VAR d,p:REAL);

{This procedure computes the probability that a standard normal random
 variable is greater than or equal to k (1-F(k)).  It also compute the
 ordinate of the normal density at k, f(k).  Maximum error is 0.0000007.
 The approximation is based on C. Hastings. }

  VAR
    tempk,w:            REAL;
  BEGIN
    tempk:=ABS(k);
    w:=1.0/(1.0+0.2316419*tempk);
    d:=0.3989423*EXP(-(tempk*tempk)/2.0);
    p:=1.0-d*w*((((1.330274*w-1.821256)*w+1.781478)*w-0.3565638)*w+
                0.3193815);
    IF(k>0)THEN p:=1.0-p;
  END;

{************************************************************************}
{*                            multiObjective                            *}
{************************************************************************}

PROCEDURE multiObjective;
  VAR
    i,j:                    INTEGER;
    lhs:                    ARRAY [1..30]OF REAL;
    bj,kj,mag,prob,temp:    REAL;
  BEGIN

    { Find left hand side of all rows (constraints) }
    FOR i:=1 TO nrows DO
      BEGIN
        lhs[i]:=0.0;
        FOR j:=1 TO nvars DO
          lhs[i]:=lhs[i]+acoef[i,j]*decnR[j];
      END;

    { Find left hand side for the nonlinear constraints }

    lhs[stoxRow]:=0.0;
    n:=1;
```

```
FOR j:=1 TO ndestns DO
  BEGIN
    bj:=0.0;
    FOR i:= n TO n+nsources-1 DO
        bj:=bj+decnR[i];

    IF(demand='N')THEN              { Normal distribution of demands }
      BEGIN
        kj:=(bj-mu[j])/sigma[j];
        NormalCDF(kj,mag,prob);
        lhs[stoxRow]:=lhs[stoxRow]+sigma[j]*(Ocost[j]+Ucost[j])*
                      (mag-kj*prob)+Ocost[j]*(bj-mu[j]);
      END
    ELSE                            { Uniform distribution of demands }
      BEGIN
        lhs[stoxRow]:=lhs[stoxRow]+1/(2*(Ubound[j]-Lbound[j]))*
             ((Ocost[j]+Ucost[j])*SQR(bj)+Ocost[j]*SQR(Lbound[j])+
             Ucost[j]*SQR(Ubound[j])-2*(Ocost[j]*Lbound[j]+Ucost[j]*
             Ubound[j])*bj);
      END;
    n:=n+nsources;
  END;

{ Compute the deviational variables }

FOR i:=1 TO nrows DO
  BEGIN
    temp:=rhs[i]-lhs[i];
    IF(temp >=0.0)THEN
      BEGIN
        neg[i]:=temp;
        pos[i]:=0.0;
      END
    ELSE
      BEGIN
        pos[i]:=-temp;
        neg[i]:=0.0;
      END;
  END;

{ Calculate the achievement values }

FOR i:=1 TO nprt DO
  BEGIN
    prty[i]:=0.0;
    FOR j:=1 TO tprt DO
      BEGIN
        IF(obj[j].priority=i)THEN
          BEGIN
            IF(obj[j].sign='P')THEN
              prty[i]:=prty[i]+pos[obj[j].row]*obj[j].weight
            ELSE
              prty[i]:=prty[i]+neg[obj[j].row]*obj[j].weight;
```

```
            END;
          END;
      END;
    INC(nIteration);
  END;

{*************************************************************************}
{*                                compare                                *}
{*************************************************************************}

PROCEDURE compare;
  VAR
    i,j:                    INTEGER;
  BEGIN
    better:=FALSE;
    different:=FALSE;
    j:=0;
    REPEAT
      INC(j);
      IF ABS(prty[j] - prtyUB[j]) >= delta[j] THEN
        different:=TRUE;
    UNTIL(j=nprt)OR(different);
    IF(different)AND(prty[j] < prtyUB[j])THEN
      BEGIN
        better:=TRUE;
        FOR i:=1 TO nvars DO
          intDecn[i]:=decnR[i];
        FOR i:= 1 TO nrows DO
          BEGIN
            intNeg[i]:=neg[i];
            intPos[i]:=pos[i];
          END;
        FOR i:=1 TO nprt DO
          prtyUB[i]:=prty[i];
      END;
  END;

{.........................................................................}
{.                          PatternSearch                                .}
{.........................................................................}

   BEGIN
     maxob:=500;              { Maximum number to evaluate  objectives }
     maxss:=6;                { Maximum number to reduce the step size }
     alpha:=1.0;                              { Acceleration factor }
     beta:=0.5;                             { Step reduction factor }
     nIteration:=0;               { Counter for objective evaluation  }
     jcount:=0;                   { Counter for step size reduction   }
     ncount:=0;                   { Counter for number of coordinates }
     kflag:=TRUE;               { Indicates if pattern move is successful }
     terminate:=FALSE;          { Indicates if convergence criteria is met}

     { Set the error in achievement vector to be reached before an
```

```
  achievement vector can dominates the upper bound }

FOR i:=1 TO nprt DO
  delta[i]:=0.1;

{ Read the initial starting point. decnR[ ] is used so it can be
 used as a real type variable instead of decn[ ] which is extended}

FOR i:=1 TO nvars DO
  decnR[i]:=decn[i];
FOR i:=1 TO nrows DO
  FOR j:=1 TO nvars DO
    acoef[i,j]:=0.0;
{ Set nonzero coefficients }

FOR j:=1 TO ntc DO
  acoef[coef[j].row,coef[j].column]:=coef[j].value;

{ Find the nonlinear constraint row, nonlinear constraint is a row
  with all coefficients zero }

i:=0;
REPEAT
  nonLinearRow:=TRUE;    { Indicates if the nonlinear row is found }
  INC(i);
  j:=0;
  REPEAT
    INC(j);
    IF(acoef[i,j]<>0)THEN nonLinearRow:=FALSE;
  UNTIL(NOT nonLinearRow)OR(j=nvars);
UNTIL(nonLinearRow);
stoxRow:=i;
FOR i:=1 TO nvars DO
  BEGIN
    q[i]:=decnR[i];
    qq[i]:=decnR[i];
  END;
multiObjective;
{ Initialize the initial optimal (upper bound) solution }
FOR i:=1 TO nvars DO
  intDecn[i]:=decnR[i];
FOR i:= 1 TO nrows DO
  BEGIN
    intNeg[i]:=neg[i];
    intPos[i]:=pos[i];
  END;
FOR i:=1 TO nprt DO
  BEGIN
    wPrev[i]:=prty[i];
    prtyUB[i]:=prty[i];
  END;

{ Start the search }
```

```
REPEAT
  FOR i:=1 TO nprt DO
     CurntValue[i]:=prty[i];

  { Establish the search pattern }

  icount:=0;
  FOR i:=1 TO nsources*ndestns DO
    IF(varChange[i]='V')THEN
      BEGIN
        INC(icount);
        TempPoint:=decnR[i];
        decnR[i]:=decnR[i]+step[i];
        multiObjective;
        compare;
        IF(better)THEN
          BEGIN
            FOR j:=1 TO nprt DO
              curntValue[j]:=prty[j];
            qq[i]:=decnR[i];
          END
        ELSE

        { Search the opposite direction of the current coordinate }

          BEGIN
            decnR[i]:=decnR[i]-2.0*step[i];
            IF(decnR[i]<0.0)THEN decnR[i]:=0.0;
            multiObjective;
            compare;
            IF(better)THEN
              BEGIN
                FOR j:=1 TO nprt DO
                  curntValue[j]:=prty[j];
                qq[i]:=decnR[i];
              END
            ELSE
           { Search in current coordinate unsuccessful, backtrack }
              BEGIN
                INC(ncount,1);
                decnR[i]:=TempPoint;
                qq[i]:=decnR[i];
              END;
          END;
      END;

  { Test to determine termination of the program }

  GOTOXY(1,3);
  WRITE(' Iteration........',nIteration:4);
  GOTOXY(1,4);
  WRITE(' Step Size.......',step[1]:5:2);
```

```
        IF(nIteration >= maxob)OR(jcount >= maxss)THEN
           terminate:=TRUE
        ELSE
          BEGIN

            { IF search for all axes fail, reduce the step size }

            IF(ncount = icount)THEN
              BEGIN
              { If search failed due to pattern move, retrack to previous
                point }
                IF(NOT kflag)THEN
                  BEGIN
                    kflag:=TRUE;
                    FOR i:=1 TO nsources*ndestns DO
                      decnR[i]:=q[i];
                  END;
                { Reduce the step size }
                INC(jcount);
                FOR i:= 1 TO nsources*ndestns DO
                   step[i]:=step[i]*beta;
              END
            ELSE
              { Perform a pattern move }
              FOR i:=1 to nvars-nsources DO
                BEGIN
                  decnR[i]:=decnR[i] + alpha*(decnR[i]-q[i]);{New point }
                  IF(decnR[i]<0.0)THEN decnR[i]:=0.0;
                  q[i]:=qq[i];                          { Previous Point }
                END;
            ncount:=0;
            FOR i:=1 TO nprt DO
               wPrev[i]:=curntValue[i];
            multiObjective;
            compare;
            IF(NOT better)THEN
              kflag:=FALSE;
            { Pattern move or step size reduction is unsuccessful }
          END;
      UNTIL(terminate);
  END;                                          { End of PatternSearch }

{*************************************************************************}
{*                              UpperCase                                *}
{*************************************************************************}

FUNCTION UpperCase(inString:STRING):STRING;

{ This function returns an uppercase version of the string it receives }

     VAR
        outString:       STRING;
```

```
    BEGIN
       outString := '';
       FOR i:=1 TO LENGTH(inString) DO
         BEGIN
           outString:=outString + UPCASE(instring[i]);
         END;
       Uppercase:=outString;
    END;

{*************************************************************************}
{*                                 Cursor                                *}
{*************************************************************************}

