To the Graduate Council:

I am submitting herewith a thesis written by David Diedrich Holmes entitled "A Multiobjective Optimization Model for the Land Allocation Problem." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Management Science.

Richard E. Rosenthal

Richard E. Rosenthal
Major Professor

We have read this thesis and recommend its acceptance:

Kenneth C. Gilbert

Pat L. Fisher

Robert S. Markland

Accepted for the Council:

L. Evans Roth

Vice Chancellor
Graduate Studies and Research

A MULTIOBJECTIVE OPTIMIZATION MODEL FOR THE LAND ALLOCATION PROBLEM

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

David Diedrich Holmes

March 1982

3057965

ABSTRACT

This thesis presents the development and application of a multiobjective optimization model for the land use allocation process. An integer programming model was developed for locating land allocations based on the cost of the proposed development, proximity to existing land uses, and the shape of the final allocation. The solution technique uses an implicit enumeration scheme to iteratively identify efficient solutions. The model and solution technique were implemented in an interactive computer system designed for use by the land use planning professional. The system uses a grid cell structure to represent geographic locations for the final allocation. To test the system, allocations were made for a thirteen acre residential development across a 2,250 acre study area.

TABLE OF CONTENTS

CHAPTER		PAGE
1.	BACKGROUND OF APPLICATION FIELD AND INTRODUCTION TO MODEL	1
	1.1. Use of Computers in Land Use Planning	1
	1.2. Models for Land Use Planning	1
	1.3. Objective of This Study	7
	1.4. Geographic Information Systems	7
2.	REVIEW OF MULTIOBJECTIVE OPTIMIZATION TECHNIQUES	9
	2.1. The Difficulty of Defining a "Best" Decision	9
	2.2. Utility Functions	10
	2.3. Efficiency	12
	2.4. Classification of Techniques	14
3.	MULTIOBJECTIVE OPTIMIZATION MODEL FOR THE LAND ALLOCATION PROBLEM	20
	3.1. Overview of Model	20
	3.2. Description of Objectives	21
	3.3. Use of Efficiency	25
4.	A NEW ALGORITHM FOR PRODUCING EFFICIENT SOLUTIONS	27
	4.1. Description of Algorithm	27
	4.2. Proof of Efficiency	31
5.	MOLANDA—A MULTIOBJECTIVE LAND ALLOCATION SYSTEM	33
	5.1. Introduction to System and Solution Techniques	33
	5.2. Background on Implicit Enumeration	34
	5.3. Enumeration Technique Selected	35
	5.4. Cost Subproblem	39
	5.5. A Lower Bound in the Cost Subproblem	45
	5.6. Proximity Subproblems	47
	5.7. Shape Subproblem	52
	5.8. Alternative Enumeration Methods	54
6.	EXAMPLE OF USING THE MOLANDA SYSTEM	59
	6.1. MOLANDA Test Description	59
	6.2. Computational Considerations	70

CHAPTER PAGE

7. CONCLUSION 78

LIST OF REFERENCES 79

APPENDIX 83

VITA 119

LIST OF FIGURES

FIGURE		PAGE
1.	Hierarchical Framework of References Based on Information Requirements	16
2.	Example of Calculating Shape Values of Two Possible Configurations	24
3.	Cells for Cost Subproblem Example	41
4.	Cells for Proximity Subproblem Example	49
5.	Chart Showing Cost-Rank Relationship for Three Enumeration Methods	56
6.	Map of Definition Layer for the MOLANDA Test Example	60
7.	Map of Cost Layer for the MOLANDA Test Example	61
8.	Optimal Shape Configuration Using Five Cells	63
9.	Listing of MOLANDA Example Run	64
10.	Listing of Solution File for MOLANDA Example Run	71
11.	Eight Solutions Found in the MOLANDA Test Example	74

CHAPTER 1

BACKGROUND OF APPLICATION FIELD AND INTRODUCTION TO MODEL

1.1. Use of Computers in Land Use Planning

During the past few years, land use planners and managers have come to rely on the computer as an important tool in the land use allocation process. This is due primarily to the ever-increasing amount of natural resource, economic, cultural, and environmental information required to decide effectively on the use of the land. The task of allocating certain areas of land for particular uses by manually compiling and assimilating all necessary data has become a formidable task. The use of the computer, however, can facilitate this process by providing a planner with the ability to combine, analyze, and evaluate the various data in a systematic process. This powerful tool can be used as a significant aid to the decision-making process involved in the land use planning and management field.

1.2. Models for Land Use Planning

A common problem approached in the land use planning process is locating contiguous tracts of land for new development or other particular purposes. Ideally, the resulting land allocation should have the best combination of economic, physical, proximity, and shape characteristics for the proposed land use.

Typically, computerized land allocation models have involved descriptive analyses of physical land capability and proximity to other existing land uses. Howard and Baxter (1976), described a systematic approach to resource management by combining and weighting geographic data variables to indicate suitable areas for the desired land use. An example of this approach might interpret data variables such as soil type, elevation, and water to produce data variables such as prime farmland, topographic slope, and flood areas which are combined and indexed to form suitability maps for the proposed land use. Berry and Tomlin (1979), compared the process of cartographic modeling to algebraic evaluation in which primitive data variables are mathematically interpreted to produce cumulative expressions which represent the land use suitability maps. They also described complex descriptive analyses such as studying the effect of computing the slope of a slope map to indicate areas of topographic transition.

This type of land allocation model can be applied on a site-specific scale as well as a regional scale. Baxter et al. (1975), demonstrated the use of descriptive land use planning models in various applications at different scales. For example, regional maps were produced indicating suitable counties for industrial locations as well as wildlife habitat maps indicating suitable areas for habitat improvement.

The common procedure for creating the suitability maps involves overlaying the various basic and interpreted data variables to indicate areas of high coincidences. By analyzing these coincidences, the planner produces the final suitability maps which can be used to select the locations for the proposed land uses.

The final step in the descriptive analysis technique is to allocate manually the areas for the desired land uses based on the various suitability maps produced. This final step contains two inherent problems which affect the quality of the overall land use plan. First, there is no systematic method for making final allocations from suitability maps. The decision maker must manually compile and assimilate the numerous data variables that went into the land suitability model and select the best allocation plan from multiple possibilities. At this stage of the planning process, it is difficult to understand the relationships between the component data that went into the model in order to select the best location for the desired land use. The second problem involves the land allocation process itself. Often, a suitability map will indicate desirable areas which do not form contiguous areas large enough for the proposed land use. The decision maker is forced to select both desirable and nondesirable areas in order to form a contiguous land tract which meet his size requirements. This introduces subjectivity and uncertainty in the final allocation, since various levels of suitability will be combined in the final allocation. Also in this process, the decision maker is faced with numerous possible combinations of desirable and nondesirable areas. This adds to the uncertainty of the final allocation.

Another type of computerized land allocation model involves a mathematical process using optimization techniques. This type of model uses the physical, economic, and proximity parameters of geographically-fixed areas such as land tracts to allocate all or portions of these

areas to particular land uses. One typical result of this process is an acreage allocation of the land tracts in order to meet predefined levels of the various parameters.

Bammi and Bammi (1979), described such a model used to allocate regional acreages for a comprehensive county land use plan. This model used multiple objectives and constraints on desired growth patterns for the area to arrive at optimal acreages. The objectives considered were conflicts between different land uses, travel distances, tax costs, environmental impacts, and capital costs of community facilities. Constraints were set on each type of land use based on demographic, socioeconomic, and natural resource considerations in order to guide the solution process along desirable growth patterns. The solution process involved optimizing each objective individually and then generating a composite objective function by adding scaled values of the various objective functions. The process used the composite objective function to determine the optimal solution. The decision makers were allowed to vary the weights applied to the different objective functions, and thus played an integral role in the determination of the most satisfactory county land use plan.

Another similar model, Leuschner et al. (1975), allocated various land use activities to aggregated land units to meet desired objectives and constraints. This model dealt with production objectives such as timber and hunting and land management activities such as cutting timber or improving wildlife habitat. The model was formulated as a single-objective linear programming model using timber production as the

objective function and treating the other objectives as constraints. Other constraints included financial and physical resource limitations. The variables used were expressed as activity levels in aggregated land units which related them to geographic locations. Multiple use planning was accomplished by forcing the linear program to meet other production objective levels in the form of constraints while optimizing on timber production. Sensitivity analyses were performed to better understand the interactions between the different production objectives. The model was successfully tested on a seventy thousand acre region in Jefferson National Forest.

Other optimization models dealing strictly with acreage allocations have been proposed by Bottoms and Bartlett (1975), Schular and Meadows (1975), and Dress and Field (undated). In the Schular paper, for example, goal programming is used to determine optimal acreage allocations based on different resources such as domestic forage, wildlife forage, fish, lodgepole pine, and spruce-fir. These are used to produce various products such as cow and steer grazing, recreation days for camping, board feet for timber, and wildlife grazing. In addition, several land management alternatives were considered which change the levels of the various products as well as the resources. The model was solved several times using different orders of goal priorities which allowed the planners to evaluate different strategies based on the importance of the various production objectives.

As in the descriptive analysis technique, the mathematical modeling technique also contains an inherent problem when addressing the land

allocation problem. The problem is that land allocation is made on predefined units across the study area. This does not allow the decision maker the flexibility of locating spatially new areas for proposed land uses. Rather, he must predefine the possible land units and use the model to aid in the activity allocation on these areas.

One exception to this group of models was a solution technique to a political districting problem by Garfinkel and Nemhauser (1970). They designed a program which spatially allocated population units into contiguous districts based on population requirements. The model was formulated as an integer programming problem and solved using implicit enumeration techniques. The model was successfully used on the State of Washington in which 39 counties were allocated into seven political districts.

There are advantages and disadvantages to both model types which limit their effectiveness in the land use planning process. The descriptive analysis technique can consider the spatial location of a land allocation but cannot optimally determine the appropriate land use activities. The mathematical modeling technique can produce an optimal allocation plan based on predefined constraints and objectives, but cannot consider new spatial locations of the land use activities in the allocation. Both types, however, have been used extensively and effectively in the land use planning field.

1.3. Objective of This Study

The purpose of this study was to design and develop a computerized tool for land allocation which avoided the inherent problems in the previous techniques. The thesis model would be an attempt to combine an optimization technique with a descriptive analysis technique which included spatial variables. The model would allocate contiguous areas which met criteria based on the economic, physical, proximity, and shape characteristics for the proposed land use. Professional planners and resource analysts could use the model to effectively aid the decision making process of land management.

1.4. Geographic Information Systems

The descriptive analysis models discussed previously typically use a type of computer system referred to as a Geographic Information System (GIS). A GIS can be used as an effective tool in analyzing and evaluating the various geographic data involved in the land use allocation model.

Basically, a GIS is any system capable of storing, managing, retrieving, analyzing, and displaying natural resource, physical, cultural, socioeconomic, and other data which are referenced to a place on the earth. The thesis model is based on the use of a grid cell GIS which partitions the area to be studied into a uniform grid of rows and columns forming equal-sized cells. Each cell represents individual units of land which can be addressed by the model. Typically, a layer of cells is used to define a particular geographic variable, such as slope or land

cover. For example, each cell in a slope layer contains a value describing either the actual slope percentage of the land unit covered by the cell or a category of slope percentages (e.g., 1 = 0-10%, 2 = 11-20%, etc.). In the descriptive land allocation model, cell layers are combined and analyzed to determine those cells possessing the most favorable levels of each geographic variable for the proposed land use. This type of GIS has been highly successful in a variety of land use planning and management applications, Holmes and Jolly (1980).

CHAPTER 2

REVIEW OF MULTIOBJECTIVE OPTIMIZATION TECHNIQUES

2.1. The Difficulty of Defining a "Best" Decision

A common problem faced by decision makers is choosing the "best" or "optimal" course of action among feasible alternatives. However reasonable this desire may sound, in some cases it can lead to disagreement. The basic problem is in defining the "best" solution.

This is not always a difficulty. There have been many management science applications in which the best course of action is well-defined. In these cases, the optimal solution is one which maximizes or minimizes a well defined real-valued objective function.

But in other decision making problems, there is no single objective function which can adequately measure the difference in desirability between feasible solutions. Multiobjective optimization is the subfield of management science which deals with these types of problems. (A recent textbook is by Zeleny (1982).)

The land allocation problem addressed in this thesis belongs to this class of problems. Rosenthal (1979) described the class of multiobjective problems as important, but ill-defined. He presented a general definition of the problem: given a decision variable $x \in R^n$, a set of feasible decisions $X \subset R^n$, and a set of real-valued objective functions $f_i(x)$, $i=1, \ldots, k$, find an $x \in X$ such that the most preferred vector of objective function values $F(x) = (F_1(x), \ldots, F_k(x))$ is attained. The problem is referred to as ill-defined because the phrase

"most preferred" has no universally applicable definition and thus depends on subjective human judgment. As a result, in the presence of multiple objectives, there is no absolute meaning of a "best" or "optimal" course of action.

One exception to this idea is the unrealistic case in which none of the objectives are in conflict with each other. In this unlikely case, a solution exists which optimizes each objective independently and thus is the undisputed "best" solution to the multiobjective problem.

More realistic cases involve objectives which conflict with each other so that no one solution optimizes all of them simultaneously. For example, a labor allocation problem in a firm with a highly fluctuating workload must consider objectives such as high productivity per worker and low turnover which might be in conflict with each other. As a result, any "preferred" solution is some compromise between objectives which can only be determined by the decision maker's subjective value judgment.

2.2. Utility Functions

There is a second exception to the statement that no absolute meaning of "best" decision exists. This is when a utility function is used to model the conflicts of the multiple objectives. This theoretically ideal approach to the multiobjective optimization problem is to solve:

$$\underset{x \in X}{\text{Max}} \; U(f_1(x), \ldots, f_K(x))$$

where the utility function $U: R^k \rightarrow R$ has the property that: given two feasible alternatives $y, z \in X$,

$$U(F(y)) > U(F(z))$$

if and only if the vector of objective attainments F(y) is "preferred" to the vector F(z) (Rosenthal (1979)). If the function U can be explicitly defined, the problem reduces to a well-defined single-objective maximization problem. Unfortunately, determining a suitable utility function is usually difficult for most applications and renders this ideal approach impractical. There have been, however, applications where explicit utility functions have been successfully derived and used. In most of these instances, there is a very small number of feasible solutions which are completely enumerated prior to the "optimization." Such problems may be more correctly described as problems in decision theory rather than multiobjective optimization. For example, Keeney (1978) used an explicit utility function approach for the evaluation of ten potential pumped storage sites for generating electricity.

One other approach of using utility functions is described by Geoffrion et al. (1972). They do not define an explicit utility function but assume that an implicit utility function exists and solve the corresponding maximization problem. This technique will be discussed further in the classification section.

2.3. Efficiency

In all further discussion of the multiobjective problem, it is assumed that an appropriate explicit utility function is not available. This section renews the concept of efficiency which is similar to utility in that it provides an objective method of determining in some cases preference between two alternative solutions.

The monotonicity assumption must hold in order for the concept of efficiency to be applicable. This simply means that for every objective function $f_k(x)$ (possibly redefined as $-f_k(x)$), a decision maker's satisfaction will never decrease as $f_k(x)$ increases. This is a reasonable assumption for most applications including the land allocation problem presented in this thesis. An exception in using this assumption is the deer population objective used in the forest management model of Harrison and Rosenthal (1981). For recreational purposes, more deer are normally preferred except when the population is large enough to defoliate the vegetation understory.

The concept of efficiency classifies feasible solutions as either efficient (also known as noninferior, nondominated, or Pareto optimal) or dominated (also known as inferior or inefficient). Efficient solutions are preferred to dominated ones, but there is no way to determine preference between two efficient solutions without introducing additional considerations or subjectivity on the part of the decision maker. As a result, multiobjective techniques which generate efficient solutions usually depend on the decision maker's input to determine a most preferred solution or set of solutions. In comparison to utility functions,

efficiency is less discriminating in determining preference between alternative solutions, but has been found to be more applicable.

Efficient solutions are defined as follows: given solutions y, z ϵ X, z is dominated by y if and only if

$$f_k(y) \geq f_k(z), \quad k=1, \ldots, K$$

with strict inequality for at least one objective. (With the monotonicity assumption, a rational decision maker would prefer solution y over solution z.) A solution x ϵ X is efficient if and only if there exists no other solution x ϵ X which dominates it.

There has been extensive literature written on the theory and use of efficient solutions. For example, Soland (1979) presented a general characterization of efficient solutions in terms of optimal solutions to particular scalar optimization problems. Ecker and Kouada (1978) and Zeleny (1982) presented a method for generating the entire set of efficient solutions for a multiobjective linear programming problem. However, the efficient set is typically quite large so methods are required for identifying partial sets of efficient solutions which are meaningful to the decision maker. A paper by Ecker and Shoemaker (1980) described an algorithm for selecting desirable subsets of the efficient set for multiobjective linear programming problems. One type of desirable subset that they generate is based on minimizing the maximum deviation between the objective function values and some ideal vector. The ideal vector may be defined as the maximum value individually attainable for each objective function.

The algorithm presented in this thesis also generates desirable subsets of the efficient set. The algorithm is discussed in detail in Chapter 4. The following section discusses several other methods for generating partial sets of efficient solutions.

2.4. Classification of Techniques

A number of different multiobjective techniques which do not require explicit utility functions and provide methods for determining meaningful solutions are available. Ho (1979b) presented a classification of multiobjective techniques based on the demand for information required of the decision maker. He examined the human interaction involved in multiobjective methods by assessing the decision maker's ability to express his value judgment towards alternative solutions. In the following is a brief review of current methods based on the decision maker's role in the solution process. Three different types of information requirements used by the various methods are presented first.

Tradeoffs can be used in cases where it is desirable to estimate marginal utility but an explicit utility function is not available. In evaluating a solution, utility can be estimated by determining from the decision maker the amount of change in one objective he or she would consider acceptable compensation for a specified change in another.

In some cases, it is only necessary to rely on a decision maker's preference judgment to evaluate alternative solutions. That is, given two alternative solutions, the decision maker is able to determine preference for one of them or indifference between them.

Another information requirement useful in some problems is for the decision maker to specify a priority ranking of the objectives. This forces the decision maker to define certain objectives as being more important than others. In the case of preemptive priorities, the most preferred solution is one which optimizes the objectives lexicographically. That is, the highest priority objective is optimized individually first. If there is more than one optimal solution, the second highest priority objective is used to break ties. If there is still more than one optimal solution, the third objective is used to break ties. This is repeated until all objectives are used or only one optimal solution is found. Rosenthal (1979) has pointed out that this method usually has two drawbacks. It does not allow tradeoff of a small decrement in a high priority objective for large improvement in a low priority objective and it is limited in applicability to problems which have alternate optimal solutions after each individual optimization.

Tradeoffs, preferences, and priorities have all been used in variety of multiobjective techniques, either individually or in combinations with each other. The following will briefly describe several techniques using these information requirements. Figure 1 illustrates the applications referenced with their corresponding information requirements.

A method using only tradeoff information is presented by Zionts and Wallenius (1976). Their algorithm for solving multicriterion linear programs involves generating efficient solutions in each iteration by maximizing a weighted sum of the criteria. Marginal tradeoffs of

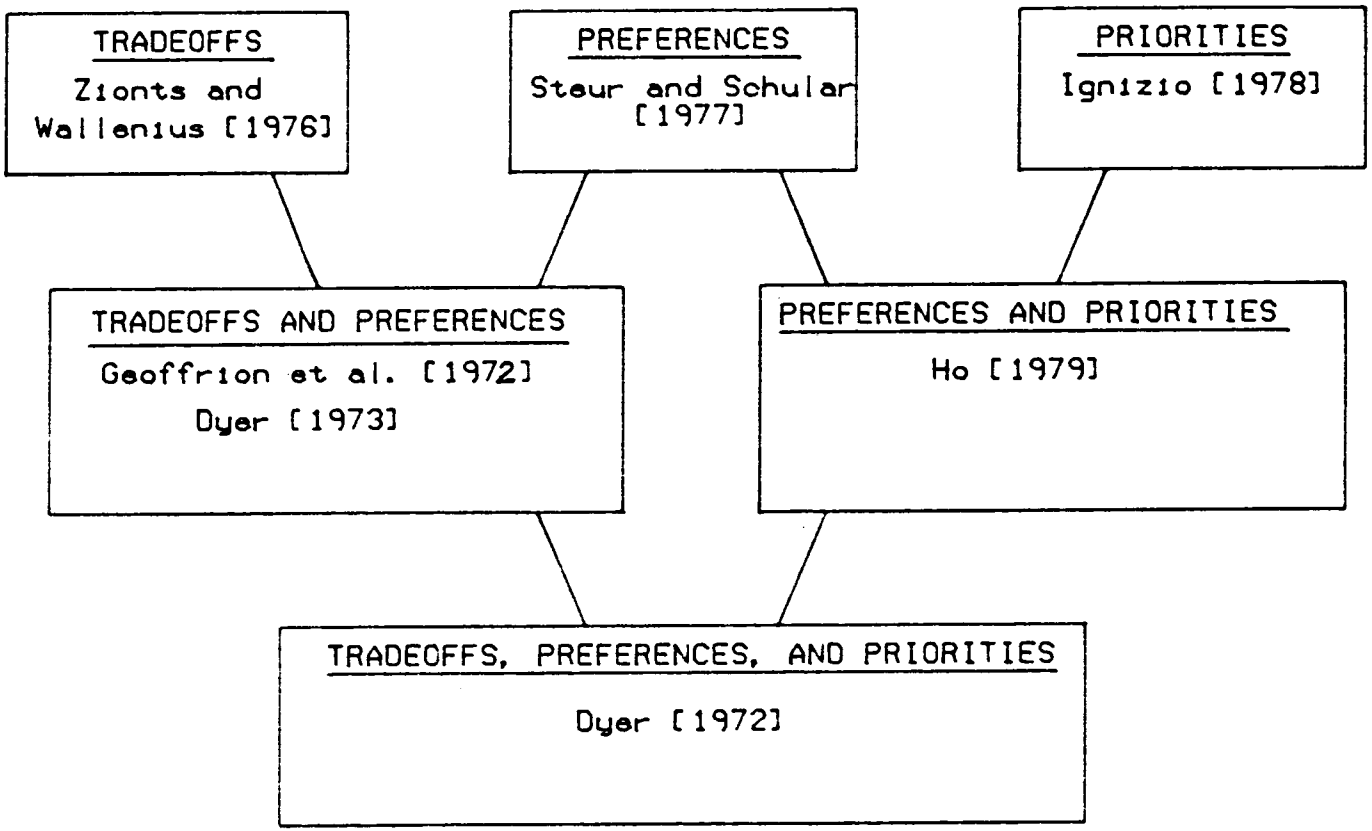


Figure 1. Hierarchical framework of references based on information requirements.

the objectives are computed and presented to the decision maker. Using this evaluation of the tradeoffs, new efficient solutions are generated and the process is repeated.

Using both tradeoffs and preferences is demonstrated by Geoffrion et al. (1972) and Dyer (1973). They developed an implementation of the Frank-Wolfe gradient-step technique (Wolfe (1972)) which uses tradeoffs to determine the direction of improvement over the current solution. The tradeoff evaluation is based on determining changes in objective levels of the current solution that are considered indifferent by the decision maker. The amount of improvement of the current solution is determined by the decision maker choosing a preferred solution over several alternatives. This is used to determine a new set of feasible solutions and the process is repeated.

An application using tradeoffs, preferences, and priorities is the interactive goal programming method presented by Dyer (1972). Initially, the decision maker uses his priority judgment to set goals for each objective. Higher priority objectives would have goals closer to the optimal values of the individual objectives. A goal programming problem is solved which minimizes a weighted sum of the deviations below these goals. As in the Geoffrion et al. (1972) method, the weights represent marginal tradeoffs between objectives. In each iteration, the decision maker can change these weights which are used in another goal programming problem to define a new search direction. The decision maker is allowed to evaluate feasible solutions along the resulting search direction and his most preferred solution is used to start the next iteration.

A method using preferences only is presented by Steur and Schular (1977). In each iteration, several efficient solutions are presented to the decision maker from which the most preferred is selected. This is used to generate a new set of efficient solutions based on exploring the surrounding feasible region. They applied this method to determine a forest management plan of greatest acceptability.

