Reasoning About Spatial Structure in Landscapes with Geographic Information Systems

Claude Saunders and Art Farley

CIS-TR-91-20
September 12, 1991

Computer and Information Science Department
University of Oregon

REASONING ABOUT SPATIAL STRUCTURE IN LANDSCAPES WITH GEOGRAPHIC INFORMATION SYSTEMS

Claude W. Saunders
Arthur M. Farley

ABSTRACT

Geographic Information Systems offer a new means to store, transform, and present cartographic information. Persons involved in landscape ecology, planning, and management use these systems to transform and examine data pertaining to a given geographical region. Unfortunately, there is often little correspondence between the language of landscape analysis and the operations provided by most Geographic Information Systems.

This thesis describes a language based on the landscape element abstractions commonly used by landscape architects and ecologists. With this language, a task specific landscape definition may be expressed in terms of matrix, patch, and corridor elements. The landscape definition is then used to guide the interpretation of cartographic data.

This thesis presents an extension to an existing Geographic Information System which provides the ability to locate instances of specified spatial structures in cartographic data. This extension represents an initial step towards being able to reason directly about the spatial relations amongst elements of interest in a landscape. The definitions of two landscape analysis tasks are given in the landscape language and solutions generated by our system are presented.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS

The author expresses sincere appreciation to his advisor, Professor Art Farley, for his inspiration and guidance in developing the ideas of this thesis. I wish to thank Professor David Hulse for his conceptual assistance and interest in our work. I would also like to thank Kit Larsen for donating macGIS and for providing technical assistance. In addition, I owe fellow student Zheng-Yang Liu a debt of gratitude for his help in formatting this manuscript.

CHAPTER I

INTRODUCTION

The wealth of information contained in cartographic data poses many challenges for those who develop methodologies and computational aids for its storage, analysis, and presentation. The advent of Geographic Information Systems (GIS) has provided a basic architecture for dealing with cartographic data. There has been a continual evolution of GIS technology towards providing more sophisticated storage, analysis, and presentation capabilities. Persons involved in landscape ecology, planning, and management utilize such capabilities to transform and examine data pertaining to a given geographical region.

Unfortunately, there is often little correspondence between the language of landscape analysis and the operations provided by most GIS. Users must now map their problems onto operations provided by the GIS. For example, to create a map overlay which shows the proximity of agricultural regions to rivers, one might use the following sequence of GIS data transformation operations: "Isolate Landuse assign 1 to agricultural for temp-map1. Isolate Waterbodies assign 1 to rivers for temp-map2. Add temp-map1 to temp-map2 for Resultant-map." This indirect, procedural method of analysis has prevented the use of GIS in some problem domains.

This thesis investigates the role of landscape structure concepts in facilitating GIS interaction and analysis. Central to the thesis is the notion that analysis of data should be carried out in the language domain of landscape structure, not in the language domain of GIS architecture. One assumption underlying this approach is that the spatial structure of a landscape is a sufficiently powerful organizing principle on which to base analysis tasks. This thesis describes a mapping from low level cartographic data to the higher level language elements of landscape structure. It presents an extension to an existing GIS which provides the ability to locate instances of specified spatial structures in a landscape.

This chapter briefly introduces the central operative topics: Geographic Information Systems, landscape structure, and methods for spatial analysis of landscapes using Geographic Information Systems. The chapter concludes with an organizational overview of the remaining chapters.

Working with Geographic Information Systems

A GIS represents the evolution of the map from paper form to digital form. The information once only in paper form now can benefit from the flexibility of access and manipulation by computer. The rate of updating and producing maps is increased greatly. Just as with any paper map, qualitative as well as quantitative data can be represented. Information regarding altitude and distance are what we would consider quantitative, while information such as soil composition or zoning are more qualitative. All this information, once in the form of a vast array of general purpose maps and task specific thematic maps, can now be stored in an integrated fashion in the computer. This information can then be selected and combined to produce particular thematic maps, as desired.

Geographic Information Systems store cartographic information internally, using one or both of two major classes of data representation: raster (or tessellation) and vector [Burrough 1986].

Raster representation is basically a two-dimensional grid of cells, each cell corresponding to a fixed, square area of the region being mapped. Each cell contains the value for some attribute over that area. Vector representation describes the features of a region using points, lines, and polygons. A relational record is often associated with each point, line, and polygon by means of a unique identifier. These relational records can contain non-spatial attribute information, as well as topological relationships.

A GIS is similar in many ways to a conventional database system. There are modules for data input and verification, data storage and management, data presentation, data transformation, and user interaction [Burrough 1986]. Of primary concern in this thesis are the data transformation and user interaction modules. More specifically, an addition to the data transformation module is presented, and its potential to facilitate user interaction and some analysis tasks is examined.

A common use of GIS is in land-use planning and management. Given a regional landscape, one wants to allocate land of various types to human and natural activities. Tracts of land must be found which meet certain requirements. Various interests often compete for land and an optimal balance of allocation can be difficult to find. The ability of GIS to generate custom maps from the transformation or combination of this information facilitates such planning and management.

The map overlay is a general type of operation common to most GIS transformation modules. Given several specialized maps (stored digitally within the GIS), a new map can be generated from the overlay of existing maps. For example, given a zoning map and soil composition map, an overlay of the two would allow the user to locate regions which have certain combinations of zoning and soil characteristics. Another common operation is the ability to recode the attribute values in a map. For example, a map which depicts private, local, state, and federal land ownership could be recoded to depict only private and government ownership. This can be accomplished by assigning a single value to the collection of values representing local, state, and federal ownership.

While GIS provide an integrated source for geographic information, the users must still map their analysis problems onto the structures and operations provided by the system. For some problems this task is simple, while for others the user must have an intimate knowledge of the data representation and operations of the system. Current interactions with GIS are procedural in nature. The user solves a landscape problem by piecing together a sequence of primitive GIS operations and often locating complex configurations off-line.

This thesis, in proposing and implementing a mapping from low-level GIS data to high-level primitives of a landscape language, hopes to facilitate many analysis tasks. Analysis should no longer require mapping a task onto a sequence of system specific operations. The landscape language allows a more direct statement of the problem through a description of desired landscape configurations. This is the notion of declarative versus procedural interaction. One declares the problem and allows some interpreter to evaluate it and generate a procedural solution. In short, the interaction with a GIS can be in terms of elements of the problem domain, not primitive GIS operations.

A landscape language provides a means to represent and store knowledge about landscapes. Knowledge about specific landscape classifications and characteristics can be expressed explicitly. A typical GIS is in essence a database of spatially related information. The criteria guiding interpretation of that information is implicit in the user's mind and the choice of primitive GIS operations. A landscape language provides a means to formally and explicitly express the landscape definitions guiding interpretation.

Landscape Structure

The notion of a landscape has several different definitions depending on the field of study and the scale being considered. This thesis considers the landscape as that defined by the fields of Landscape Architecture and Landscape Ecology. Here a landscape is a heterogeneous land area of the scale typically seen in aerial photography. This ranges from tens of meters to hundreds of kilometers [Forman 1986]. The exclusion of small scale elements results in the fine details of physical systems comprising the landscape being abstracted away. Landscapes of a continental scale are also excluded, for if included, one would be approaching the description of global landscape. The landscapes considered are typically those which regional land-use planning examines. The purpose is not to examine individual physical sub-systems on a small scale, nor to attempt description on a global scale. Rather, it is to examine systems on a spatial and temporal scale which is most heavily impacted by human decision making.

Landscapes can be examined in terms of structure, function, and dynamics [Forman 1986]. This thesis restricts itself to landscape structure, although function and dynamics do remain a consideration. The structure of a landscape involves the spatial relationships between its basic elements. Spatial characteristics, eg. size and shape, of the individual elements are also considered. Landscape function involves the causal relationships between and within elements while landscape dynamics examines the evolution of structure and function over time.

Researchers in landscape ecology and landscape architecture have developed a language in which one can describe landscape structure. The language is tailored to describing surface ecosystems at the landscape level. The primitives of the language are based primarily on the spatial structure. Several elements of a landscape are identified. The matrix of a landscape is typically the dominant land component. The matrix generally provides the ability of a landscape to recover equilibrium after perturbation. Patches of varying types are scattered within the matrix. These patches can be "natural", such as lakes and tracts of forest, or can be "introduced", such as farm fields and landfills. Corridors are linear elements of a landscape such as rivers, hedgerows, and roads. Each of these element types has defining characteristics. In addition, the landscape as a whole has characteristics which represent the spatial relationships among the primitive element types [Forman 1986].

The characteristics necessary to classify a landscape and its components are by no means universally agreed upon. Two significant factors influence what characteristics are used. One, the culture of the region determines what features of the land are considered significant in evaluating usage. Two, the specific goals of the person or persons performing an evaluation of a landscape will bias the choice of characteristics. As such, this thesis is not promoting any particular characteristics over any others. A small, working subset of characteristics will be chosen with no attempt to represent them as universal.

Spatial Analysis - A Comparison

A cartographic model is defined to be a sequence of GIS data transformation operations representing a particular analysis task. Cartographic modelling is the process of analyzing cartographic data by developing sequences of GIS data transformation operations [Burrough 1986]. One type of cartographic modelling is the analysis of spatial relations among regions (or objects) in a landscape. Here we will examine one example of such spatial analysis. The analysis problem and available GIS data will be presented first. Following this, a cartographic model based on the modelling language of macGIS will be used to solve the problem. A cartographic model based on the operational landscape language developed in this thesis will then be given.

We consider the following hypothetical problem: You are given the task of locating a region, within a given area, for relocating members of an endangered species. This region must be undeveloped land and outside the floodplain of any river or lake in the area. The floodplain is defined to be 300 feet around any body of water. You are given the two GIS maps shown in Figure 1. Thus, the transformation operations of the cartographic model must produce a final map which identifies regions of undeveloped land which are at least 300 feet from any river or lake.