PROCEDURE Cursor(OnOff:BOOLEAN);

   { This procedure turns the cursor on/off }

   VAR
      reg:     REGISTERS;
   BEGIN
      IF(OnOff)THEN
         IF MEM[0:$449]=7 THEN
            reg.CX:=$0C0D
         ELSE
            reg.CX:=$0607
      ELSE
         reg.CX:=$2000;
      reg.AX:=$0100;
      INTR($10,reg)
   END;

{*************************************************************************}
{*                               lineDraw                                *}
{*************************************************************************}

PROCEDURE lineDraw(lsize:BYTE;lchar:CHAR);

{This procedure draws a line of length 'lsize' using character 'lchar' }
   VAR
      i:             BYTE;
   BEGIN
      FOR i:=1 TO lsize DO
         WRITE(lchar);
      WRITELN;
   END;

{*************************************************************************}
{*                                blank                                  *}
{*************************************************************************}

PROCEDURE blank(x,y,l:BYTE);
   VAR
      i:                BYTE;
```

```
   { This procedure blanks a specified entry }

   BEGIN
      GOTOXY(x,y);
      FOR i:=1 TO l DO
         WRITE(' ');
      GOTOXY(x,y);
   END;

{*********************************************************************}
{*                              drawBox                              *}
{*********************************************************************}

PROCEDURE drawBox(xupL,yupL,xloR,yloR:INTEGER);

   { This procedure draws a box on the screen }

   CONST
      upLcor=#201;                                  { Upper left corner }
      loLcor=#200;                                  { Lower left corner }
      loRcor=#188;                                 { Lower right corner }
      upRcor=#187;                                 { Upper right corner }
      horizl=#205;
      vertil=#186;

   BEGIN
      GOTOXY(xupL,yupL);
      WRITE(upLcor);
      j:=xloR-xupL-1;
      FOR i:=1 TO j DO
         WRITE(horizl);                             { Draw the top line }
      WRITE(upRcor);
      FOR i:=yupL+1 TO yloR-1 DO
         BEGIN
            GOTOXY(xloR,i);
            WRITE(vertil);
            GOTOXY(xupL,i);
            WRITE(vertil);
         END;
      GOTOXY(xupL,yloR);
      WRITE(loLcor);
      FOR i:=1 TO j DO
         WRITE(horizl);                          { Draw the bottom line }
      WRITE(loRcor);
   END;

{*********************************************************************}
{*                              message                              *}
{*********************************************************************}

PROCEDURE message(prompt:       STRING;
                  action:       CHAR;
```

```
                insett:      setType;
                VAR inCh:    CHAR);

   { This procedure displays a given prompt on the last line of screen }

   BEGIN
      WINDOW(1,1,80,25);
      TEXTBACKGROUND(7);                                  { Light gray }
      TEXTCOLOR(4);                                             { Red }
      GOTOXY(2,25);
      WRITE(prompt);
      CASE action OF
         '0':
            BEGIN
               TEXTCOLOR(0);                                   { Black }
               GOTOXY(50,25);
               WRITE('Press any Key to Continue...');
               answer:=READKEY;
               GOTOXY(1,25);
               CLREOL;                              { Clear the message }
            END;
         '1':
            BEGIN
               inCh:=inputChar(insett);
               GOTOXY(1,25);
               CLREOL;                              { Clear the message }
            END;
         '2':
            BEGIN
               WRITE(bell);
               DELAY(1000);                           { Delay 1 Second }
               GOTOXY(1,25);
               CLREOL;                              { Clear the message }
            END;
         '3':
            BEGIN
               GOTOXY(1,25);
               CLREOL;                              { Clear the message }
            END;
         { If '4', It only displays the prompt without erasing it }
      END;

      TEXTBACKGROUND(1);                                       { Blue }
      TEXTCOLOR(14);                                         { Yellow }
   END;

{*************************************************************************}
{*                            inputInteger                               *}
{*************************************************************************}

FUNCTION inputInteger(xpos,ypos:    BYTE;
                      LoLmt,UpLmt:  INTEGER):INTEGER;
```

```
   { This function accepts a valid integer number }

   VAR
      temp:                  STRING[30];
      tempInteger,code:      INTEGER;
      CkRange:               BOOLEAN;

   BEGIN                                                { inputInteger }
      GOTOXY(xpos,ypos);
      REPEAT
         CkRange:=TRUE;
         READLN(temp);
         VAL(temp,tempInteger,code);
         IF(LENGTH(temp)=0)THEN                { Enter has been pressed }
            BEGIN
               inputInteger:=0;
               code:=0;
            END;
         IF(code = 0)THEN                                { Check the range }
            IF((tempInteger < LoLmt)OR(tempInteger > UpLmt))THEN
               CkRange:=FALSE;

         IF((code<>0)OR(CkRange=FALSE))THEN
            BEGIN
               message('Data Out of Range.....','2',validSet4,answer);
               blank(xpos,ypos,LENGTH(temp));
            END;
      UNTIL ((code=0)AND(CkRange));
      inputInteger:=tempInteger;
   END;                                                 { inputInteger }

{*************************************************************************}
{*                              inputReal                                *}
{*************************************************************************}

FUNCTION inputReal(xpos,ypos:BYTE):EXTENDED;

   { This function accepts a valid real number }

   VAR
      temp:          STRING[20];
      tempReal:      EXTENDED;
      code:          INTEGER;

   BEGIN                                                  { inputReal }
      REPEAT
         REPEAT
            GOTOXY(xpos,ypos);
            READLN(temp);
         UNTIL(LENGTH(temp) > 0);                { Do not except Enter }
         VAL(temp,tempReal,code);
         IF(code<>0)THEN
            BEGIN
```

```
            WRITE(bell);
            blank(xpos,ypos,LENGTH(temp));
         END;
      UNTIL (code=0);
      inputReal:=tempReal;
   END;                                                    { inputReal }

{*************************************************************************}
{*                              inputChar                                *}
{*************************************************************************}

FUNCTION inputChar(inSet:setType):CHAR;

   { This function accepts a valid character }

   VAR
      ansr,ansr2:          CHAR;

   BEGIN
      REPEAT
         ansr:=UPCASE(READKEY);

         { If a key with extended code (Function Keys, Arrows,
           Ctl-,Alt-) has been pressed, discard the second character }

         IF(ansr=#0)THEN ansr2:=READKEY;

         IF NOT(ansr IN inSet)THEN
            WRITE(bell);
      UNTIL(ansr IN inSet);

      WRITE(ansr);                                  { Echo back the input }
      inputChar:=ansr;
   END;

{*************************************************************************}
{*                              lineCount                                *}
{*************************************************************************}

PROCEDURE lineCount;

   { This procedure stops the screen from scrolling }

   BEGIN
      INC(lcount);
      IF(lcount>20)THEN
         BEGIN
            lcount:=0;
            message('','0',validSet4,inCh);
            WINDOW(3,2,48,23);
            GOTOXY(1,22);
         END;
   END;
```

```
{*************************************************************************}
{*                              saveInput                                *}
{*************************************************************************}

PROCEDURE saveInput;
  BEGIN
 { Save necessary input information which may be altered by integer
   routine }
     orig_ntc:=ntc;
     orig_nprt:=nprt;
     orig_nrows:=nrows;
     FOR i:=1 TO ntc DO
        orig_coef[i]:=coef[i];
     FOR i:=1 TO nvars DO
        orig_num[i]:=num[i];
  END;

{*************************************************************************}
{*                              sortCoef                                 *}
{*************************************************************************}

PROCEDURE sortCoef(inTemp:coefArray);
  BEGIN

  { Sorts the information columnwise if they were entered rowwise. Also,
    finds number of coefficients (decision variable) in each column }

     c:=0;
     k:=0;
     FOR j:=1 TO nvars DO
        BEGIN
           FOR i:=1 TO ntc DO
              BEGIN
                 IF (inTemp[i].column = j) THEN
                    BEGIN
                       INC(c,1); {Counter for dec. variable in column j}
                       INC(k,1);
                       coef[k] := inTemp[i];
                    END;
              END;
           num[j]:=c;
           c:=0;
        END;
  END;
END.                                                    { smlautil }
```

```
UNIT dbasUtil;

   INTERFACE

      USES CRT,DOS,PRINTER,smlaUtil;
      PROCEDURE CreateDataBase;
      PROCEDURE LoadDataBase;
      PROCEDURE SaveDataBase;
      PROCEDURE DisplayDataBase;
      PROCEDURE OutputResult;
      PROCEDURE FinalZjCj;
      PROCEDURE SensiAnaly;

   IMPLEMENTATION

{*************************************************************************}
{*                              NInvCDF                                  *}
{*************************************************************************}

PROCEDURE NInvCDF(mu,sigma,ccprob: REAL; VAR ccRhs: REAL);

   { This procedure computes the inverse cumulative distribution
     function of a random variable Z, distributed normally with mean
     zero and variance one. ie. Z=F**(-1)(prob). }

   VAR
      tempProb,w,w2,z:                  REAL;

   BEGIN
      tempProb:=ccProb;
      IF(tempProb > 0.5)THEN tempProb:=1.0-tempProb;
      w2:=LN(1.0/SQR(tempProb));
      w:=SQRT(w2);
      { Z value for unit normal distribution }
      z:=w-(2.515517+0.802853*w+0.010328*w2)/
           (1.0+1.432788*w+0.189269*w2+0.001308*w*w2);
      IF(ccProb<=0.5)THEN z:=-z;

      { Find corresponding x value for the given normal distribution }

      ccRhs:=mu+z*sigma;
      IF(FRAC(ccRhs) > 0.001)THEN
         ccRhs:=INT(ccRhs+1.0)
      ELSE
         ccRhs:=INT(ccRhs);
   END;

{*************************************************************************}
{*                              UInvCDF                                  *}
{*************************************************************************}