The use of preferences and priorities is demonstrated by Ho (1979) in the HOPE method for solving multiobjective linear programming problems. Parametric linear programs are solved to determine partial sets of efficient solutions. The decision maker's ordering of the objectives is used to structure the parameterizations of weights on the objectives. At each iteration, the decision maker selects the most preferred efficient solution from the available set. Weights corresponding to the selected solution are individually parameterized in successive linear programs to generate a new partial set of efficient solutions.

The use of priorities only is demonstrated by preemptive goal programming as presented by Ignizio (1978). Goal programming is based on minimizing weighted deviations from specified targets for each objective. The preemptive case of goal programming applications uses the preemptive priority concept of lexicographically optimizing the objectives. This implies that the weight of the first priority is considerably greater than the weight of the second priority which is considerably greater than the weight of the third priority, and so on.

As discussed, there are many different approaches to solving the multiobjective optimization problem. Each technique must be evaluated on its merit as applied to particular applications. A current theme,

however, is an interactive role by the decision maker in determining a most preferred solution or satisfactory subset of efficient solutions. This capability is essential to the future success of implementing optimization tools in real-world problems.

CHAPTER 3

MULTIOBJECTIVE OPTIMIZATION MODEL FOR THE LAND ALLOCATION PROBLEM

3.1. Overview of Model

The land allocation model allocates a contiguous area for the proposed land use based on four objectives: total development costs, proximity to desirable and undesirable existing land uses, and the shape of the final allocation of land. The model uses a grid cell structure to represent the study area being considered. Therefore, the final allocation will be a set of contiguous cells which meet particular levels of the four objectives. Desirable land uses are referred to as amenity land uses and undesirable land uses are referred to as detractor land uses. Before the model is applied, the decision maker designates particular cells in the study area to represent either amenity or detractor land uses.

The general formulation for the multiobjective land allocation model is as follows:

Given a set of cells G defining the study area, find a set of contiguous cells $I \subset G$ so as to

$$\text{Minimize } \sum_{i \in I} c_i,$$

$$\text{Minimize } \{\text{maximum } a_i,\ i \in I\},$$

$$\text{Maximize } \{\text{minimum } d_i,\ i \in I\},$$

$$\text{Minimize } S(I),$$

$$I \subset X,$$

$$||I|| = k$$

where c_i = cost of using cell i for proposed land use,

a_i = distance from cell i to nearest amenity land use cell,

d_i = distance from cell i to nearest detractor land use cell,

$S(I)$ = compactness (shape) value of solution I,

X = set of feasible cells in G.

3.2. Description of Objectives

The cost objective is based on minimizing the sum of the individual cell costs over the final cell allocation. Each cell cost represents all costs involved for the proposed land use. For example, if a residential area is to be located, the costs should include site preparation and construction costs on each cell. These costs are typically derived from a weighted index of various geographic variables such as slope, land cover, soil type, depth to bedrock, and accessibility. The required variables depend on the type of land use to be located and will change considerably for each one.

The proximity objectives are measured by the distance between the solution set I and cells designated as amenity land uses or detractor land uses. The amenity objective is measured by the maximum distance between any cell in I and the nearest amenity land use cell. By

minimizing the maximum distance, the decision maker is guaranteed that all other cells in the set I are at least as close or closer to an amenity land use. The detractor objective is measured by the minimum distance between any cell in I and the nearest detractor land use cell. By maximizing the minimum distance, the decision maker is guaranteed that all other cells in the set I are at least as far or farther away from a detractor land use. By using bottleneck objectives, the model attempts to provide equity of distance between the cells in the final allocation. As a result, the distances to desirable and undesirable land uses are distributed so that no one cell is considerably better or worse than any others.

The cells designated either as amenity or detractor land uses do not have to be contiguous but rather can be located individually or in groups anywhere in the study area. For example, a system of rivers, ponds, and lakes could be used as the amenity land use category. The model allows for one amenity and one detractor objective corresponding to one amenity and one detractor land use category. Additional proximity objectives could possibly be added to the model; however, too many objectives can easily confuse the decision maker in evaluating different land allocations. One possible approach allows for different land use types but uses only one amenity and one detractor objective. The land uses are weighted according to their importance in the location of the proposed land use. Distances would be affected by the weights so that different land uses would appear further from or closer to a land allocation depending on their importance. This addition is left to future research.

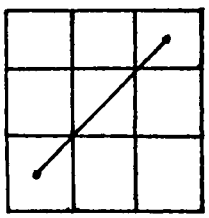
The shape objective is a measure of the compactness of the configuration of allocated cells. This value is calculated as the product of the perimeter of the shape and its diameter. The perimeter is the number of outside edges in the shape and the diameter is the maximum distance between any two cells in the shape. That is, compactness is measured by:

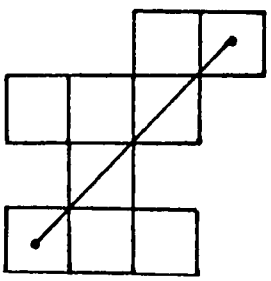
$$S(I) = \sum_{j=1}^{k} y_j * (\max_{i,\ell \in I} [b(i,\ell)]),$$

where y_j is the number of outside edges of cell j not adjacent to another cell in I and $b(i,\ell)$ is the Euclidean distance between cells i and ℓ. Note that S(I) is a nonseparable function since a complete allocation of k cells must be made before it can be computed. In contrast, the cost objective is separable since it can be computed on a partial solution (and use that information in a solution procedure).

As illustrated in Figure 2, the shape value will decrease as the configuration of cells in I form a more compact shape. Therefore, since this objective is minimized, an underlying assumption to the model is that a more compact shape is preferred over a dispersed one.

Other compactness values that were considered have been used for similar problems. Garfinkel and Nemhauser (1970), used the diameter squared divided by the area of an allocation to indicate compactness. This measurement also decreases as the shape becomes more compact. The IMGRID System, Holmes and Jolly (1980), includes a shape measurement which increases as the shape becomes more compact. The square root of





Number of outside edges:	12	20
Maximum distance:	$\sqrt{8}$	$\sqrt{18}$
Shape value:	33.94	84.85

Figure 2. Example of calculating shape values of two possible configurations.

the shape's area is divided by its perimeter to measure the compactness. This measurement, like the one used in the thesis model, is designed to penalize shapes with uneven edges or many protrusions.

Typically, area is used in measuring the compactness of a land allocation. The compactness measurement in the thesis model, however, does not require the area of an allocation since it is fixed for any allocation of k equal-sized cells. The diameter of an allocation provides a suitable measure of compactness and the perimeter provides a measure of shape. Thus, the product of the two provides an accurate measurement of shape and compactness for the land allocation model.

The feasible set of cells X represent all cells in the study area where the proposed land use could be located. Cells representing either the amenity or detractor land use are not considered feasible. Other cells not to be considered due to external reasons are referred to as reject cells and are excluded from the set X.

3.3. Use of Efficiency

The overall approach to the solution of the land allocation model will be to generate a set of efficient solutions as defined in Chapter 2. Underlying the use of efficient solutions is the monotonicity assumption which indicates that the more of each objective the decision maker can achieve, the happier he or she will be with the final solution. In the case of the land allocation model, the decision maker will always prefer lower values for the cost, amenity, and shape objectives, and higher values for the detractor objective.

Therefore, the technique of generating efficient solutions for the land allocation model is appropriate. Only efficient solutions will be presented to the decision maker for evaluation and comparison. By using his professional judgment, the most preferred efficient solution or set of efficient solutions can be chosen for the final land allocation.

CHAPTER 4

A NEW ALGORITHM FOR PRODUCING EFFICIENT SOLUTIONS

4.1. Description of Algorithm

A new algorithm is presented for generating a subset of the efficient solutions to the problem. Numerous authors including Ecker and Shoemaker (1980), Hultz et al. (1981), Soland (1979), Ho (1979a), and Steur and Schular (1978) have suggested that using the approach of partial generation of the efficient set is the most advisable approach to some multiobjective problems. The algorithm presented here takes this approach and makes an attempt to generate "good" efficient solutions in some sense.

The new algorithm at each iteration involves solving single-objective subproblems which yield efficient solutions and presenting these solutions to the decision maker. By changing specifications of the subproblems, numerous efficient solutions can be generated in the process. The decision maker controls these specifications so that he can guide the algorithm to efficient solutions he considers "good." In terms of the classification of algorithms in Chapter 2, this process assumes the decision maker can provide preferences, but does not require tradeoffs or priorities. Therefore, given two efficient solutions, he is able to determine preference for one or indifference between them.

Each iteration begins with a previously determined efficient solution, whose vector of objective function values is referred to

as the current point. The decision maker is required to select one objective to improve over its current level. If there are k objectives, k-1 subproblems are then solved. Each subproblem attempts to strictly improve the selected objective, while exactly one of the other objectives is allowed to degrade. (Improvement of the selected objective without some degradation in another objective is impossible, because the current point is efficient.)

To describe this idea mathematically, let the current efficient solution be x_0, with objective function vector:

$$f(x_0) = \begin{bmatrix} f_1(x_0) \\ f_2(x_0) \\ \vdots \\ f_k(x_0) \end{bmatrix}$$

and suppose f_j is selected for improvement. Assume that all k objectives are to be maximized in the model. Then the following k-1 subproblems are solved in an iteration of the new algorithm:

For $m = 1, \ldots, k$, $m \neq j$, solve the problem

$$\text{Maximize } f_m(x)$$

$$\text{such that } f_j(x) \geq f_j(x_0) + \varepsilon, \quad (\varepsilon > 0),$$

$$f_i(x) \geq f_i(x_0), \quad (i = 1, \ldots, k, \text{ except } i \neq j, i \neq m).$$

$$x \subset X$$

(ε could be chosen by the user or as in the case of the land allocation model which has discrete valued objective functions, it is the smallest possible value of improvement.)

It will be shown that if E is the set of alternate optimal solutions to a subproblem, then at least one member of E will be efficient. Moreover, a new efficient point, if it exists, as compared to the current point, will have an improved value of f_j, a degraded value of f_m, and equal or better values of the remaining k-2 objectives.

The new efficient solutions generated in the subproblems are returned to the decision maker for evaluation. Also presented are differences in the levels of particular objectives between the current solution point and each new one. The differences are determined for the objective selected to improve and the one allowed to degrade in each subproblem. The ratio of these differences represents tradeoff values between the current efficient solution and each new one. By comparing the tradeoff values, the decision maker can determine his next step. He may choose one of these efficient solutions or any other previously generated efficient solution as the new current point. It is not guaranteed that any given subproblem will produce a new efficient solution (because some subproblems may be infeasible). Therefore, on occasion, the decision maker must resort to previously generated efficient solutions for the new current point or to a different choice of the objective to be improved. The process is repeated until the decision maker is satisfied with one or more of the efficient solutions found.

The new algorithm attempts to meet some of the required criteria for an acceptable multiobjective optimization technique, as defined by Hultz et al. (1981). It allows the decision maker to play an integral role in the process of producing efficient solutions. The process capitalizes on the decision maker's understanding of the objectives and their interrelationships, and uses this insight in working towards the most satisfactory solution. It also ensures the acceptability of the resulting solutions since the decision maker is closely involved in the process.

A statement of the new algorithm is presented:

1. Initialization: Determine an initial efficient solution. (One method is presented in the following section.) Go to step 2.

2. Selection: Present the current efficient solution to the decision maker and request that one objective be selected for improvement over its current level. Go to step 3.

3. Optimization: Using the selected objective, solve k-1 subproblems for new efficient solutions. Go to step 4.

4. Evaluation: Present the new efficient solutions (if any) to the decision maker for evaluation. Request that either one of the new efficient solutions or any previously generated efficient solution be considered the new current solution. Either return to step 2 and continue the process or stop if the decision maker is satisfied with the set of efficient solutions generated so far.

4.2. Proof of Efficiency

To prove that only efficient solutions are being generated by the new algorithm, it is necessary to show that any of the k possible subproblems is only generating efficient solutions. A by-product of this proof will be a method for generating an initial efficient solution. The proof is summarized in the following problem definition and theorem.

Define the scalar maximization subproblem (R) as:

$$\text{Maximize } f_m(x)$$

such that:

$$f_i(x) \geq b_i, \; i=1, \ldots, k, \; i \neq m,$$

$$x \subset X$$

Theorem: If a point $x' \subset X$ is an optimal solution in (R) and is not dominated by any alternate optimal solution in (R), then x' is efficient.

Proof (by contradiction): Assume that the point x' is an optimal solution in (R) and is not dominated by any alternate optimal solutions in (R). Suppose x' is not efficient. Then there must exist $x'' \subset X$ such that x'' dominates x' and is thus feasible in (R). By assumption, x'' cannot be an alternate optimal solution in (R). Therefore,

$$f_m(x'') < f_m(x').$$

This contradicts x'' dominating x' and therefore x' must be efficient.

In the subproblems of the algorithm, b_i will take on values

$$b_j = f_j(x_0) + \varepsilon \quad (\varepsilon > 0),$$

$$b_i = f_i(x_0), \; i=1, \ldots, k, \; i \neq j,m,$$

where x_0 is an efficient point. For initialization of the algorithm, an efficient solution can be obtained by setting the b_i's to arbitrary, attainable values for k-1 objectives.

This proof provides sufficient conditions for efficiency. Therefore, we are guaranteed that only efficient solutions will be generated from the subproblems and presented to the decision maker. The proof, however, does not provide necessary conditions for generating efficient solutions. As a result, there may exist some efficient solutions which cannot be found using the subproblem (R). This is due to the constraint values b_i being objective levels from previously generated efficient points.

CHAPTER 5

MOLANDA—A MULTIOBJECTIVE LAND ALLOCATION SYSTEM

5.1. Introduction to System and Solution Techniques

This chapter describes a computer implementation for the algorithm described in Chapter 4 for solution of the model in Chapter 3. The computer program is the Multiobjective Land Allocation (MOLANDA) program and is used to generate efficient solutions for the land allocation process. Four subproblems are solved, one for each of the objectives: cost, amenity distance, detractor distance, and shape. Each integer programming subproblem is solved using an implicit enumeration technique to identify an optimal solution.

Given N feasible cells in the study area and an allocation requirement of k cells, the possible number of solutions is $M = \binom{N}{k}$. With this many potential solutions, it is essential that the enumeration technique be as efficient as possible in locating an optimal one. The technique enumerates solutions in an order which facilitates fathoming, i.e., provides a means of skipping many suboptimal solutions. There are two ordering schemes used for the various subproblems in the model. One is referred to as the natural lexicographic order and and the other is referred to as the bottleneck lexicographic order. These will be explained in the following sections.

5.2. Background on Implicit Enumeration

Enumeration methods are a common approach to solving integer programming problems where the set of values of the integer variables is finite. The set of values may become quite large so techniques are required to partially enumerate possible solutions and skip the remaining ones. When a solution is skipped during an enumeration scheme, it is said to have been implicitly enumerated. Garfinkel and Nemhauser (1972), discuss two such enumeration techniques.

Branch and bound uses a tree structure to represent the possible solutions. Each solution is represented by a unique path of edges and vertices which define values for the integer variables. Each edge represents a constraint on one of the variables and each vertex represents a constraint set of edges up to that vertex. The technique searches through the tree enumerating possible solutions. At each vertex, a bound on the problem's objective function is determined which is compared to previously calculated bounds. This is used to possibly fathom the vertex which eliminates from the enumeration any possible solutions using this vertex. After a vertex is fathomed, the searching process branches to another one and continues the enumeration.

The other technique discussed by Garfinkel and Nemhauser (1972) uses a lexicographic enumeration method for the case of a binary integer programming problem. This technique represents the possible solutions as vectors of binary values and enumerates them in a lexicographically increasing order. This ordering is based on the total integer value represented by the components of the binary vector. Therefore, each new

binary vector enumerated has a total integer value equal to the integer value of the last binary vector plus one. The problem's objective function is used to calculate bounds so that possible binary vectors can be skipped from the enumeration. A general form of the lexicographic enumeration method is used for the MOLANDA subproblems and will be discussed in the following section.

Both of these techniques use the objective function to accelerate the enumeration process by skipping large sets of possible solutions. By implicitly enumerating these solutions and only examining a portion of the entire solution set, large integer programming problems can be solved.

5.3. Enumeration Technique Selected

The MOLANDA subproblems use two lexicographic enumeration schemes to identify optimal solutions. A solution is represented by a vector of cell indices with the indices placed in increasing order. For example, if k = 4, (1,3,6,8) is a valid solution vector. In both enumeration schemes, the solutions are lexicographically ordered (described by Lawler (1976)) so that the first solution vector is (1,2,..., k) and the last one is (N-k+1, N-k+2, ..., N). However, the methods differ in the enumeration of the solutions between the first and last ones.

For the following description of the algorithms in the MOLANDA program, the following notation is required:

N = ||G|| where G is as defined in Chapter 3 and || is cardinality.

J = $(j_1, j_2, \ldots, j_m)$; this vector represents the first m cells in a solution under construction, $m \leq k$. The elements in J are distinct cells in increasing order. At any point in the algorithm, the vector J will have been chosen as a tentative solution and some cell i will be considered for the m-th position of J.

I = $(i_1, i_2, \ldots, i_k)$; this vector represents the cells in a completed solution.

M = $\binom{N}{k}$ possible solutions.

The technique enumerates a new solution by incrementing one vector component at a time. In the natural lexicographic method, the rightmost component is incremented for each new solution, if possible. When this component reaches N, the second rightmost component is incremented and the rightmost component is reset to one more than the new second rightmost component. That is, given a current solution

$$I^q = (i_1^q, i_2^q, \ldots, i_k^q),$$

$$\text{with } i_1^q < i_2^q < \ldots < i_k^q \text{ and } i_k^q < N,$$

the next solution would be

$$I^{q+1} = (i_1^q, i_2^q, \ldots, i_{k-1}^q, i_k^q + 1).$$

If $i_k^q = N$ and $i_{k-1}^q < N-1$, the next solution would be

$$I^{q+1} = (i_1^q, i_2^q, \ldots, i_{k-2}^q, i_{k-1}^q+1, i_{k-1}^q+2).$$

The process continues to increment the rightmost component until it reaches N. Each time the rightmost component reaches N, the component furthest to the right (position s) and not equal to N-(k-s) is incremented and each component to the right of it is reset to one more than its left neighbor. The overall method is a recursive procedure, described as follows:

Step 0: Set $I^q = (i_1, i_2, \ldots, i_k) = (1, 2, \ldots, k)$.

Step 1: Set $m = 1$.

Step 2: If $i_k < N$, then

set $i_k = i_k + 1$; $I^{q+1} = (i_1, i_2, \ldots, i_k)$;

$q = q + 1$. Repeat step 2.

Otherwise, go to step 3.

Step 3: If $i_{k-m} < N-m$, then

set $i_{k-m} = i_{k-m} + 1$;

set $i_\ell = i_{\ell-1} + 1$ for $\ell = k-m+1, \ldots, k$;

$I^{q+1} = (i_1, i_2, \ldots, i_k)$;

go to step 1.

Otherwise, set $m=m+1$ and repeat step 3.

This process is continued until the leftmost component (i_1) reaches N-k+1. Using the above example of N=5 and k=3, the M=10 possible solutions would be enumerated in the following order:

1.	(1,2,3)	6.	(1,4,5)
2.	(1,2,4)	7.	(2,3,4)
3.	(1,2,5)	8.	(2,3,5)
4.	(1,3,4)	9.	(2,4,5)
5.	(1,3,5)	10.	(3,4,5)

In the bottleneck lexicographic method, the rightmost component of a solution vector is held at its initial value (called the bottleneck value b) while the remaining components are incremented using a modified natural lexicographic order for vectors of k-1 cells with N = b. The modified natural lexicographic order increments the leftmost component which is less than its right neighbor minus one and resets all of its left neighbors to their corresponding index position values. The bottleneck value is not incremented until the leftmost component (i_1) reaches b - k + 1.

That is, suppose there exists a current solution

$$I^q = (i_1^q, i_2^q, \ldots, i_{k-1}^q, i_k^q).$$

$$\text{with } i_1^q < i_2^q < \ldots < i_k^q, \text{ and } b = i_k^q.$$

The next solution would be formed by incrementing the leftmost component i_ℓ^q such that $i_\ell^q < i_{\ell+1}^q - 1$, and by resetting $i_1, \ldots, i_{\ell-1}$ to $1, \ldots, \ell-1$. Therefore, the next solution would be

$$I^{q+1} = (1, 2, \ldots, \ell-1, i_\ell^q+1, i_{\ell+1}^q, \ldots, i_k^q).$$

When i_1 reaches b - k + 1, the next solution would be

$$(1, 2, \ldots, k-1, b+1)$$

and the process is repeated until the bottleneck value has reached N + 1. Using the above example, the M possible solutions would be enumerated in the following order:

1.	(1,2,3)	6.	(1,3,5)
2.	(1,2,4)	7.	(2,3,5)
3.	(1,3,4)	8.	(1,4,5)
4.	(2,3,4)	9.	(2,4,5)
5.	(1,2,5)	10.	(3,4,5)

5.4. Cost Subproblem

The cost subproblem is formulated as follows:

Given a, d, and $\bar{s}$ as the current amenity distance, detractor distance, and shape value, find a set of k contiguous cells so as to

$$\text{Minimize } C(I) = \sum_{i \in I} c_i$$

$$I \subset G(d,a)$$

$$S(I) \leq \bar{s}$$

where c_i = cost of cell i,

$G(d,a) = \{g \in X \mid d_g \geq d, a_g \leq a\}$; d_g is the distance from cell g to nearest detractor cell; a_g is the distance from cell g to nearest amenity cell,

S(I) = compactness value for solution I, calculated by the perimeter times the diameter, as described in Chapter 3,

X = set of feasible cells in study area.

The cost subproblem is solved using the natural lexicographic method to enumerate feasible solutions. The feasible cells are indexed from 1 to N in order of increasing cost, so that cell i is the i^{th} most costly cell. Ties are handled arbitrarily.

The cost subproblem is to find one or more sets of k feasible cells, each of which are contiguous, meet the compactness requirement, and minimize the total cost. Since the cells are ordered by cost, large blocks of solutions can be implicitly enumerated using the natural lexicographic method. Suppose the current solution is

$$I^q = (i_1^q, i_2^q, \ldots, i_k^q)$$

and has a total cost of $C(I^q)$. If this solution meets the contiguity and compactness requirements, set $\bar{c} = C(I^q)$. If $i_k^q < N$, the next solution in the natural lexicographic order is:

$$I' = (i_1^q, i_2^q, \ldots, i_{k-1}^q, i_k^q+1).$$

However, if

$$c(i_k^q+1) > c^q(i_k) \text{ and } C(I') > C(I^q), \text{ then}$$

solution I' can be skipped in the enumeration. Also, all remaining solutions which differ from I^q only in the last (k^{th}) component can be skipped. Thus the next solution that should be enumerated after I^q (if $i_{k-1}^q + 1$, $i_{k-1}^q + 2$ exist) is:

$$I^{q+1} = (i_1^q, i_2^q, \ldots, i_{k-2}^q, i_{k-1}^q+1, i_{k-1}^q+2).$$

By continuing this process, only solutions which can possibly produce a lower or equal total cost are enumerated. After a feasible solution is found, the process is continued until all possible alternate solutions with the same cost are enumerated. These alternate optima are checked for dominance against each other. All efficient solutions found are presented to the decision maker.

For example, consider six cells with the costs and spatial distribution illustrated in Figure 3.

Cell i:	1	2	3	4	5	6
Cost:	3	5	6	6	10	11

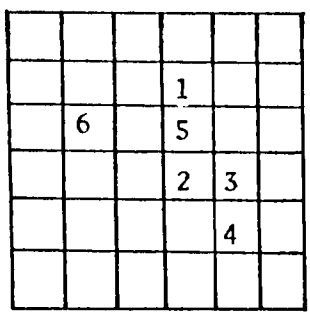


Figure 3. Cells for cost subproblem example.

If k = 3, the order of enumerating solutions for this problem is given in Table 1. (Initially, set $\bar{c} = \infty$.)

Table 1. Enumeration of complete solutions for the cost subproblem.