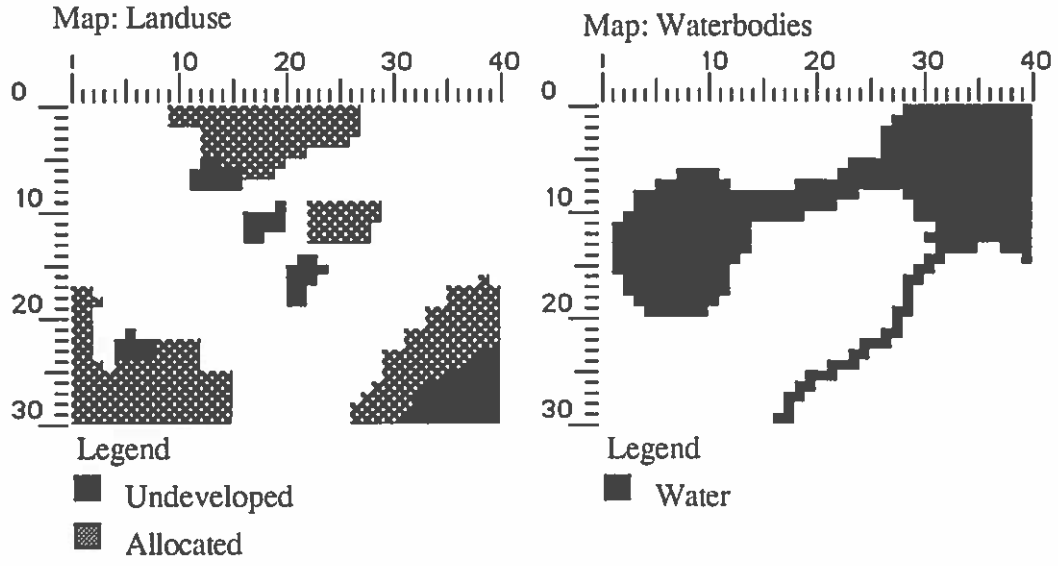


Figure 1. Raster GIS Maps

MacGIS provides a text-based modelling language in addition to a full point-and-click graphical interface. Data transformation sequences may be stored in files and executed as a unit [Larsen and Hulse 1989]. A possible sequence for solving the spatial analysis problem in macGIS is as follows:

1. Isolate Landuse assign 1 to undeveloped for Undeveloped-Land.
2. Spread Waterbodies to 4 for Floodplain-temp.
3. Isolate Floodplain-temp assign 1 to 0 through 3 for Floodplain.
4. Add Floodplain plus Undeveloped-Land times 8 for Relocation-Map.

The basic operations applied are: Isolate, Spread, Isolate, and Add. The first operation produces a new map called Undeveloped-Land whose cell values are 1 (black) for the corresponding cell in Landuse which has the value "undeveloped". All other cells have the value 0 (white). The second operation produces a map called Floodplain-temp in which cells of distance one, two, three, and four from the water are given special values. Note that the user must be aware that a cell distance of three corresponds to a 300 foot distance in the real world. The upper two graphics in Figure 2 show the results of these transformations. The third operation modifies the Floodplain-temp map so that the water and spread area are all coded black. The fourth operation then overlays the results of the first and third operations, with some cell coding tricks, in order to produce the final map shown at the bottom of Figure 2.

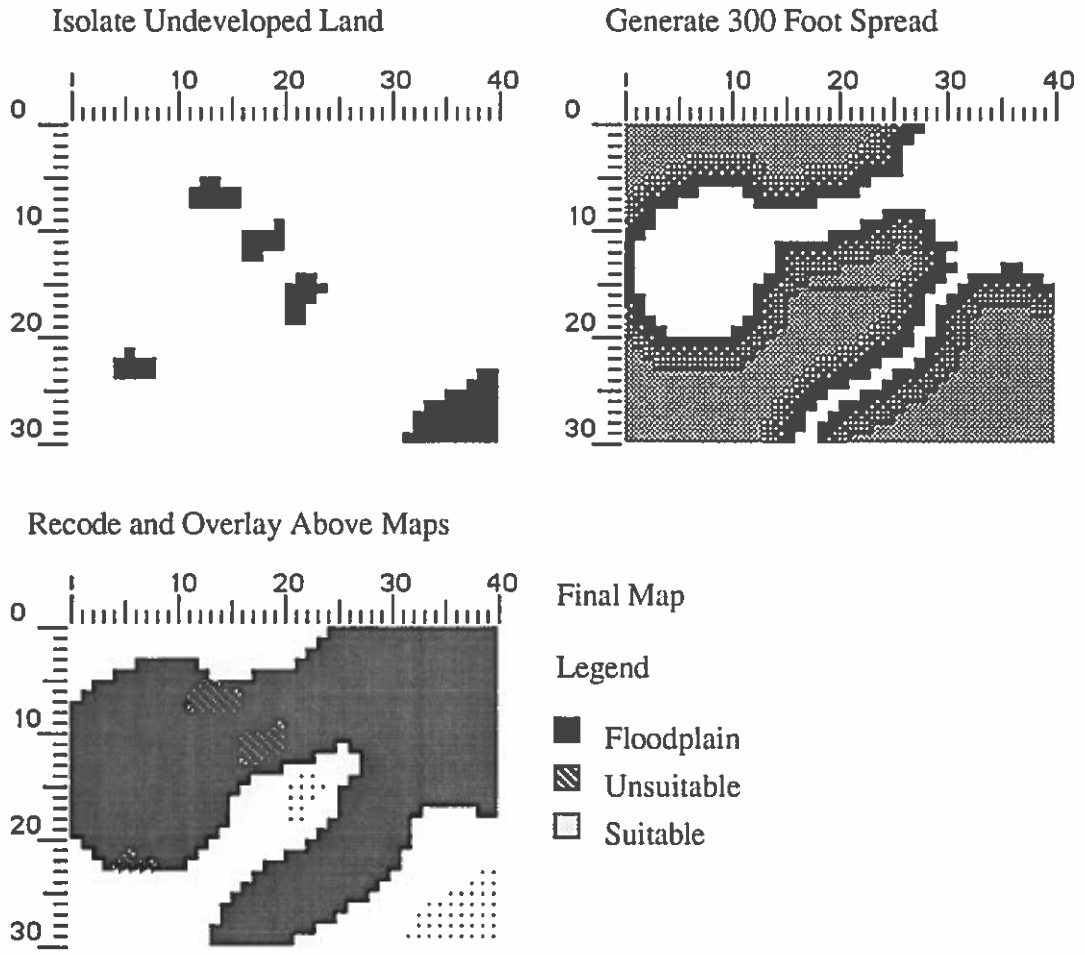


Figure 2. Solution Using GIS Operations

The macGIS cartographic model shown is quite involved, even for a simple spatial analysis task. Note the lack of correspondence between the language of the analysis goal and the operation sequence. This is not a problem with macGIS itself, but rather with the general approach of "map algebra" common to many GIS systems. There are analysis tasks to which this approach is better suited, but there are also clearly a number of weaknesses, most notably in the domain of spatial analysis. The person performing analysis must have an intimate knowledge of the data representation and operations of the GIS.

This thesis presents an operational language for spatial analysis of landscapes. The above analysis problem in terms of this language is as follows:

```
(def-data-layer      undeveloped-land
    creation-script =  (   ("Isolate Landuse assign 1 to undeveloped for Undev.")
                           (load "Undev")
                       ))

(def-data-layer      water
    creation-script =  (   ("Isolate Waterbodies assign 1 to water for Water.")
                           (load "Water")))
```

```
(def-element   patch       undeveloped-patch
   data-layer      =  undeveloped-land
   constraint-list =  (   (> area (10000 "sq feet")) ))

(def-element   corridor    river-corridor
   data-layer      =  water
   constraint-list =  (   (> width (100 "feet"))
                          (< width (300 "feet")) ))

(def-element   patch       lake-patch
   data-layer      =  water
   constraint-list =  (   (> area (30000 "sq feet")) ))

(def-landscape     species-relocation
   component-list     =  (   (patch undeveloped-patch)
                             (corridor river-corridor)
                             (patch lake-patch)
                          )
   constraint-list =        ((undeveloped-patch ?species-relocation-zone)
                             (! (for-all river-corridor
                                (>  (min-dist ?species-relocation-zone river-corridor)
                                    (300 "feet"))))
                             (! (for-all lake-patch
                                (>  (min-dist ?species-relocation-zone lake-patch)
                                    (300 "feet"))))
                          ))

(define landscape-instances (generate-landscapes species-relocation 'full))
(send landscape-instances generate-GIS-map)
```

The first two expressions specify the two "types" of land (or water) which are pertinent to the analysis task. The next four expressions define the desired landscape configuration. The last two expressions generate instances of the landscape definition and produce an output map for each. In this case, two output maps are produced, each of which depicts a potential region for species relocation which conforms to the constraints of the definition.

The landscape language solution may at first seem more involved than the first solution. The first two expressions which define the two "types" of land seem much like the method used before, but there is an important distinction. The creation-scripts do not consider any spatial properties, they simply define a type of land on a cell by cell basis. A library of task independent data layer defintions may be built up and referenced as needed. As such, custom data layers do not need to be created for every analysis task.

The landscape definition also demonstrates some additional capabilities which are simply not available in the "map algebra." Here we are able to place specific constraints on the spatial characteristics within and between the undeveloped land, rivers, and lakes. Using the "map algebra" approach, work is required off-line by the user to determine group sizes and intergroup distances that may be relevant to the task at hand.

Organizational Overview

Chapter II delves into Geographic Information Systems in more detail. The architecture, data representations, and data transformations utilized will be examined. Specifics of macGIS will also be presented. Chapter III describes the language of landscape structure as defined by those in the field of landscape architecture and landscape ecology. Also presented will be a number of spatial characteristics of landscapes. An operational landscape language will be introduced and two analysis tasks will be expressed in this language. Chapter IV describes in detail the semantics of the operational landscape language. The representations and algorithms invoked during interpretation of the language will be given. The final section will be devoted to presenting the prototype application which is built around the landscape language.