PROCEDURE UInvCDF(Lbound,Ubound,ccProb: REAL; VAR ccRhs: REAL);
```

```
   { This procedure computes the inverse probability function of a
     random variable distributed uniformly between Lbound and Ubound. }

   BEGIN
      ccRhs:=(Ubound-Lbound)*ccProb + Lbound;
      IF(FRAC(ccRhs) > 0.001)THEN
         ccRhs:=INT(ccRhs+1.0)
      ELSE
         ccRhs:=INT(ccRhs);
   END;

{*************************************************************************}
{*                              CreateDataBase                           *}
{*************************************************************************}

PROCEDURE CreateDataBase;
   CONST
     f = #196;                                  { ASCII Character for - }
   VAR
      x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,
      y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11:         BYTE;

{*************************************************************************}
{*                               Stochastic                              *}
{*************************************************************************}

PROCEDURE Stochastic;
  BEGIN

    { Input rhs values }

    FOR i:= 1 TO nrows DO
      BEGIN
        constType[i]:='D';
        blank(x8,y8,18);
        GOTOXY(6,18);
        WRITE ('RHS  for Constraint ',i:2,' '+f+f+CHR(16)+' ');
        REPEAT
          rhs[i]:=inputReal(x8,y8);                    { Input the rhs }
          IF(rhs[i]<0)THEN
            BEGIN
              message('RHS Must be >= 0','2',validSet4,answer);
              blank(x8,y8,18);
            END;
        UNTIL (rhs[i] >= 0);
      END;
    { Input number of sources and destinations }
    REPEAT
      nsources:=inputInteger(x9,y9,1,29);
      ndestns:=inputInteger(x10,y10,1,29);
      IF((nsources*ndestns+nsources)>30)THEN
        BEGIN
          message('Problem is Too Big ... ','2',validSet4,answer);
```

```
      STR(nsources,s);
      blank(x9,y9,LENGTH(s));
      STR(ndestns,s);
      blank(x10,y10,LENGTH(s));
    END;
UNTIL(nsources*ndestns+nsources <= 30);
GOTOXY(x11,y11);
demand:=inputChar(validSet7);

{ Input the demand parameters }

FOR i:=1 TO ndestns DO
  BEGIN
    blank(x1,y1,15);
    blank(x2,y2,15);
    CASE demand of
      'N':
        BEGIN
          GOTOXY(6,22);
          WRITE('Destination ',i:2,' Mean '+f+CHR(16)+' ');
          x1:=WHEREX;y1:=WHEREY;
          GOTOXY(6,23);
          WRITE('Destination ',i:2,' S.D. '+f+CHR(16)+' ');
          x2:=WHEREX;y2:=WHEREY;
          mu[i]:=inputReal(x1,y1);
          sigma[i]:=inputReal(x2,y2);
        END;
      'U':
        BEGIN
          GOTOXY(6,22);
          WRITE('Destination ',i:2,' Lower Bound '+f+CHR(16)+' ');
          x1:=WHEREX;y1:=WHEREy;
          GOTOXY(6,23);
          WRITE('Destination ',i:2,' Upper Bound '+f+CHR(16)+' ');
          x2:=WHEREX;y2:=WHEREY;
          REPEAT
            REPEAT
              Lbound[i]:=inputReal(x1,y1);
              IF(Lbound[i]<0.0)THEN
                BEGIN
                  message('Lower Bound Must be >= 0 ','2',validSet4,
                          answer);
                  blank(x1,y1,15);
                END;
            UNTIL(Lbound[i]>=0);
            REPEAT
              Ubound[i]:=inputReal(x2,y2);
              IF(Ubound[i]<0.0)THEN
                BEGIN
                  message('Upper Bound Must be >= 0 ','2',validSet4,
                          answer);
                  blank(x2,y2,15);
                END;
```

```
            UNTIL(Ubound[i]>=0);
            IF(Lbound[i]>=Ubound[i])THEN
              BEGIN
                message('Lower Bound Must be < Upper Bound ','2',
                        validSet4,answer);
                blank(x1,y1,15);
                blank(x2,y2,15);
              END;
          UNTIL(Lbound[i] < Ubound[i]);
        END;
    END;
  END;
{ input penalty costs }
WINDOW(2,19,50,23);
CLRSCR;
WINDOW(1,1,80,25);
FOR i:= 1 TO ndestns DO
  BEGIN
    blank(x1,y1,12);
    blank(x2,y2,12);
    GOTOXY(6,20);
    WRITE('Oversupply  Cost at Destn. ',i:2,' '+f+CHR(16)+' ');
    x1:=WHEREX;y1:=WHEREY;
    GOTOXY(6,21);
    WRITE('Undersupply Cost at Destn. ',i:2,' '+f+CHR(16)+' ');
    x2:=WHEREX;y2:=WHEREY;
    REPEAT
      Ocost[i]:=inputReal(x1,y1);
      IF(Ocost[i]<0)THEN
        BEGIN
          message('Cost Must be Nonnegative ... ','2',validSet4,
                  answer);
          blank(x1,y1,12);
        END;
    UNTIL(Ocost[i]>=0.0);
    REPEAT
      Ucost[i]:=inputReal(x2,y2);
      IF(Ucost[i]<0)THEN
        BEGIN
          message('Cost Must be Nonnegative ... ','2',validSet4,
                  answer);
          blank(x2,y2,12);
        END;
    UNTIL(Ucost[i]>=0.0);
  END;

{ Initialize all decision variables to zero }
FOR i:=1 TO (nsources*ndestns)+nsources DO
  BEGIN
    decn[i]:=0.0;
    varChange[i]:='F';
  END;
```

```
    { Input the index of open sources }
    GOTOXY(6,22);
    WRITE('Index for an Open Source '+f+CHR(16)+' ');
    x1:=WHEREX;y1:=WHEREY;
    REPEAT
      i:=inputInteger(x1,y1,1,nsources);
      decn[(nsources*ndestns)+i]:=1.0;
      message('More Open Sources? (Y/N) -'+CHR(16)+' ','1',validSet4,
               answer);
      IF(answer='Y')THEN
        blank(x1,y1,2);
    UNTIL(answer='N');

    { Input the allocation variables }
    FOR i:=1 TO nsources DO
      BEGIN
        FOR j:=1 TO ndestns DO
          BEGIN
            IF(decn[(nsources*ndestns)+i]=1.0) THEN
              BEGIN
                varChange[(j-1)*nsources+i]:='V';
                GOTOXY(16,23);
                WRITE('ENTER X(',i:2,j:2,') '+f+CHR(16)+' ');
                x1:=WHEREX;y1:=WHEREY;
                REPEAT
                  temp:=inputReal(x1,y1);
                  IF(temp<0.0)THEN
                    BEGIN
                      message('Allocations Must be Nonnegative ... ',
                              '2',validSet4,answer);
                      STR(temp,s);
                      blank(x1,y1,LENGTH(s));
                    END;
                UNTIL(temp>=0.0);
                decn[(j-1)*nsources+i]:=temp;
                STR(temp,s);
                blank(x1,y1,LENGTH(s));
              END;
          END;
      END;
  END;

{*************************************************************************}
{*                              ChanceConst                              *}
{*************************************************************************}

PROCEDURE ChanceConst(i:INTEGER);

{ This procedure accepts RHS information for chance-Const. formulation }

   VAR
      mu, sigma, serviceLvl, rhsTemp,
      Lbound,Ubound:                          REAL;
```

```
   ansr:                                    CHAR;
   x1,y1,x2,y2,x3,y3,x4,y4:                 BYTE;

BEGIN
   GOTOXY(6,19);
   WRITE('#',i:2,' Probabilistic (Y/N) '+f+CHR(16)+' ');
   ansr:=inputChar(validSet4);
   IF(ansr='N')THEN                        { Deterministic constraint }
     BEGIN
       GOTOXY(6,20);
       WRITE ('RHS  for Constraint ',i:2,' '+f+f+CHR(16)+' ');
       x1:=WHEREX;y1:=WHEREY;
       REPEAT
         rhs[i]:=inputReal(x1,y1);                  { Input the rhs }
         IF(rhs[i]<0)THEN
           BEGIN
             message('RHS Must be >= 0 ','2',validSet4,answer);
             blank(x1,y1,18);
           END;
       UNTIL (rhs[i] >= 0);
     END
   ELSE                                    { Probabilistic constraint }
     BEGIN
       GOTOXY(6,20);
       WRITE('Normal or Uniform (N/U) '+f+CHR(16)+' ');
       ansr:=inputChar(validSet7);
       GOTOXY(10,21);
       WRITE('Enter Service Level '+f+CHR(16)+' ');
       x2:=WHEREX;y2:=WHEREY;
       IF(ansr='N')THEN                                  { Normal }
         BEGIN
           GOTOXY(10,22);
           WRITE('Enter Mean '+f+f+f+f+f+f+f+f+f+f+f+CHR(16)+' ');
           x3:=WHEREX;y3:=WHEREY;
           GOTOXY(10,23);
           WRITE('Standard Deviation '+f+f+CHR(16)+' ');
           x4:=WHEREX;y4:=WHEREY;
         END
       ELSE                                              { Uniform }
         BEGIN
           GOTOXY(10,22);
           WRITE('Enter Lower Bound '+f+f+f+CHR(16)+' ');
           x3:=WHEREX;y3:=WHEREY;
           GOTOXY(10,23);
           WRITE('Enter Upper Bound '+f+f+f+CHR(16)+' ');
           x4:=WHEREX;y4:=WHEREY;
         END;
       { Input the service level probability }
       REPEAT
         serviceLvl:=inputReal(x2,y2);
         IF(serviceLvl<0.0)OR(serviceLvl>1.0)THEN
           BEGIN
             message('Service Level Must be Between 0 and 1 ','2',
```

```
                validSet4,answer);
        blank(x2,y2,18);
      END;
UNTIL(serviceLvl>=0.0)AND(serviceLvl<=1.0);