Solution	Reason for skipping or considering
(1,2,3)	Not contiguous (NC)
(1,2,4)	NC
(1,2,5)	Feasible, $\bar{c}$ = 18
(1,3,4)	NC; Notice that (1,2,6) was skipped due to the upper bound $\bar{c}$.
(1,3,5)	NC
(1,3,6)	NC
(1,4,5)	NC
(1,4,6)	NC
(1,5,6)	NC
(2,3,4)	Feasible, $\bar{c}$ = 17
(2,4,5)	NC; Notice that (2,3,5) and (2,3,6) were skipped due to c.
(2,4,6)	NC
(2,5,6)	NC
(3,4,5)	NC
(3,4,6)	NC
(3,5,6)	NC
(4,5,6)	NC; Stop process.

The process actually does not create an entire solution before checking the cost and contiguity conditions. Each cell i is checked individually before it is added to a solution being built. That is given a partial solution.

$$J^n = (j_1^n, j_2^n, \ldots, j_m^n), \ (m < k),$$

the next cell added would create

$$J^{n+1} = (j_1^n, j_2^n, \ldots, j_m^n, j_{m+1}^n).$$

As the cells are added, the total cost is compared with the current $\bar{c}$ and the contiguity requirement is checked. Using this process, the above example actually uses the enumeration order given in Table 2. ($k = 3$, $N = 6$.)

The cost subproblem algorithm is presented:

1. Initialization: Order cells in G(d,a) increasingly by cost. Set J=(1), $\bar{c}=\infty$, i=1, m=2, $C(J)=c_1$. Go to 3.

2. Backtrack on position index: Set m=m-1, $i=j_m$, $C(J)=\sum_{\ell=1}^{m-1} c_\ell$. If m = 0, go to 9.

3. Increment cell index: Set i=i+1. If i > N, go to 2.

4. Check cost: If $C(J) + c_i > \bar{c}$, go to 2.

5. Check contiguity: If cell i cannot be contained with $(j_1, j_2, \ldots, j_{m-1})$ in a contiguous solution, go to 3.

6. Add cell and increment position index: Set $j_m = i$, $C(J)= \sum_{\ell=1}^{m} c_\ell$, m=m+1. If $m \leq k$, go to 3.

Table 2. Enumeration of partial solutions for the cost subproblem.

Solution	Reason for skipping or considering
(1)	Contiguous, continue. (CC)
(1,2)	CC
(1,2,3)	NC
(1,2,4)	NC
(1,2,5)	Feasible, $\bar{c}$ = 18, skip.
(1,3)	NC
(1,4)	NC
(1,5)	CC
(1,5,6)	NC
(2)	CC
(2,3)	CC
(2,3,4)	Feasible, $\bar{c}$ = 17, skip.
(2,4)	CC
(2,4,5)	NC
(2,4,6)	NC
(2,5)	CC
(2,5,6)	NC
(3)	CC
(3,4)	CC
(3,4,5)	NC
(3,4,6)	NC
(3,5)	CC
(3,5,6)	NC
(4)	CC
(4,5)	NC
(4,5,6)	NC; Stop. Optimal solution is (2,3,4) with c = 17.

7. Check final solution for compactness: k cells have been chosen. If $S(J) > \bar{s}$, set m=m-1, and go to 3.

8. Store solution: A feasible solution has been found. If $C(J) < \bar{c}$, store the solution as the best found so far and ignore any previously found optimal solutions. If $C(J) = \bar{c}$, this is an alternate optimal solution. Compare it to other optimal solutions to check for efficiency. If it dominates any existing optimal solution, replace the dominated one with this one. If it is dominated by any existing solution, ignore this solution. If it is nondominated, add it as another alternate optimal solution. Set m=m-1 and go to 3.

9. Stop: If no feasible solutions have been found, there are none using the current constraints. Otherwise, one or more efficient solutions have been found.

5.5. A Lower Bound in the Cost Subproblem

An additional feature was incorporated into the algorithm to help eliminate suboptimal solutions from the enumeration process. The feature involves computing a lower bound on the cost of a solution being constructed and comparing it with an upper bound on the cost of a feasible solution already found. Any feasible solution identified in the enumeration scheme provides such an upper bound. Given a partial solution

$$J = (j_1, j_2, \ldots, j_m),\ m < k,$$

a lower bound on the cost of any subsequent solution containing the cells in J is given by

$$\underline{c} = \sum_{\ell=1}^{m} c_{j_\ell} + \sum_{\ell=1}^{k-m} c_{(j_m+\ell)}.$$

If $\underline{c}$ is greater than the upper bound $\bar{c}$, then it is not necessary to consider any subsequent solutions containing the cells of J. In such a case, the m-1st index of J can be incremented (steps 2 and 3 of cost algorithm). The next partial solution would be

$$J = (j_1, j_2, \ldots, j_{m-2}, j_{m-1}+1).$$

For example, consider the following cells and their associated costs:

Cell i:	1	2	3	4	5	6
Cost:	1	3	5	8	10	11

Suppose the following feasible solution was found in the enumeration scheme (k = 4):

$$I^1 = (1,2,4,6)$$

which had a total cost of 23. The next partial solution to be considered would be:

$$J^1 = (1,2,5)$$

with cell 5 being considered for addition to the current solution being built. By calculating $\underline{c}$:

$$\underline{c} = c_1 + c_2 + c_5 + c_6 = 25,$$

we see that any solution constructed using J^1 will be dominated due to cost. Therefore, this cell can be skipped and the next partial solution to be considered would be:

$$J^2 = (1,3)$$

with $$\underline{c} = c_1 + c_3 + c_4 + c_5 = 24.$$

Since $\underline{c} > C(I^1)$, we can skip cell 3 and the next partial solution to be considered is:

$$J^3 = (2)$$

with $$\underline{c} = c_2 + c_3 + c_4 + c_5 = 26.$$

Thus, any solution using J^3 or any subsequent solution would be dominated. Therefore, the enumeration is completed and I^1 is the optimal solution.

5.6. Proximity Subproblems

The amenity distance subproblem is formulated as follows:

Given d, $\bar{c}$, and $\bar{s}$ as the current detractor distance, cost, and shape values, find a set of k contiguous cells so as to:

$$\text{Minimize } \{\text{maximum}_{i \in I} a_i\}$$

such that:

$$I \subset G(d)$$

$$S(I) \leq \bar{s}$$

$$C(I) \leq \bar{c}$$

where: a_i = distance from cell i to nearest amenity cell,

$$G(d) = \{g \in X \mid d_g \geq d\}.$$

The detractor distance subproblem is formulated as follows: Given a, $\bar{c}$, and $\bar{s}$ as the current amenity distance, cost, and shape values, find a set of k contiguous cells so as to

$$\text{Maximize } \{\underset{i \in I}{\text{minimum}}\ d_i\}$$

such that:

$$I \subset G(a)$$

$$S(I) \leq \bar{s}$$

$$C(I) \leq \bar{c}$$

where: d_i = distance from cell i to nearest detractor cell,

$G(d) = \{g \in X \mid a_g \leq a\}$.

The proximity subproblems, amenity and detractor distance, use the bottleneck lexicographic enumeration method to find the optimal solutions. Since these problems will be to either minimize or maximize the amenity or detractor distances, respectively, the N cells are ordered by distance to the particular land use areas. The cells are ordered increasingly from the amenity land use areas for the amenity problem and decreasingly from the detractor land use areas for the detractor problem. The proximity subproblems are to find one or more contiguous sets of feasible cells, each of which meet the current compactness and cost requirements.

These subproblems set either the minimum or maximum distance as the bottleneck value in the enumeration process. The method starts with $i_k = k$ and holds this while all possible solutions are enumerated using

k as the minimum or maximum distance, depending on the problem. It is then incremented to k+1 and the process is repeated until a feasible solution is found or $i_k > N$.

This method is able to stop after one or more feasible solutions are found with the same bottleneck value. This is due to the solutions being enumerated in order of minimum or maximum bottleneck distance so that any feasible ones found will be the best using the current bottleneck value. Alternate optimal solutions are identified by continuing the enumeration after the first optimal solution is found until the bottleneck distance changes. If several alternate optimal solutions are found, they are checked for dominance and all efficient solutions are returned to the decision maker.

For example, consider the following feasible cells for the amenity distance subproblem and their associated costs, as illustrated in Figure 4.

Cell:	1	2	3	4	5	6	7	8
Cost:	10	10	10	8	5	8	10	3

		1	A	A	A
		5	2	A	A
			6	3	4
				7	8

Figure 4. Cells for proximity subproblem example.

If $k = 3$ and $\bar{c} = 26$, the order of enumerating the 56 possible solutions is given in Table 3. (Assume all detractor distances are equal.)

Even though this method is able to stop after one or more optimal solutions are found with the same bottleneck value, it does not allow blocks of solutions to be skipped as in the natural lexicographic enumeration method. Since the bottleneck distance controls the order in which solutions are enumerated, the cost of each solution is not nondecreasing and thus cannot be used to eliminate dominated solutions. Also, cells included in partial solutions are rearranged in subsequent solutions and so contiguity cannot be used to eliminate solutions from the process.

The proximity subproblem algorithm is presented:

1. Initialization: For the amenity subproblem, order the cells in G(d) increasingly in distance from the nearest amenity land uses. For the detractor subproblem, order the cells in G(a) decreasingly in distance from the nearest detractor land uses. Set $J=\emptyset$, $i=k$, $m=k$, $C(J)=0$. If this is the amenity subproblem, set $b = \infty$. If this is the detractor subproblem, set $b = 0$. Go to 4.

2. Backtrack on index position: Set $m=m+1$, $i=j_m$.

3. Increment cell index: Set $i=i+1$. If this is the amenity subproblem, $m=k$, and distance from cell i to nearest amenity cell is $> b$, go to 9. If this is the detractor subproblem, $m=k$, and distance from cell i to nearest detractor cell $< b$, go to 9. If $i > N$, go to 8. If $i = j_{m+1}$, go to 2.

Table 3. Enumeration of complete solutions for the proximity subproblem.

Solution	Reason for skipping or continuing	Cost
(1,2,3)	NC, and over $\bar{c}$ (OC)	$30
(1,2,4)	NC, OC	28
(1,3,4)	NC, OC	28
(2,3,4)	NC, OC	28
(1,2,5)	Contiguous, feasible.	25
(1,3,5)	NC	25
(2,3,5)	NC	25
(1,4,5)	NC	23
(2,4,5)	NC	23
(3,4,5)	NC	23
(1,2,6)	NC, OC	28
(1,3,6)	NC, OC	28
(2,3,6)	Contiguous, but dominated in cost by (1,2,5).	28
(1,4,6)	NC, OC	26
(2,4,6)	NC, OC	26
(3,4,6)	Contiguous, but dominated in cost by (1,2,5).	26
(1,5,6)	NC	23
(2,5,6)	Contiguous and dominates (1,2,5) in cost.	23
(3,5,6)	NC	23
(4,5,6)	NC	21
(1,2,7)	Stop since bottleneck distance is incremented. Optimal solution is (2,5,6).	

4. Check cost: If $C(J) + c_i \geq \bar{c}$, go to 3.

5. Check contiguity: If cell i cannot be contained with J in a contiguous solution, go to 3.

6. Add cell and increment position index: $j_m = i$, $C(J)=C(J) + c_i$, $m=m-1$, $i=m-1$. If $m > 0$, go to 3.

7. Check solution for compactness requirement: k cells have been chosen. If $S(I) > s$, set $i=j_1$, $m=1$, $C(J)=C(J)-c_i$, and go to 3.

8. If all possible solutions have been enumerated, stop with no feasible optimal solution. If a feasible solution has been found, compare it with any previously found optimal solutions for dominance. Set b = current bottleneck value and return to 3.

9. Stop process with current efficient solution(s) found.

5.7. Shape Subproblem

The shape subproblem is formulated as follows:

Given a, d, and $\bar{c}$ as the current amenity distance, detractor distance, and cost values, find a set of k contiguous cells so as to

$$\text{Minimize } S(I)$$

$$\text{such that: } I \subset G(d,a)$$

$$C(I) \leq \bar{c}$$

Since the shape objective value cannot be measured until after a solution of cells has been constructed, the shape subproblem cannot directly use either of the lexicographic enumeration techniques. There is no cell-by-cell approach that can be applied to allocate a

set of cells with the best shape value. Therefore, the cost subproblem is used with a controlled shape constraint for the shape subproblem. The shape subproblem is to find one or more contiguous sets of k feasible cells which meet the cost requirement, proximity requirements, and minimize the compactness (shape) value. This problem is solved by iteratively increasing the shape constraint in the cost subproblem until one or more feasible solutions are found. The first one(s) found will be the optimal solution(s) for the shape subproblem. As in the previous subproblems, alternate optimal solutions in the cost subproblem are checked for efficiency and all efficient solutions are returned to the decision maker.

For each iteration of the cost subproblem, the shape value constraint is incremented by 1.0. The process is repeated until any feasible solutions are found or the shape value constraint becomes greater than the worst possible shape value that could be achieved for k cells. If no solution has been found at this point, then there exists no solution which strictly improves one of the other objectives and allows the shape objective to degrade.

The shape subproblem algorithm is presented:

1. Initialization: Let S(J) = shape value of current solution. Let $\bar{s}_k$ be the worst shape value (upper bound) that can be achieved for k cells allocated.

2. Increment: Set $\hat{s} = S(J) + 1.0$. If $\hat{s} > \bar{s}_k$, go to 5.

3. Use the cost subproblem to find any feasible solutions such that:

$$C(I) \leq \bar{c}$$

$$I \subset G(d,a)$$

$$S(I) \leq \hat{s}$$

4. If any feasible solutions have been found with the same shape value, check each one for efficiency. Stop and return all efficient solutions to the decision maker. Otherwise, go to 2.

5. Stop with no feasible solution found.

5.8. Alternative Enumeration Methods

During the research of developing the enumeration schemes in the MOLANDA System, two additional methods were considered, but not implemented. These methods do not use any kind of lexicographic structure like the two already discussed. They will be referred to as the total rank sum and total cost sum methods. Both methods would only be appropriate for the cost subproblem since they rely on the cost objective for eliminating solutions from enumeration.

The total rank sum method involves enumerating the solutions in increasing order of the total rank sum of the cell indices. For example, if k = 3, the first solution to be enumerated would be (1,2,3) whose rank is 6. Next, all solutions with rank sums of 7, 8, ... would be enumerated until all solutions have either been checked or shown to be dominated.

This method can be compared with the lexicographic methods by considering the likelihood of obtaining an optimal solution at the early stages of the enumeration process. The relative performances of the three methods in this respect can be predicted under certain assumptions concerning the cost structure of the feasible cells. Figure 5 illustrates the three possible cost structures.

If the cost of each cell is a linear function of its rank in the initial ordering, then the total rank sum method could be expected to perform best. In this case, minimizing the sum of the ranks is equivalent to minimizing the total cost. Therefore, the first feasible solution will be the optimal one.

If the cost is a concave function of the rank, then the bottleneck lexicographic method could be expected to perform the best. The cost of the cells is increasing at an increasing rate as the rank increases. Therefore, only one or two high ranked cells in a solution may make its cost very large. This method would enumerate the solutions in order of maximum cell rank and thus more nearly approximate an enumeration in order of increasing cost.

If the cost is a convex function of the rank, then the natural lexicographic method could be expected to perform the best. The cost increases at a decreasing rate as the rank increases. If a cell has to be changed in a solution, the smallest cost increase would result from increasing the index of the most costly cell as is done in this method. Therefore, for a convex cost function, the natural lexicographic method will more nearly approximate an enumeration in order of increasing cost.

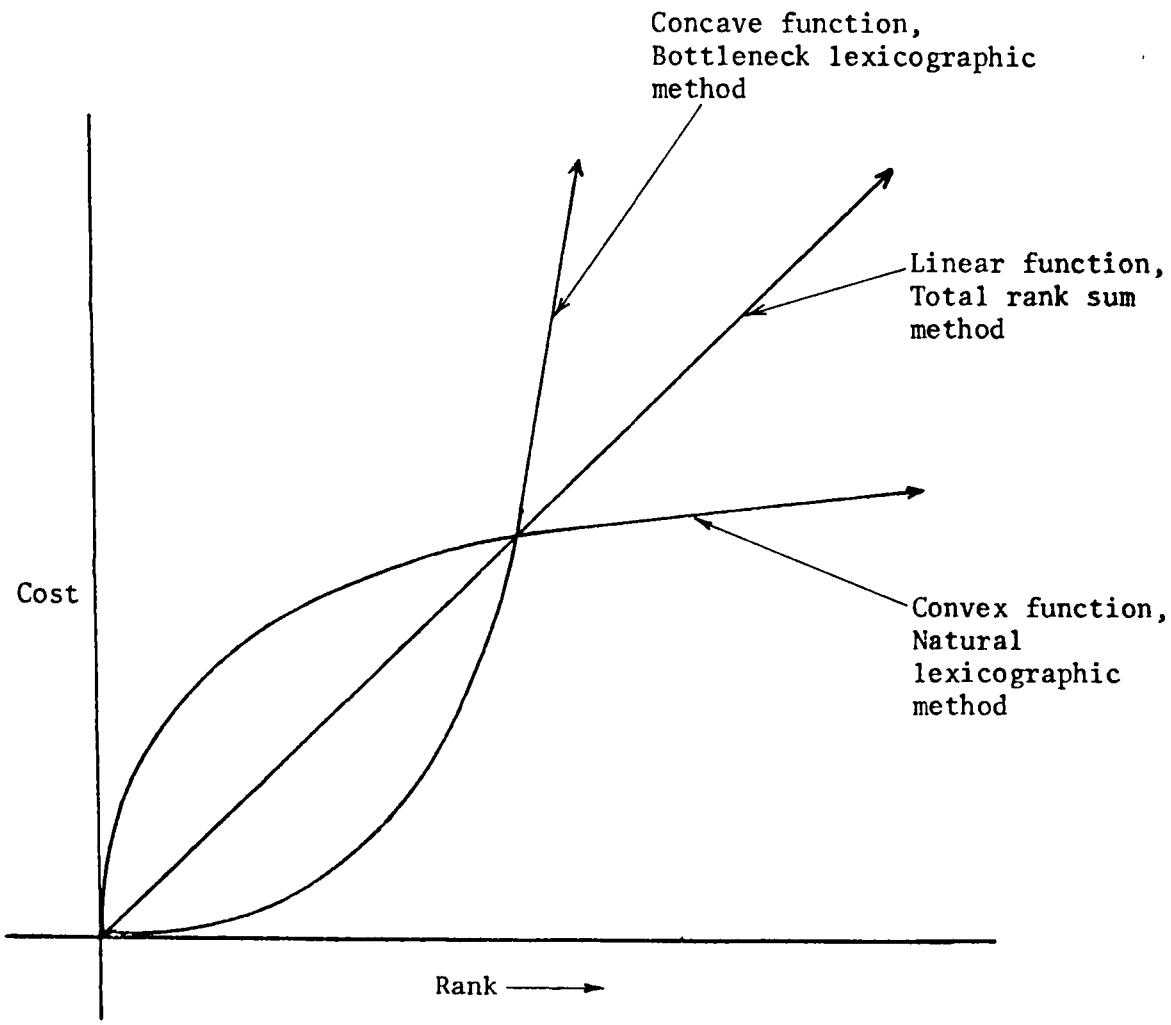


Figure 5. Chart showing cost-rank relationship for three enumeration methods.

The total cost sum method enumerates the solutions in order of nondecreasing total cost. Like the bottleneck lexicographic method, this method could stop after finding the first feasible solution. Since the solutions are enumerated in order of nondecreasing total cost and the cost subproblem minimizes the total cost, the first feasible solution enumerated would be optimal. One disadvantage to this method is the difficulty of generating the solutions in order of total cost as compared to the other methods. Presented is a conceptual approach for this method.

This approach uses the individual cell indices to enumerate subsequent solutions. To describe the technique mathematically, let I^m be the m^{th} lowest cost solution (break ties arbitrarily). Let L(m) =

$$\{I^1, I^2, \ldots, I^m\}.$$

Define a <u>successor</u> of a solution $(i_1, i_2, \ldots, i_k)$ as any one of the k solutions of the form

$$(i_1, i_2, \ldots, i_{\ell-1}, i_{\ell+1}, i_{\ell+2}, \ldots, i_\ell + k - \ell + 1),\ 1 \leq \ell \leq k.$$

Let S(I) denote the set of successors of solution I. For example, S(1,2,3) = {(2,3,4), (1,3,4), (1,2,4)}. Let C(m) be the set of all solutions which are successors of one or more solutions in L(m) but which are not themselves in L(m).

Set $C(m) = \bigcup_{i=1}^{m} S(I^i) - L(i)$. Then I^{m+1} is contained in C(m) and can be found by evaluating the cost of all the solutions in C(m). This suggests the following algorithm:

1. Let m = 1, I^m = (1,2, ..., k), and C(1) and L(1) as defined above. Go to step 2.

2. If I^m is feasible, it is optimal. Otherwise, go to step 3.

3. Choose the lowest cost solution in C(m) and denote it as $\hat{I}$. Set $I^{m+1} = \hat{I}$ and

$$C(m) = C(m) \cup S(\hat{I}) - \hat{I} - L(m).$$

Set m = m + 1, $L(m) = I^m$, and go to step 2.

The following is an example of the algorithm using k = 3 and the following cells:

Cell i:	1	2	3	4	5	6
Cost:	1	3	5	6	10	11

The first solution generated would be

$$I^1 = (1,2,3),$$

with cost = 9 and C(1) = {(1,2,4), (1,3,4), (2,3,4)}. If I^1 was not feasible, I^2 would be (1,2,4) with cost = 10 and C(2) = {(1,3,4), (2,3,4), (1,2,5)}. At this point, I^4 could either be (1,2,5) or (2,3,4) since they have equal cost. This process is repeated until a feasible solution is found which will be the optimal solution. The implementation of this algorithm is left for future research.

CHAPTER 6

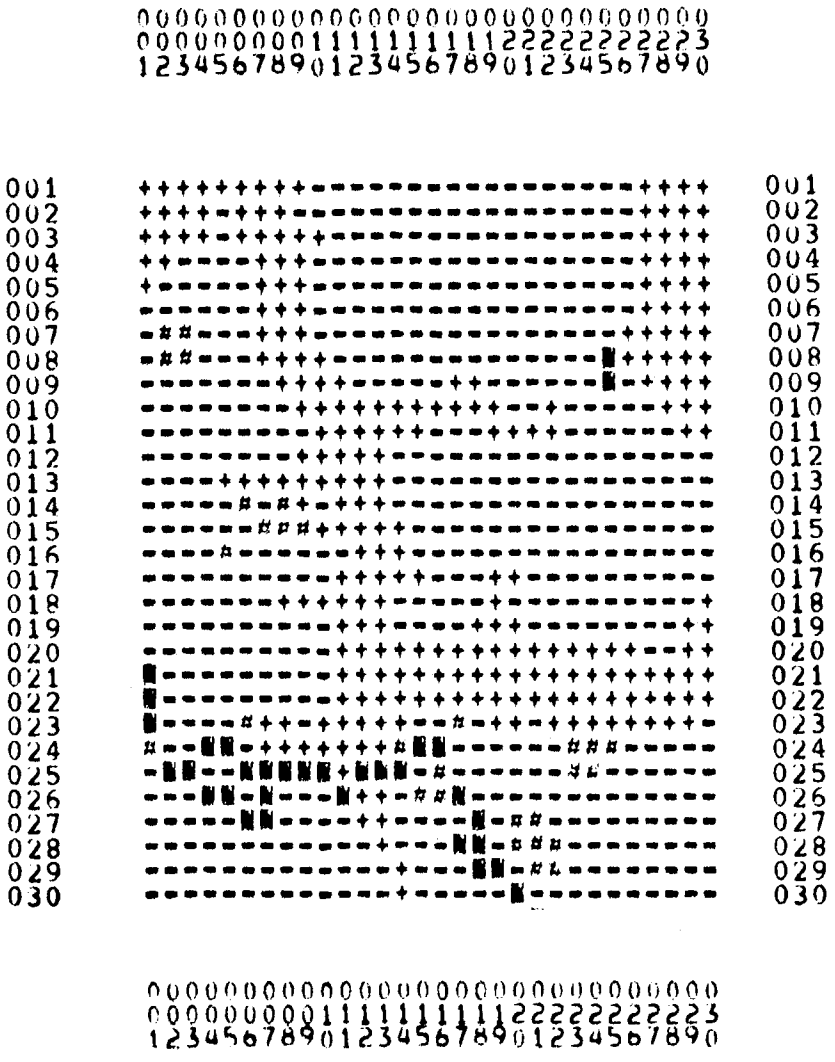
EXAMPLE OF USING THE MOLANDA SYSTEM

6.1. MOLANDA Test Description

To demonstrate the MOLANDA program, a test study area surrounding the town of Norris, Tennessee, was selected and the appropriate data generated for locating a new residential development. The grid cell GIS, IMGRID, Holmes and Jolly (1980), was used to store, prepare, and display the required cost and definition layers. A map of the definition layer showing the study area is presented in Figure 6. The dashes (-) represent the feasible cells, pluses (+) represent Norris Reservoir which is the amenity land use, pounds (#) represent industries or commercial areas which are the detractor land uses, and the solid symbols (■) represent the roads and public lands which are reject areas.