Chapter V presents solutions to the two analysis tasks given in Chapter III. The solutions have been developed using the prototype application. Chapter VI presents an evaluation of our research to date. Chapter VII concludes by briefly examining the issue of incorporating non-spatial data into our system. In closing, we relate our efforts to other recent research.

CHAPTER II

GEOGRAPHIC INFORMATION SYSTEMS

The name Geographic Information System encompasses a wide variety of software and hardware designed to support the storage and analysis of cartographic data. The special nature of the data and analysis tasks set GIS apart from traditional database models and spatial design systems such as CAD [Burrough 1986; Cowen 1988]. GIS data is composed of a wide variety of spatial information, such as distance, area, altitude, and distribution, and non-spatial information, such as soil composition, zoning, etc. A traditional database model, eg. the relational model, is certainly capable of representing such information. The tabular data format, however, does not readily support spatial transformations. While traditional data base models are not used as the primary representation, they are often used to hold the non-spatial information. The extensive, special purpose analytical capabilities of GIS are what separate them from CAD systems.

This chapter begins by defining the five major software sub-systems of a GIS. The following section describes the abstract data models and internal data structures used by most GIS. The last section addresses, in more detail, the software subsystem which is of primary concern to this thesis, data transformation.

Architecture

A GIS is composed of five software components: data input and verification, data presentation, query and analysis support, data storage and management, and data transformation [Burrough 1986]. Figure 3 shows these components and the flow of information between them. While any particular cartographic system may support these components to different degrees, they all must be present for a system to be considered a GIS. A unified user-interface component is included here, however the interaction with GIS may occur through the separate components as well. This section will briefly consider each component in turn.

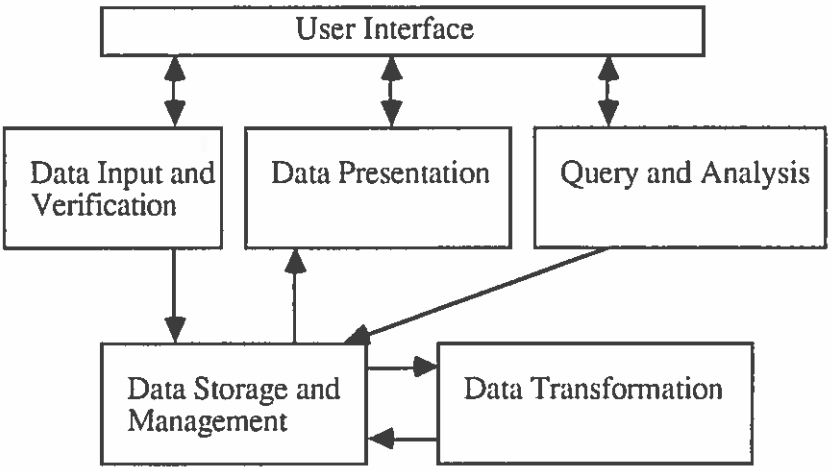


Figure 3. Subsystems of a GIS

Data input and verification is the process of converting maps into digital form. There are four basic steps to this process: entering spatial data, entering non-spatial attributes, linking spatial with non-spatial data, and verifying the results [Burrough 1986]. Entering spatial data can be done manually, with digitizing tablets, or by using image scanners. This process is closely tied to the internal data representation. Non-spatial attributes may be entered in a linear form, much like a map legend, or in more sophisticated structures such as database records. Linking spatial to non-spatial data requires an indexing scheme. This indexing can be done using spatial coordinates (Cartesian), or by explicitly giving unique identifiers to spatial data components.

The process of data input is subject to a number of errors. A flawed encoding of the map can arise from human error in entry, resolution limitations of input devices, distortions in sensing equipment, and errors inherent in coordinate system transformations. The various sources of error must be located and corrected in some fashion.

The task of presenting GIS data, either directly from the database, or after extensive analysis operations, is as broad as the field of scientific visualization. Among the most basic requirements are the ability to spatially locate and window areas of interest, and to output a labelled map to a display or hardcopy device. The output can make use of color schemes or three-dimensional graphics to emphasize thematic data. The display of spatial data can be derived directly from the internal representation without much difficulty. The display of associated non-spatial attribute information, however, raises a number of difficult problems.

A primary benefit of having maps in digital form is the ability to created custom, thematic maps from the transformation of existing maps. Transformations of spatial data require corresponding changes in the linkages to non-spatial data. A critical presentation problem which arises is the placement of map labels. Given changes in the arrangement of spatial data elements, how does one alter the legend or relocate map labels? If the original data made use of color schemes to emphasize elements of a theme, how does one recode the colors for the new map? One benefit of the extensions to the transformation module presented in this thesis is the automatic labelling of objects in the output map. The landscape definitions guiding interpretation of the GIS data prescribe the labelling of the output.

The query and analysis component of a GIS supplies a language in which the user can express retrieval and analysis goals or explicitly execute a sequence of operations. This language can support visual interaction, eg. direct manipulation of the map images, natural language interaction much like that which is applied to relational databases, command-line interaction, menu-based interaction, or some combination of the above. Retrieval involves requesting the presentation of data explicitly represented by the system. For example, what is at location (x,y)? or, what locations have attribute value z? Analysis involves transformations of the explicitly represented data in order to derive new information. For example, what locations above a certain altitude have attribute value z? The wide variety of analytical capabilities has given rise to an equally wide variety of languages for their support.

The process of turning an analysis goal into an expression in the query and analysis language is referred to as cartographic modelling [Burrough 1986]. Burrough emphasizes the importance of explicitly representing the structure of an analysis task. To this end, we make use of a language of landscapes which allows the explicit declaration of landscape analysis goals. The analysis goal is then "compiled" into operations on the GIS data.

Data storage and management involves the full spectrum of issues from physical storage medium to data structure to abstract data model. The high volume of data places great demands on physical storage and access. The various modules of GIS place demands on the data structures used to hold the volume of data in memory. What may prove to be a successful data structure for presentation may be inefficient with respect to certain analysis and transformations tasks. Still further up the abstraction spectrum lies the abstract data model. As discussed in database literature, it is important to form a distinct boundary between the abstract data model and the underlying data

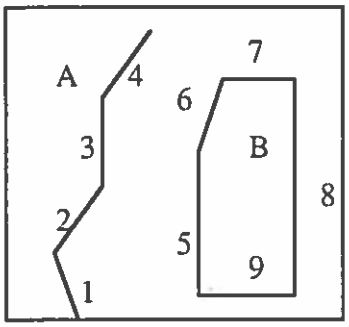
structures. GIS stand to benefit from the theoretical work in data base management systems (DBMS), but also have a number of unique problems.

The data transformation module encompasses a large number of analysis and processing operations. Under requests from the input, presentation, and analysis modules, the data storage and management and data transformation modules work together to assemble the necessary data. Most operations are highly dependent on the internal data representation. Section three of this chapter categorizes transformation operations into seven major groups.

Data Representation

Two major abstract data models for representing cartographic data are currently in use: the tessellation (or raster) model and the vector model [Peuquet 1984]. Each makes a fundamentally different commitment in representing the analog data: the paper map, areal photograph, or remote sensing. Each has a number of advantages and disadvantages. The proliferation of these models in commercial systems confirms the utility of both. A number of systems also support the conversion of data between models. The models are implemented using a variety of data structures and data compression techniques. This section will briefly examine some implementations with respect to their parent abstract data model.

The vector model represents the points, lines, and regions of a map with line segments of varying length. A point is a segment of length zero, a line is a chain of segments, while a region is represented as a polygon. Non-spatial data can be associated with each segment, or with the higher-level constructs of point, line, and polygon. Figure 4 provides a brief example. The vector model is high-level in that the entities of the source map are almost directly represented. For example, a polygon representing an agricultural field can have non-spatial data such as crop type, property value, and owner directly associated with it. A key disadvantage of this, however, is that the record of attribute data essentially fixes the interpretation of the map. When entering data, one generally must have a-priori knowledge regarding what real-world entities need to be explicitly represented. Consider the problem of representing continuously varying values over a region, such as altitude or soil acidity. There may be no a-priori interpretation of this data in terms of chains or regions. The vector model is entirely appropriate for many uses, however the tessellation model is an important alternative.



Segment	Endpts
1	x1,y1 x2,y2
2	...
.	
.	
.	
9	

Chains	Segments
A	1 - 4

Polygons	Segments
B	5 - 9

Object	Name	Attributes
A	River	flow rate, destination
B	Field	crop type, property value

Figure 4. Vector Data Representation

The vector model can be implemented as a simple list of line segments. Each segment can be given a unique identifier under which the endpoint coordinates are stored. A simple compression technique called chain-coding can be applied. Chain-coding involves storing, for each disjoint chain or polygon, a starting coordinate followed by a series of (direction vector, length) pairs. This removes the redundancy associated with common line segment endpoints. A hierarchical organization can also be built in which regions and chains, with associated database records, are represented as lists of segment identifiers. The topology of points, chains, and regions may also be stored explicitly at time of data input.

The vector model provides compact storage of data. Output can be produced using pen plotters, giving accurate and very readable maps. Precise boundaries are well represented, such as property lines or zoning. Data for regions can be retrieved graphically by selecting a point inside the region of interest and allowing the system to compute which polygon the point lies within. In addition to the disadvantage mentioned above, the vector model also does not lend itself well to the operation of map overlay. A number of difficulties arise from the intersection of line segments in the respective maps [Burrough 1986].

The tessellation data model, also known as the raster model, has a number of incarnations. The model represents data with a regular grid of cells. The cells are most commonly square, however any other covering shape can be used, such as the triangle or hexagon. Due to the predominance of square tessellation models, all future mention of tessellation or raster will refer to this cell shape. Each cell corresponds to a small, fixed area of the region being mapped. Each cell contains the value for some attribute of that area, or more commonly, an index into a map legend which defines the value. The spatial information is represented by the coordinates of the cell, while the non-spatial information is represented by the contents. Since the grid of cells represents the values of only a single attribute over a given region, there is a separate grid, or data-layer, for each attribute being recorded. Figure 5 provides an example.