{ Read parameters for normal distribution }

IF(ansr='N')THEN                                          { Normal }
  BEGIN
    constType[i]:='N';
    IF(serviceLvl=0.0)THEN serviceLvl:=0.001;
    IF(serviceLvl=1.0)THEN serviceLvl:=0.999;
    REPEAT
      mu:=inputReal(x3,y3);
      sigma:=inputReal(x4,y4);
      NInvCDF(mu,sigma,serviceLvl,rhsTemp);
      IF(rhsTemp<=0)THEN
        BEGIN
          message('Invalid Parameters - Enter Again','2',
                  validSet4,answer);
          STR(mu,s);
          blank(x3,y3,LENGTH(s));
          STR(sigma,s);
          blank(x4,y4,LENGTH(s));
        END;
    UNTIL(rhsTemp>0.0);
    rhs[i]:=rhsTemp;
  END;

{ Read parameters for uniform distribution }

IF(ansr='U')THEN                                          { Uniform }
  BEGIN
    constType[i]:='U';
    REPEAT
      REPEAT
        Lbound:=inputReal(x3,y3);
        IF(Lbound<0.0)THEN
          BEGIN
            message('Lower Bound Must be >= 0 ','2',validSet4,
                    answer);
            blank(x3,y3,18);
          END;
      UNTIL(Lbound>=0);
      REPEAT
        Ubound:=inputReal(x4,y4);
        IF(Ubound<0.0)THEN
          BEGIN
            message('Upper Bound Must be >= 0 ','2',validSet4,
                    answer);
            blank(x4,y4,18);
          END;
      UNTIL(Ubound>=0);
```

```
                IF(Lbound>=Ubound)THEN
                  BEGIN
                    message('Lower Bound Must be < Upper Bound ','2',
                            validSet4,answer);
                    blank(x3,y3,18);
                    blank(x4,y4,18);
                  END;
              UNTIL(Lbound < Ubound);
              UInvCDF(Lbound,Ubound,serviceLvl,rhsTemp);
              rhs[i]:=rhsTemp;
            END;
        END;
      blank(x8,y8,1);
      blank(x9,y9,1);
      WINDOW(2,20,50,23);
      CLRSCR;
      WINDOW(1,1,80,25);
  END;                                          { End of ChanceConst }

{*************************************************************************}
{*                              inputData                                *}
{*************************************************************************}

PROCEDURE inputData;
  LABEL a,b,c,d;
  VAR
    s:                                          STRING;
    temp:                                       coefArray;
    nnrows:                                     BYTE;

  BEGIN                                                  { inputData }
    { Set up the input entry display }

    GOTOXY(3,3);
    TEXTCOLOR(10);
    WRITELN ('SET 1 - PRIORITY STRUCTURE:');
    GOTOXY(3,10);
    WRITELN ('SET 2 - TECHNOLOGICAL COEFFICIENTS:');
    GOTOXY(3,16);
    WRITELN ('SET 3 - CONSTRAINTS SIGN AND RHS VALUES:');
    TEXTCOLOR(14);

    { Information for the priority structure }

    GOTOXY(6,5);
    WRITE ('Sign ''P'' or ''N'''+CHR(196)+CHR(16)+' ');
    x1:=WHEREX;y1:=WHEREY;
    GOTOXY(6,6);
    WRITE('Row Number '+f+f+f+f+f+CHR(16)+' ');
    x2:=WHEREX;y2:=WHEREY;
    GOTOXY(6,7);
    WRITE ('Priority '+f+f+f+f+f+f+f+CHR(16)+' ');
    x3:=WHEREX;y3:=WHEREY;
```

```
GOTOXY(6,8);
WRITE ('Weight '+f+f+f+f+f+f+f+f+f+f+CHR(16)+' ');
x4:=WHEREX;y4:=WHEREY;

{ Information for the technological coefficients }

GOTOXY(6,12);
WRITE ('Row Number '+f+f+f+f+f+f+CHR(16)+' ');
x5:=WHEREX;y5:=WHEREY;
GOTOXY(6,13);
WRITE ('Column Number '+f+f+CHR(16)+' ');
x6:=WHEREX;y6:=WHEREY;
GOTOXY(6,14);
WRITE ('Coefficient '+f+f+f+f+f+CHR(16)+' ');
x7:=WHEREX;y7:=WHEREY;

{ Information for the constraints sign and rhs values }

CASE modelType OF
  'D':                                              { Deterministic }
    BEGIN
      GOTOXY(6,18);
      WRITE ('Sign for Constraint  1 '+f+f+CHR(16)+' ');
      x8:=WHEREX;y8:=WHEREY;
      GOTOXY(6,19);
      WRITE('RHS  for Constraint  1 '+f+f+CHR(16)+' ');
      x9:=WHEREX;y9:=WHEREY;
    END;
  'C':                                        { Chance-constrained }
    BEGIN
      GOTOXY(6,18);
      WRITE ('Sign for Constraint  1 '+f+f+CHR(16)+' ');
      x8:=WHEREX;y8:=WHEREY;
      GOTOXY(6,19);
      WRITE('# 1 Probabilistic (Y/N) '+f+CHR(16)+' ');
      x9:=WHEREX;y9:=WHEREY;
    END;
  'S':                                                { Stochastic }
    BEGIN
      GOTOXY(6,18);
      WRITE('RHS  for Constraint  1 '+f+f+CHR(16)+' ');
      x8:=WHEREX;y8:=WHEREY;
      GOTOXY(6,19);
      WRITE('Number of Sources '+f+f+f+f+f+f+f+f+CHR(16)+' ');
      x9:=WHEREX;y9:=WHEREY;
      GOTOXY(6,20);
      WRITE('Number of Destinations '+f+f+CHR(16)+' ');
      x10:=WHEREX;Y10:=WHEREY;
      GOTOXY(6,21);
      WRITE('Normal or Uniform Demands (N/U) '+f+CHR(16)+' ');
      x11:=WHEREX;y11:=WHEREY;
    END;
END;
```

```
      { Start reading the information }

      { Input the achievement function. Read in the sign, row, priority,
        and weight of deviational variables in the objective function.
        Also, compute total number of priorities present. }

a:    nprt := 0;
      tprt := 0;
      nnrows:=0;

b:    GOTOXY(x1,y1);                          { Input the achievement sign }
      INC(tprt,1);                                  { Increment tprt by 1 }
      WITH obj[tprt] DO
        BEGIN
          sign := inputChar(validSet1);
          IF(sign <> enter) THEN
            BEGIN
              row:=inputInteger(x2,y2,1,30);      { Input the row number }
              IF(row > nnrows)THEN nnrows:=row;
              { Input priority number }
              priority:=inputInteger(x3,y3,1,10);
              REPEAT
                 weight:=inputReal(x4,y4);             { Input the weight }
                 IF(weight <= 0)THEN
                   BEGIN
                     message('Weight Must be > 0 ','2',validSet4,answer);
                     STR(weight,s);
                     blank(x4,y4,LENGTH(s));
                   END;
              UNTIL (weight>0.0);
              IF (priority > nprt) THEN nprt:=priority;
              blank(x1,y1,1);
              STR(row,s);
              blank(x2,y2,LENGTH(s));
              STR(priority,s);
              blank(x3,y3,LENGTH(s));
              STR(weight,s);
              blank(x4,y4,LENGTH(s));
              GOTO b;                          { Read the next priority }
            END;
        END;

      DEC(tprt,1);                                  { Decrement tprt by 1 }
      IF (nprt = 0) THEN
         BEGIN
            message('Number of priorities must be > 0','0',validSet4,
                    inCh);
            GOTO a;
         END;

      { Input the technological coefficients- (row, column, value).
        Also, calculate number of rows and decision variables. }
```

```
c:    ntc:= 0;
      nvars:= 0;
      nrows:= 0;

d:    INC (ntc, 1);
      WITH temp[ntc] DO
        BEGIN
          row:=inputInteger(x5,y5,0,30);          { Input the row number }
          WHILE (row <> 0) DO
            BEGIN
              { Input the column number }
              column:=inputInteger(x6,y6,1,30);
              REPEAT
                value:=inputReal(x7,y7);              { Input the value }
                IF(value = 0)THEN
                  BEGIN
                    message('Value Must be > 0 ','2',validSet4,answer);
                    blank(x7,y7,1);
                  END;
              UNTIL (value<>0.0);
              IF (column > nvars) THEN nvars := column;
              IF (row > nrows) THEN nrows := row;
              STR(row,s);
              blank(x5,y5,LENGTH(s));
              STR(column,s);
              blank(x6,y6,LENGTH(s));
              STR(value,s);
              blank(x7,y7,LENGTH(s));
              GOTO d;                        { Read the next coefficient }
            END;
          blank(x5,y5,1); { Clear the 0 if it was used instead of return}
        END;
      DEC (ntc, 1);
      IF (ntc = 0) THEN
        BEGIN
          message('Number of Variables Must be > 0 ','0',validSet4,inCh);
          GOTO c;
        END;

      { The following is true for stochastic models when the nonlinear
        constraint is the last row }

      IF(nnrows>nrows)THEN nrows:=nnrows;

      { Sort the information columnwise if they were entered rowwise.
        Also, find number of coefficients (decision variables) in each
        column }

      sortCoef(temp);

      { Input the sign for each constraint:

              E - System (rigid) equality constraints
```

```
              G - System > = constraints
              L - System < = constraints
              B - Goal constraints }

      IF(modelType='S')THEN
        Stochastic
      ELSE
        BEGIN
          FOR i:= 1 TO nrows DO
            BEGIN
              constType[i]:='D';
              GOTOXY(6,18);
              WRITE ('Sign for Constraint ',i:2,' '+f+f+CHR(16)+' ');
              GOTOXY(x8,y8);                  { Input the constraint sign }
              csign[i] := inputChar(validSet2);