A map of the cost layer is presented in Figure 7. The symbols represent cost categories for developing a residential area on each cell. Table 4 presents the distribution of costs for the map symbols. The model uses the actual costs, but categories are used for the purposes of display.

As indicated by the maps, the study area is represented by 900 cells configured using 30 rows and 30 columns. Each cell represents approximately 2.5 acres so the entire study area covers 2,250 acres. The required parameters are represented in the definition layer using the following:



LEGEND

1. Feasible Cells
2. Roads and Public Lands
3. Industry or Commercial
4. Norris Reservoir

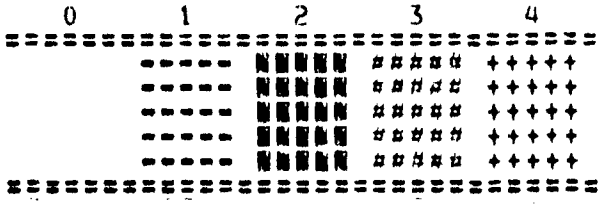


Figure 6. Map of definition layer for the MOLANDA test example.



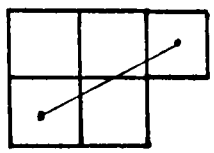
Figure 7. Map of cost layer for the MOLANDA test example.

Table 4. Distribution of costs for the cost map in Figure 7.

Map value	Inverval
0	0 - 7.1
1	7.2-14.2
2	14.3-21.3
3	21.4-28.4
4	28.5-35.5
5	35.6-42.6
6	42.7-49.7
7	49.8-56.8
8	56.9-63.9
9	64.0-71.0

Value	Definition
1	Feasible cells
2	Reject cells
3	Detractor land use cells
4	Amenity land use cells

For this example, a thirteen acre contiguous tract is to be located for development of a residential area. Therefore, five cells will be allocated by the model. The optimal shape configuration that can be achieved using five cells has four cells in a square and the fifth one adjacent to any of the four. This is illustrated in Figure 8.



Number of outside edges: 10
Maximum distance between two cells: $\sqrt{5}$
Shape value: 22.36

Figure 8. Optimal shape configuration using five cells.

In the program run, all calculated shape values will be normalized by dividing them by 22.36 so that the user will recognize the best possible shape value as 1.0.

The program is conversational in nature so that the user is prompted for all required information. Figure 9 is a listing of the

```
*
LAND USE ALLOCATION PROGRAM
Enter study area NAME (8 CHARACTERS):
NORRIS
ENTER NUMBER OF ROWS:
30
ENTER NUMBER OF COLUMNS:
30
ENTER LAND USE AMENITY CODE:
4
ENTER DETRACTOR LAND USE CODE:
3
enter reject land use code:
2
CURRENT INPUT VALUES
study area name:  NORRIS
Number of rows and columns:    30    30
Amenity land use code:  4
Detractor land use code:  3
reject land use code:  2
Do you want to change these?
Enter y or n
N
PROBLEM SELECTION:

ENTER SIZE OF SHAPE TO BE ALLOCATED
(NUMBER OF CELLS):
5

INITIALIZE OBJECTIVE LEVELS FOR PROBLEM:
ENTER INITIAL AMENITY DISTANCE LEVEL:
3
ENTER INITIAL DETRACTOR DISTANCE LEVEL:
15
ENTER INITIAL SHAPE LEVEL:
3

WOULD YOU LIKE TO STORE THE INITIAL SOLUTION?
ENTER Y OR N:
Y

TABLE SOLUTION    1 STORED AS NUMBER    1
```

Figure 9. Listing of MOLANDA example run.

```
YOUR CURRENT SOLUTION IS:

                  **************************
                  *                        *
(1) COST          *     12.00              *
(2) AMEN DIST     *      3.00              *
(3) DET DIST      *     17.49              *
(4) SHAPE         *      2.14              *
                  *                        *
                  **************************

SELECT OBJECTIVE TO IMPROVE (1,2,3,4):
2

USING THE CURRENT SOLUTION AND OBJECTIVE  2
THE TRADEOFF TABLE IS:

                                OBJECTIVES TO BECOME WORSE

                                    COST        DET DIST    SHAPE
*************************************************************************
OBJ TO
BE IMPROVED:      AMN DIST        -1.000        -1.000      *****
*************************************************************************
                                   70.000       -11.492      *****

*************************************************************************

                                     ( 1)         ( 2)         ( 3)


                                    COST        DET DIST    SHAPE
*************************************************************************
OBJ TO
BE IMPROVED:      AMN DIST       NONE          -1.000       NONE
*************************************************************************
                                   NONE        -11.492       NONE

*************************************************************************

                                     ( 4)         ( 5)         ( 6)
```

Figure 9 (Continued)

```
                                    COST        DET DIST    SHAPE
*****************************************************************
OBJ TO
BE IMPROVED:      AMN DIST      NONE       -1.000        NONE
*****************************************************************
                                    NONE      -11.492        NONE


*****************************************************************

                                     ( 7)       ( 8)       ( 9)


YOU HAVE THE FOLLOWING OPTIONS:
     1-USE ONE OF THE SOLUTIONS INDICATED BY THE
       TRADEOFF TABLE,
     2-RE-INITIALIZE THE PROBLEM AND RESTART,
     3-STORE ONE OF THE SOLUTIONS IN THE TABLE,
     4-RETRIEVE A SOLUTION PREVIOUSLY STORED,
     5-VIEW ONE OF THE TABLE SOLUTIONS,
     6-VIEW A STORED SOLUTION,
     7-VIEW THE CELLS OF A TABLE SOLUTION,
     8-VIEW THE CELLS OF A STORED SOLUTION,
     9-STOP PROCESS.
    10-PRINT CURRENT SOLUTION FILE.
ENTER AN OPTION NUMBER:
3
ENTER TABLE SOLUTION NUMBER TO BE STORED:
1

TABLE SOLUTION    1 STORED AS NUMBER    2
ENTER AN OPTION NUMBER:
3
ENTER TABLE SOLUTION NUMBER TO BE STORED:
2

TABLE SOLUTION    2 STORED AS NUMBER    3
ENTER AN OPTION NUMBER:
3
ENTER TABLE SOLUTION NUMBER TO BE STORED:
5

TABLE SOLUTION    5 STORED AS NUMBER    4
ENTER AN OPTION NUMBER:
3
ENTER TABLE SOLUTION NUMBER TO BE STORED:
8

TABLE SOLUTION    8 STORED AS NUMBER    5
```

Figure 9 (Continued)

```
ENTER AN OPTION NUMBER:
1
ENTER TABLE SOLUTION NUMBER TO BE USED:
1


YOUR CURRENT SOLUTION IS:

                  ************************
                  *                      *
(1) COST          *     82.00            *
(2) AMEN DIST     *      2.00            *
(3) DET DIST      *     20.00            *
(4) SHAPE         *      1.20            *
                  *                      *
                  ************************


SELECT OBJECTIVE TO IMPROVE (1,2,3,4):
1



USING THE CURRENT SOLUTION AND OBJECTIVE  1
THE TRADEOFF TABLE IS:

                              OBJECTIVES TO BECOME WORSE

                                    AMN DIST    DET DIST    SHAPE
*****************************************************************
OBJ TO
BE IMPROVED:      COST          -15.999       -5.000      *****
*****************************************************************
                                   1.000       -7.000      *****

*****************************************************************

                                    ( 1)        ( 2)        ( 3)
```

Figure 9 (Continued)

```
                                    AMN DIST    DET DIST    SHAPE
*****************************************************************
OBJ TO
BE IMPROVED:      COST           -24.000       NONE        NONE
*****************************************************************
                                   1.000        NONE        NONE


*****************************************************************

                                     ( 4)        ( 5)        ( 6)

YOU HAVE THE FOLLOWING OPTIONS:
    1-USE ONE OF THE SOLUTIONS INDICATED BY THE
      TRADEOFF TABLE,
    2-RE-INITIALIZE THE PROBLEM AND RESTART,
    3-STORE ONE OF THE SOLUTIONS IN THE TABLE,
    4-RETRIEVE A SOLUTION PREVIOUSLY STORED,
    5-VIEW ONE OF THE TABLE SOLUTIONS,
    6-VIEW A STORED SOLUTION,
    7-VIEW THE CELLS OF A TABLE SOLUTION,
    8-VIEW THE CELLS OF A STORED SOLUTION,
    9-STOP PROCESS.
   10-PRINT CURRENT SOLUTION FILE.
ENTER AN OPTION NUMBER:
3
ENTER TABLE SOLUTION NUMBER TO BE STORED:
1

TABLE SOLUTION    1 STORED AS NUMBER    6
ENTER AN OPTION NUMBER:
3
ENTER TABLE SOLUTION NUMBER TO BE STORED:
2

TABLE SOLUTION    2 STORED AS NUMBER    7
ENTER AN OPTION NUMBER:
3
ENTER TABLE SOLUTION NUMBER TO BE STORED:
4

TABLE SOLUTION    4 STORED AS NUMBER    8
ENTER AN OPTION NUMBER:
10
SOLUTION FILE WRITTEN TO PRTFILE.
```

Figure 9 (Continued)

example run. The first section involves the user entering information about the study area and the requested allocation. After the initial objective levels are entered, the cost subproblem is solved to find the first efficient solution. If no feasible solutions exist, the user is requested to respecify the initial objective levels. Once a solution is found, the user may store it for future reference. Storing solutions refers to writing the solution cell positions and objective levels out to an external file for future use. After the initial efficient solution is stored, it is considered the current solution and presented. The user is requested to select one of the objectives to be improved. The corresponding three subproblems are solved and the tradeoff table is presented. The table is repeated for each possible set of new efficient solutions. Each one may contain up to three new solutions. If a solution is not present, it is indicated by NONE printed in the corresponding space. Within each table, the first row represents the values of the selected objective in each of the new efficient solutions. The second row represents the values of the corresponding objectives allowed to degrade in the subproblems. A star (*) value indicates that no feasible solution could be found regardless of how much the unconstrained objective was allowed to degrade. In Figure 9, the first tradeoff table indicates that four new efficient solutions were found with the following tradeoffs:

Solution 1—Compared to the current solution, this solution is one cell closer to the nearest amenity land use cell and costs $70 more.

Solutions 2, 5, and 8—Compared to the current solution, these solutions are one cell closer to the nearest amenity land use cell and 11.492 cell units closer to the nearest detractor land use cell.

Solution 3—No feasible solution could be found which allowed the shape objective to degrade, improved the amenity objective, and held or improved the cost and detractor objectives.

After the tradeoff table, a list of options is presented to the user. He may view the other objective levels or cell locations of the new efficient solutions or any previously found solution, or he may store any of the new solutions for future use, or he may use any of the new solutions or any previously generated solutions as the next current solution. He may also print all of the stored solutions or stop the process. The example run stores the tradeoff table solutions and uses the first solution as the next current solution. In this iteration, cost is selected to be improved. The resulting tradeoff table indicates that only subproblems which allowed the amenity distance and detractor distance objectives to degrade found new efficient solutions. These three new solutions are stored and the entire solution file is printed. Figure 10 is the resulting file of efficient solutions. Figure 11 displays the eight solutions using the definition layer map.

6.2. Computational Considerations

The time required to solve the subproblems in the MOLANDA program is dependent primarily on the number of feasible cells and the spatial distribution of those cells in the study area. If the feasible cells

```
        MULTI-OBJECTIVE LAND ALLOCATION MODEL
          CURRENT SOLUTION FILE

            *** SOLUTION NUMBER    1 ***

                    COST:    12.000
          AMENITY DISTANCE:     3.000
        DETRACTOR DISTANCE:    17.492
                   SHAPE:     2.146

          SOLUTION CELLS:

             114              144              174              84    54

            *** SOLUTION NUMBER    2 ***

                    COST:    82.000
          AMENITY DISTANCE:     2.000
        DETRACTOR DISTANCE:    20.000
                   SHAPE:     1.200

          SOLUTION CELLS:

             25               26               55               85    86

            *** SOLUTION NUMBER    3 ***

                    COST:     8.000
          AMENITY DISTANCE:     2.000
        DETRACTOR DISTANCE:     6.000
                   SHAPE:     1.697

          SOLUTION CELLS:

             501              531              532              533   534
```

Figure 10. Listing of solution file for MOLANDA example run.

*** SOLUTION NUMBER 4 ***

COST:	0.000
AMENITY DISTANCE:	2.000
DETRACTOR DISTANCE:	6.000
SHAPE:	2.146

SOLUTION CELLS:

531 532 533 534 535

*** SOLUTION NUMBER 5 ***

COST:	0.000
AMENITY DISTANCE:	2.000
DETRACTOR DISTANCE:	6.000
SHAPE:	2.146

SOLUTION CELLS:

532 536 533 534 535

*** SOLUTION NUMBER 6 ***

COST:	66.000
AMENITY DISTANCE:	3.000
DETRACTOR DISTANCE:	20.518
SHAPE:	1.000

SOLUTION CELLS:

26 56 25 55 24

*** SOLUTION NUMBER 7 ***

COST:	77.000
AMENITY DISTANCE:	2.000
DETRACTOR DISTANCE:	12.999
SHAPE:	1.000

SOLUTION CELLS:

263 293 294 323 324

Figure 10 (Continued)

```
          *** SOLUTION NUMBER    8 ***

                    COST:    58.000
      AMENITY DISTANCE:     3.000
  DETRACTOR DISTANCE:      20.000
                   SHAPE:     1.200

      SOLUTION CELLS:

           26              25              55              85   24
```

Figure 10 (Continued)

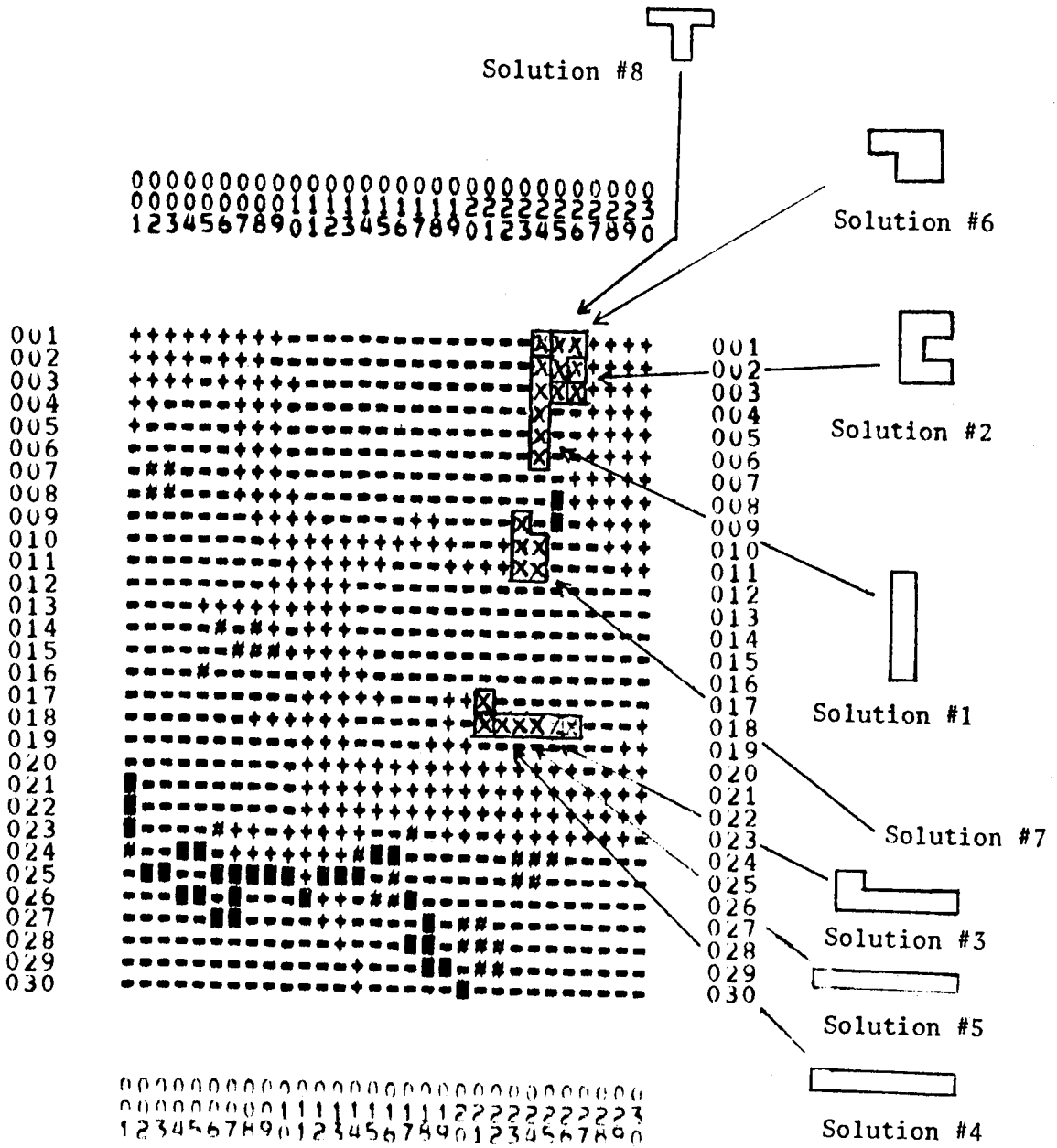


Figure 11. Eight solutions found in the MOLANDA test example.

are scattered across the study area, the program will run faster since most cells will be ignored due to contiguity requirements. If the feasible cells are relatively close to each other, the process will take much longer since many possible configurations must be checked.

In the example run presented in Figure 9, page 64, the initial problem was as follows:

Minimize C
A $\leq$ 3.0
D $\geq$ 15.0
S $\leq$ 3.0

These proximity specifications produced 27 feasible cells for the cost subproblem. Since the example run was searching for five cell solutions, there were $M = \binom{27}{5} = 80{,}730$ possible solutions in the study area. The algorithm only had to examine 32 cells to identify the optimal solution and required approximately 20 seconds* to do so.

The first tradeoff table in Figure 10, page 71, presents four new solutions which required the following two subproblems:

(1) Minimize C
A < 3.0
D > 17.49
S $\leq$ 2.14

(2) Maximize D
A < 3.0
C < 12.0
S $\leq$ 2.14

These two problems had the following computational characteristics:

---

*All time estimates given are average response times running the program on a DEC-10 computer. Response time can vary depending on the number of users present on the computer during runtime.

Problem	(1)	(2)
Number of feasible cells:	13	298
Number of possible solutions (M):	1287	2 × 10 exp 10
Number of cells examined:	143	16358
Approximate time required:	20 seconds	30 seconds

The problems represented in the second tradeoff table in the example run demonstrate the effect that the cell spatial distribution has on the speed of the algorithm. The two problems for the three solutions in the second tradeoff table are as follows:

(1) Minimize A	(2) Maximize D
C < 82.0	C < 82.0
D ≥ 20.0	A ≤ 2.0
S ≤ 1.2	S ≤ 1.2

These two problems had the following computational characteristics:

Problem	(1)	(2)
Number of feasible cells:	8	250
Number of possible solutions (M):	56	78 × 10 exp 8
Number of cells examined:	109	36151
Approximate time required:	20 seconds	40 seconds

Notice that problem (2) had less number of feasible cells than problem (2) of the first tradeoff table but required considerably more cells to be examined and time to find the optimal solution.

In the program, the identification of feasible cells using proximity constraints is the first major time requirement of any subproblem. In the example run, every subproblem solved averaged twenty seconds for this initial process which therefore could be considered a fixed time requirement. In the example run as well as other runs using the Norris study area, response time for a set of three subproblems initiated by a selected objective averaged one and a half minutes. It is the author's opinion that this response time

categorizes the MOLANDA System as an interactive computer system and thus results in an effective man-machine session for aiding the decision maker in the land use allocation process.

CHAPTER 7

CONCLUSION

The multiobjective land allocation model and system presented here represents an important tool for anyone involved in the land use planning and management field. The use of the computer to generate meaningful alternative solutions based on several objectives is an effective approach for the land allocation process. By combining this approach with the professional judgment of a planner or resource analyst, intelligent decisions can be made concerning the future use of our lands.

LIST OF REFERENCES

LIST OF REFERENCES

Bammi, Deepak and Dalip Bammi (1979), "Development of a Comprehensive Land Use Plan by Means of a Multiple Objective Mathematical Programming Model." Interfaces, Vol. 9, No. 2, pp. 50-63.

Baxter, F. P., T. L. Cox, and R. P. Gregory (1975), "Land Analysis Systems." Technical Note No. B9, Tennessee Valley Authority, Norris, Tennessee.

Berry, Joseph K. and C. Dana Tomlin (1979), "An Academic Approach to Cartographic Modeling in Natural Resource Management." Harvard Library of Computer Graphics, Mapping Collection, Vol. 11, pp. 13-19, Harvard University, Cambridge, Mass.

Bottoms, Kenneth E. and E. T. Bartlett (1975), "Resource Allocation through Goal Programming." Journal of Range Management 28(6), November 1975, pp. 442-447.

Dress, Peter E. and Richard C. Field (undated manuscript), "Multi-criterion Decision Methods in Forest Resource Management." University of Georgia, Athens, Georgia.

Dyer, James S. (1972), "Interactive Goal Programming." Man. Sci. 19, pp. 62-70.

Dyer, James S. (1973), "An Empirical Investigation of a Man-Machine Interactive Approach to the Solution of the Multiple Criteria Problem," in Multiple Criteria Decision Making, (J. L. Cochrane and M. Zeleny, eds.). University of South Carolina Press, Columbia, pp. 202-216.

Ecker, J. G. and Nancy E. Shoemaker (1980), "Multiple Objective Linear Programming and the Trade-Off Compromise Set," in Multiple Criteria Decision Making Theory and Applications, (G. Fundel and T. Gal, eds.). Springer-Verlag, New York. Pp. 60-73.

Garfinkel, R. S. and G. L. Nemhauser (1970), "Optimal Political Districting by Implicit Enumeration Techniques." Man. Sci. 16, pp. B495-B508.

Garfinkel, R. S. and G. L. Nemhauser (1972), Integer Programming. John Wiley & Sons, New York, N.Y.

Geoffrion, A. M., J. S. Dyer, and A. Feinburg (1972), "An Interactive Approach for Multi-Criteria Optimization, with an Application to the Operation of an Academic Department." Management Science, Vol. 19, No. 4, pp. 357-368.

Harrison, Terry P. and Richard E. Rosenthal (1981), "A Multiobjective Optimization Approach for Scheduling Timber Harvests in Nonindustrial Private Forest Lands." Working Paper Series, No. 119, The University of Tennessee, Knoxville.

Ho, James K. (1979a), "Holistic Preference Evaluation in Multiple Criteria Optimization." U.S. Dept. of Energy.

Ho, James K. (1979b), "Multiple Criteria Optimization: A Unified Framework." Presented at the Conference on Human Aided Optimization, August 6-8, 1980, The Wharton School, Philadelphia.

Holmes, David D. and Rebecca L. Jolly (1980), "IMGRID Version 3.5." Tennessee Valley Authority, Norris, Tennessee.

Howard, E. Estill and F. Paul Baxter (1976), "A Team Approach to Resource Management Decision-Making." Technical Note No. B16, Tennessee Valley Authority, Norris, Tennessee.

Hultz, John W., Darwin D. Klingman, G. Terry Ross, and Richard M. Soland (1981), "An Interactive Computer System for Multicriteria Facility Location." Computers & Operations Research, Vol. 8, No. 4, pp. 249-261.