Soil Ph

2	3	3	1	1	0	2
1	3	2	2	0	2	2
1	2	0	0	2	2	2
2	0	2	2	2	1	1
0	2	3	3	0	0	0
3	2	3	3	0	0	2
3	2	1	1	0	1	2

Legend

0	no value
1	acid
2	neutral
3	base

Waterbodies

0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
1	0	0	0	2	2	2
0	0	0	0	2	2	0
0	0	0	0	2	0	0

Legend

0	land
1	rivers
2	lakes

Figure 5. Tessellation Data Representation

The tessellation model is a low-level representation in comparison to the vector model. Composite entities such as chains or regions are not explicitly represented. As such, no commitment to interpretation is made beyond the choice of attributes (data-layers). Continuous valued attributes, such as altitude, are better represented. An important benefit of continuity is the ability to represent degrees of suitability over a region with respect to some analysis task. The continuity is, of course, an approximation subject to the resolution of the grid.

The tessellation data model is conveniently implemented with two-dimensional arrays and a corresponding set of tables for the map legends. A sufficiently high grid resolution can result in a tremendous volume of data, similar to the storage required for 16 bit color images. Many compression techniques can be applied to mitigate this problem, however this forces changes in the transformation algorithms, or potentially time consuming decompression/compression operations. Two common compression techniques are run-length encoding and quadtrees. Run-length encoding traverses the grid of values and replaces contiguous sequences of like-valued cells with a suitably encoded (value,length) pair. Quadtrees seek to recursively decompose a region by successive halving in both dimensions. The process stops when all cells of a subregion contain the same value.

The tessellation model simplifies an increasingly common form of data input, remote sensing. Data physically collected from surface sensors or satellites may be directly entered. Differences in resolution can be dealt with via averaging or interpolation. The regular nature of the grid also simplifies many transformation algorithms [Burrough 1986].

Data Transformation

The vast array of GIS operations can be grouped into a number of categories: map generalization, map abstraction, map sheet manipulation, buffer generation, map overlay, measurement, and digital terrain analysis [Dangermond 1983]. Most of the operations in these categories are highly dependent on the data structures used (eg. raster or vector).

Map generalization is a set of operations which reduces the volume of data in a map. These operations are most often used in conjunction with changes in map scale. As an example, consider a line or area in a map composed of a lengthy chain of vectors. If the scale of the map is increased greatly, it may be sufficient to represent the lines and areas with a fewer number of vectors. Another possibility is to simply remove vectors from the map as scale is increased.

Map abstraction involves fundamental changes in the representation of data. A number of GIS operations fall under this category: converting from raster to vector, or vector to raster, representing closed boundaries by center of area, representing data points by isolines, and reclassifying a set of regions as one. This thesis proposes an additional abstraction based on the "landscape element". A set of spatial data and associated non-spatial attributes may be collected into a single object, per user definition, called a landscape element. The map thus becomes a collection of landscape elements, each with explicitly defined characteristics. The abstraction is formed for the purpose of facilitating spatial landscape analysis. Chapter III introduces this abstraction in more detail.

Map sheet manipulation involves changes in the spatial data independent of content. Among these operations are scale change, projections, rotation, translation, and transformations to remove distortions.

Buffer generation involves generating spatial expansions or contractions of points, lines, and regions. A point may be expanded into a circle or square. A line may be expanded into a strip, and a region may be uniformly expanded or contracted. These transformations may serve a number of purposes. For example, a landfill may be represented by a region on the map. By uniformly expanding this region, a graphical representation of a safety buffer zone may be created.

Map overlay is the composition of individual maps. The methods for performing this differ considerably, depending on whether the representation is raster based or vector based. The overlay of vector based maps involves such issues as vector intersection, redefinition of closed boundaries, and alignment. The overlay of raster based maps is simplified by the regular structure. A form of "map algebra" for raster maps has been developed [Burrough 1986]. Many of the overlay operations for raster maps are analogous to boolean image operations used in computer graphics. Cell values in maps may be "weighted" by adding or multiplying with other map layers. The applications of map overlay vary widely. For example, a map delimiting congressional districts may be overlayed with a more traditional map of streets to enable the planning of house to house canvassing.

Measurement operations summarize spatial characteristics of the data. Common measures are: point-to-point distance, distance along chains, area, and perimeter. Among the more specialized measurements are: volume estimates given altitude information, or least cost paths along networks of chains. A number of spatial characteristics are discussed in Chapter III in the context of landscape structure.

A number of GIS operations make use of altitude information over a region. These fall under the category of digital terrain analysis. The representation of altitude, known as the digital elevation model, has a number of incarnations. Two common models are point data and contour. Point data is readily implemented in the raster representation, each cell containing the average altitude of the covered area. Contour elevation models may be implemented using nested polygons. Point data may be interpolated to generate contours as well. Digital terrain analysis can

derive new information such as viewshed, slope, and aspect, as well as generate three-dimensional terrain views or terrain cross-sections.

MacGIS

MacGIS is a raster based Geographic Information System for the Macintosh which was authored at the University of Oregon by Kit Larsen and David Hulse [1989]. It provides the ability to create, edit, view, and transform cartographic data in raster form. The prototype landscape analysis module presented in this thesis interacts closely with macGIS. As such, we will briefly examine some aspects of data representation, data transformation, and interaction in macGIS relevant to our implementation. This section is not intended to serve as a summary of macGIS capabilities.

MacGIS represents cartographic data as a set of raster data layers. Each data layer represents the values of a single attribute over a given region. For example, a data layer can represent land ownership in a county, or crop types in an agricultural region. These data layers are stored individually in files and read into memory as needed for viewing, editing, and transformation operations. These files provide convenient access to the GIS data needed by the prototype system of this thesis.

Each data layer has two components: a two-dimensional array for holding integer cell values, and a legend which associates each cell value with an arbitrary label and visual pattern. A cell value is not an actual attribute value, but rather an index into the legend. The labels in the legend represent the attribute values. The patterns are utilized when the map is displayed on screen. Instead of showing a dense array of cell numbers, the value in each cell is replaced with its associated pattern. No data compression techniques are utilized, thereby simplifying use of this information by our implementation.

New data layers may be created by applying a variety of transformation operations to existing data layers. The transformation operations are divided into three groups: Point, Arithmetic, and Neighborhood. The Point group contains three operations: Isolate, Recode, and Slice. These operations create a new data layer by changing the cell values and legend contents of a single input data layer. Isolate is used to combine one or more attribute values of the input layer into a single value for the new layer. For example, isolate the landuse layer, assigning '1' to housing and heavy industry. The result is a binary data layer where cells of value one represent the presence of housing or heavy industry, and cells of value zero represent the absence. The Recode and Slice operations allow similar changes.

The Arithmetic group contains eight operations: Add, Average, Cover, Divide, Maximize, Minimize, Multiply, and Subtract. These operations create a new data layer by computing new cell values as a function of the values in two or more input data layers. The functions operate on a cell by cell basis. For example, Add computes a cell value of the new data layer by summing the corresponding two cell values of the input data layers. The other operations work similarly.

The Neighborhood group contains eight operations: Clump, Differentiate, Orient, Radiate, Scan, Score, Smooth, and Spread. These functions vary widely in purpose, but are similar in that they compute new cell values as a function of a neighborhood of cell values in one or more input layers. For example, Spread takes a binary layer and distance value as input, and generates a new layer such that every region of the input layer is expanded uniformly by the distance value. The Scan operation computes new cell values based on a specified function of a specified neighborhood of cells in the input layer.

The transformation operations of primary interest to this thesis are in the Point and Arithmetic groups. The def-data-layer expression of the landscape language makes use of these operations to define "types" of land by the combination of attributes on a cell by cell basis. The analysis of spatial properties, normally implemented with Neighborhood operations, will be handled outside macGIS by our system.

MacGIS provides two interfaces to its collection of data editing and transformation operations: graphical direct manipulation and a text based command language. The direct manipulation interface is the menu and dialog box interface common to many Macintosh applications. In addition, macGIS provides the ability to examine and edit data layers by direct manipulation of the displayed map and legend.

The text based command language provides the text equivalent of all the transformation operations and most of the data file management operations. These commands may be entered via a command log window, or by reading them from a text data file. This was a critical feature for our prototype implementation. Our application obtains GIS data by supplying text commands to macGIS and retrieving the resulting data files.

CHAPTER III

THE LANGUAGE OF LANDSCAPE STRUCTURE

In the study of complex natural systems, a language must be developed for describing the essential components and processes in the domain. In particular, studying systems of many variables at a macroscopic level necessitates the development of an abstraction of the domain. Inevitably, multiple competing abstractions develop in a field, and one must make an ontological commitment before commencing experimentation. The chosen abstraction focuses the researcher on particular elements of structure and function in the system.

This thesis makes use of a language developed by researchers in landscape ecology and landscape architecture. The language is tailored to describing surface ecosystems at the landscape level. The primitives of the language are based primarily on the spatial structure of landscapes. Research is currently underway to correlate the spatial characteristics of landscapes with measures of ecological health or historical significance.

This chapter introduces the field of landscape ecology, and then proceeds to discuss landscape structure in some detail. The following section introduces a sample of spatial landscape characteristics and some algorithms for their computation. The last section presents the operational landscape language utilized by the Landscape Reasoning Module. Also presented in the last section are two analysis tasks expressed in the language.

Landscape Ecology

Landscape ecology considers a landscape to be a collection of elements, each of which represents either a separate ecosystem or a connection between ecosystems. The dynamics of the landscape are determined by the flows of energy and materials among and through these elements [Forman 1986]. There are three basic element types: the matrix, the patch, and the corridor.

As an example, consider a landscape consisting of waterbodies, such as ponds and streams, and small tracts of dense growth well distributed in a primarily agricultural region. The ponds and tracts of dense growth are instances of the patch element type. The streams connecting the ponds are instances of corridors. Perhaps less obviously, hedgerows which connect some of the dense growth patches are also considered corridors. The dominant component of the region (both in area as well as ecological significance), is the vast area of groomed agricultural land. This constitutes an instance of the matrix element type.