              { Input the rhs for each constraint }

              CASE modelType OF
                'D':
                  BEGIN
                    GOTOXY(6,19);
                    WRITE ('RHS  for Constraint ',i:2,' '+f+f+CHR(16)+
                           ' ');
                    REPEAT
                      rhs[i]:=inputReal(x9,y9);          { Input the rhs }
                      IF(rhs[i]<0)THEN
                       BEGIN
                        message('RHS Must be >= 0','2',validSet4,answer);
                        blank(x9,y9,18);
                       END;
                    UNTIL (rhs[i] >= 0);
                    blank(x8,y8,1);
                    blank(x9,y9,18);
                  END;
                'C':
                  { Call chance-constrained routine for input }
                  ChanceConst(i);
              END;
            END;
        END;
    END;                                          { End of inputData }

{.....................................................................}
{.                        CreateDataBase                             .}
{.....................................................................}

    BEGIN                                          { CreateDataBase }

      TEXTCOLOR(11);
      drawBox(1,1,80,24);
      GOTOXY(30,1);
      WRITELN (' CREATE A NEW MODEL ');
```

```
GOTOXY(51,2);
CASE modelType OF
  'D':  WRITELN('Type: Deterministic');
  'C':  WRITELN('Type: Chance-Constrained');
  'S':  WRITELN('Type: Stochastic');
END;
TEXTCOLOR(14);

{ Open up the help window }

TEXTBACKGROUND(11);
drawbox(51,4,74,22);
GOTOXY(59,4);
WRITELN(' HELP ');
window(52,5,73,21);
CLRSCR;
TEXTCOLOR(0);
WRITELN(' SET 1');
WRITELN('   0 <    Row   '+CHR(243)+' 30');
WRITELN('   0 < Priority '+CHR(243)+' 10');
WRITELN('   0 <  Weight');
WRITELN;
WRITELN(' SET 2');
WRITELN('   0 <    Row   '+CHR(243)+' 30');
WRITELN('   0 < Variable '+CHR(243)+' 30');
WRITELN('   0 < Coefficient');
WRITELN;
WRITELN(' SET 3');
IF(modelType<>'S')THEN
  BEGIN
    WRITELN('   Sign');
    WRITELN('        E ..... =');
    WRITELN('        G ..... '+CHR(242));
    WRITELN('        L ..... '+CHR(243));
    WRITELN('        B ..... GOAL');
  END;
WRITE('   RHS '+CHR(242),' 0');
IF(modelType='S')THEN
  BEGIN
     WRITELN;
     WRITELN(' Sources*Destinations');
     WRITELN(' + Sources '+CHR(242),' 0');
  END;

{ End of the help window }

WINDOW(1,1,80,25);
TEXTBACKGROUND(1);
TEXTCOLOR(14);
inputData;                                  { Get all the input data }
saveInput;                                  { Save the original data }
flg3:=TRUE;
message('Save This Model? (Y/N) -'+CHR(16),'1',validSet4,answer);
```

```
      IF(answer='Y')THEN
         SaveDataBase
      ELSE
         filename:='Test';
   END;                                                { CreateDataBase }

{*************************************************************************}
{*                          LoadDataBase                                 *}
{*************************************************************************}

PROCEDURE LoadDataBase;
   BEGIN                                                { LoadDataBase }
     flg5:=FALSE;        { Indicates if the current load is successful }
     ASSIGN (datafile, filename);   { Identify the file's name on disk }
     cursor(FALSE);
     message('Loading File '+filename+' ','4',validSet4,inch);

     { Check if the file exist }

     {$I-}
     APPEND(datafile);
     {$I+}
     IF(IORESULT <> 0)THEN
        BEGIN
           cursor(TRUE);
           message('','3',validSet4,inch);
           message('File does not exist','0',validSet4,inCh);
           EXIT;
        END
     ELSE
        CLOSE(datafile);                                  { File exist }
     RESET (datafile);                      { Open the file for reading }
     READLN(datafile,modelType);
     READLN(datafile, tprt, nprt);

     FOR i:=1 TO tprt DO
        WITH obj[i] DO
           READLN(datafile, sign, row, priority, weight);
     READLN(datafile, ntc, nvars, nrows);
     FOR i:= 1 TO ntc DO
        WITH coef[i] DO
           READLN(datafile, row, column, value);
     IF(modelType<>'S')THEN
       BEGIN
         FOR i:=1 TO nrows DO
              READLN(datafile, csign[i], constType[i], rhs[i]);
         FOR i:=1 TO nvars DO
              READLN(datafile, num[i]);
       END
     ELSE                                              { Stochastic model }
       BEGIN
         FOR i:=1 TO nrows DO
            READLN(datafile,constType[i],rhs[i]);
```

```
          READLN(datafile,demand,nsources,ndestns);
          FOR i:=1 TO ndestns DO
            BEGIN
              IF(demand='N')THEN
                READLN(datafile,mu[i],sigma[i],Ocost[i],Ucost[i])
              ELSE
                READLN(datafile,Lbound[i],Ubound[i],Ocost[i],Ucost[i]);
            END;
          FOR i:=1 TO (nsources*ndestns)+nsources DO
            READLN(datafile,varChange[i], decn[i]);
        END;
      CLOSE (datafile);
      message('','3',validSet4,inch);
      cursor(TRUE);
      saveInput;                                  { Save the original data }
      flg3:=TRUE;
      flg5:=TRUE;
    END;                                            { End of LoadDataBase }

{*************************************************************************}
{*                            SaveDataBase                               *}
{*************************************************************************}

PROCEDURE SaveDataBase;
    VAR
       rightFile:                          BOOLEAN;
       x10,y10:                            BYTE;

    BEGIN
      GOTOXY(3,23);
      WRITE ('Enter the output file name -'+CHR(16)+' ');
      x10:=WHEREX;y10:=WHEREY;
      rightFile:=FALSE;
      REPEAT
         READLN (filename);
         ASSIGN (datafile, filename);
         cursor(FALSE);
         message('Saving File '+filename+' ','4',validSet4,inch);

         { Check if the file exist }

         {$I-}
         APPEND(datafile);
         {$I+}
         IF(IORESULT <> 0)THEN                    { File does not exist }
            rightFile:=TRUE
         ELSE
            BEGIN
               CLOSE(datafile);
               cursor(TRUE);
               message('File Already Exist....'+
                     '                                    Overwrite (Y/N) -'+
                    CHR(16)+' ','1',validSet4,answer);
```

```
          IF(answer='N')THEN
             blank(x10,y10,LENGTH(filename))
          ELSE
             BEGIN
                message('Saving File '+filename+' ','4',validSet4,
                         inch);
                rightFile:=TRUE;
                cursor(FALSE);
             END;
       END;
UNTIL rightFile;

{ Write all the information into the file }

{$I-}
REWRITE (datafile);                    { create the output data file }
{$I+}
IF (IORESULT <>0) THEN
   BEGIN
      WRITE(bell);
      message('','3',validSet4,inch);
      message('Can not Open File','0',validSet4,inCh);
      cursor(TRUE);
      EXIT;
   END;
WRITELN(datafile,modelType);
WRITELN(datafile, tprt, ' ', nprt);
FOR i:= 1 TO tprt DO
   WITH obj[i] DO
      BEGIN
         WRITELN (datafile,sign,' ',row,' ',priority,' ',weight);
      END;
WRITELN (datafile,ntc,' ',nvars,' ',nrows);
FOR i:= 1 TO ntc DO
  WITH coef[i] DO
    BEGIN
      WRITELN (datafile, row,' ',column,' ',value);
    END;
IF(modelType<>'S')THEN
  BEGIN
    FOR i:= 1 TO nrows DO
       WRITELN(datafile, csign[i],constType[i],' ',rhs[i]);
    FOR i:=1 TO nvars DO
       WRITELN(datafile, num[i]);
  END
ELSE
  BEGIN
    FOR i:=1 TO nrows DO
       WRITELN(datafile,constType[i],' ',rhs[i]);
    WRITELN(datafile,demand,' ',nsources,' ',ndestns);
    FOR i:=1 TO ndestns DO
     BEGIN
       IF(demand='N')THEN
```

```
            WRITELN(datafile,mu[i],' ',sigma[i],' ',Ocost[i],' ',
                                Ucost[i])
          ELSE
            WRITELN(datafile,Lbound[i],' ',Ubound[i],' ',Ocost[i],' ',
                                Ucost[i]);
        END;
       FOR i:=1 TO (nsources*ndestns)+nsources DO
         WRITELN(datafile,varChange[i],' ',decn[i]);
     END;
   CLOSE (datafile);
   message('','3',validSet4,inch); { Clear the message from last line}
   cursor(TRUE);
 END;

{*************************************************************************}
{*                          DisplayDataBase                              *}
{*************************************************************************}

PROCEDURE DisplayDataBase;

 BEGIN
   TEXTBACKGROUND(11);
   IF(modelType<>'S')THEN
     BEGIN
       drawBox(49,5,73,16);
       WINDOW(50,6,72,15);
     END
   ELSE
     BEGIN
       drawBox(49,5,73,19);
       WINDOW(50,6,72,18);
     END;
   CLRSCR;
   TEXTCOLOR(0);                                              { Black }
   WRITELN('     INPUT SUMMARY');
   WRITELN;
   WRITELN(' # of Priorities...',nprt:3);
   WRITELN(' # of Rows.........',nrows:3);
   WRITELN(' # of Variables....',nvars:3);
   WRITELN(' # of Tech. Coeff..',ntc:3);
   IF(modelType='S')THEN
     BEGIN
       WRITELN(' # of Sources......',nsources:3);
       WRITELN(' # of Destinations.',ndestns:3);
       WRITELN(' Demand Distribution:',demand);
     END;
   WRITELN;
   WRITELN(' Current Model '+CHR(26)+' ',filename);
   CASE modelType OF
     'D':  WRITELN(' Type: Deterministic');
     'C':  WRITELN(' Type: Chance-Const.');
     'S':  WRITELN(' Type: Stochastic');
   END;
```