Ignizio, James P. (1978), "A Review of Goal Programming: A Tool for Multiobjective Analysis." Journal of Operational Research Society 29, pp. 1109-1119.

Lawler, Eugene L. (1976), Combinatorial Optimization: Networks and Matroids. Holt, Rinehart and Winston, N.Y. Pp. 18-19.

Leuschner, William A., John R. Porter, Marion R. Reynolds, and Harold E. Burkhart (1975), "A Linear Programming Model for Multiple-Use Planning." Can. Journal of Forest Resources, Vol. 5, pp. 485-491.

Keeney, Ralph L. (1978), "Evaluation of Proposed Storage Sites." Operations Research, Vol. 27, No. 1, pp. 48-64.

Rosenthal, Richard E. (1979), "Critique of Goal Programming." Working Paper Series, No. 81, The University of Tennessee, Knoxville.

Schular, A. and J. C. Meadows (1975), "Planning Resource Use on National Forests to Achieve Multiple Objectives." Journal of Environmental Management, 8, pp. 351-366.

Soland, Richard M. (1979), "Multicriteria Optimization: A General Characterization of Efficient Solutions." Decision Sciences, Vol. 10, 1979, pp. 26-38.

Steuer, Ralph E. and Albert T. Schular (1978), "An Interactive Multiple-Objective Linear Programming Approach to a Problem in Forest Management." Operations Research, Vol. 26, No. 2, pp. 254-269.

Wolfe, P. (1970), "Convergence Theory in Nonlinear Programming," in Integer and Nonlinear Programming, (J. Abadie, ed.). North-Holland Publishing Co., Amsterdam.

Zeleny, Milan (1982), Multiple Criteria Decision Making, McGraw-Hill Book Company, New York, N.Y.