The function and dynamics of the landscape described above may be expressed in terms of the landscape elements. For instance, several species of animals may reside in the dense growth patches. The hedgerow corridors may provide a means for them to migrate from patch to patch. The pond patches and stream corridors are an integrated system as well. Changes in the agricultural matrix can be examined in terms of the impact on the various patches and corridors in the region. It has been observed that the intra and inter element spatial characteristics play an important role in determining the ecological dynamics of a region. As such, having a language which focuses on the spatial structure of a landscape has become increasingly important.

Landscape Structure

A landscape may be partitioned into a number of regions, each belonging to one of the three basic element types: the matrix, patch, and corridor. While there are numerous variants of these element types, such as the network, ring, and peninsula, this thesis will restrict itself to the three basic types.

Given a landscape, how does one partition it into a matrix, patches, and corridors? This depends on the definition of the element types and one's goal in understanding any particular landscape. I will begin by discussing the general definition of each element type. These types are as defined in *Landscape Ecology* [Forman 1986].

Matrix

The matrix is the dominant element of a landscape. Needless to say, the definition of dominant is highly context sensitive. Three characteristics are used by Forman to identify dominance. The most obvious characteristic is total area. The matrix of a landscape is generally the type of land occupying the largest percentage of area in a region. The next most important characteristic is connectivity. The matrix usually exhibits high connectivity. While this is not an absolute requirement, one can see that a matrix with very low connectivity is perhaps better described as a collection of patches. The third characteristic is degree of control over landscape dynamics. The matrix is the land type which dominates the flow of materials and energy through a landscape. It generally provides the ability of a landscape to recover equilibrium after perturbation. Since this characteristic is outside landscape structure, it will not be considered.

Having identified the matrix in a landscape using area and connectivity, additional characteristics can be utilized for description. The porosity of a matrix is the number of closed boundaries surrounding patches of a given type. The boundary shape is the degree of concavity or convexity of the boundaries of the matrix. In essence, is the boundary rounded, convoluted, or dendritic. Each of these reflects different functionality. A related characteristic is the perimeter-to-area ratio.

Patch

A patch is defined as a non-linear bounded area of the landscape which differs from its surroundings. While there is only one matrix, a landscape may have many patches. The primary characteristics of a patch are area, shape, boundary type, origin, and heterogeneity. Shape takes such values as circular, elliptical, square, rectangular, irregular. Boundary type is like that for matrix; a boundary can be rounded, convoluted, or dendritic. Origin refers to the source of the patch's existence. A patch may be the result of a natural disturbance, human endeavor, or an environmental resource. This characteristic, while having descriptive value, is generally supplied explicitly. For this reason, it does not need to be interpreted from GIS data. Heterogeneity reflects the diversity of composition in a patch. This characteristic, like dominance, is context-sensitive and must be defined for each landscape.

Corridor

A corridor is defined as a narrow strip of landscape which differs in type from the matrix. The definition of narrow will decide whether some part of a landscape is a patch or a corridor.

Thus the key characteristic is width, and not area. Other spatial characteristics are curvilinearity, connectivity, and node distribution. A corridor may originate and/or terminate in patches or even other corridors. A corridor may also have breaks and narrowings. Connectivity is therefore a more difficult term to define. The frequency of patches or widenings in a corridor define the node distribution. A corridor has non-spatial characteristics as well. Origin follows the same as that for patches. A corridor may also be identified by its altitude relative to the surroundings. A hedgerow is as much a corridor as a drainage ditch is.

Landscape as Collection of Landscape Elements

At this point, the semantics and spatial characteristics of landscape elements have been defined. The spatial characteristics alone, however, are insufficient to allow the definition of any particular landscape. Any given landscape has many different interpretations depending on the goals of the description. For example, one could take a landscape of some region consisting of forest and bodies of water. If the goal is to describe and examine the flow of water through the landscape, one could imagine the patches as lakes and ponds, while the corridors would be the rivers and streams flowing between. All other land area, eg. the forest, could be a part of the matrix. On the other hand, if the goal is to describe and examine logging practices, one may consider the landscape as a collection of cut and uncut patches of forest. The corridors may be the streams and logging roads. Here it is perhaps more difficult to define the matrix. The point is that the user must at some point explicitly describe what they are looking for in a landscape. One cannot interpret a map from purely spatial characteristics and come up with a landscape classification. Any automated system must be informed of the land type that is to constitute the matrix, patches, and corridors. The last section of this chapter describes the method applied in the Landscape Reasoning Module for defining the landscape elements.

In addition to finding the component elements of a landscape, the landscape as a whole has a number of characteristics. The distribution of patches and corridors can vary considerably. A landscape can be very regular in appearance, or it can be dominated by a distinctive feature such as a town. Characteristics can be more numerical, such as average patch size, or average patch to corridor distance, etc. These are inter-element characteristics. Previously discussed were the intra-element characteristics. The full description of a landscape incorporates both.

Sample of Spatial Characteristics

Quantitative spatial measures of a landscape serve to summarize the spatial data. Given a set of spatial characteristics, one can compare landscapes, or correlate with other data sources. The spatial characteristics utilized depend on the particular analysis task. This section presents a sample of characteristics gleaned from a number of sources. Where possible, an algorithm for computing the value will be given.

The algorithms given in this section assume a square tessellation raster representation. Since the non-spatial attributes are not considered here, the data is assumed to be binary values in a two-dimensional array. The data is not encoded in any fashion, eg. quad-tree, run length encoding, chain code, etc. Equivalent algorithms for vector data representation could be made as well. Alternately, there exist methods for converting from vector to raster data.

The spatial characteristics can be divided into two categories, intra-element and inter-element. Intra-element characteristics describe a single, isolated landscape element such as a patch or corridor. Inter-element characteristics describe relationships between landscape elements. The

algorithms for intra-element characteristics assume that the landscape element is already isolated from the remaining image elements. In other words, the algorithms do not consider the problem of separating one landscape element, such as a patch, from an adjacent element, such as a corridor. This will be addressed in Chapter IV. The landscape element under consideration is assumed to be contained in a minimal bounding box. The minimal bounding box is the smallest rectangle inscribing the landscape element.

Intra-Element Characteristics

The majority of intra-element spatial characteristics refer only to a generic contiguous region. As such, the bulk of these characteristics apply to all three specific element types, the patch, matrix, and corridor. The common characteristics considered here are area, edge length (perimeter), interior to edge ratio, edge density, shape index, fractal dimension, length breadth ratio, and compaction index. The matrix specific characteristics considered are connectivity and porosity. The corridor specific characteristics considered are width, extent, and curvilinearity. Before discussing their computation, it is necessary to define what is meant by a contiguous region.

A contiguous, or connected, region in a square tessellation data model can be four-connected or eight-connected. Figure 6 provides an example of both. For a cell (i,j) to be considered a member of a four-connected region, it must be adjacent to that region by at least one of the (i-1,j), (i,j+1), (i+1,j), or (i,j-1) cells. For a cell (i,j) to be considered a member of a eight-connected region, it must be adjacent to that region by at least one of the (i-1,j), (i,j+1), (i+1,j), (i,j-1), (i-1,j+1), (i+1,j+1), (i+1,j-1), or (i-1,j-1) cells.

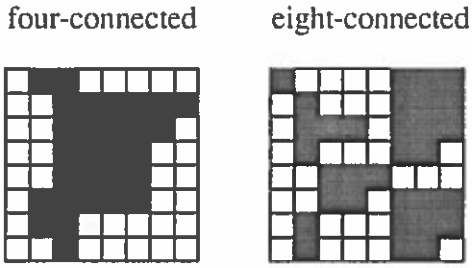


Figure 6. Connectedness of Regions

The computation of spatial characteristics, most notably perimeter, is more efficient if one assumes a landscape element will always be represented as a four-connected region. The nature of GIS data precludes this, however. The low resolution of a GIS data layer relative to the landscape results in frequent occurrences of eight-connected regions.

Two general algorithms for computing the perimeter of a region will be given. One assumes a four-connected region, the other makes no assumption. A four-connected region allows one to follow the boundary of the area, summing cell edges as one progresses. The same algorithm applied to an eight-connected region carries the possibility of missing cells and getting caught in cycles. The algorithm can be modified to handle these problems, but many details are introduced [Ballard and Brown 1982]. The algorithm for a possible eight-connected region will thus examine every cell of the region, including interior cells. A count of cell edges is returned. This can easily be converted to the units of the GIS data layer.

The perimeter algorithm for a four-connected region is as follows:

```
procedure four-perimeter(bound box b)
cell edges c;

    c = 1;
    scan b beginning with upper left cell, stop when first region cell encountered
    repeat
        if current cell is region cell then
            c = c + 1
            turn left and step to next cell
        else
            turn right and step to next cell
    until current cell = first cell
    return c;

end four-perimeter;
```

The perimeter algorithm for an eight-connected region is as follows:

```
procedure eight-perimeter(bound box b)
cell edges c;
index i,j;

    c = 0;
    scan b row by row, for each region cell (i,j) encountered
        if cell (i-1,j) is non-region then c = c + 1
        if cell (i,j+1) is non-region then c = c+ 1
        if cell (i+1,j) is non-region then c = c + 1
        if cell (i,j-1) is non-region then c = c + 1
    return c;

end eight-perimeter;
```

The algorithm for area simply scans the bound box b and totals the number of region cells. This value can be converted to the units of the GIS data layer.

The perimeter and area of a landscape element can be used to compute a number of second order measures which characterize the shape. The shape index (SI) measures deviation from a perfect square. It is computed directly from the area and perimeter values as follows:

$$SI = \frac{0.25 * \text{Perimeter}}{\sqrt{\text{Area}}}$$

The compaction index (CI) measures deviation from a perfect circle. The compaction index requires finding the smallest circle inscribing the region in question. Given this, the index is computed as follows:

$$CI = \frac{\text{Area}}{\text{Area of smallest inscribing circle}}$$

The length breadth ratio measures deviation from either a circle or a square. The length value is the length of the longest line between edge cells of the region. The breadth value is the length of the longest line perpendicular to the length line. A perfect circle or square both result in unity for this measure.