```
WINDOW(3,2,48,23);
TEXTCOLOR(14);
TEXTBACKGROUND(1);
WRITELN('ACHIEVEMENT FUNCTION:');
lineDraw(21,CHR(196));
WRITELN('SIGN   ROW   PRIORITY   WEIGHT');
lcount:=3;
FOR i:= 1 TO tprt DO
   WITH obj[i] DO
      BEGIN
         WRITELN(sign:2,row:7,priority:9,weight:11:2);
         lineCount;
      END;
WRITELN;lineCount;
WRITELN('TECHNOLOGICAL COEFFICIENTS:');lineCount;
lineDraw(27,CHR(196));lineCount;
WRITELN ('ROW   COLUMN    VALUE');lineCount;
FOR i:= 1 TO ntc DO
  WITH coef[i] DO
    BEGIN
      WRITELN(row:2,column:7,value:12:2);
      lineCount;
    END;
WRITELN;
lineCount;
WRITELN('RIGHT HAND SIDE:');
lineCount;
lineDraw(16,CHR(196));
lineCount;
IF (modelType<>'S')THEN
  BEGIN
    WRITELN('ROW   SIGN      VALUE    TYPE');
    lineCount;
    FOR i:= 1 TO nrows DO
      BEGIN
        WRITELN(i:2,csign[i]:7,rhs[i]:12:2,'       ',constType[i]);
        lineCount;
      END;
  END
ELSE
  BEGIN
    WRITELN('ROW   TYPE      VALUE');
    lineCount;
    FOR i:=1 TO nrows DO
      BEGIN
        WRITELN(i:2,constType[i]:7,rhs[i]:12:2);
        lineCount;
      END;
    WRITELN;lineCount;
    IF(demand='N')THEN
      WRITELN('Dest. Mean   S.D. Over S. Cost  Under S. Cost')
    ELSE
      WRITELN('Dest. Lbound  Ubound  Oversup. $  Undersup. $');
```

```
    lineCount;
    FOR i:=1 TO ndestns DO
      BEGIN
        IF(demand='N')THEN
          WRITELN(i:2,mu[i]:8:2,sigma[i]:7:2,Ocost[i]:9:2,
                  Ucost[i]:15:2)
        ELSE
          WRITELN(i:2,Lbound[i]:9:2,Ubound[i]:9:2,Ocost[i]:9:2,
                  Ucost[i]:12:2);
        lineCount;
      END;
    WRITELN;lineCount;
    WRITELN('STARTING SOLUTION:');lineCount;
    lineDraw(18,CHR(196));lineCount;
    WRITELN('Variable     Value    Fixed/Variable');lineCount;
    FOR i:=1 TO nsources DO
      FOR j:=1 TO ndestns DO
        BEGIN
          WRITELN('X(',i:2,j:2,') = ',decn[(j-1)*nsources+i]:7:2,
                  '           ',varChange[(j-1)*nsources+i]);
          lineCount;
        END;
    FOR i:=1 TO nsources DO
      BEGIN
        WRITELN('  Y(',i:2,') = ',decn[(nsources*ndestns)+i]:7:2,
                '           ',varChange[(nsources*ndestns)+i]);
        lineCount;
      END;
  END;
message('Print (Y/N)? -'+CHR(16),'1',validSet4,answer);
IF(answer='Y')THEN
  BEGIN
    WRITELN(LST,'-----------------------------------------------');
    WRITELN(LST,'           INPUT DATA FILE: ',filename);
    WRITELN(LST,'-----------------------------------------------');
    CASE modelType OF
      'D':  WRITELN(LST,' Type: Deterministic');
      'C':  WRITELN(LST,' Type: Chance-Constrained');
      'S':  WRITELN(LST,' Type: Stochastic');
    END;
    WRITELN(LST);
    WRITELN(LST,' Number of Priorities...',nprt:3);
    WRITELN(LST,' Number of Rows.........',nrows:3);
    WRITELN(LST,' Number of Variables....',nvars:3);
    WRITELN(LST,' Number of Tech. Coeff..',ntc:3);
    IF(modelType='S')THEN
      BEGIN
        WRITELN(LST,' Number of Sources......',nsources:3);
        WRITELN(LST,' Number of Destinations.',ndestns:3);
        WRITELN(LST,' Demand Distributions...  ',demand);
      END;
    WRITELN(LST);
    WRITELN(LST,'ACHIEVEMENT FUNCTION:');
```

```
      WRITELN(LST,'SIGN   ROW   PRIORITY   WEIGHT');
      FOR i:= 1 TO tprt DO
         WITH obj[i] DO
            WRITELN(LST,sign:2,row:7,priority:9,weight:12:2);
      WRITELN(LST);
      WRITELN(LST,'TECHNOLOGICAL COEFFICIENTS:');
      WRITELN (LST,'ROW   COLUMN    VALUE');
      FOR i:= 1 TO ntc DO
         WITH coef[i] DO
            WRITELN(LST,row:2,column:7,value:12:2);
      WRITELN(LST);
      WRITELN(LST,'RIGHT HAND SIDE:');
      IF(modelType<>'S')THEN
        BEGIN
          WRITELN(LST,'ROW   SIGN      VALUE    TYPE');
          FOR i:= 1 TO nrows DO
            WRITELN(LST,i:2,csign[i]:7,rhs[i]:12:2,'      ',
                        constType[i]);
        END
      ELSE
        BEGIN
          WRITELN(LST,'ROW   TYPE      VALUE');
          FOR i:=1 TO nrows DO
              WRITELN(LST,i:2,constType[i]:7,rhs[i]:12:2);
          WRITELN(LST);
          IF(demand='N')THEN
            WRITELN(LST,'Dest. Mean   S.D. Over S. Cost  Under S. '+
                        'Cost')
          ELSE
            WRITELN(LST,'Dest. Lbound  Ubound  Oversup. $  '+
                        'Undersup. $');
          FOR i:=1 TO ndestns DO
            IF(demand='N')THEN
              WRITELN(LST,i:2,mu[i]:8:2,sigma[i]:7:2,Ocost[i]:9:2,
                          Ucost[i]:15:2)
            ELSE
              WRITELN(LST,i:2,Lbound[i]:9:2,Ubound[i]:9:2,
                          Ocost[i]:9:2,Ucost[i]:12:2);
          WRITELN(LST);
          WRITELN(LST,'STARTING SOLUTION:');
          WRITELN(LST,'Variable     Value   Fixed/Variable');
          FOR i:=1 TO nsources DO
            FOR j:=1 TO ndestns DO
              WRITELN(LST,'X(',i:2,j:2,') = ',
                          decn[(j-1)*nsources+i]:8:2,
                          '          ',varChange[(j-1)*nsources+i]);
          FOR i:=1 TO nsources DO
            WRITELN(LST,'  Y(',i:2,') = ',
                        decn[(nsources*ndestns)+i]:8:2,
                        '          ',varChange[(nsources*ndestns)+i]);
        END;
    END;
END;                                        {End of DisplayDataBase}
```

```
{*************************************************************************}
{*                              OutputResult                             *}
{*************************************************************************}

PROCEDURE OutputResult;
   BEGIN
      flg4:=TRUE;        { A continuous or integer solution is obtained }
      WINDOW(3,2,48,23);
      TEXTCOLOR(14);                                              { Yellow }
      TEXTBACKGROUND(1);                                            { Blue }
      WRITELN;
      WRITELN('ANALYSIS OF MULTIPLE OBJECTIVES');
      WRITELN('Priority   Under-Achievement');
      lcount:=3;
      FOR i:=1 TO nprt DO
        BEGIN
          WRITELN(i:4,prty[i]:14:2);
          lineCount;
        END;
      WRITELN;
      lineCount;
      WRITELN('ANALYSIS OF DECISION VARIABLES');
      lineCount;
      FOR i:= 1 TO nvars DO
         BEGIN
            WRITELN(' x(',i:2,')=',decn[i]:10:2);
            lineCount;
         END;
       WRITELN;
      lineCount;
      WRITELN('ANALYSIS OF DEVIATIONAL VARIABLES');
      lineCount;
      WRITELN('Const./Goal #          d-          d+');
      lineCount;
      FOR i:=1 TO nrows DO
         BEGIN
            WRITE(i:8,neg[i]:17:2);
            WRITELN(pos[i]:12:2);
            lineCount;
         END;
      message('Print? (Y/N) -'+CHR(16)+' ','1',validSet4,answer);
      IF(answer='Y')THEN
      BEGIN
         WRITELN(LST,'*****************************************');
         IF(option='E')THEN          { Continuous Solution is selected }
               WRITELN(LST,'*          CONTINUOUS SOLUTION          *')
         ELSE
               WRITELN(LST,'*            INTEGER SOLUTION           *');
         WRITELN(LST,'*****************************************');
         WRITELN(LST,'Model Name: ',filename);
         WRITELN(LST,'Iteration : ',nIteration:8);
         IF(option='F')THEN
```

```
            BEGIN
               WRITELN(LST,'Nodes Generated:',nNodGe:4);
               WRITELN(LST,'Nodes Evaluated:',nNodEv:4);
               WRITELN(LST,'U.B. Updates:',ubUpdate:7);
            END;
          WRITELN(LST,'CPU:',elapsed:18:2,' SECONDS');
         WRITELN(LST);
         WRITELN(LST,'ANALYSIS OF MULTIPLE OBJECTIVES');
         WRITELN(LST,'Priority   Under-Achievement');
         FOR i:=1 TO nprt DO
            WRITELN(LST,i:5,prty[i]:16:2);
         WRITELN(LST);
         WRITELN(LST,'ANALYSIS OF DECISION VARIABLES');
         FOR i:=1 TO nvars DO
            WRITELN(LST,' X(',i:2,')=',decn[i]:14:2);
         WRITELN(LST);
         WRITELN(LST,'ANALYSIS OF DEVIATIONAL VARIABLES');
         WRITELN(LST,'Const./Goal #          d-          d+');
         FOR i:=1 TO nrows DO
         BEGIN
            WRITE(LST,i:8,neg[i]:17:2);
            WRITELN(LST,pos[i]:12:2);
         END;
      END;
   END;                                          { End of OutputResult }