APPENDIX

MOLANDA PROGRAM LISTING

```
  10   {***********************************************
  10
  20
  30            M O L A N D A
  40
  50            MULTIOBJECTIVE
  60         LAND ALLOCATION PROGRAM
  70
  80
  90       RESEARCH WORK FOR MGT. SCI. thesis
 100
 110       Spring, 1981
 120
 130       David D. Holmes
 140
 150
 160       UNIVERSITY OF TENNESSEE, KNOXVILLE, TENNESSEE
 170
 180   ***********************************************}
 190
 200   {GLOBAL DEFINITIONS:
 210         IA-ARRAY OF CELL COST VALUES,
 220         IB-ARRAY OF CELL LAND USE VALUES,
 230         IC-ARRAY OF ADMISSABLE CELLS,
 240         AMCODE-AMENITY LAND USE CODE VALUE,
 250         DECODE-DETRACTOR LAND USE CODE VALUE,
 260         SIZE-NUMBER OF CELLS TO BE ALLOCATED,
 270         NAMDCELLS-NUMBER OF AMENITY LAND USE CELLS,
 280         NDETCELLS-NUMBER OF DETRACTOR LAND USE CELLS,
 290         INTCOST,INTAMD,INTDED,INTSHP-INITIAL OBJECTIVE LEVELS,
 300         CURCOST,CURAMD,CURDED,CURSHP-CURRENT OBJECTIVE LEVELS,
 310         TRADE-ARRAY OF TRADEOFF TABLE VALUES,
 320         ADMCELLS-ARRAY OF AMENITY CELLS,
 330         DETCELLS-ARRAY OF DETRACTOR CELLS,
 340         OPTSHP-OPTIMAL SHAPE VALUE FOR SIZE TO BE ALLOCATED,
 350         DISKSOLN-SOLUTION NUMBER STORED ON DISK,
 360         CURSOLCELLS-ARRAY OF CURRENT SOLUTION CELLS (CELL,PROBLEM,SOLUTION
),
 370         CURSOLLEV-ARRAY OF CURRENT SOLUTION OBJECTIVE LEVELS (OBJECTIVE,PR
OBLEM,SOLUTION),
 380         WOSHP-UPPER BOUND ON SHAPE VALUE FOR K CELLS.
 390         SOLLIST-TABLE OF BINARY VALUES INDICATING VALID SOLUTIONS DURING A
 SUBPROBLEM.
 400         TABSOL-TABLE (PROBLEM,SOLUTION) OF BINARY VALUES INDICATING PRESEN
CE OF SOLUTIONS
 410                 WHEN TRADEOFF TABLE IS PRESENTED.
 420
 430                                                          *)
 440
 450   TYPE
 460      REL=0.00..5000.00;
 470       SZE=0..3000;
 480       CST=0..5000;
 490       USE=0..50;
 500       INFILE= FILE OF CHAR;
 510       WKFILE= FILE OF CHAR;
 520       DTFILE=FILE OF CHAR;
 530       PRFILE=FILE OF CHAR;
 540
 550   VAR
 560      IA:PACKED ARRAY [1..1000] OF REAL;
 570      IB:PACKED ARRAY [1..1000] OF USE;
```

```
580       IC:PACKED ARRAY [1..800,1..3] OF REAL;
590       ICPOINT:PACKED ARRAY [1..800] OF SZE;
600       NROWS,NCOLS,COSTELE,LANDELE,REJECT,AMCODE,DECODE,OPTION,
610       SIZE,NCELLS,NAMDCELLS,NDETCELLS,NIC,SELECT,MAXLEV:INTEGER;
620       INTCOST,INTAMD,INTDED,INTSHP,CURCOST,CURAMD,CURDED,
630       CURSHP,WOSHP,OPTSHP:REAL;
640       DISKSoIN,TAB1,TAB2,TAB3,WHCNT:INTEGER;
650       MRKCELL: ARRAY [1..50] OF INTEGER;
660       TRADE: ARRAY [1..2,1..3,1..20] OF REAL;
670       SOLLIST:PACKED ARRAY [1..20] OF USE;
680       ADMCELLS:PACKED ARRAY [1..500] OF SZE;
690       DETCELLS:PACKED ARRAY [1..500] OF SZE;
700       CURSOLCELLS: PACKED ARRAY [1..50,1..3,1..20] OF SZE;
710       CURSOLLEV: PACKED ARRAY [1..4,1..3,1..20] OF REAL;
720       FANS:PACKED ARRAY [1..50] OF SZE;
730       OSHAPE:PACKED ARRAY [1..50] OF REAL;
740       WOSHAPE:PACKED ARRAY [1..50] OF REAL;
750       COMB:PACKED ARRAY [1..51] OF SZE;
760       FCOMB:PACKED ARRAY [1..51] OF SZE;
770       CURMXD,FEAMXD:REAL;
780       TABSOL:PACKED ARRAY [1..3,1..20] OF USE;
790       SOLFILE:INFILE;
800       WRKFILE:WKFILE;
810       PRTFILE:PRFILE;
820       DATFILE:DTFILE;
830
840
850     PROCEDURE RDINT(VAR K:INTEGER);
860     (* ******************************************************
870
880       RDINT PROCEDURE
890
900       THIS PROC READS IN AN INTEGER VARIABLE.
910
920     ****************************************************** *)
930
940     BEGIN
950     BREAK(TTYOUTPUT);
960     RESET(TTY);
970     READ(TTY,K);
980     END;
990
1000    PROCEDURE INITMODEL;
1010    (* ******************************************************
1020
1030       INITMODEL PROCEDURE
1040
1050    THIS PROC INITIALIZES THE PROGRAM RUN BY SETTING UP THE
1060    OPTIMAL SHAPE VALUES FOR CELL SIZES FROM 1 TO 5.
1070    THE COST AND DEFINITION LAYERS ARE READ IN HERE ALSO.
1080
1090    ****************************************************** *)
1100    VAR I:INTEGER;
1110        ID:ARRAY [1..25] OF INTEGER;
1120        J,N:INTEGER;
1130
1140    BEGIN
1150
1160    RESET (DATFILE);
1170    N:=0;
1180    FOR I:=1 TO 30 DO
1190      BEGIN
1200      FOR J:=1 TO 25 DO
1210        BEGIN
1220        READ(DATFILE,ID[J]);
1230        N:=N+1;
```

```
1240        IA[N]:=ID[J];
1250        END;
1260      READLN(DATFILE);
1270      FOR J:=1 TO 5 DO
1280        BEGIN
1290        READ(DATFILE,ID[J]);
1300        N:=N+1;
1310        IA[N]:=ID[J];
1320        END;
1330      READLN(DATFILE);
1340      END;
1350    N:=0;
1360    FOR I:=1 TO 30 DO
1370      BEGIN
1380      FOR J:=1 TO 25 DO
1390        BEGIN
1400        READ(DATFILE,ID[J]);
1410        N:=N+1;
1420        IB[N]:=ID[J];
1430        END;
1440      READLN(DATFILE);
1450      FOR J:=1 TO 5 DO
1460        BEGIN
1470        READ(DATFILE,ID[J]);
1480        N:=N+1;
1490        IB[N]:=ID[J];
1500        END;
1510      READLN(DATFILE);
1520      END;
1530    DISKSOLN:=0;
1540    OSHAPE[1]:=4.0;
1550    OSHAPE[2]:=6.0;
1560    OSHAPE[3]:=11.3137;
1570    OSHAPE[4]:=11.3137;
1580    OSHAPE[5]:=22.36;
1590    WOSHAPE[1]:=1.5;
1600    WOSHAPE[2]:=1.5;
1610    WOSHAPE[3]:=2.4;
1620    WOSHAPE[4]:=3.4;
1630    WOSHAPE[5]:=3.0;
1640    END;
1650
1660    PROCEDURE RDREL(VAR K:REAL);
1670    (* ***************************************************************
1680
1690       RDREL PROCEDURE
1700
1710      THIS PROC READS A REAL VARIABLE FROM THE USER.
1720
1730    *************************************************************** *)
1740
1750    BEGIN
1760      BREAK(TTYOUTPUT);
1770      RESET(TTY);
1780      READ(TTY,K);
1790    END;
1800
1810
1820    PROCEDURE INITRUN;
1830    (* ***************************************************************
1840
1850       This procedure initializes the run for the user.  All
1860        necessary parameters for the run are entered and
1870        verified.
1880
1890     *************************************************************** *)
```

```
1900
1910    VAR    I,K,L,NAMELN:INTEGER;
1920           NAME:PACKED ARRAY [1..8] OF CHAR;
1930           ANS,N :char;
1940
1950   begin
1960      writeln (tty,'LAND USE ALLOCATION PROGRAM');
1970
1980      break(ttyoutput);
1990      ANS:='Y';
2000      while (ANS = 'Y') do
2010         begin
2020         writeln(tty,'Enter study area NAME (8 CHARACTERS):');
2030         break(ttyoutput);
2040         reset(tty);
2050         NAMELN:=0;
2060         WHILE NOT EOLN(TTY) AND (NAMELN <= 8) DO
2070           BEGIN
2080           READ(TTY,N);
2090           NAMELN:=NAMELN+1;
2100           NAME[NAMELN]:=N;
2110           END;
2120         WRITELN(TTY,'ENTER NUMBER OF ROWS:');
2130         RDINT(NROWS);
2140         WRITELN(TTY,'ENTER NUMBER OF COLUMNS:');
2150         RDINT(NCOLS);
2160         WRITELN(TTY,'ENTER LAND USE AMENITY CODE:');
2170         RDINT(AMCODE);
2180         WRITELN(TTY,'ENTER DETRACTOR LAND USE CODE:');
2190         RDINT(DECODE);
2200         writeln(tty,'enter reject land use code:');
2210         rdint(reject);
2220
2230         writeln(tty,'CURRENT INPUT VALUES');
2240         write(tty,'study area name:');
2250         WRITELN(TTY,'  ',NAME:NAMELN);
2260         writeln(tty,'Number of rows and columns:',NROWS:5,NCOLS:5);
2270         writeln(tty,'Amenity land use code:',AMCODE:3);
2280         writeln(tty,'Detractor land use code:',DECODE:3);
2290         writeln(tty,'reject land use code:',reject:3);
2300         writeln(tty,'Do you want to change these?');
2310         writeln(tty,'Enter y or n');
2320         break(ttyoutput);
2330         reset(tty);
2340         read(tty,ANS);
2350      end;
2360
2370      L:=0;  K:=0;
2380      NCELLS:=NROWS*NCOLS;
2390      FOR I:=1 TO NCELLS DO
2400        BEGIN
2410        IF IB[I] = AMCODE THEN
2420          BEGIN
2430          L:=L+1;
2440          ADMCELLS[L]:=I;
2450          END;
2460        IF IB[I] = DECODE THEN
2470          BEGIN
2480          K:=K+1;
2490          DETCELLS[K]:=I;
2500          END;
2510        END;
2520     NAMDCELLS:=L;
2530     NDETCELLS:=K;
2540   END;
2550
```

```
2560   PROCEDURE BLINE(K:INTEGER);
2570   (* ************************************************************
2580
2590        BLINE PROCEDURE
2600
2610     THIS PROC SPACES THE OUTPUT TO THE TTY K LINES.
2620
2630   ************************************************************ *)
2640   VAR I:INTEGER;
2650
2660   BEGIN
2670     FOR I:=1 TO K DO
2680       BEGIN
2690       WRITELN(TTY);
2700       END;
2710   END;
2720
2730
2740   PROCEDURE SELSIZ;
2750     (* ************************************************************
2760
2770      SELSIZ PROCEDURE
2780
2790     THIS PROC DETERMINES THE SIZE OF THE SHAPE THAT IS
2800     TO BE ALLOCATED.  IT ALSO FINDS THE OPTIMAL SHAPE VALUE
2810     PREVIOUSLY STORED.
2820
2830   ************************************************************ *)
2840
2850   BEGIN
2860     WRITELN(TTY,'PROBLEM SELECTION:');
2870     BLINE(1);
2880     WRITELN(TTY,'ENTER SIZE OF SHAPE TO BE ALLOCATED');
2890     WRITELN(TTY,'(NUMBER OF CELLS):');
2900     RDINT(SIZE);
2910     OPTSHP:=OSHAPE[SIZE];
2920     WOSHP:=WOSHAPE[SIZE];
2930   END;
2940
2950   FUNCTION NOTADJCHK(TROW,TCOL:INTEGER):BOOLEAN;
2960    {*************************************************
2970
2980      NOTADJCHK FUNCTION
2990   THIS FUNCTION CHECKS WHETHER OR NOT A CELL IS
3000   ADJACENT TO ANY OTHER CELLS IN A FEASIBLE SOLUTION.
3010
3020   *************************************************}

3030
3040   VAR POS,P2,I:INTEGER;
3050
3060   BEGIN
3070     POS:=((TROW-1)*NCOLS)+TCOL;
3080     NOTADJCHK:=TRUE;
3090     FOR I:=1 TO SIZE DO
3100       BEGIN
3110       P2:=ICPOINT[COMB[I]];
3120       IF (POS=P2) THEN NOTADJCHK:=FALSE;
3130      END;
3140   END;
3150
3160   PROCEDURE COMPSHP(VAR BSHAPE:REAL);
3170   {*************************************************
3180
3190      COMPSHP PROCEDURE
```

```
3200
3210    THIS PROC COMPUTES THE SHAPE VALUE OF A FEASIBLE SHAPE.
3220    THE FUNCTION NOTADJCHK IS USED TO DETERMINE ADJACENCY
3230    BETWEEN CELLS.
3240
3250    ************************************************************}
3260
3270    VAR ROW,POS,COL,I,J,K,EDGE,TROW,TCOL:INTEGER;
3280        DIST,MAX:REAL;
3290
3300    BEGIN
3310      EDGE:=0;
3320      FOR I:=1 TO SIZE DO
3330        BEGIN
3340        POS:=ICPOINT[COMB[I]];
3350        ROW:=POS DIV NCOLS;
3360        IF POS # (ROW*NCOLS) THEN ROW:=ROW+1;
3370        COL:=POS - ((ROW-1)*NCOLS);
3380        TROW:=ROW;
3390        TCOL:=COL+1;
3400        IF (NOTADJCHK(TROW,TCOL)) THEN EDGE:=EDGE+1;
3410        TCOL:=COL-1;
3420        IF (NOTADJCHK(TROW,TCOL)) THEN EDGE:=EDGE+1;
3430        TCOL:=COL;
3440        TROW:=ROW+1;
3450        IF (NOTADJCHK(TROW,TCOL)) THEN EDGE:=EDGE+1;
3460        TROW:=ROW-1;
3470        IF (NOTADJCHK(TROW,TCOL)) THEN EDGE:=EDGE+1;
3480      END;
3490    MAX:=0;
3500    FOR I:= 1 TO (SIZE-1) DO
3510      BEGIN
3520      POS:=ICPOINT[COMB[I]];
3530      ROW:=POS DIV NCOLS;
3540      IF POS # (ROW*NCOLS) THEN ROW:=ROW+1;
3550      COL:=POS - ((ROW-1)*NCOLS);
3560      FOR J:= (I+1) TO SIZE DO
3570        BEGIN
3580        POS:=ICPOINT[COMB[J]];
3590        TROW:=POS DIV NCOLS;
3600        IF POS # (TROW*NCOLS) THEN TROW:=TROW+1;
3610        TCOL:=POS - ((TROW-1)*NCOLS);
3620        DIST:=SQRT(SQR(COL-TCOL)+SQR(ROW-TROW));
3630        IF (DIST > MAX) THEN MAX:=DIST;
3640        END;
3650      END;
3660    BSHAPE:=(EDGE*MAX)/OPTSHP;
3670    END;
3680
3690
3700    FUNCTION CONTIG(KIN,TEMP:INTEGER):BOOLEAN;
3710    {************************************************************
3720
3730          CONTIG FUNCTION
3740
3750    THIS FUNCTION DETERMINES WHETHER OR NOT A FULL SHAPE
3760    COULD BE COMPLETED USING THE LATEST  POSSIBLE CELL AND
3770    ALL THE CURRENT CELLS IN THE SHAPE.
3780
3790    ************************************************************}
3800
3810    VAR I,J,K,L,ROW,POS,FLAG,COL,AROW,DROW,DCOL,ACOL:INTEGER;
3820        DIST,R:REAL;
3830
3840    BEGIN
3850      POS:=ICPOINT[TEMP];
```

```
3860      ROW:=POS DIV NCOLS;
3870      IF (POS # (ROW*NCOLS)) THEN ROW:=ROW+1;
3880      COL:=POS - ((ROW-1)*NCOLS);
3890      CONTIG:=FALSE;
3900      FLAG:=0;
3910      I:=1;
3920      WHILE (FLAG = 0) AND (I < KIN) DO
3930        BEGIN
3940        POS:=ICPOINT[COMB[I]];
3950        AROW:=POS DIV NCOLS;
3960        IF (POS # (AROW*NCOLS)) THEN AROW:=AROW+1;
3970        ACOL:=POS - ((AROW-1)*NCOLS);
3980        DROW:=AROW-ROW;
3990        DCOL:=ACOL-COL;
4000        L:=ABS(DROW) + ABS(DCOL) -1;
4010        IF L <= (SIZE-KIN) THEN
4020            BEGIN
4030            CONTIG:=TRUE;
4040            FLAG:=1;
4050            END;
4060        I:=I+1;
4070      END;
4080    END;
4090    PROCEDURE SNEIGHBOR(VAR CELL,POS:INTEGER);
4100    (* ******************************************************
4110
4120       SNEIGHBOR PROCEDURE
4130
4140    THIS RECURSIVE PROC IS USED TO CHECK EACH CELL IN A
4150    SHAPE THAT HAS BEEN PUT TOGETHER BY COST AND ODIST.
4160    THE ROUTINE RECURSIVELY SEARCHES THROUGH THE SHAPE
4170    AND CHECKS EACH CELL'S 4 NEIGHBORS AS IT GOES.
4180
4190    ****************************************************** *)
4200
4210    VAR I,J,L,ROW,COL,NPOS,FLAG:INTEGER;
4220
4230    BEGIN
4240      ROW:=POS DIV NCOLS;
4250      IF (POS # (ROW*NCOLS)) THEN ROW:=ROW+1;
4260      COL:=POS-((ROW-1)*NCOLS);
4270      FOR J:=1 TO 4 DO
4280        BEGIN
4290        FLAG:=0;
4300        CASE J OF
4310        1: BEGIN
4320           NPOS:=((ROW-1)*NCOLS) +(COL-1);   (* WEST *)
4330           IF (COL-1) = 0 THEN FLAG:=1;
4340           END;
4350        2: BEGIN
4360           NPOS:=((ROW-2)*NCOLS) + COL;      (* NORTH *)
4370           IF (ROW-2) < 0 THEN FLAG:=1;
4380           END;
4390        3: BEGIN
4400           NPOS:=((ROW-1)*NCOLS) + (COL+1); (* EAST *)
4410           IF (COL+1) > NCOLS THEN FLAG:=1;
4420           END;
4430        4: BEGIN
4440           NPOS:=(ROW*NCOLS) + COL;          (* SOUTH *)
4450           IF (ROW = NROWS) THEN FLAG:=1;
4460           END
4470        END;
4480        IF FLAG = 0 THEN
4490          BEGIN
4500          FOR L:= 1 TO SIZE DO
4510            BEGIN
```

```
4520          IF ((L#CELL) AND (MRKCELL[L]=0)) THEN
4530            BEGIN
4540            IF (NPOS = ICPOINT[COMB[L]]) THEN
4550              BEGIN
4560              MRKCELL[L]:=1;
4570              WHCNT:=WHCNT+1;
4580              SNEIGHBOR(L,NPOS);
4590              END;
4600            END;
4610          END;
4620        END;
4630      END;
4640  END;
4650
4660  FUNCTION WHSHAPE:BOOLEAN;
4670  (* ******************************************************
4680
4690     WHSHAPE FUNCTION
4700
4710  THIS FUNCTION CONTROLS THE SNEIGHBOR PROC WHICH DETERMINES
4720  WHETHER A WHOLE SHAPE HAS BEEN FOUND OR ONE WITH HOLES
4730  IN IT.
4740
4750  ****************************************************** *)
4760
4770  VAR POS,CELL,I,J,K,L:INTEGER;
4780
4790  BEGIN
4800    FOR I:=1 TO 50 DO
4810      MRKCELL[I]:=0;
4820    WHCNT:=1;
4830    POS:=ICPOINT[COMB[1]];
4840    CELL:=1;
4850    MRKCELL[1]:=1;
4860    SNEIGHBOR(CELL,POS);
4870    IF WHCNT < SIZE THEN WHSHAPE:=FALSE
4880                    ELSE WHSHAPE:=TRUE;
4890  END;
4900
4910  PROCEDURE SWITCH(VAR I,K:INTEGER);
4920  {******************************************************
4930
4940     SWITCH PROCEDURE
4950
4960  THIS PROC IS USED IN THE SORTING OF THE ADMISSABLE CELLS
4970  TO SWITCH TWO CELLS. A COMMON BUBBLE SORT  IS USED.
4980
4990  ******************************************************}
5000
5010  VAR T:REAL;
5020      L:INTEGER;
5030
5040  BEGIN
5050    K:=1;
5060    L:=ICPOINT[I];
5070    ICPOINT[I]:=ICPOINT[I+1];
5080    ICPOINT[I+1]:=L;
5090    FOR L:=1 TO 3 DO
5100      BEGIN
5110      T:=IC[I,L];
5120      IC[I,L]:=IC[I+1,L];
5130      IC[I+1,L]:=T;
5140    END;
5150  END;
5160
5170  PROCEDURE ORDERCELLS(OPT:INTEGER);
5180  {******************************************************
```

```
5190
5200        ORDERCELLS PROCEDURE
5210
5220    THIS PROC SORTS THE ADMISSABLE CELLS BY EITHER COST,
5230    MIN DIST, OR MAX DIST.  A STANDARD BUBBLE SORT IS
5240    USED.
5250
5260    ******************************************************}
5270
5280    VAR I,J,K:INTEGER;
5290        FIN,FLAG:INTEGER;
5300
5310    BEGIN
5320      FIN:=0;
5330      WHILE FIN=0 DO
5340        BEGIN
5350        FLAG:=0;
5360        FOR I:= 1 TO (NIC-1) DO
5370          BEGIN
5380          CASE OPT OF
5390            1:  IF IC[I,1] > IC[I+1,1] THEN SWITCH(I,FLAG);
5400            2:  IF IC[I,2] > IC[I+1,2] THEN SWITCH(I,FLAG);
5410            3:  IF IC[I,3] < IC[I+1,3] THEN SWITCH(I,FLAG)
5420          END;
5430        END;
5440        IF FLAG =  0 THEN FIN:=1;
5450      END;
5460    END;
5470
5480    PROCEDURE MARKCELLS(AMD,DED:REAL);
5490    {******************************************************
5500
5510         MARKCELLS PROCEDURE
5520
5530    THIS PROC MARKS THE CELLS WHICH ARE TO BE ADMISSABLE
5540    FOR THIS SUBPROBLEM.  THE PERMISSABLE DISTANCES ARE
5550    ENTERED AND USED TO DETERMINE THE ADMISSABLE CELLS.
5560
5570    ******************************************************}
5580
5590    VAR ICPNT,I,J,K,L:INTEGER;
5600        ROW,COL,AROW,ACOL:INTEGER;
5610        DIST,MXDIST,MNDIST:REAL;
5620
5630    BEGIN
5640      NIC:=0;
5650      ICPNT:=0;
5660      FOR I:=1 TO NCells do
5670        BEGIN
5680        IF IB[I] # REJECT THEN
5690           BEGIN
5700           IF IB[I] # DECODE THEN
5710              BEGIN
5720              if ib[i] # amcode then
5730              begin
5740              ROW:=I DIV NCOLS;
5750              IF (I # (ROW*NCOLS)) THEN ROW:=ROW+1;
5760              COL:=I - ((ROW-1)*NCOLS);
5770              MNDIST:=10000;
5780              FOR J:=1 TO NAMDCELLS DO
5790                BEGIN
5800                AROW:=ADMCELLS[J] DIV NCOLS;
5810                IF (ADMCELLS[J] # (AROW*NCOLS)) THEN AROW:=AROW+1;
5820                ACOL:=ADMCELLS[J] - ((AROW-1)*NCOLS);
5830                DIST:=SQRT(SQR(ROW-AROW)+SQR(COL-ACOL));
```

```
5840              IF (DIST < MNDIST) THEN  MNDIST:=DIST;
5850            END;
5860          IF (mndist <= amd) THEN
5870            BEGIN
5880            MXDIST:=10000;
5890            FOR J:=1 TO  NDETCELLS DO
5900              BEGIN
5910              AROW:=DETCELLS[J] DIV NCOLS;
5920              IF (DETCELLS[J] # (AROW*NCOLS)) THEN AROW:=AROW+1;
5930              ACOL:=DETCELLS[J] - ((AROW-1) * NCOLS);
5940              DIST:=SQRT(SQR(AROW-ROW)+SQR(ACOL-COL));
5950              IF (DIST < MXDIST) THEN MXDIST:=DIST;
5960              END;
5970          IF (mxdist >= ded) THEN
5980            BEGIN
5990            ICPNT:=ICPNT+1;
6000            ICPOINT[ICPNT]:=I;
6010            IC[ICPNT,1]:=IA[I];
6020            IC[ICPNT,2]:=MNDIST;
6030            IC[ICPNT,3]:=MXDIST;
6040            NIC:=ICPNT;
6050            END;
6060            END;
6070          END;
6080        END;
6090      END;
6100  END;
6110  END;
6120
6130  PROCEDURE STORESOL(FCOST,FAMD,FDED,FSHAPE:REAL;SNUM,SOLNUM:INTEGER);
6140  {*********************************************************
6150
6160      STORESOL PROCEDURE
6170
6180  THIS PROC STORES A FEASIBLE SOLUTION IN THE CURRENT SOLUTION
6190  ARRAYS TO BE USED WITH THE TRADEOFF TABLE.
6200
6210  *********************************************************}
6220
6230  VAR I:INTEGER;
6240
6250  BEGIN
6260    FOR I:=1 TO SIZE DO
6270      CURSOLCELLS[I,SNUM,SOLNUM]:=FANS[I];
6280    CURSOLLEV[1,SNUM,SOLNUM]:=FCOST;
6290    CURSOLLEV[2,SNUM,SOLNUM]:=FAMD;
6300    CURSOLLEV[3,SNUM,SOLNUM]:=FDED;
6310    CURSOLLEV[4,SNUM,SOLNUM]:=FSHAPE;
6320  END;
6330
6340  PROCEDURE NEXTCELL(VAR KIN,TEMP:INTEGER);
6350  {*********************************************************
6360
6370        NEXTCELL PROCEDURE
6380
6390  THIS PROC ASSIGNS THE NEXT CELL TO THE COMB ARRAY
6400  WHICH IS BUILDING THE SHAPE ALLOCATION.
6410
6420  *********************************************************}
6430
6440  BEGIN
6450   TEMP:=TEMP+1;
6460  END;
6470
6480  PROCEDURE RESETCELL(VAR KIN,TEMP:INTEGER;VAR RCOST:REAL);
```

```
6500
6510            RESETCELL PROCEDURE
6520
6530    THIS PROC RESETS THE COMBINATION INDEX FOR THE COST SUBPROBLEM.
6540
6550    ************************************************}
6560
6570    BEGIN
6580      COMB[KIN]:=0;
6590      KIN:=KIN-1;
6600      IF KIN > 0 THEN
6610         BEGIN
6620         TEMP:=COMB[KIN];
6630         BCOST:=BCOST-IC[COMB[KIN],1];
6640         END
6650         ELSE TEMP:=0;
6660    END;
6670
6680    PROCEDURE FADMDIST(VAR MAX:REAL);
6690    (* ************************************************
6700
6710       FADMDIST PROCEDURE
6720
6730      THIS PROCEDURE FINDS THE MAX DISTANCE BETWEEN A
6740    SOLUTION STORED IN FANS AND THE ADMENITY LAND USE CELLS.
6750
6760    ************************************************ *)
6770
6780    VAR I,J,K,L:INTEGER;
6790        ROW,COL,AROW,ACOL,POS:INTEGER;
6800        DIST,MN:REAL;
6810
6820    BEGIN
6830      MAX:=0;
6840
6850      FOR I:=1 TO SIZE DO
6860        BEGIN
6870        MN:=10000;
6880        POS:=FANS[I];
6890        ROW:=POS DIV NCOLS;
6900        IF (POS # (ROW*NCOLS)) THEN ROW:=ROW+1;
6910        COL:=POS - ((ROW-1)*NCOLS);
6920
6930        FOR J:=1 TO NAMDCELLS DO
6940          BEGIN
6950          AROW:=ADMCELLS[J] DIV NCOLS;
6960          IF (ADMCELLS[J] # (AROW*NCOLS)) THEN AROW:=AROW+1;
6970          ACOL:=ADMCELLS[J] - ((AROW-1)*NCOLS);
6980          DIST:=SQRT(SQR(AROW-ROW)+SQR(ACOL-COL));
6990          IF (DIST < MN) THEN MN:=DIST;
7000          END;
7010        IF (MN > MAX) THEN MAX:=MN;
7020        END;
7030    END;
7040
7050    PROCEDURE FDETDIST(VAR MIN:REAL);
7060    (* ************************************************
7070
7080       FDETDIST PROCEDURE
7090
7100      THIS PROCEDURE FINDS THE MIN DISTANCE BETWEEN A
7110    SOLUTION STORED IN FANS AND THE DETRACTOR LAND USE CELLS.
7120
7130    ************************************************ *)
7140
7150    VAR I,J,K,L:INTEGER;
```

```
7160        ROW,COL,AROW,ACOL,POS:INTEGER;
7170        DIST:REAL;
7180
7190    BEGIN
7200      MIN:=10000;
7210
7220      FOR I:=1 TO SIZE DO
7230        BEGIN
7240        POS:=FANS[I];
7250        ROW:=POS DIV NCOLS;
7260        IF (POS # (ROW*NCOLS)) THEN ROW:=ROW+1;
7270        COL:=POS - ((ROW-1)*NCOLS);
7280
7290        FOR J:=1 TO NDETCELLS DO
7300          BEGIN
7310          AROW:=DETCELLS[J] DIV NCOLS;
7320          IF (DETCELLS[J] # (AROW*NCOLS)) THEN AROW:=AROW+1;
7330          ACOL:=DETCELLS[J] - ((AROW-1)*NCOLS);
7340          DIST:=SQRT(SQR(AROW-ROW)+SQR(ACOL-COL));
7350          IF (DIST < MIN) THEN MIN:=DIST;
7360          END;
7370        END;
7380    END;
7390
7400
7410    PROCEDURE DETCOST(VAR SCOST:REAL;VAR KIN,TEMP:INTEGER);
7420    (* ************************************************
7430
7440      DETCOST PROCEDURE
7450
7460    THIS PROC COMPUTES THE MINIMUM COST THAT CAN BE
7470    ACHIEVED BY USING THE CURRENT CELLS IN COMB.  THIS IS
7480    DONE BY ADDING THE COST OF TEMP AND THE COST OF TEMP + I
7490    WHERE I GOES FROM KIN+1 TO SIZE.
7500
7510    ************************************************ *)
7520
7530    VAR L,J,I:INTEGER;
7540        TCST:REAL;
7550
7560    BEGIN
7570      TCST:=0.0;
7580      IF KIN>1 THEN BEGIN
7590        FOR L:=1 TO (KIN-1) DO
7600          TCST:=TCST + IC[COMB[L],1];
7610        END;
7620      SCOST:=TCST + IC[TEMP,1];
7630      IF (KIN < SIZE) THEN BEGIN
7640        FOR L:=1 TO (SIZE-KIN) DO
7650          BEGIN
7660          J:=TEMP+L;
7670          SCOST:=SCOST+IC[J,1];
7680          END;
7690        END;
7700    END;
7710
7720    PROCEDURE EFFCHK(BCOST,MAX,MIN,SHP:REAL;VAR SOLCNT,SNUM,EFLAG:
7730                     INTEGER);
7740    (* ************************************************
7750
7760      EFFCHK PROCEDURE
7770
7780    THIS PROC CHECKS A FEASIBLE SOLUTION AGAINST ANY OTHER ALTERNATE
7790    OPTIMAL SOLUTIONS FOR EFFICIENCY.  EFLAG IS RETURNED AS FOLLOWS:
7800
```

```
7810          EFLAG= 1:  NEW SOLUTION IS DOMINATED BY EXISTING OPTIMAL
7820                      SOLUTION.
7830                 2:  NEW SOLUTION DOMINATES EXISTING OPTIMAL SOLUTION.