The boundary of a landscape element may be characterized by interior to edge ratio, edge density, or by fractal dimension. The interior to edge ratio is simply the area divided by the perimeter. Edge density measures perimeter per unit area and is simply the inverse of the interior to edge ratio. Fractal dimension (FR) provides another characterization of a region's boundary and is computed as follows [Li 1989]:

$$FR = \frac{2\,\mathrm{Log}(\mathrm{Perimeter}/4)}{\mathrm{Log}(\mathrm{Area})}$$

Two spatial characteristics unique to the matrix landscape element are identified: connectivity and porosity. The connectivity of a region has been restricted to one so far. In other words, for a cell to be considered a part of a region, it can be at most one cell away from a region cell. The matrix of a landscape is the dominant element, however it may be desirable to consider it as a collection of possibly disconnected regions. Connectivity is not measured after the matrix element is isolated, but rather it is used to constrain the process of searching for the matrix element. This issue will be addressed in Chapter IV, along with other aspects of isolating landscape elements in a GIS data layer.

Porosity measures the number of closed boundaries in the matrix. Computing this characteristic requires following the edge of all non-interior cells in the bounding box. If a cycle results, a closed boundary has been found and the porosity can be increased by one. The process of following edge cells is subject to the same difficulties as computing the perimeter of a region. If a four-connected region cannot be assumed and/or the connectivity is greater than one, the algorithm is complicated considerably.

Three spatial characteristics unique to the corridor landscape element are identified: width, extent, and curvilinearity. Maximum, minimum, or average corridor width can be computed. Maximum corridor width can be found by uniformly shrinking the region (removing perimeter cells) until it disappears. Similarly, minimum corridor width can be found by uniformly shrinking the region until it becomes disconnected. This technique is addressed in Chapter IV in the context of isolating corridors from adjacent patches. Extent characterizes the length of the corridor, while curvilinearity characterizes deviation from a straight line. A discussion related to these measures can be found in *Digital Picture Processing* [Rosenfeld and Kak 1976].

Inter-Element Characteristics

The inter-element characteristics apply to one or more landscape elements of one type or across element types. In addition, these characteristics can apply to specific sub-types of the matrix, patch, and corridor such as forest matrix, lake patch, and river corridor. Those considered here are: total number of elements, element density, average element size, inter-element distance (minimum distance and distance between center of area), distribution, and adjacency. It is assumed that all elements of interest are contained in separate minimal bounding boxes. How this is accomplished is the subject of Chapter IV.

The total number of elements and average element size are simple to compute given all elements are contained in separate minimal bounding boxes. Element density can be given in terms of the entire landscape area, or in terms of the matrix area only.

Inter-element distance can be computed as the minimum distance between two landscape elements, or as the distance between the center of area of the two elements. The algorithm used in this thesis for computing minimum distance involves comparing every cell of one region with every cell of the other. The shortest distance represented by $d = \mathrm{Sqrt}((x1 - x2)^2 + (y1 - y2)^2)$ is chosen as minimum distance. While this method is computationally expensive, a more efficient algorithm which just examines perimeter cells must either assume four-connected regions, or incorporate enough information to prevent cycles and missed cells.

Computing distance between center of area involves first computing the center of area of each element. Center of area, also called centroid, is the intersection of a measure of center along the x-axis, and along the y-axis. These are computed as follows:

$$CAx = \frac{\sum_{i=1}^{c} x_i w_i}{N} \qquad CAy = \frac{\sum_{i=1}^{r} y_i w_i}{N}$$

where c is the number of columns of the bounding box, x_i is the x coordinate, w_i is the number of region cells in column i, and N is the total number of cells comprising the region. Similarly for CAy. The pair (CAx, CAy) after rounding or truncation represents the coordinates of center of area. Inter-element distance is then the distance between the center of area of each element.

Distribution of landscape elements can be described by a number of statistical measures [Lounsbury, Sommers, and Fernald 1981]. One such measure extends the center of area measure to incorporate multiple landscape elements. The standard deviation of this center of area can then be used to characterize aggregation of these elements. The standard deviation of center of area, also referred to as standard distance, is the two-dimensional equivalent of standard deviation; it represents the radius of a circle about the center of area. Computing the center of area of a collection of landscape elements is identical to the above method, only the minimal bounding box is now assumed to inscribe all of the elements. Given the pair (CAx, CAy), the standard deviation (distance) is computed as follows:

$$\text{standard distance} = \sqrt{\frac{\sum_{i=1}^{c} (x_i - CAx)^2 w_i + \sum_{i=1}^{r} (y_i - CAy)^2 w_i}{N}}$$

where r, c, x_i, y_i, w_i, and N are the same as above.

Adjacency measures the length of common boundary between different element types (or sub-types). This is especially important for characterizing the potential for flow of materials and energy in a landscape.

An Operational Language for Landscape Analysis

This section introduces an operational language of landscape structure modelled after the definitions presented earlier in the chapter. The language enables the definition of landscape and landscape element "types", and provides a means to invoke interpretation of GIS data subject to the definitions. Also provided is a means to query the result of any interpretation. The syntax of the language is covered first. A brief exposition of the semantics is then covered, followed by two

specific analysis tasks expressed in this language. Some knowledge of the scheme programming language is necessary to work with portions of the landscape language. A complete application, described in Chapter IV, was built to alleviate this requirement. To fully understand the utility of the language, however, it is necessary to present it sans user-interface.

The Language

The constructs of the landscape language are much like those in the scheme language. Namely, each expression is delimited by balanced parentheses. The expressions fall into three major functional groups: definition, instantiation, and query. The syntax will be presented now and the following section will address their usage. The syntax presentation is informal; see the appendix for a complete BNF definition.

Three expressions provide the ability to define the landscape and GIS data source: def-data-layer, def-element, and def-landscape. They are as follows:

```
(def-data-layer      <name>
    creation-script =  (   (   <macGIS-commands>  )
                           (load <filename>)
                       )
)

(def-element   <element-type>    <name>
    data-layer       =  <name>
    constraint-list  =  ( <intra-el-constraint-exps> )
)

(def-landscape      <name>
    component-list =  ( <components> )
    constraint-list  =  ( <inter-el-constraint-exps> )
)
```

Two expressions provide the ability to generate instances of defined objects: make-inst and generate-landscapes. They are as follows:

```
(make-inst      <name>)
(generate-landscapes <name>  <mode>)
```

where:
<name> refers to a previously defined object
<mode> is either 'full, 'non-data-layer, or 'satisfy-only

A single expression form allows queries and commands to be sent to instances generated as a result of the above expressions. This form is as follows:

```
(send <inst>  <message>  ...<arguments>...)
```

where:
<inst> is an instance returned by make-inst or generate-landscapes
<message> is one of a number of queries or commands

Expressing an Analysis Goal in the Language

The landscape language is designed to facilitate spatial analysis of a landscape in terms of its component landscape elements. The particular task supported here is the ability to locate instances of specified spatial structures in the GIS data. Analysis proceeds in the following order: landscape definition, landscape instantiation, and landscape query.

The expressions def-data-layer, def-element, and def-landscape are used to supply the complete set of definitions for analysis. Using the def-data-layer expression, one begins by defining the various "types" of land from which landscape elements will be composed. Each def-data-layer expression associates an arbitrary <name> with a sequence of macGIS commands. The macGIS commands are supplied with the creation-script. The commands should extract GIS data, perform some transformations, and ultimately produce a binary data-layer (binary in that the only cell values are zero and one). The final entry of the creation-script, "(load <filename>)", loads the resultant binary data-layer. A typical example would be:

```
(def-data-layer  undeveloped-land
    creation-script =  (    ("isolate Landuse assign 1 to undeveloped for
                                Undeveloped-Land.")
                            (load "Undeveloped-Land"))
)
```

The purpose of each def-data-layer expression is to define a single type of land (or water). Cells with the value one represent the presence of this type while cells with the value zero represent the absence. When constructing a sequence of macGIS commands to produce a binary layer, no consideration should be given to spatial relationships in the data. The idea is simply to define a type of land by the combination of attributes on a cell by cell basis. The macGIS commands Add, Multiply, Cover, Isolate, and Recode are useful for this. The def-data-layer expression is essentially a macro. It allows the user to associate a single label with a command sequence. Future references to a particular type of land may now utilize the more meaningful label, as opposed to a potentially long sequence of commands. A "library" of these macros can be built up and drawn upon by many analysis tasks.

Given that a set of land types has been defined, the def-element expression may now be used to define landscape elements. As before, each definition is given an arbitrary <name>. Each definition will be a specialization of one of the three major classes: matrix, patch, and corridor. This is specified via the <element-type>. Each definition will reference one of the land types previously defined, via the data-layer <name>, and contain a set of intra-element spatial constraints, via the constraint-list. Any connected component, or clump, of the referenced land type which meets the spatial constraints will be considered an instance of that definition. A typical example would be:

```
(def-element  patch  undeveloped-patch
    data-layer       =   undeveloped-land
    constraint-list  =   ((> area (10000 "sq feet")))
)
```

Intra-element spatial constraints are specified as a list of zero or more individual constraints. Each constraint is of the following form:

(<relation> <characteristic> (<value> <units>))

Typical examples would be: (< area (2000 "sq feet")), or (< width (50 "meters")). The current list of characteristics available is: area, width, shape-index, i-e-ratio, and connectivity. The definition of a corridor must contain, at a minimum, a constraint on the maximum allowable width, and a constraint on the minimum allowable width. The definition of a matrix must contain, at a minimum, a constraint on the connectivity. The definition of a patch may have zero or more constraints. The semantic details of defining landscape elements of different types is covered in Chapter IV.