{*************************************************************************}
{*                              FinalZjCj                                *}
{*************************************************************************}

PROCEDURE FinalZjCj;

{ This procedure calculates and stores the optimum Zj-Cj matrix for
  tradeoff analysis in sensitivity analysis module }

   LABEL s;
   VAR
      tempZmax:                        EXTENDED;

   BEGIN
     IF(flg1)THEN nprt1:=nprt+1
     ELSE
       nprt1:=nprt;
     FOR k:=1 TO ncols DO
       BEGIN
         IF(currentBasic[k] <> 0) THEN
           BEGIN
             FOR p:=1 TO nprt1 DO
               OptZjCj[p,k]:=0.0;
               GOTO s;
           END;
         FOR i:=1 TO nrows DO y[i]:=0.0;
```

```
          { Construct the original a column }

          FOR i:=start[k] TO start[k]+n[k]-1 DO
            y[arow[i]]:=avalue[i];

          { Update the 'a' column }

          IF (nelemty <> 0)THEN
            BEGIN
              FOR i:=1 TO nelemty DO
                BEGIN
                  ar:=y[position[i]];
                  y[position[i]]:=0.0;              { This is the a-hat }
                  IF(ABS(ar) > 1.0E-10) THEN
                    BEGIN
                      indx1:=ElCount[i];
                      indx2:=ElCount[i+1]-1;
                      FOR j:=indx1 TO indx2 DO
                        BEGIN
                          ij:=ElRow[j];
                          y[ij]:=y[ij]+ar*ElValue[j];
                        END;
                    END;
                END;
            END;

          FOR p:=1 to nprt1 DO
            BEGIN
              tempzmax:=0.0;

              { Calculate zj-cj for the current variable and priority }

              FOR i:=1 TO nrows DO
                IF(pwBasis[i].priority = p) THEN
                  tempzmax:=tempzmax+pwBasis[i].weight * y[i];
              IF(pw[k].priority = p)THEN
                tempzmax:=tempzmax-pw[k].weight;              { Zj-Cj }
              OptZjCj[p,k]:=tempzmax;
            END;
s:      END;                                        { End of column loop }
    END;

{************************************************************************}
{*                              SensiAnaly                              *}
{************************************************************************}

PROCEDURE SensiAnaly;
    VAR
       option2:                          CHAR;
       s:                                STRING;
       conflict:                         ARRAY[2..11,1..11]OF BOOLEAN;
       tradeoff:                         ARRAY[2..11,1..11]OF EXTENDED;
       tempTrade:                        EXTENDED;
```

```
      serviceLvl,mu,sigma,Lbound,
      Ubound,tempRhs:                    REAL;
      x1,y1,x2,y2,x3,y3,x4,y4:           BYTE;
      conflictFlag,change:               BOOLEAN;

{*********************************************************************}
{*                          ListAchievmt                             *}
{*********************************************************************}

PROCEDURE ListAchievmt;
  BEGIN
    FOR i:=1 TO 2 DO
      WRITE(' Goal #   Desired Level  Actual Level ');
    WRITELN;
    FOR i:=1 TO 2 DO
      WRITE(' ------------------------------------ ');
    WRITELN;
    IF(nrows<=15)THEN
      BEGIN
        FOR i:=1 TO nrows DO
          WRITELN(i:4,rhs[i]:16:2,rhs[i]-neg[i]+pos[i]:17:2);
      END
    ELSE
      BEGIN
        FOR i:=1 TO 15 DO
          WRITELN(i:4,rhs[i]:16:2,rhs[i]-neg[i]+pos[i]:17:2);
        j:=1;
        FOR i:=16 TO nrows DO
          BEGIN
            GOTOXY(39,j+3);
            WRITELN(i:4,rhs[i]:16:2,rhs[i]-neg[i]+pos[i]:17:2);
            INC(j);
          END;
      END;
    message('','O',validSet4,inCh);
  END;

{*********************************************************************}
{*                         TradeoffAnly                              *}
{*********************************************************************}

PROCEDURE TradeoffAnly;

   BEGIN

     { Initialization }

     IF(solution='c')THEN   { If integer,'i',this is done in 'updateUB }
       BEGIN
         keepFlg1:=flg1;
         FinalZjCj;
       END;
     conflictFlag:=FALSE;          { Indicates if a conflict is present }
```

```
  FOR i:=2 TO nprt1 DO
    FOR j:=1 TO nprt1 DO
      BEGIN
        conflict[i,j]:=FALSE;
        tradeoff[i,j]:=1.0E20;
      END;

  IF(KeepFlg1)AND(firstTime)THEN
    BEGIN

    {Set the flag so the program does not execute these statements
     more than once for the current solution.  This can happen if
     this option is selected more than once for the current solution}

      firstTime:=FALSE;
      FOR i:=nprt1 DOWNTO 2 DO
        prty[i]:=prty[i-1];
      prty[i]:=0.0;
    END;
  FOR p:=2 TO nprt1 DO
    BEGIN
      IF(prty[p] >= 1.0E-10)THEN     {Goal is not fully achieved}
        BEGIN
          FOR k:=1 TO ncols DO
            BEGIN
              IF(OptZjCj[p,k] >= 1.0E-10)THEN    {Positive Zj-Cj}
                BEGIN
                  FOR i:=1 TO p-1 DO
                    IF(OptZjCj[i,k] <= -1.0E-10)THEN  {Neg Zj-Cj}
                      BEGIN
                        conflict[p,i]:=TRUE;
                        conflictFlag:=TRUE;
                        tempTrade:=-OptZjCj[i,k]/OptZjCj[p,k];
                        IF(tempTrade < tradeoff[p,i])THEN
                          tradeoff[p,i]:=tempTrade;
                      END;
                END;
            END;
        END;
    END;

 {If conflict exist, list conflicting objectives and their tradeoffs}

IF(conflictFlag)THEN
 BEGIN
  WRITELN(' Priority   <Conflicts with>   Priority','   Trade-Offs');
  WRITELN(' ------------------------------------------------------');
  FOR p:=2 TO nprt1 DO
    BEGIN
      FOR i:=p-1 DOWNTO 1 DO
        BEGIN
          IF(conflict[p,i])THEN
            BEGIN
```

```
                    IF(keepFlg1)THEN
                      BEGIN
                        j:=i-1;k:=p-1;
                      END
                    ELSE
                      BEGIN
                        j:=i;k:=p;
                      END;
                    IF(j=0)THEN
                      WRITELN(k:6,'           ..... System Constraints')
                    ELSE
                      BEGIN
                        WRITE(k:6,'           ..... ',j:12);
                        WRITELN(tradeoff[p,i]:13:2);
                      END;
                  END;
              END;
          END;
      END;

       IF(NOT conflictFlag)THEN
         message('*** WARNING ***  No Conflict is Present','0',validSet4,
                 inCh)
       ELSE
         message('','0',validSet4,inCh);
    END;

{*************************************************************************}
{*                              ChangePri                                *}
{*************************************************************************}

PROCEDURE ChangePri;
  LABEL a;

  BEGIN
    WRITELN(' Priority      Under      Current      New');
    WRITELN('   NAME     Achievement  Priority    Priority');
    WRITELN(' ------------------------------------------');
    FOR i:=1 TO nprt DO
      WRITELN(' Prty ',i:2,prty[i]:14:2,prtyOrder[i]:8);
    message('Change Priority? (Y/N) -'+CHR(16)+' ','1',validset4,
            answer);
    { Window is now (1,1,80,25) }
    IF(answer='Y')THEN
      BEGIN
        GOTOXY(4,22);
        WRITE('Swap Priority Level -'+CHR(16)+' ');
        x1:=WHEREX;y1:=WHEREY;
        GOTOXY(4,23);
        WRITE('With Priority Level -'+CHR(16)+' ');
        x2:=WHEREX;y2:=WHEREY;
a:      i:=inputInteger(x1,y1,1,nprt);
        j:=inputInteger(x2,y2,1,nprt);
```

```
        FOR k:=1 TO nprt DO
          BEGIN
            IF(prtyOrder[k]=i)THEN
              prtyOrder[k]:=j
            ELSE
              IF(prtyOrder[k]=j)THEN
                prtyOrder[k]:=i;
          END;
        flg4:=FALSE;
        FOR k:=1 TO nprt DO
          BEGIN
            GOTOXY(41,k+5);
            WRITELN(prtyOrder[k]:2);
          END;

        { Exchange the priorities }

        FOR k:=1 TO tprt DO
          BEGIN
            IF(obj[k].priority=i)THEN
              obj[k].priority:=j
            ELSE
              IF(obj[k].priority=j)THEN
                obj[k].priority:=i;
          END;
        message('More Changes? (Y/N) -'+CHR(16)+' ','1',validSet4,
                 answer);
        IF(answer='Y')THEN
          BEGIN
            blank(x1,y1,2);
            blank(x2,y2,2);
            GOTO a;
          END;
      END;
  END;

{*************************************************************************}
{*                              ChangeRhs                                *}
{*************************************************************************}

PROCEDURE ChangeRhs;
  LABEL a;
  VAR
     x1,y1,x2,y2,x3,y3,x4,y4:              BYTE;
  BEGIN
    FOR i:=1 TO 2 DO
      WRITE('  Row #    Current Value    New Value ');
    WRITELN;
    FOR i:=1 TO 2 DO
      WRITE('  ---------------------------------- ');
    WRITELN;
    IF(nrows<=15)THEN
      FOR i:=1 TO nrows DO
```