7840                 3:  NEW SOLUTION IS NONDOMINATED.
7850
7860    ********************************************************** *)
7870
7880    BEGIN
7890      EFLAG:=0;
7900      IF (BCOST <= CURSOLLEV[1,SNUM,SOLCNT]) THEN
7910        BEGIN
7920        IF (MAX <= CURSOLLEV[2,SNUM,SOLCNT]) THEN
7930          BEGIN
7940          IF (MIN >= CURSOLLEV[3,SNUM,SOLCNT]) THEN
7950            BEGIN
7960            IF (SHP <= CURSOLLEV[4,SNUM,SOLCNT]) THEN
7970              EFLAG:=EFLAG+2;
7980            END;
7990          END;
8000        END;
8010      IF (BCOST >= CURSOLLEV[1,SNUM,SOLCNT]) THEN
8020        BEGIN
8030        IF (MAX >= CURSOLLEV[2,SNUM,SOLCNT]) THEN
8040          BEGIN
8050          IF (MIN <= CURSOLLEV[3,SNUM,SOLCNT]) THEN
8060            BEGIN
8070            IF (SHP >= CURSOLLEV[4,SNUM,SOLCNT]) THEN
8080              EFLAG:=EFLAG+1;
8090             END;
8100          END;
8110        END;
8120      IF (EFLAG = 0) THEN EFLAG:=3;
8130    END;
8140
8150    PROCEDURE COMPRESS(VAR SNUM,SOLNUM:INTEGER);
8160    (* ***********************************************************
8170
8180      COMPRESS PROCEDURE
8190
8200    THIS PROC COMPRESSES THE ALTERNATE OPTIMAL SOLUTIONS IN
8210    CURSOLLEV. EMPTY SLOTS OCCUR WHEN SUBSEQUENT SOLUTIONS
8220    ARE FOUND TO BE DOMINATED AND REMOVED FROM THE ARRAY.
8230    IT IS THEN NECESSARY TO COMPRESS THE ARRAY TO REMOVE
8240    ANY EMPTY SLOTS.
8250
8260    ********************************************************** *)
8270
8280    VAR I,J,K,INFLAG,OUTFLAG:INTEGER;
8290
8300    BEGIN
8310      OUTFLAG:=0;
8320      I:=0;
8330      WHILE (OUTFLAG =0) DO
8340        BEGIN
8350        I:=I+1;
8360        IF SOLLIST[I]=0 THEN
8370          BEGIN
8380          INFLAG:=0;
8390          J:=I;
8400          WHILE (INFLAG =0) DO
8410            BEGIN
8420            J:=J+1;
8430            IF J < 21 THEN
8440              BEGIN
8450              IF (SOLLIST[J] = 1) THEN
8460                BEGIN
```

```
8470              INFLAG:=1;
8480              CURSOLLEV[1,SNUM,I]:=CURSOLLEV[1,SNUM,J];
8490              CURSOLLEV[2,SNUM,I]:=CURSOLLEV[2,SNUM,J];
8500              CURSOLLEV[3,SNUM,I]:=CURSOLLEV[3,SNUM,J];
8510              CURSOLLEV[4,SNUM,I]:=CURSOLLEV[4,SNUM,J];
8520              FOR K:=1 TO SIZE DO
8530                CURSOLCELLS[K,SNUM,I]:=CURSOLCELLS[K,SNUM,J];
8540              SOLLIST[I]:=1;
8550              SOLLIST[J]:=0;
8560              END;
8570            END
8580            ELSE BEGIN
8590                 OUTFLAG:=1;
8600                 INFLAG:=1;
8610                 SOLNUM:=I-1;
8620                 END;
8630          END;
8640        END;
8650      END;
8660  END;
8670
8680  PROCEDURE COST(var TSHP:REAL;VAR STAT:INTEGER;TOPT,SNUM,SOPT:INTEGER);
8690  {*****************************************************
8700
8710      COST PROCEDURE
8720
8730  THIS PROC SOLVES THE COST SUBPROBLEM.
8740
8750  *****************************************************}
8760
8770  VAR BCOST,BSHAPE,FCOST,TCOST,FSHAPE:REAL;
8780      FIN,LFLG,FLAG,COMP,I,J,K,L,TEMP,KIN,NCEX:INTEGER;
8790      TCAMD,tcded,shp,MIN,SCOST,TCST,MAX:real;
8800      PFLAG,EFLAG,SOLNUM,NALTSOL,SOLCNT,POS,TOTALT:INTEGER;
8810
8820  BEGIN
8830    case topt of
8840     1:begin
8850        tcded:=curded;
8860        tcamd:=curamd- 0.5;
8870        end;
8880     2:begin
8890       tcded:=curded+ 0.5;
8900       tcamd:=curamd;
8910       end;
8920     3:begin
8930       tcded:=curded;
8940       tcamd:=curamd;
8950       END;
8960     4:BEGIN
8970       TCDED:=CURDED;
6980       TCAMD:=CURAMD;
8990       END
9000    end;
9010
9020    (* DETERMINE FEASIBLE CELLS AND ORDER THEM BY COST. *)
9030
9040    MARKCELLS(tcamd,tcded);
9050    IF NIC > 1 THEN ORDERCELLS(1);
9060
9070    IF SOPT = 1 THEN FCOST:=100000.0
9080                ELSE FCOST:=CURCOST;
9090    FLAG:=0;
9100    FIN:=0;
9110    TEMP:=0;
```

```
9140      BCOST:=0.0;
9150      STAT:=0;
9160      FSHAPE:=curshp;
9170      BSHAPE:=0.0;
9180      SOLNUM:=0;
9190      NALTSOL:=0;
9200      FOR I:=1 TO 50 DO
9210        COMB[I]:=0;
9220
9230      WHILE FIN = 0 DO
9240        BEGIN
9250        COMP:=0;
9260        WHILE COMP = 0 DO
9270          BEGIN
9280
9290          (* RETRIEVE NEXT AVAILABLE CELL FOR CURRENT PARTIAL SOLUTION. *)
9300
9310          NEXTCELL(KIN,TEMP);
9320          NCEX:=NCEX+1;
9330          LFLG:=0;
9340
9350          (* IF CELL INDEX IS GREATER THAN N, BACKTRACK ON POSITION. *)
9360
9370          WHILE (LFLG = 0) DO
9380          BEGIN
9390          IF TEMP > NIC THEN BEGIN
9400                             RESETCELL(KIN,TEMP,BCOST);
9410                             NEXTCELL(KIN,TEMP);
9420                             END
9430                        ELSE LFLG:=1;
9440          END;
9450          IF KIN > 0 THEN
9460          BEGIN
9470          BCOST:=BCOST+IC[TEMP,1];
9480
9490          (* DETERMINE LOWER BOUND ON COST OF PARTIAL SOLUTION AND COMPARE T
O
9500             CURRENT FEASIBLE UPPER BOUND. *)
9510
9520          DETCOST(SCOST,KIN,TEMP);
9530          IF ((SOPT = 1) AND (SCOST <= FCOST)) OR
9540       .     ((SOPT = 2) AND (TOPT = 4) AND (SCOST < FCOST)) OR
9550             ((SOPT = 2) AND (TOPT < 3) AND (SCOST <= FCOST)) THEN
9560            BEGIN
9570            IF KIN > 1 THEN
9580              BEGIN
9590
9600              (* CHECK FOR CONTIGUITY USING THIS CELL. *)
9610
9620              IF CONTIG(KIN,TEMP) THEN
9630              BEGIN
9640
9650                   (* IF YOU REACH HERE, THE CELL IN TEMP CAN BE ADDED DUE
9660                      TO COST AND CONTIGUITY. OTHERWISE, START RESETTING
9670                      POSITIONS IN CURRENT PARTIAL SOLUTION. *)
9680
9690                   COMB[KIN]:=TEMP;
9700                   KIN:=KIN+1;
9710                   IF (KIN > SIZE) THEN COMP:=1;
9720              END
9730                ELSE BCOST:=BCOST-IC[TEMP,1];
9740            END
9750            ELSE IF TEMP < NIC THEN BEGIN
9760                 COMB[KIN]:=TEMP;
9770                 KIN:=KIN+1;
9780                 END;
```

```
9790          END
9800          ELSE
9810            BEGIN
9820              BCOST:=BCOST-IC[TEMP,1];
9830              RESETCELL(KIN,TEMP,BCOST);
9840              IF KIN = 0 THEN
9850                BEGIN
9860                FIN:=1;
9870                COMP:=1;
9880                END;
9890          END;
9900          END
9910          ELSE BEGIN
9920                FIN:=1;
9930                COMP:=1;
9940                END;
9950        END;
9960        IF FIN = 0 THEN
9970        BEGIN
9980
9990        (* WHEN A COMPLETE SOLUTION IS BUILT, CHECK FOR OVERALL CONTIGUITY.
*)
10000
10010       IF WHSHAPE THEN
10020       BEGIN
10030
10040       (* CALCULATE SHAPE VALUE AND COMPARE FOR FEASIBILITY. *)
10050
10060       COMPSHP(BSHAPE);
10070       IF ((SOPT = 1) AND (TOPT < 3) AND (BSHAPE <= CURSHP)) OR
10080          ((SOPT = 1) AND (TOPT = 3) AND (BSHAPE < CURSHP)) OR
10090          ((SOPT = 2) AND (BSHAPE <= CURSHP)) THEN
10100         begin
10110
10120         (* DETERMINE MAX AND MIN DISTANCE, STORE CELLS, AND CHECK IF THIS
10130            IS THE FIRST OPTIMAL SOLUTION FOUND WITH CURRENT COST. *)
10140
10150         shp:=bshape;
10160          FLAG:=1;
10170          FOR I:=1 TO SIZE DO
10180           FANS[I]:=ICPOINT[COMB[I]];
10190          FADMDIST(MAX);
10200          FDETDIST(MIN);
10210            PFLAG:=0;
10220            IF ((SOPT=1) AND (BCOST=FCOST) AND (SOLNUM>0)) OR
10230               ((SOPT=2) AND (SHP=CURSHP) AND (SOLNUM>0)) THEN
10240               BEGIN
10250
10260               (* IF MORE THAN ONE OPTIMAL SOLUTION HAS BEEN FOUND WITH
10270                  SAME COST, COMPARE IT TO OTHERS FOR DOMINATION. *)
10280
10290               NALTSOL:=NALTSOL+1;
10300               TOTALT:=NALTSOL-1;
10310               SOLCNT:=0;
10320               EFLAG:=3;
10330               WHILE ((SOLCNT < TOTALT) AND (EFLAG > 1)) DO
10340                 BEGIN
10350                 SOLCNT:=SOLCNT+1;
10360                 EFLAG:=0;
10370                 EFFCHK(BCOST,MAX,MIN,SHP,SOLCNT,SNUM,EFLAG);
10380                 IF (EFLAG > 1) THEN PFLAG:=1
10390                                ELSE PFLAG:=0;
10400                 IF (EFLAG = 2) THEN SOLLIST[SOLCNT]:=0;
10410                 IF ((EFLAG = 2) OR (EFLAG=1)) THEN
10420                                NALTSOL:=NALTSOL-1;
```

```
10430                 END;
10440               END
10450               ELSE BEGIN
10460
10470                 (* IF A NEW SOLUTION IS FOUND WITH CHEAPER COST, REMOVE
10480                    ALL PREVIOUSLY FOUND SOLUTIONS. *)
10490
10500                 FOR I:=1 TO 20 DO
10510                   SOLLIST[I]:=0;
10520                 NALTSOL:=1;
10530                 SOLNUM:=0;
10540                 PFLAG:=1;
10550                 END;
10560             IF PFLAG = 1 THEN BEGIN
10570
10580                   (* STORE ANY NONDOMINATED SOLUTIONS. *)
10590
10600                 SOLNUM:=SOLNUM+1;
10610                 FCOST:=BCOST;
10620                 STORESOL(FCOST,MAX,MIN,SHP,SNUM,SOLNUM);
10630                 SOLLIST[SOLNUM]:=1;
10640                 END;
10650
10660             (* REMOVE ANY EMPTY SLOTS IN SOLUTION LIST. *)
10670
10680             COMPRESS(SNUM,SOLNUM);
10690             KIN:=KIN-1;
10700             BCOST:=BCOST-IC[COMB[SIZE],1];
10710           END
10720
10730           (* AFTER SOLUTION HAS BEEN CONSIDERED, RESET CELL INDEX AND CONTIN
UE. *)
10740
10750           ELSE
10760                BEGIN
10770                KIN:=KIN-1;
10780                BCOST:=BCOST-IC[COMB[SIZE],1];
10790                END;
10800         END
10810         ELSE
10820           BEGIN
10830           KIN:=KIN-1;
10840           BCOST:=BCOST-IC[COMB[SIZE],1];
10850           END;
10860   END;
10870   END;
10880   IF SOPT=1 THEN POS:=1
10890             ELSE POS:=3;
10900   IF FLAG = 1 THEN
10910     begin
10920
10930       (* IF ANY OPTIMAL SOLUTIONS WERE FOUND, DETERMINE CORRESPONDING
10940          TRADEOFFS AND FILL TRADEOFF TABLE SLOTS. *)
10950
10960       FOR I:=1 TO 20 DO
10970         BEGIN
10980         IF SOLLIST[I] = 1 THEN
10990           BEGIN
11000           IF (I > MAXLEV) THEN MAXLEV:=MAXLEV+1;
11010           STAT:=0;
11020           TABSOL[POS,I]:=1;
11030           IF (SOPT=1) THEN TRADE[2,POS,I]:=CURSOLLEV[1,POS,I]-CURCOST
11040                       ELSE TRADE[2,POS,I]:=CURSOLLEV[4,POS,I]-TSHP;
11050           CASE TOPT OF
11060            1: TRADE[1,POS,I]:=CURSOLLEV[2,POS,I]-CURAMD;
11070            2: TRADE[1,POS,I]:=CURSOLLEV[3,POS,I]-CURDED;
```

```
11080             3: TRADE[1,POS,I]:=CURSOLLEV[4,POS,I]-CURSHP;
11090             4: TRADE[1,POS,I]:=CURSOLLEV[1,POS,I]-CURCOST
11100           END;
11110         END
11120         ELSE TABSOL[POS,I]:=0;
11130      END;
11140      END
11150     ELSE BEGIN
11160          STAT:=1;
11170          FOR I:=1 TO 20 DO
11180             TABSOL[POS,I]:=0;
11190          END;
11200   END;
11210
11220   FUNCTION DCONTIG(KIN,TEMP:INTEGER):BOOLEAN;
11230   {************************************************************
11240
11250         DCONTIG FUNCTION
11260
11270   THIS FUNCTION IS CALLED FROM THE ODIST ROUTINE AND PERFORMS
11280   THE FUNCTION FOR THE ADMENITY AND DETRACTOR SUBPROBLEMS.  IT
11290   IS SLIGHTLY DIFFERENT FROM THE CONTIG FUNCTION.
11300
11310   THIS FUNCTION DETERMINES WHETHER OR NOT A FULL SHAPE
11320   COULD BE COMPLETED USING THE LATEST  POSSIBLE CELL AND
11330   ALL THE CURRENT CELLS IN THE SHAPE.
11340
11350   ************************************************************}
11360
11370   VAR I,J,K,L,ROW,POS,FLAG,COL,AROW,DROW,DCOL,ACOL:INTEGER;
11380       DIST,R:REAL;
11390
11400   BEGIN
11410     POS:=ICPOINT[TEMP];
11420     ROW:=POS DIV NCOLS;
11430     IF (POS # (ROW*NCOLS)) THEN ROW:=ROW+1;
11440     COL:=POS - ((ROW-1)*NCOLS);
11450     CONTIG:=FALSE;
11460     FLAG:=0;
11470     I:=SIZE;
11480     WHILE (FLAG = 0) AND (I > KIN) DO
11490       BEGIN
11500       POS:=ICPOINT[COMB[I]];
11510       AROW:=POS DIV NCOLS;
11520       IF (POS # (AROW*NCOLS)) THEN AROW:=AROW+1;
11530       ACOL:=POS - ((AROW-1)*NCOLS);
11540       DROW:=AROW-ROW;
11550       DCOL:=ACOL-COL;
11560       L:=ABS(DROW) + ABS(DCOL) -1;
11570       IF L <= KIN THEN
11580           BEGIN
11590           CONTIG:=TRUE;
11600           FLAG:=1;
11610           END;
11620       I:=I-1;
11630     END;
11640   END;
11650
11660   PROCEDURE DRESETCELL(VAR KIN,TEMP:INTEGER;VAR BCOST:REAL);
11670   (* ************************************************************
11680
11690      DRESETCELL PROCEDURE
11700
11710   THIS PROCEDURE RESETS THE COMBINATION CELL COUNTER FOR
11720   THE ODIST (ADM AND DET SUBPROBLEMS) ROUTINE.
11730
11740   ************************************************************ *)
```

```
11750
11760   VAR FLAG:INTEGER;
11770
11780   BEGIN
11790     FLAG:=0;
11800     WHILE FLAG = 0 DO
11810       BEGIN
11820       KIN:=KIN+1;
11830       TEMP:=COMB[KIN];
11840       BCOST:=BCOST-IC[COMB[KIN],1];
11850       IF ((COMB[KIN]+1) < COMB[KIN+1]) OR
11860          (KIN = SIZE) THEN
11870          FLAG:=1
11880          ELSE
11890            COMB[KIN]:=0;
11900       END;
11910   END;
11920
11930   PROCEDURE DNEXTCELL(VAR KIN,TEMP:INTEGER;VAR BCOST:REAL);
11940   (* ************************************************************
11950                                    I
11960      DNEXTCELL PROCEDURE
11970
11980   THIS PROCEDURE DETERMINES THE NEXT CELL TO TRY IN THE
11990   CURRENT COMBINATION BEING BUILT IN THE ODIST (ADM AND DET
12000   SUBPROBLEMS) ROUTINE.
12010
12020   ************************************************************ *)
12030
12040   VAR FLAG:INTEGER;
12050
12060   BEGIN
12070       FLAG:=0;
12080       WHILE FLAG = 0 DO
12090         BEGIN
12100         TEMP:=TEMP+1;
12110         IF (TEMP < KIN) THEN TEMP:=KIN;
12120         IF KIN < SIZE THEN
12130           BEGIN
12140           IF (TEMP = COMB[KIN+1]) THEN
12150             BEGIN
12160             KIN:=KIN+1;
12170             BCOST:=BCOST-IC[COMB[KIN],1];
12180             COMB[KIN]:=0;
12190             END
12200             ELSE FLAG:=1;
12210           END
12220           ELSE FLAG:=1;
12230         END;
12240   END;
12250
12260   PROCEDURE ODIST(POS:INTEGER;VAR STAT:INTEGER;TOP1,SNUM:INTEGER);
12270   (* ************************************************************
12280
12290        ODIST PROCEDURE
12300
12310   THIS ROUTINE SOLVES THE AMENITY AND DETRACTOR SUBPROBLEMS.
12320   THE LOGIC IS SIMILAR TO THE COST SUBPROBLEM AND THEREFORE SIMILAR
12330   COMMENTS WILL BE LEFT OUT.  MAJOR DIFFERENCES ARE NOTED.
12340
12350   ************************************************************ *)
12360
12370   VAR BCOST,BSHAPE,FCOST,TCOST,FSHAPE:REAL;
12380       FIN,FLAG,COMP,I,J,K,L,TEMP,KIN,NCEX,TOTALT:INTEGER;
12390       TCAMD,TCDED,SHP,MIN,MAX:REAL;
```

```
12400       BOTTLE:REAL;
12410       SOLNUM,NALTSOL,SOLCNT,EFLAG,PFLAG,SOLFLAG:INTEGER;
12420
12430   BEGIN
12440     CASE TOPT OF
12450      1:BEGIN
12460        TCDED:=CURDED;
12470        TCAMD:=10000.0;
12480        END;
12490      2:BEGIN
12500        TCDED:=CURDED+0.5;
12510        TCAMD:=10000.0;
12520        END;
12530      3:BEGIN
12540        TCDED:=CURDED;
12550        TCAMD:=10000.0;
12560        END;
12570      4: BEGIN
12580         TCAMD:=CURAMD;
12590         TCDED:=0.0;
12600         END;
12610      5: BEGIN
12620         TCAMD:=CURAMD-0.5;
12630         TCDED:=0.0;
12640         END;
12650      6: BEGIN
12660         TCAMD:=CURAMD;
12670         TCDED:=0.0;
12680         END
12690     END;
12700     MARKCELLS(TCAMD,TCDED);
12710     IF NIC > 1 THEN
12720       BEGIN
12730       IF TOPT <= 3 THEN ORDERCELLS(2)
12740                    ELSE ORDERCELLS(3);
12750       END;
12760
12770     (* SET INITIAL BOTTLENECK VALUES, DEPENDING ON PROBLEM.  NOTE THAT
12780        KIN STARTS AT THE RIGHT SIDE OF THE COMBINATION. *)
12790
12800     IF TOPT <= 3 THEN BOTTLE:=10000.0
12810                  ELSE BOTTLE:=0.0;
12820     FCOST:=CURCOST;
12830     FIN:=0;
12840     TEMP:=0;
12850     NCEX:=0;
12860     KIN:=SIZE;
12870     BCOST:=0.0;
12880     STAT:=0;
12890     FSHAPE:=CURSHP;
12900     BSHAPE:=0.0;
12910     SOLNUM:=0;
12920     NALTSOL:=0;
12930     SOLFLAG:=0;
12940
12950     FOR I:=1 TO 50 DO
12960       COMB[I]:=0;
12970
12980     WHILE FIN=0 DO
12990       BEGIN
13000       COMP:=0;
13010       WHILE COMP=0 DO
13020         BEGIN
13030         NCEX:=NCEX+1;
13040
13050           (* RETRIEVE NEXT AVAILABLE CELL. CHECK IF BOTTLENECK VALUE SURPASS
```

```
ES
13060           CURRENT BOTTLENECK VALUE OF A FEASIBLE SOLUTION. *)
13070
13080         DNEXTCELL(KIN,TEMP,BCOST);
13090         IF ((IC[TEMP,2] <= BOTTLE) AND (TOPT <=3)) OR
13100            ((IC[TEMP,3] >= BOTTLE) AND (TOPT > 3)) THEN
13110         BEGIN
13120         IF TEMP <= NIC THEN
13130           BEGIN
13140           BCOST:=BCOST+IC[TEMP,1];
13150
13160             (* CHECK FOR COST FEASIBILITY. *)
13170
13180             IF ((TOPT = 1) AND (BCOST < FCOST)) OR
13190                ((TOPT = 4) AND (BCOST < FCOST)) OR
13200              (((TOPT=2) OR (TOPT=3) OR (TOPT=5) OR (TOPT=6))
13210                AND (BCOST <=FCOST)) THEN
13220               BEGIN
13230               IF KIN < SIZE THEN
13240                 BEGIN
13250
13260                 (* CHECK FOR CONTIGUITY FEASIBILITY. *)
13270
13280                 IF DCONTIG(KIN,TEMP) THEN
13290                   BEGIN
13300                     COMB[KIN]:=TEMP;
13310                     KIN:=KIN-1;
13320                     TEMP:=0;
13330                     IF KIN = 0 THEN COMP:=1;
13340                   END
13350                   ELSE BCOST:=BCOST-IC[TEMP,1];
13360                 END
13370                 ELSE BEGIN
13380                      COMB[KIN]:=TEMP;
13390                      KIN:=KIN-1;
13400                      TEMP:=0;
13410                      END;
13420               END
13430               ELSE BEGIN
13440                    BCOST:=BCOST-IC[TEMP,1];
13450                    IF TEMP > NIC THEN
13460                      BEGIN
13470                      FIN:=1;
13480                      COMP:=1;
13490                      END;
13500                    END;
13510             END
13520             ELSE
13530               BEGIN
13540               FIN:=1;
13550               COMP:=1;
13560               END;
13570         END
13580         ELSE BEGIN
13590              FIN:=1;
13600              COMP:=1;
13610              END;
13620         END;
13630         IF FIN=0 THEN
13640           BEGIN
13650
13660             (* AT THIS POINT, WE HAVE A COMPLETE SOLUTION.  CHECK FOR OVERAL
L
13670                CONTIGUITY AND SHAPE FEASIBILITY.  IF THIS IS AN OPTIMAL
13680                SOLUTION, CHECK FOR DOMINATION WITH OTHER OPTIMAL SOLUTIONS
13690                USING SAME METHOD AS IN COST SUBPROBLEM. *)
  700
```

```
13710           IF WHSHAPE THEN
13720           BEGIN
13730           COMPSHP(BSHAPE);
13740           IF ((TOPT < 3) AND (BSHAPE <= CURSHP)) OR
13750              ((TOPT = 3) AND (BSHAPE < CURSHP)) OR
13760              (((TOPT = 4)OR(TOPT = 5))AND (ESHAPE <= CURSHP))  OR
13770              ((TOPT = 6) AND (BSHAPE < CURSHP)) THEN
13780             BEGIN
13790             SOLFLAG:=1;
13800             SHP:=BSHAPE;
13810             ECOST:=BCOST;
13820             FOR I:=1 TO SIZE DO
13830               FANS[I]:=ICPOINT[COMB[I]];
13840             FADMDIST(MAX);
13850             FDETDIST(MIN);
13860             IF (TOPT <= 3) THEN BOTTLE:=MAX
13870                            ELSE BOTTLE:=MIN;
13880             PFLAG:=0;
13890             IF (SOLNUM > 0) THEN BEGIN
13900               NALTSOL:=NALTSOL+1;
13910               TOTALT:=NALTSOL-1;
13920               SOLCNT:=0;
13930               EFLAG:=3;
13940               WHILE ((SOLCNT < TOTALT) AND (EFLAG > 1)) DO
13950                 BEGIN
13960                 SOLCNT:=SOLCNT+1;
13970                 EFLAG:=0;
13980                 EFFCHK(ECOST,MAX,MIN,SHP,SOLCNT,SNUM,EFLAG);
13990                 IF (EFLAG > 1) THEN PFLAG:=1
14000                                ELSE PFLAG:=0;
14010                 IF (EFLAG = 2) THEN SOLLIST[SOLCNT]:=0;
14020                 IF ((EFLAG=2) OR (EFLAG=1)) THEN
14030                                  NALTSOL:=NALTSOL-1;
14040                 END;
14050             END
14060             ELSE BEGIN
14070                  FOR I:=1 TO 20 DO
14080                    SOLLIST[I]:=0;
14090                  NALTSOL:=1;
14100                  SOLNUM:=0;
14110                  PFLAG:=1;
14120                  END;
14130           IF PFLAG = 1 THEN BEGIN
14140                     SOLNUM:=SOLNUM+1;
14150                     STORESOL(ECOST,MAX,MIN,SHP,SNUM,SOLNUM);
14160                     SOLLIST[SOLNUM]:=1;
14170                     END;
14180           COMPRESS(SNUM,SOLNUM);
14190           DRESETCELL(KIN,TEMP,BCOST);
14200           END
14210           ELSE DRESETCELL(KIN,TEMP,BCOST);
14220         END
14230         ELSE DRESETCELL(KIN,TEMP,BCOST);
14240         END;
14250       END;
14260       IF (SOLFLAG = 1) THEN
14270         BEGIN
14280
14290         (* IF ANY OPTIMAL SOLUTIONS WERE FOUND, PLACE THEM IN THE
14300            APPROPRIATE TRADEOFF TABLE SLOTS. *)
14310
14320         FOR I:=1 TO 20 DO
14330           BEGIN
14340           IF SOLLIST[I] = 1 THEN
14350             BEGIN
```

```
14360             STAT:=0;
 1370             IF (I > MAXLEV) THEN MAXLEV:=MAXLEV+1;
14380             TABSOL[POS,I]:=1;
14390             IF TOPT <= 3 THEN TRADE[2,POS,I]:=CURSOLLEV[2,POS,I]-CURAMD
 1400                          ELSE TRADE[2,POS,I]:=CURSOLLEV[3,POS,I]-CURDED;
14410             CASE TOPT OF
14420               1: TRADE[1,POS,I]:=CURSOLLEV[1,POS,I]-CURCOST;
 1430               2: TRADE[1,POS,I]:=CURSOLLEV[3,POS,I]-CURDED;
14440               3: TRADE[1,POS,I]:=CURSOLLEV[4,POS,I]-CURSHP;
14450               4: TRADE[1,POS,I]:=CURSOLLEV[1,POS,I]-CURCOST;
 1460               5: TRADE[1,POS,I]:=CURSOLLEV[2,POS,I]-CURAMD;
14470               6: TRADE[1,POS,I]:=CURSOLLEV[4,POS,I]-CURSHP
14480             END;
 1490             END
14500             ELSE TABSOL[POS,I]:=0;
14510           END;
 1520         END
14530         ELSE BEGIN
14540              STAT:=1;
 1550              FOR I:=1 TO 20 DO
14560                TABSOL[POS,I]:=0;
14570              END;
 1580   END;
14590   PROCEDURE CSHAPE(VAR STAT:INTEGER;
14600                    TOPT,SNUM:INTEGER);
 1610   (* ************************************************
14620

      14630        CSHAPE PROCEDURE
      14640
      14650    THIS PROC SOLVES THE SHAPE SUBPORBLEM.  IT ITERATIVELY
      14660    USES THE COST SUBPROBLEM UNTIL A FEASIBLE SOLUTION IS
      14670    FOUND.
      14680
      14690    ************************************************** *)
      14700
      14710    VAR NUM,FLG:INTEGER;
      14720        TSHP:REAL;
      14730
      14740    BEGIN
      14750      FLG:=0;
      14760      STAT:=1;
      14770      TSHP:=CURSHP;
      14780      WHILE (FLG=0) AND (STAT = 1) DO
      14790        BEGIN
      14800        CURSHP:=CURSHP+1.0;
      14810        IF CURSHP < WOSHP THEN
      14820               COST(TSHP,STAT,TOPT,SNUM,2)
      14830               ELSE FLG:=1;
      14840        END;
      14850    END;
      14860
      14870    PROCEDURE INITPRBSETUP(VAR STAT:INTEGER);
      14880    {*****************************************************
      14890
      14900         INITPRBSETUP PROCEDURE
      14910
      14920    THIS PROC SETS UP FOR THE INITIAL SUBPROBLEM.
      14930
      14940    *****************************************************}
      14950
      14960    VAR DUM:REAL;
      14970
      14980    BEGIN
      14990      CURCOST:=100000.0;
      15000      CURAMD:=INTAMD+0.5;
```

```
15010     CURDED:=INTDED;
15020     CURSHP:=INTSHP;
15030     COST(DUM,STAT,1,1,1);
15040     CURCOST:=CURSOLLEV[1,1,1];
15050     CURAMD:=CURSOLLEV[2,1,1];
15060     CURDED:=CURSOLLEV[3,1,1];
15070     CURSHP:=CURSOLLEV[4,1,1];
15080   END;
15090
15100   PROCEDURE SETUP(VAR K:INTEGER);
15110   {************************************************************