Assuming now that a set of landscape elements have been defined, a desired spatial configuration of these elements may now be specified with the def-landscape expression. The defined landscape, again given an arbitrary <name>, consists of landscape elements specified by the component-list which meet inter-element constraints specified by the constraint-list. Consider the following example which expresses the definition of a landscape suitable for location of a water park:

```
(def-landscape  water-park
    component-list     =   ((patch undeveloped-patch) (patch lake-patch)
                            (corridor river-corridor))
    constraint-list  =   ((undeveloped-patch ?u1) (lake-patch ?l1)
                          (river-corridor ?rc)
                          (! (adjacent ?u1 ?rc))
                          (! (< (min-dist ?u1 ?l1) (400 "feet")))
)
```

The component-list consists of entries of the following form:

```
(<element-type> <name>)
```

The names undeveloped-patch and river-corridor are landscape elements that were previously defined. The component-list does not say which elements must be in the landscape, but rather which element types will be considered by the constraints.

The constraint-list is a list of zero or more individual constraints. The above constraints express the following statement, "This landscape must contain an undeveloped patch, a lake patch, and a river corridor, such that the undeveloped patch is adjacent to the river, and within 400 feet of the lake." There are a number of syntactical forms and operators allowed for entries in the constraint-list. The variety is best expressed by a series of examples:

```
(lake-patch ?a) (undeveloped-patch ?b) (river-corridor ?c) (forest-matrix ?d)
(! (adjacent ?a ?b))
(! (> (area ?a) (area ?b)))
(! (< (i-e-ratio ?a) (i-e-ratio ?b)))
(! (< (min-dist ?a ?b) (2000 "feet")))
(! (< (min-dist ?a ?b) (min-dist ?b ?c)))
(! (for-all lake-patch (< (min-dist ?b lake-patch) (600 "feet"))))
```

All but the last are self-explanatory. The "for-all" construct is a form of universal quantification over a given element type. This last constraint requires that any undeveloped patch (?b) found must be within 600 feet of all lake patches. Clearly there is room for considerable expansion of the constraint language. A small but sufficient subset has been implemented for this thesis.

The landscape definition is possible due to the prior definition of landscape elements, which in turn are possible due to the prior definition of land types. It is important to realize that the

landscape expression is truly just a definition. There may be zero or more instances of it in the GIS data. The same holds for landscape element definitions.

The next step in analysis is landscape instantiation. The generate-landscapes expression requests that the system find all instances of a defined landscape. For example, the expression (generate-landscapes water-park 'full) returns a list of landscape instances. This list may be "stored" by using the above expression in conjunction with the define expression of scheme, giving (define landscape-list (generate-landscapes water-park 'full)). Each element on this list is a unique landscape instance representing a unique interpretation of the GIS data subject to the constraints of the water-park definition.

The final step in analysis is landscape query. Each instance returned by generate-landscapes may be examined. The most useful way to do this is to request that each instance produce a macGIS map. This is expressed using the send expression in conjunction with a scheme language construct called for-each:

```
(for-each
    (lambda (instance)
        (send instance generate-GIS-map))
    landscape-list))
```

Here, each instance in the landscape-list is sent a "generate-GIS-map" command. A series of data files will be written out which may subsequently be viewed from within macGIS. Each data file is a map representing the desired landscape configuration, with each element labelled as in the definitions.

Two Analysis Tasks

Two analysis tasks will be presented here without solution. Our goal is to demonstrate the use of the landscape language in specifying landscape types. Chapter V will be devoted to solving these tasks. The solutions will be developed using LRM, a prototype application. The first task involves GIS data which was contrived for the purpose of testing the landscape language. The second task involves GIS data collected from the Mt. Pisgah region of Lane County, Oregon.

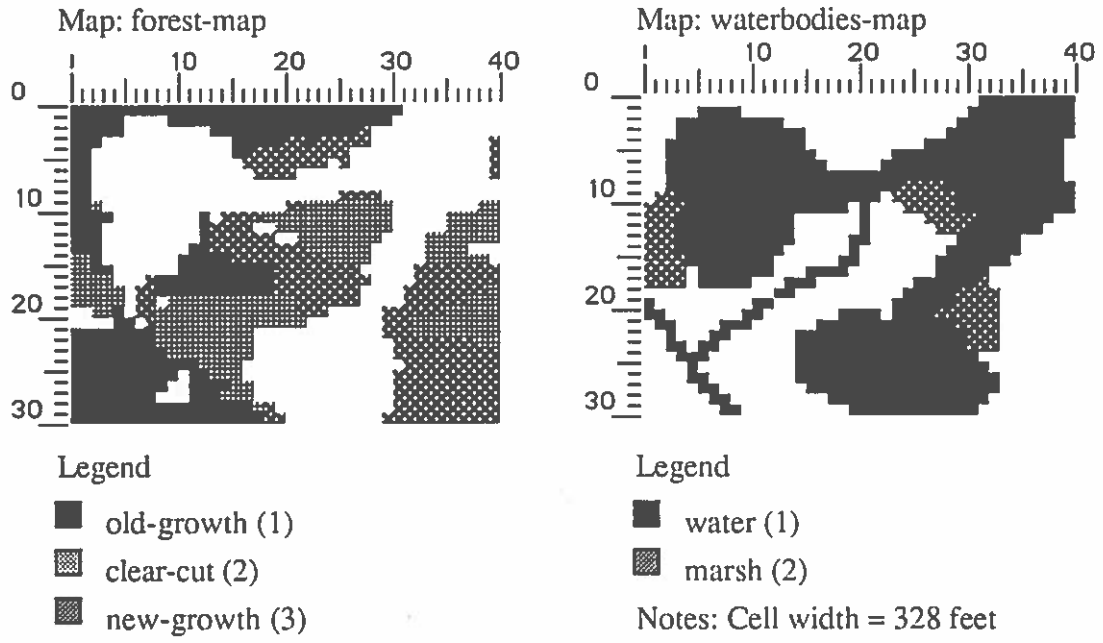


Figure 7. Maps for Analysis Task 1

The first analysis task is based on the two GIS maps shown in Figure 7. The first map, called forest-map, shows the distribution of old-growth, new-growth, and clear-cut forest for a fictional region. The second map, called waterbodies-map, shows the lake and river system for the same region. The values shown next to the map legend entries are the actual values contained in the cells of the map.

The task is to locate all lake-river-lake instances in a forested area such that one lake is the source of water, and the other lake the sink. In other words, lake-river-lake instances such that one lake feeds the other. Assume, for example, that the larger lake is always the source in such a system. Also, narrow streams are to be excluded. This goal is expressed as follows:

```
(def-data-layer forest-land
   creation-script =  (("isolate forest-map assign 1 to 1 through 3 for forest-layer.")
                       (load "forest-layer"))
)
(def-data-layer water
   creation-script =  (("isolate waterbodies-map assign 1 to 1 for waterbody-layer.")
                       (load "waterbody-layer"))
)

(def-element corridor river-corridor
   data-layer      =  water
   constraint-list =  ((> width (400 "feet")) (< width (2500 "feet")))
)

(def-element patch lake-patch
   data-layer      =  water
   constraint-list =  ((> area (12000000 "sq feet"))))
```

```
(def-element matrix forest-matrix
   data-layer      =  forest-land
   constraint-list =  ((< connectivity (400 "feet")))
)

(def-landscape lake-river-lake-system
   component-list  =  ((corridor river-corridor)
                       (patch lake-patch)
                       (matrix forest-matrix))


   constraint-list =  ((lake-patch ?l1) (lake-patch ?l2)
                       (river-corridor ?r) (forest-matrix ?fm)
                       (! (adjacent ?l1 ?r)) (! (adjacent ?l2 ?r))
                       (! (> (area ?l1) (area ?l2))))
)
```

The second analysis task is based on the GIS maps shown in Figure 8. The map legends are omitted due to size. The first map, landuse, gives detailed information about the use of land for the Mt. Pisgah region. Among the data are such items as housing, heavy industry, mining, picnic grounds, timber harvesting, etc. The second map, vegetation, shows the distribution of forest, grassland, agricultural land, and orchards. The third map, waterbodies, simply displays all open water in the region.