```
        WRITELN(i:4,rhs[i]:16:2)
      ELSE
        BEGIN
          FOR i:=1 TO 15 DO
            WRITELN(i:4,rhs[i]:16:2);
          j:=1;
          FOR i:=16 TO nrows DO
            BEGIN
              GOTOXY(39,j+3);
              WRITELN(i:4,rhs[i]:16:2);
              INC(j);
            END;
        END;
      message('Change RHS? (Y/N) -'+CHR(16)+' ','1',validset4,answer);
      { After above message the window is (1,1,80,25) }
      IF(answer='Y')THEN
        BEGIN
          GOTOXY(4,20);
          TEXTCOLOR(15);
          WRITE('Enter Row Number    -'+CHR(16)+' ');
          TEXTCOLOR(14);
          x1:=WHEREX;y1:=WHEREY;
a:        i:=inputInteger(x1,y1,1,nrows);
          IF(constType[i]='D')THEN       { Deterministic Constraint }
            BEGIN
              GOTOXY(4,21);
              TEXTCOLOR(15);
              WRITE('Enter New RHS       -'+CHR(16)+' ');
              TEXTCOLOR(14);
              x2:=WHEREX;y2:=WHEREY;
              REPEAT
                tempRhs:=inputReal(x2,y2);
                IF(tempRhs < 0.0)THEN
                  BEGIN
                    message('RHS Must be > 0 ','2',validSet4,answer);
                    STR(tempRhs,s);
                    blank(x2,y2,LENGTH(s));
                  END;
              UNTIL(tempRhs>=0.0);
            END;
          IF(constType[i]='N')THEN                              { Normal }
            BEGIN
              GOTOXY(4,21);
              TEXTCOLOR(15);
              WRITE('Enter Service Level -'+CHR(16)+' ');
              x2:=WHEREX;y2:=WHEREY;
              GOTOXY(4,22);
              WRITE('Enter Mean          -'+CHR(16)+' ');
              x3:=WHEREX;y3:=WHEREY;
              GOTOXY(4,23);
              WRITE('Standard Deviation  -'+CHR(16)+' ');
              x4:=WHEREX;y4:=WHEREY;
              TEXTCOLOR(14);
```

```
END;
IF(constType[i]='U')THEN                                    { Uniform }
  BEGIN
    GOTOXY(4,21);
    TEXTCOLOR(15);
    WRITE('Enter Service Level -'+CHR(16)+' ');
    x2:=WHEREX;y2:=WHEREY;
    GOTOXY(4,22);
    WRITE('Enter Lower Bound   -'+CHR(16)+' ');
    x3:=WHEREX;y3:=WHEREY;
    GOTOXY(4,23);
    WRITE('Enter Upper Bound   -'+CHR(16)+' ');
    x4:=WHEREX;y4:=WHEREY;
    TEXTCOLOR(14);
  END;
IF(constType[i]='N')OR(constType[i]='U')THEN
  BEGIN
    REPEAT
      serviceLvl:=inputReal(x2,y2);
      IF(serviceLvl<0.0)OR(serviceLvl>1.0)THEN
        BEGIN
          message('Service Level Must be Between 0 and 1',
                  '2',validSet4,answer);
          STR(serviceLvl,s);
          blank(x2,y2,LENGTH(s));
        END;
    UNTIL(serviceLvl>=0.0)AND(serviceLvl<=1.0);
  END;
IF(constType[i]='N')THEN
  BEGIN
    IF(serviceLvl=0.0)THEN serviceLvl:=0.001;
    IF(serviceLvl=1.0)THEN serviceLvl:=0.999;
    REPEAT
      mu:=inputReal(x3,y3);
      sigma:=inputReal(x4,y4);
      NInvCDF(mu,sigma,serviceLvl,tempRhs);
      IF(tempRhs<=0)THEN
        BEGIN
          message('Invalid Parameters - Enter Again','2',
                          validSet4,answer);
          STR(mu,s);
          blank(x3,y3,LENGTH(s));
          STR(sigma,s);
          blank(x4,y4,LENGTH(s));
        END;
    UNTIL(tempRhs>0.0);
  END;
IF(constType[i]='U')THEN
  BEGIN
    REPEAT
      REPEAT
        Lbound:=inputReal(x3,y3);
        IF(Lbound<0.0)THEN
```

```
                    BEGIN
                      message('Lower Bound Must be >= 0 ','2',validSet4,
                              answer);
                      STR(Lbound,s);
                      blank(x3,y3,LENGTH(s));
                    END;
                  UNTIL(Lbound>=0);
                  REPEAT
                    Ubound:=inputReal(x4,y4);
                    IF(Ubound<0.0)THEN
                      BEGIN
                        message('Upper Bound Must be >= 0 ','2',validSet4,
                                answer);
                        STR(Ubound,s);
                        blank(x4,y4,LENGTH(s));
                      END;
                  UNTIL(Ubound>=0);
                  IF(Lbound>Ubound)THEN
                    BEGIN
                      message('Lower Bound Must be < Upper Bound ','2',
                              validSet4,answer);
                      STR(Lbound,s);
                      blank(x3,y3,LENGTH(s));
                      STR(Ubound,s);
                      blank(x4,y4,LENGTH(s));
                    END;
                UNTIL(Lbound < Ubound);
                UInvCDF(Lbound,Ubound,serviceLvl,tempRhs);
              END;
            rhs[i]:=tempRhs;
            change:=TRUE;
            flg4:=FALSE;
            IF(i<=15)THEN
              GOTOXY(22,i+4)
            ELSE
              GOTOXY(60,i-15+4);
            WRITELN(rhs[i]:16:2);
            message('More Changes? (Y/N) -'+CHR(16)+' ','1',validSet4,
                    answer);
            IF(answer='Y')THEN
              BEGIN
                blank(x1,y1,2);
                WINDOW(3,21,78,23);
                CLRSCR;
                WINDOW(1,1,80,25);
                GOTO a;
              END;
        END;
    END;

{.....................................................................}
{.                          SensiAnaly                               .}
{.....................................................................}
```

```
BEGIN
  change:=FALSE; {Indicates if rhs or priority structure has changed}
  REPEAT
    WINDOW(2,2,79,23);
    CLRSCR;
    TEXTCOLOR(15);                          { Select white characters }
    GOTOXY(51,2);
    WRITELN('Current Model '+CHR(26),' ',filename);
    GOTOXY(51,3);
    CASE modelType OF
        'D':  WRITELN('Type: Deterministic');
        'C':  WRITELN('Type: Chance-Constrained');
        'S':  WRITELN('Type: Stochastic');
    END;
    TEXTCOLOR(14);                         { Select yellow characters }
    GOTOXY(3,4);
    WRITELN('[A]  List Actual vs. Desired Goals');
    GOTOXY(3,5);
    WRITELN('[B]  Perform Trade-off Analysis');
    GOTOXY(3,6);
    WRITELN('[C]  Change Priority Structure');
    GOTOXY(3,7);
    WRITELN('[D]  Change RHS of Goal/Real Constraints');
    GOTOXY(3,13);
    WRITELN('[E]  Return to Main Menu');
    message('Enter Option -'+CHR(16)+' ','1',validSet5,option2);
    WINDOW(2,2,79,23);
    CLRSCR;
    CASE option2 OF
      'A':
        BEGIN
          IF(change)THEN
            message('Model has been Changed, Resolve....','0',
                    validSet4,inch)
          ELSE
            BEGIN;
              TEXTCOLOR(15);
              GOTOXY(54,1);
              WRITELN(' Actual vs Desired Goals');
              TEXTCOLOR(14);
              ListAchievmt;
            END;
        END;

      'B':
        BEGIN
          IF(modelType='S')THEN
             message('This Option is Not Available for Stochastic '+
                    'Model','0',validSet4,inCh)
          ELSE
            BEGIN
              TEXTCOLOR(15);
              GOTOXY(54,1);
```

```
                WRITELN(' Trade-Off Analysis');
                TEXTCOLOR(14);
                TradeoffAnly;
              END;
          END;

        'C':
          BEGIN
            TEXTCOLOR(15);
            GOTOXY(52,1);
            WRITELN(' Change Priority Structure');
            TEXTCOLOR(14);
            ChangePri;
          END;

        'D':
          BEGIN
            TEXTCOLOR(15);
            GOTOXY(54,1);
            WRITELN(' Change RHS Values');
            TEXTCOLOR(14);
            ChangeRhs;
          END;
      END                                  { End of the case statement }
    UNTIL(option2='E');
    WINDOW(1,1,80,25);
  END;
END.                                                  { End of dbasUtil }
```

VITA

MORTEZA ABTAHI

Candidate for the Degree of

Doctor of Philosophy

Thesis: AN INTERACTIVE MULTICRITERIA APPROACH TO FACILITY LOCATION-ALLOCATION MODELS UNDER STOCHASTIC DEMAND

Major Field: Industrial Engineering and Management

Biographical:

Personal Data: Born in Isfahan, Iran, March 12, 1957, the son of Nahid Madani and Hossein Abtahi. Married to Marzieh Torabian on January 12, 1985.

Education: Graduated from Ershad High School, Tehran, Iran in June 1975; received Associate Degree in Electronics from Tehran Technical College, Iran in June, 1977; received Bachelor of Science Degree in Electronics Engineering Technology from Oklahoma State University in May, 1981; received Master of Science Degree in Industrial Engineering from Oklahoma State University in May, 1983; completed requirements for the Doctor of Philosophy degree at Oklahoma State University in July, 1989.

Professional Experience: Teaching and Research Associate, School of Industrial Engineering and Management, Oklahoma State University, Fall, 1982 to Summer 1989. System Manager for HP-3000 minicomputer, and microcomputers, School of Industrial Engineering and Management, Oklahoma State University, Fall 1984 to Fall 1988.

Professional Organizations: Institute of Industrial Engineers, Operations Research Society of America, Alpha Pi Mu, Tau Beta Pi.