15120
15130         SETUP PROCEDURE
15140
15150   THIS PROC INITIATES THE SUBPROBLEM SOLUTION PROCESS.
15160
15170   ************************************************************}
15180
15190   VAR STAT,I,J,L:INTEGER;
15200       DUM:REAL;
15210
15220   BEGIN
15230     BLINE(3);
15240     TAB1:=0;
15250     TAB2:=0;
15260     TAB3:=0;
15270     FOR J:=1 TO 4 DO
15280      BEGIN
15290      FOR L:=1 TO 20 DO
15300       BEGIN
15310         FOR I:=1 TO 3 DO
15320           CURSOLLEV[J,I,L]:=0.0;
15330       END;
15340     END;
15350     FOR J:=1 TO 20 DO
15360       BEGIN
15370       FOR I:=1 TO 3 DO
15380         BEGIN
15390         FOR L:=1 TO 50 DO
15400           CURSOLCELLS[L,I,J]:=0;
15410         END;
15420       END;
15430     MAXLEV:=1;
15440    case k of
15450     1:begin
15460       ODIST(1,TAB1,1,1);
15470       QDIST(2,TAB2,4,2);
15480       CSHAPE(TAB3,3,3);
15490       end;
15500     2:begin
15510       COST(DUM,tab1,1,1,1);
15520       ODIST(2,TAB2,5,2);
15530       CSHAPE(TAB3,1,3);
15540       end;
15550     3:begin
15560       COST(DUM,tab1,2,1,1);
15570       ODIST(2,TAB2,2,2);
15580       CSHAPE(TAB3,2,3);
15590       end;
15600     4:begin
15610       COST(DUM,tab1,3,1,1);
15620       ODIST(2,TAB2,3,2);
15630       ODIST(3,TAB3,6,3);
15640       end
15650    end;
15660    END;
```

```
15670
15680   PROCEDURE CURSOL;
15690   {************************************************************
15700
15710       CURSOL PROCEDURE
15720
15730   THIS PROC PRESENTS THE CURRENT SOLUTION TO THE USER AND
15740   DETERMINES THE OBJECTIVE TO IMPROVE.  WITH THIS, THE
15750   CORRESPONDING 3 SUBPROBLEMS ARE SOLVED.
15760
15770   ************************************************************}
15780
15790
15800   BEGIN
15810     BLINE(3);
15820     WRITELN(TTY,'YOUR CURRENT SOLUTION IS:');
15830     BLINE(1);
15840     WRITELN(TTY,'                    ************************');
15850     WRITELN(TTY,'                    *                      *');
15860     WRITELN(TTY,'(1) COST            *   ',CURCOST:6:2,'             *');
15870     WRITELN(TTY,'(2) AMEN DIST       *   ',CURAMD:6:2,'             *');
15880     WRITELN(TTY,'(3) DET DIST        *   ',CURDED:6:2,'             *');
15890     WRITELN(TTY,'(4) SHAPE           *   ',CURSHP:6:2,'             *');
15900     WRITELN(TTY,'                    *                      *');
15910     WRITELN(TTY,'                    ************************');
15920
15930     BLINE(2);
15940     WRITELN(TTY,'SELECT OBJECTIVE TO IMPROVE (1,2,3,4):');
15950     RDINT(SELECT);
15960     CASE SELECT OF
15970       1:SETUP(SELECT);
15980       2:SETUP(SELECT);
15990       3:SETUP(SELECT);
16000       4:SETUP(SELECT)
16010     END;
16020   END;
16030
16040   PROCEDURE INITLEV;
16050   {************************************************************
16060
16070       INITLEV PROCEDURE
16080
16090   THIS PROC INITIALIZES THE PROBLEM BY READING IN THE
16100   INITIAL OBJECTIVE LEVELS.
16110
16120   ************************************************************}
16130   VAR N,FLAG,STAT:INTEGER;
16140
16150   BEGIN
16160     FLAG:=0;
16170     WHILE FLAG = 0 DO
16180     BEGIN
16190     BLINE(1);
16200     WRITELN(TTY,'INITIALIZE OBJECTIVE LEVELS FOR PROBLEM:');
16210     INTCOST:=0.0;
16220     WRITELN(TTY,'ENTER INITIAL AMENITY DISTANCE LEVEL:');
16230     RDINT(N);
16240     INTAMD:=N;
16250     WRITELN(TTY,'ENTER INITIAL DETRACTOR DISTANCE LEVEL:');
16260     RDINT(N);
16270     INTDED:=N;
16280     WRITELN(TTY,'ENTER INITIAL SHAPE LEVEL:');
16290     RDINT(N);
16300     INTSHP:=N;
16310
16320     STAT:=0;
```

```
16330     INITPRESETUP(STAT);
16340     IF (STAT > 0) THEN
16350       BEGIN
16360       BLINE(2);
16370
16380       WRITELN(TTY,'NO EFFICIENT SOLUTIONS CAN BE FOUND USING',
16390        ' INITIAL LEVELS SPECIFIED.');
16400       WRITELN(TTY,'RE-ENTER INITIIAL LEVELS.');
16410     END
16420     ELSE FLAG:=1;
16430   END;
16440   END;
16450
16460   PROCEDURE STORDISK(SOLNUM:INTEGER);
16470   {*********************************************************
16480
16490        STORDISK PROCEDURE
16500
16510   THIS PROC STORES A TABLE SOLUTION ON DISK FOR FUTURE
16520   USE BY THE USER.
16530
16540   *********************************************************}
16550
16560   VAR
16570       I,J,NUM:INTEGER;
16580       diskcell: array [1..50] of INTEGER;
16590       disklev: array [1..4] of REAL;
16600       ROW,COL:INTEGER;
16610
16620   BEGIN
16630     ROW:=SOLNUM DIV 3;
16640     IF (ROW*3 # SOLNUM) THEN ROW:=ROW+1;
16650     COL:=SOLNUM-((ROW-1)*3);
16660     IF TABSOL[COL,ROW]=1 THEN
16670     BEGIN
16680     RESET(SOLFILE);
16690     REWRITE(WRKFILE);
16700     IF DISKSOLN > 0 THEN
16710       BEGIN
16720       FOR I:=1 TO DISKSOLN DO
16730         BEGIN
16740         READ(SOLFILE,NUM);
16750         WRITE(WRKFILE,NUM);
16760         FOR J:=1 TO 4 DO
16770           BEGIN
16780           READ(SOLFILE,DISKLEV[J]);
16790           WRITE(WRKFILE,DISKLEV[J]);
16800           END;
16810         FOR J:=1 TO 50 DO
16820           BEGIN
16830           READ(SOLFILE,DISKCELL[J]);
16840           WRITE(WRKFILE,DISKCELL[J]);
16850           END;
16860         READLN(SOLFILE);
16870         WRITELN(WRKFILE);
16880         END;
16890       END;
16900     DISKLEV[1]:=CURSOLLEV[1,COL,ROW];
16910     DISKLEV[2]:=CURSOLLEV[2,COL,ROW];
16920     DISKLEV[3]:=CURSOLLEV[3,COL,ROW];
16930     DISKLEV[4]:=CURSOLLEV[4,COL,ROW];
16940     FOR I:=1 TO SIZE DO
16950       DISKCELL[I]:=CURSOLCELL[I,COL,ROW];
16960     DISKSOLN:=  DISKSOLN+1;
16970     WRITE(WRKFILE,DISKSOLN);
16980     FOR I:=1 TO 4 DO
```

```
16990       WRITE(WRKFILE,DISKLEV[I]);
17000     FOR I:=1 TO 50 DO
17010       WRITE(WRKFILE,DISKCELL[I]);
17020     WRITELN(WRKFILE);
17030     RESET(WRKFILE);
17040     REWRITE(SOLFILE);
17050     FOR I:=1 TO DISKSOLN DO
17060       BEGIN
17070       READ(WRKFILE,NUM);
17080       WRITE(SOLFILE,NUM);
17090       FOR J:=1 TO 4 DO
17100         BEGIN
17110         READ(WRKFILE,DISKLEV[J]);
17120         WRITE(SOLFILE,DISKLEV[J]);
17130         END;
17140       FOR J:=1 TO 50 DO
17150         BEGIN
17160         READ(WRKFILE,DISKCELL[J]);
17170         WRITE(SOLFILE,DISKCELL[J]);
17180         END;
17190       READLN(WRKFILE);
17200       WRITELN(SOLFILE);
17210       END;
17220     bline(1);
17230     WRITELN(TTY,'TABLE SOLUTION',SOLNUM:4,' STORED AS',
17240                 ' NUMBER',DISKSOLN:4);
17250     END
17260     ELSE WRITELN(TTY,'INVALID NUMBER SPECIFIED.');
17270   END;
17280
17290   PROCEDURE RETRDISK(SOLNUM:INTEGER);
17300   {*********************************************************
17310
17320       RETRDISK PROCEDURE
17330
17340   THIS PROC RETRIEVES A SOLUTION STORED OFFLINE BY STORESOL.
17350
17360   *********************************************************}
17370
17380   VAR DISKLEV: ARRAY [1..4] OF REAL;
17390       DISKCELL: ARRAY [1..50] OF INTEGER;
17400       i,NUM,ROW,COL:INTEGER;
17410
17420   BEGIN
17430     if solnum <= disksoln then
17440     BEGIN
17450     RESET(SOLFILE);
17460     IF SOLNUM > 1 THEN
17470       BEGIN
17480       FOR I:=1 TO (SOLNUM-1) DO
17490        READLN(SOLFILE,NUM);
17500       END;
17510       READ(SOLFILE,NUM);
17520       FOR I:=1 TO 4 DO
17530         READ(SOLFILE,DISKLEV[I]);
17540       FOR I:=1 TO 50 DO
17550         READ(SOLFILE,DISKCELL[I]);
17560     CURSOLLEV[1,1,1]:=DISKLEV[1];
17570     CURSOLLEV[2,1,1]:=DISKLEV[2];
17580     CURSOLLEV[3,1,1]:=DISKLEV[3];
17590     CURSOLLEV[4,1,1]:=DISKLEV[4];
17600     FOR I:=1 TO SIZE DO
17610       CURSOLCELL[I,1,1]:=DISKCELL[I];
17620     TABSOL[1,1]:=1;
17630     bline(1);
17640     writeln(tty,'solution',solnum:4,' retrieved and');
```

```
17650     writeln(tty,'held as table solution number 1.');
17660     end
17670     else writeln(tty,'invalid solution number requested.');
17680   END;
17690
17700   PROCEDURE RETSOL(var k,flag:INTEGER);
17710   {*********************************************************
17720
17730        RETSOL PROCEDURE
17740
17750   THIS PROC RETRIEVES A TRADEOFF SOLUTION AND MAKES IT
17760   THE CURRENT SOLUTION TO BE USED BY THE USER IN THE NEXT
17770   ITERATION.
17780
17790   *********************************************************}
17800
17810   VAR ROW,COL:INTEGER;
17820
17830   BEGIN
17840     ROW:=K DIV 3;
17850     IF (ROW*3 # K) THEN ROW:=ROW+1;
17860     COL:=K-((ROW-1)*3);
17870     IF TABSOL[COL,ROW] = 1 THEN
17880     begin
17890    CURCOST:=CURSOLLEV[1,COL,ROW];
17900    CURAMD:=CURSOLLEV[2,COL,ROW];
17910    CURDED:=CURSOLLEV[3,COL,ROW];
17920    CURSHP:=CURSOLLEV[4,COL,ROW];
17930     end
17940     else begin
17950       writeln(tty,'invalid solution number requested.');
17960       flag:=0;
17970       end;
17980   END;
17990   PROCEDURE VWTSOL(K:INTEGER);
18000   (*
18010
18020      VWTSOL PROCEDURE
18030
18040   THIS PROC DISPLAYS THE LEVELS OF A SOLUTION REPRESENTED
18050   IN THE TRADEOFF TABLE.
18060
18070   ********************************************* *)
18080
18090   VAR ROW,COL:INTEGER;
18100
18110   BEGIN
18120       ROW:=K DIV 3;
18130       IF (ROW*3 # K) THEN ROW:=ROW+1;
18140       COL:=K-((ROW-1)*3);
18150     IF TABSOL[COL,ROW] = 1 THEN
18160       BEGIN
18170       ROW:=K DIV 3;
18180       IF (ROW*3 # K) THEN ROW:=ROW+1;
18190       COL:=K-((ROW-1)*3);
18200       BLINE(2);
18210       WRITELN(TTY,'           TABLE SOLUTION NUMBER',K:4);
18220       WRITELN(TTY);
18230       WRITELN(TTY,'    ','COST     ',CURSOLLEV[1,COL,ROW]:8:3);
18240       WRITELN(TTY,'    ','AMN. DIST',CURSOLLEV[2,COL,ROW]:8:3);
18250       WRITELN(TTY,'    ','DET. DIST',CURSOLLEV[3,COL,ROW]:8:3);
18260       WRITELN(TTY,'    ','SHAPE    ',CURSOLLEV[4,COL,ROW]:8:3);
18270       BLINE(1);
18280       END
18290       ELSE
18300         WRITELN(TTY,'INVALID SOLUTION NUMBER REQUESTED.');
```

```
18310   END;
18320
18330   PROCEDURE VWSSOL(K:INTEGER);
18340   (* ***********************************************
18350
18360      VWSSOL PROCEDURE
18370
18380   THIS PROC IS USED TO DISPLAY THE LEVELS OF A SOLUTION
18390   STORED ON DISK.
18400
18410   *********************************************** *)
18420
18430   VAR
18440       DISKLEV: ARRAY [1..4] OF REAL;
18450       I,NUM:INTEGER;
18460       DISKCELL: ARRAY [1..50] OF INTEGER;
18470
18480   BEGIN
18490     IF (K <= DISKSOLN) THEN
18500     BEGIN
18510     RESET(SOLFILE);
18520     FOR I:=1 TO (K-1) DO
18530       READLN(SOLFILE,NUM);
18540     READ(SOLFILE,NUM);
18550     FOR I:=1 TO 4 DO
18560       READ(SOLFILE,DISKLEV[I]);
18570     BLINE(2);
18580       WRITELN(TTY,'           DISK SOLUTION NUMBER',K:4);
18590       WRITELN(TTY);
18600       WRITELN(TTY,'    ','COST     ',DISKLEV[1]:8:3);
18610       WRITELN(TTY,'    ','AMN. DIST',DISKLEV[2]:8:3);
18620       WRITELN(TTY,'    ','DET. DIST',DISKLEV[3]:8:3);
18630       WRITELN(TTY,'    ','SHAPE    ',DISKLEV[4]:8:3);
18640       BLINE(1);
18650       END
18660     ELSE
18670       WRITELN(TTY,'INVALID SOLUTION NUMBER REQUESTED.');
18680   END;
18690
18700   PROCEDURE VIEWTCELL(K:INTEGER);
18710   (* ***********************************************
18720
18730       VIEWTCELL PROCEDURE
18740
18750   THIS PROC DISPLAYS THE CELLS OF A SOLUTION STORED
18760   IN THE TRADEOFF TABLE.
18770
18780   *********************************************** *)
18790
18800   VAR M,J,I,L,ROW,COL,N:INTEGER;
18810
18820   BEGIN
18830     ROW:= K DIV 3;
18840     IF (ROW*3 # K) THEN ROW:=ROW+1;
18850     COL:=K-((ROW-1)*3);
18860     IF TABSOL[COL,ROW] = 1 THEN
18870     BEGIN
18880       BLINE(3);
18890       WRITELN(TTY,'TABLE SOLUTION NUMBER',K:4);
18900       N:=SIZE DIV 8;
18910       IF SIZE # (N*8) THEN N:=N+1;
18920       M:=0;
18930       L:=0;
18940       FOR I:=1 TO N DO
18950         BEGIN
18960         BLINE(1);
```

```
18970          M:=L+1;
18980          L:=L+8;
18990          IF (L > SIZE) THEN L:=SIZE;
19000          FOR J:=M TO L DO
19010            WRITE(TTY,CURSOLCELL[J,COL,ROW]:8,'     ');
19020          BLINE(1);
19030          END;
19040        END
19050        ELSE
19060          WRITELN(TTY,'INVALID SOLUTION NUMBER REQUESTED.');
19070    END;
19080    PROCEDURE FINRUN;
19090    BEGIN
19100    END;
19110
19120    PROCEDURE VIEWSSOL(K:INTEGER);
19130
19140    (* ************************************************
19150
19160       PROCEDURE VIEWSSOL
19170
19180    THIS PROC DISPLAYS THE CELLS OF A SOLUTION STORED ON
19190    DISK.
19200
19210    ************************************************** *)
19220
19230    VAR
19240        DISKLEV: ARRAY[1..4] OF REAL;
19250        DISKCELL: ARRAY[1..50] OF INTEGER;
19260        N,I,NUM,J,L,M:INTEGER;
19270
19280    BEGIN
19290      IF K <= DISKSOLN THEN
19300      BEGIN
19310        RESET(SOLFILE);
19320        FOR I:=1 TO (K-1) DO
19330          READLN(SOLFILE,NUM);
19340        READ(SOLFILE,NUM);
19350        FOR I:=1 TO 4 DO
19360          READ(SOLFILE,DISKLEV[I]);
19370        FOR I:=1 TO 50 DO
19380          READ(SOLFILE,DISKCELL[I]);
19390        BLINE(3);
19400        WRITELN(TTY,'DISK SOLUTION NUMBER',K:4);
19410        N:=SIZE DIV 8;
19420        IF SIZE # (N*8) THEN N:=N+1;
19430        M:=0;
19440        L:=0;
19450        FOR I:=1 TO N DO
19460          BEGIN
19470          BLINE(1);
19480          M:=L+1;
19490          L:=L+8;
19500          IF (L > SIZE) THEN L:=SIZE;
19510          FOR J:=M TO L DO
19520            WRITE(TTY,DISKCELL[J]:8,'     ');
19530          BLINE(1);
19540          END;
19550        END
19560        ELSE
19570          WRITELN(TTY,'INVALID SOLUTION NUMBER REQUESTED.');
19580    END;
19590
19600
19610
19620    PROCEDURE PRTSOL;
19630    (* ************************************************
```

```
.9640
19650       PRTSOL PROCEDURE
19660
19670   THIS PROC WRITES THE CURRENT SOLUTION FILE OUT TO THE
19680   PRTFILE TO BE PRINTED LATER.
19690
.9700   ********************************************************* *)
19710
19720   VAR I,K,J,NUM:INTEGER;
.9730       DISKLEV:ARRAY [1..4] OF REAL;
19740       DISKCELL: ARRAY [1..50] OF INTEGER;
19750       M,L,N:INTEGER;
.9760
19770   BEGIN
19780     RESET(SOLFILE);
.9790     REWRITE(PRTFILE);
19800
19810     FOR I:=1 TO 3 DO
.9820       WRITELN(PRTFILE);
19830   WRITELN(PRTFILE,'           MULTI-OBJECTIVE LAND ALLOCATION',
19840           ' MODEL');
.9850   WRITELN(PRTFILE,'             CURRENT SOLUTION FILE');
19860
19870     FOR I:=1 TO DISKSOLN DO
.9880       BEGIN
19890       WRITELN(PRTFILE);
19900       WRITELN(PRTFILE);
.9910       WRITELN(PRTFILE,'                *** SOLUTION NUMBER',I:4,
19920             ' ***');
19930       WRITELN(PRTFILE);
.9940       READ(SOLFILE,NUM);
19950       FOR J:=1 TO 4 DO
19960         READ(SOLFILE,DISKLEV[J]);
.9970       WRITELN(PRTFILE,'                    COST: ',DISKLEV[1]:8:3);
19980       WRITELN(PRTFILE,'     AMENITY DISTANCE: ',DISKLEV[2]:8:3);
19990       WRITELN(PRTFILE,'   DETRACTOR DISTANCE: ',DISKLEV[3]:8:3);
.0000       WRITELN(PRTFILE,'                 SHAPE: ',DISKLEV[4]:8:3);
20010       WRITELN(PRTFILE);
20020       FOR J:=1 TO 50 DO
 0030        READ(SOLFILE,DISKCELL[J]);
20040       N:=SIZE DIV 20;
20050       IF SIZE # (N*20) THEN N:=N+1;
 0060       M:=0;
20070       L:=0;
20080       WRITELN(PRTFILE,'        SOLUTION CELLS:');
 0090       WRITELN(PRTFILE);
20100       FOR J:=1 TO N DO
20110         BEGIN
 0120         M:=L+1;
20130         L:=L+20;
20140         IF (L>SIZE) THEN L:=SIZE;
 0150         FOR K:=M TO L DO
20160           WRITE(PRTFILE,'       ',DISKCELL[K]:8);
20170         WRITELN(PRTFILE);
 0180         END;
20190       END;
20200   END;
 0210
20220   PROCEDURE CURTAB;
20230   {************************************************************
 0240
20250       CURTAB PROCEDURE
20260
 0270   THIS PROC PRESENTS THE CURRENT TRADEOFF TABLE AND PRESENTS
20280   SEVERAL OPTIONS TO THE USER.
```

```
20290
 0300   ************************************************************)
20310
20320
 0330   VAR
20340     SOLNUM,I,J,FLAG:integer;
20350      CNAME,ANAME,DNAME,SNAME:PACKED ARRAY [1..8] OF CHAR;
 0360     TABNUM: PACKED ARRAY [1..60] OF INTEGER;
20370     TABSTART,TABFIN:INTEGER;
20380
 0390   BEGIN
20400     FOR I:=1 TO 60 DO
20410       TABNUM[I]:=I;
 0420     TABSTART:=0;
20430     TABFIN:=0;
20440     CNAME:='COST    ';
 0450     ANAME:='AMN DIST';
20460     DNAME:='DET DIST';
20470     SNAME:='SHAPE   ';
 0480     BLINE(2);
20490     WRITELN(TTY,'USING THE CURRENT SOLUTION AND OBJECTIVE  ',
20500               SELECT:1);
 0510     WRITELN(TTY,'THE TRADEOFF TABLE IS:');
20520     BLINE(1);
20530   FOR I:=1 TO MAXLEV DO
 0540   BEGIN
20550   IF (I=1) THEN
20560   WRITELN(TTY,'                                  OBJECTIVES TO BECOME WORSE');
 0570   BLINE(1);
20580   WRITE(TTY,'                                    ');
20590   IF SELECT # 1 THEN WRITE(TTY,CNAME:8,'   ');
 0600   IF SELECT # 2 THEN WRITE(TTY,ANAME:8,'   ');
20610   IF SELECT # 3 THEN WRITE(TTY,DNAME:8,'   ');
20620   IF SELECT # 4 THEN WRITE(TTY,SNAME:8,'   ');
 0630   WRITELN(TTY);
20640   WRITELN(TTY,'*************************************************************
*');
 0650   WRITELN(TTY,'OBJ TO ');
20660   WRITE(TTY,'BE IMPROVED:');
20670   IF SELECT = 1 THEN WRITE(TTY,'     ',CNAME:8);
 0680   IF SELECT = 2 THEN WRITE(TTY,'     ',ANAME:8);
20690   IF SELECT = 3 THEN WRITE(TTY,'     ',DNAME:8);
20700   IF SELECT = 4 THEN WRITE(TTY,'     ',SNAME:8);
 0710   IF I=1 THEN BEGIN
20720   IF TAB1 = 0 THEN WRITE(TTY,'    ',TRADE[1,1,I]:8:3)
20730               ELSE WRITE(TTY,'    ','*****');
 0740   IF TAB2 = 0 THEN WRITE(TTY,'    ',TRADE[1,2,I]:8:3)
20750               ELSE WRITE(TTY,'    ','*****');
20760   IF TAB3 = 0 THEN WRITE(TTY,'    ',TRADE[1,3,I]:8:3)
 0770               ELSE WRITE(TTY,'    ','*****');
20780   WRITELN(TTY);
20790   WRITELN(TTY,'*************************************************************
*');
20800   WRITE(TTY,'                              ');
20810   IF TAB1 = 0 THEN WRITE(TTY,'    ',TRADE[2,1,I]:8:3)
 0820               ELSE WRITE(TTY,'    ','*****');
20830   IF TAB2 = 0 THEN WRITE(TTY,'    ',TRADE[2,2,I]:8:3)
20840               ELSE WRITE(TTY,'    ','*****');
 0850   IF TAB3 = 0 THEN WRITE(TTY,'    ',TRADE[2,3,I]:8:3)
20860               ELSE WRITE(TTY,'    ','*****');
20870   END
 0880   ELSE BEGIN
20890   IF TABSOL[1,I]=1 THEN WRITE(TTY,'     ',TRADE[1,1,I]:8:3)
20900                    ELSE WRITE(TTY,'     NONE');
 0910   IF TABSOL[2,I]=1 THEN WRITE(TTY,'    ',TRADE[1,2,I]:8:3)
20920                    ELSE WRITE(TTY,'      NONE');
```

```
20930   IF TABSOL[3,I]=1 THEN WRITE(TTY,'    ',TRADE[1,3,1]:8:3) '
2 J40                    ELSE WRITE(TTY,'      NONE');
20950   WRITELN(TTY);
20960   WRITELN(TTY,'*************************************************************
* );
20970   WRITE(TTY,'                                ');
20980   IF TABSOL[1,I]=1 THEN WRITE(TTY,'    ',TRADE[2,1,I]:8:3)
2 J90                    ELSE WRITE(TTY,'     NONE');
21000   IF TABSOL[2,I]=1 THEN WRITE(TTY,'    ',TRADE[2,2,I]:8:3)
21010                    ELSE WRITE(TTY,'      NONE');
2 J20   IF TABSOL[3,I]=1 THEN WRITE(TTY,'    ',TRADE[2,3,I]:8:3)
21030                    ELSE WRITE(TTY,'      NONE');
21040   WRITELN(TTY);
2 J50   END;
21060   WRITELN(TTY);
21070   BLINE(1);
2 J80   WRITELN(TTY,'*************************************************************
*');
2109C   TABSTART:=TABFIN+1;
2 00    TABFIN:=TABFIN+3;
21110   BLINE(1);
21120   WRITE(TTY,'                          '         ');
2 30    FOR J:=TABSTART TO TABFIN DO
21140     WRITE(TTY,'      ','(',TABNUM[J]:2,')');
21150   WRITELN(TTY);
2 60    BLINE(2);
21170   END;
21180   FLAG:=0;
2 90    WHILE FLAG = 0 DO
21200   BEGIN
21210     WRITELN(TTY,'YOU HAVE THE FOLLOWING OPTIONS:');
2 20      WRITELN(TTY,'     1-USE ONE OF THE SOLUTIONS INDICATED BY THE ');
21230     WRITELN(TTY,'       TRADEOFF TABLE,');
21240     WRITELN(TTY,'     2-RE-INITIALIZE THE PROBLEM AND RESTART,');
2 50      WRITELN(TTY,'     3-STORE ONE OF THE SOLUTIONS IN THE TABLE,');
21260     WRITELN(TTY,'     4-RETRIEVE A SOLUTION PREVIOUSLY STORED,');
21270     WRITELN(TTY,'     5-VIEW ONE OF THE TABLE SOLUTIONS,');
2 80      WRITELN(TTY,'     6-VIEW A STORED SOLUTION,');
21290     WRITELN(TTY,'     7-VIEW THE CELLS OF A TABLE SOLUTION,');
21300     WRITELN(TTY,'     8-VIEW THE CELLS OF A STORED SOLUTION,');
2 10      WRITELN(TTY,'     9-STOP PROCESS.');
21320     WRITELN(TTY,'    10-PRINT CURRENT SOLUTION FILE.');
21330     WRITELN(TTY,'ENTER AN OPTION NUMBER:');
2 40      RDINT(OPTION);
21350
21360     CASE OPTION OF
2 70      1: BEGIN
21380          flag:=1;
21390          WRITELN(TTY,'ENTER TABLE SOLUTION NUMBER TO BE USED:');
2 20           RDINT(SOLNUM);
21410          RETSOL(SOLNUM,flag);
21420        END;
21430
             2: FLAG:=1;
21440        3: BEGIN
21450             WRITELN(TTY,'ENTER TABLE SOLUTION NUMBER TO BE STORED:');
21460             RDINT(SOLNUM);
21470             STORDISK(SOLNUM);
21480           END;
21490        4:  BEGIN
21500              WRITELN(TTY,'ENTER STORED SOLUTION NUMBER TO BE RETRIEVED:');
21510              RDINT(SOLNUM);
21520              RETRDISK(SOLNUM);
21530            END;
21540        5:  BEGIN
21550              WRITELN(TTY,'ENTER TABLE SOLUTION NUMBER TO BE VIEWED:');
```

```
21560            RDINT(SOLNUM);
21570            VWTSOL(SOLNUM)
21580          END;
21590      6:  BEGIN
21600            WRITELN(TTY,'ENTER STORED SOLUTION NUMBER TO BE VIEWED:');
21610            RDINT(SOLNUM);
21620            VWSSOL(SOLNUM);
21630            END;
21640      7:  BEGIN
21650            WRITELN(TTY,'ENTER TABLE SOLUTION NUMBER TO HAVE CELLS');
21660            WRITELN(TTY,'DISPLAYED:');
21670            RDINT(SOLNUM);
21680            VIEWTCELL(SOLNUM);
21690          END;
21700      8:  BEGIN
21710            WRITELN(TTY,'ENTER STORED SOLUTION NUMBER TO HAVE CELLS');
21720            WRITELN(TTY,'DISPLAYED:');
21730            RDINT(SOLNUM);
21740            VIEWSSOL(SOLNUM);
21750          END;
21760      9:  FLAG:=1;
21770     10:  BEGIN
21780          PRTSOL;
21790          WRITELN(TTY,'SOLUTION FILE WRITTEN TO PRTFILE.');
21800          END
21810     END;
21820     END;
21830   END;
21840
21850   PROCEDURE INITOPT;
21860   (* ************************************************
21870
21880       INITOPT PROCEDURE
21890
21900   THIS PROC PROVIDES THE OPPORTUNITY FOR THE USER TO
21910   STORE THE INITIAL SOLUTION FOUND BY THE MODEL.
21920
21930   ************************************************ *)
21940
21950   VAR ANS:CHAR;
21960   BEGIN
21970   BLINE(1);
21980   WRITELN(TTY,'WOULD YOU LIKE TO STORE THE INITIAL SOLUTION?');
21990   WRITELN(TTY,'ENTER Y OR N:');
22000   BREAK(TTYOUTPUT);
22010   RESET(TTY);
22020   READ(TTY,ANS);
22030   IF (ANS = 'Y') THEN STORDISK(1);
22040   END;
22050
22060   {     MAIN ROUTINE.}
22070
22080
22090
22100   begin
22110      INITMODEL;
22120      OPTION:=2;
22130      INITRUN;
22140      SELSIZ;
22150      while (OPTION = 2) do
22160         begin
22170         OPTION:=1;
22180         INITLEV;
22190         INITOPT;
22200         while (OPTION = 1) do
22210            begin
```

```
22220             CURSOL;
22230             CURTAB;
22240             end;
22250          end;
22260          FINRUN;
22270    end.
```

VITA

David Diedrich Holmes was born in Redbank, New Jersey, on October 13, 1956. Soon after, his family moved to Knoxville, Tennessee, where he attended elementary and junior high schools. He graduated from Bearden Senior High School in June 1974. The following September he entered Eastern Kentucky University and majored in mathematics. After two years, he transferred to The University of Tennessee, Knoxville, and in June 1978, he received the Bachelor of Arts degree in Liberal Arts with a double major in mathematics and computer science. In June 1977, he started work as a programmer with the Tennessee Valley Authority. This led to a systems analyst position in which he is currently employed. In September 1978, he started a part-time master's program in the Graduate School at The University of Tennessee. In March 1982, he received the Master of Science degree with a major in management science.

He is married to the former Nancy Rogers of Knoxville, Tennessee.