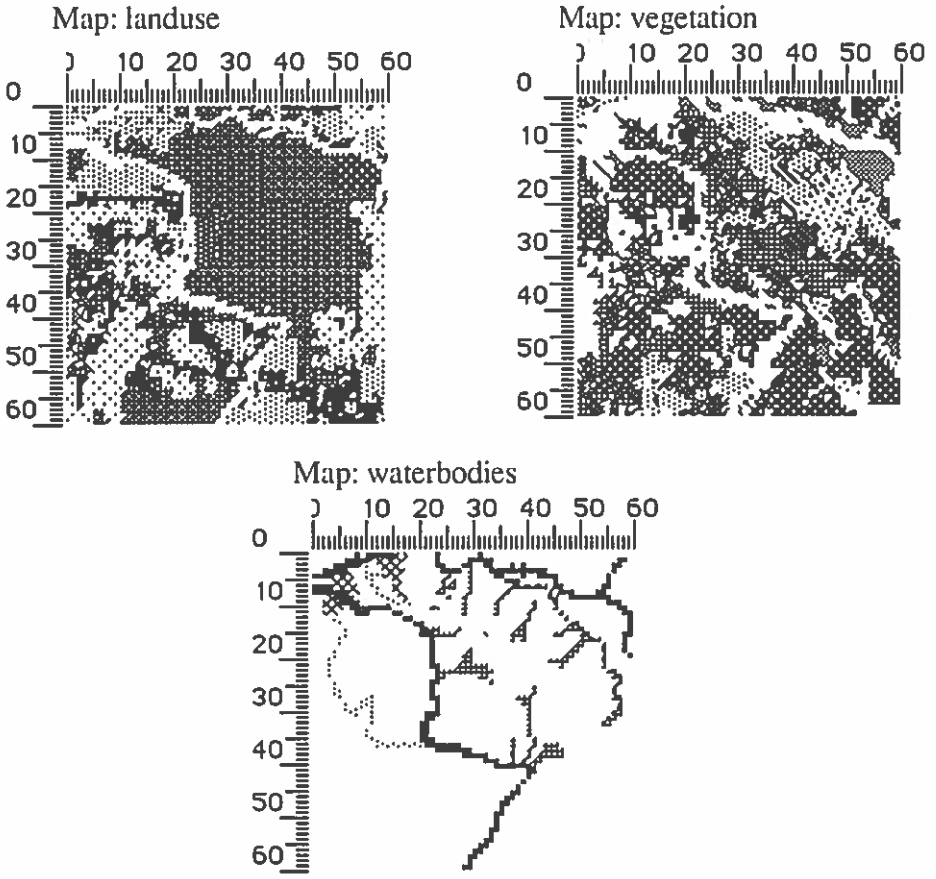


Figure 8. Maps for Analysis Task 2

The task is to locate two orchards suitable for an experiment in pest control. A natural predator will be introduced into the "test" orchard and the other orchard will serve as a control. A number of restrictions apply to the suitability of the orchards. First, they must be at least 500,000 square feet in size. Second, they must be at least 10,000 feet apart to avoid the possibility of the natural predator migrating from the test orchard to the control orchard. Third, the orchard chosen for the test must be at least 2,000 feet from any source of noise, such as mining, timber harvest, and heavy industry. Finally, since the natural predator nests at night on river snags and feeds elsewhere during the day, the test orchard must be within 2,000 feet of a river . This goal is expressed as follows:

```
(def-data-layer orchard
   creation-script =   (("isolate VEGETATION assign 1 to 1 for ORCHARD.")
                        (load "ORCHARD"))
)

(def-data-layer high-noise
   creation-script =   (("isolate LANDUSE assign 1 to 25 1 to 26 1 to 29 for NOISE")
                        (load "NOISE"))
)
```

```
(def-data-layer river-layer
   creation-script =  (("isolate WATERBODIES assign 1 to 1 for RIVERS.")
                       (load "RIVERS"))
)

(def-element corridor  river
   data-layer      =  river-layer
   constraint-list =  ((> width (50 "feet")) (< width (700 "feet")))
)

(def-element patch  orchard-patch
   data-layer      =  orchard
   constraint-list =  ((> area (500000 "sq feet")))
)

(def-element patch  noise-source
   data-layer      =  high-noise
   constraint-list =  ((> area (700000 "sq feet")))
)
```

Finally, we can express the desired landscape configuration which represents the task description.

```
(def-landscape pest-control-test
   component-list     =  ((corridor river) (patch orchard-patch) (patch noise-source))

   constraint-list    =  ((orchard-patch ?op1)          ; test orchard
                          (orchard-patch ?op2)          ; control orchard
                          (river ?r)
                          (! (> (min-dist ?op1 ?op2) (10000 "feet")))
                          (! (< (min-dist ?op1 ?r) (2000 "feet")))
                          (! (for-all noise-source
                                 (> (min-dist ?op1 noise-source) (2000 "feet")))))
)
```

CHAPTER IV

THE OPERATIONAL LANDSCAPE LANGUAGE - SEMANTICS

This chapter describes the core of the Landscape Reasoning Module, a landscape language interpreter. The interpreter has been implemented in MacScheme with SCOOPS, an object-oriented language extension. The interpreter processes expressions of the landscape language one at a time using the standard read/eval/print loop. The language expressions are divided functionally into three groups: definition of landscape, landscape element, and data-layer classes; instance generation; and instance query. Expressions of the first group serve to define a working set of landscape definitions for the analysis task. Expressions of the second group invoke interpretation of the GIS data subject to the landscape definitions. Expressions of the third group allow the results of the interpretation to be queried.

The first section of this chapter describes the general mechanism underlying the definition and instantiation of object classes that results from processing the corresponding expressions. The second section addresses, in detail, the generation of instances of the landscape element classes. This effects a mapping from GIS data to landscape elements. The third section addresses the generation of instances of the landscape class. An effort has been made to present only the central ideas and to leave out the non-critical bookkeeping details of the algorithms. The final section presents the Landscape Reasoning Module (LRM), a complete application built around the landscape language interpreter.

Definition, Interpretation, and Query

Spatial analysis with the landscape language proceeds in three steps. First, a complete landscape must be defined, including the GIS data on which it is based. Second, the specified GIS data is interpreted according to the landscape definition. The result is a set of zero or more instances of the landscape definition. Finally, the instances may be examined via commands and queries.

Class Hierarchy

The first step of spatial analysis with GIS involves developing a set of working landscape definitions and specifying the relevant GIS data-layers. In LRM, these definitions are supplied via the expressions def-data-layer, def-element, and def-landscape. These expressions define sub-classes of a pre-defined object class hierarchy. In other words, they define a number of object "types" for the system.

The power of the object-oriented approach is its ability to facilitate the grouping of data and operations into objects representative of real-world entities. This resulted in its selection as a basis for representing landscape definitions. A class-based inheritance approach is taken, so the methods and data structures of the parent class are inherited, requiring the user only to specify information unique to a sub-class.

The landscape as a whole is an object. Each element of the landscape is also an object itself. Each landscape element is derived from a corresponding data-layer object. Figure 9 shows the class hierarchy upon which objects in the system are based. This class hierarchy is a set pre-defined object classes, specializations of which are used to define a particular landscape.

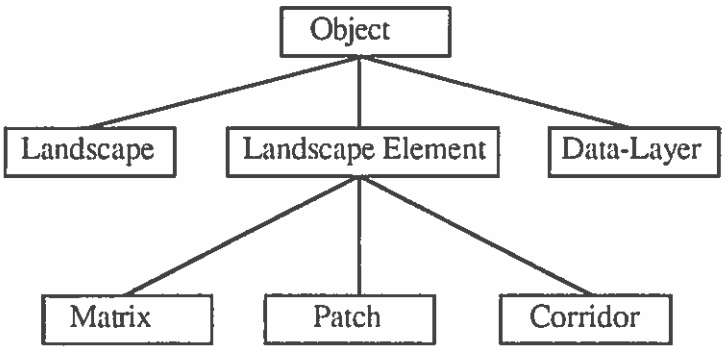


Figure 9. Landscape Class Hierarchy

Message passing is the mechanism by which objects communicate with each other. Each object has a set of methods, or procedures, which defines the messages it will respond to. The internal state of objects may not be directly manipulated, but may be set or read by sending messages. Message passing can also define the flow of execution control within a set of objects. In LRM, for example, an initialize message sent to one object often results in a propagation of initialize messages to other objects.

Given the pre-defined classes of Figure 9, the definition of a particular data-layer, landscape element, or landscape (via the expressions def-data-layer, def-element, and def-landscape) results in the creation of a new sub-class. The new sub-class inherits all the data structures and methods of its parent class. The parent class provides the functionality necessary for generating instances of the sub-class. As such, the user supplies only the information unique to that definition. Data-layer objects are defined in terms of the GIS database. Landscape element objects are defined in terms of data-layer objects and intra-element spatial constraints. Landscape objects are defined in terms of landscape element objects and inter-element constraints. In other words, to define a particular data layer, the user supplies a creation script for extracting data from the GIS. To define a particular matrix, patch, or corridor, the user supplies a reference to a data-layer definition and a set of intra-element spatial constraints. For a landscape, the user supplies a component list (references to matrix, patch, and corridor definitions), and a set of inter-element constraints.

Object Instantiation

The second step of spatial analysis involves interpreting the GIS data given the set of working landscape definitions. Interpretation of the GIS data is invoked via the make-inst and generate-landscapes expressions. These expressions generate instances of defined sub-classes. Each instance is an interpretation of the GIS data subject to the definitions of the sub-class.

Recalling the example analysis tasks of Chapters I and III, we assume the user has entered a set of landscape definitions. The interpreter now has an environment of task specific sub-classes.

The user is free to request the generation of instances of any of these sub-classes. Analysis normally proceeds by requesting that the system find all possible instances of the landscape sub-class via the generate-landscapes expression. Alternately, the user may create single instances of data-layers or landscape elements as desired using the make-inst expression.

The expression (make-inst <data-layer-def>) creates an instance of <data-layer-def> and immediately sends it the message "initialize". The initialize method for the data-layer class executes the following actions:

```
(begin
    (execute-creation-script creation-script)
    (load-resultant-GIS-data-layer)
    (load-data-layer-dimensions-and-units)
    (find-all-connected-components))
```

The result is an instance of <data-layer-def>, an object, which contains the GIS data-layer specified by the creation-script, the dimensions and units of the data-layer, and a list of all connected components in the layer.

The expression (make-inst <landscape-element-def>) creates an instance of <landscape-element-def> and immediately sends it the message "initialize". The initialize method depends on the parent class of <landscape-element-def>, eg. matrix, patch, or corridor. The initialize method attempts to locate a connected component in the associated <data-layer-def> that meets the intra-element constraints provided in <landscape-element-def>. As such, an instance of <data-layer-def> must be generated first. While the method differs somewhat between the matrix, patch, and corridor classes, the general algorithm is as follows:

```
(begin
    (set! found-valid-clump #f)
    (set! data-layer-inst (make-inst <data-layer-def>))
    (set! clump-info (send data-layer-inst get-clump-info))
    (while  (not found-valid-clump)
            (begin
            (set! next-clump (get-next-clump clump-info))
            (set! found-valid-clump
                    (verify-constraints next-clump intra-element-constraints))))
    (if  found-valid-clump
        next-clump
        #f))
```

If successful, the result is an instance of <landscape-element-def>. This instance contains a representation of the connected component, the location in the data-layer in which it was found, and the complete set of intra-element spatial characteristics computed for it. Many details of this process will be presented later in the chapter. One detail to note is that a subsequent invocation of (make-inst <landscape-element-def>) should return an instance containing a different connected component. When all possible instances of <landscape-element-def> have been found, make-inst returns #f.

The expression (generate-landscapes <landscape-def> <mode>) creates a list of one or more instances of <landscape-def>. Ignoring the possible values for <mode>, we will assume for the moment that it is "full". Generate-landscapes begins by finding all instances of the landscape element definitions in the component list of <landscape-def>. The procedure "satisfy" is then run on the resulting instances and the constraint list of <landscape-def>. This produces a set of bindings; the bindings being between variables representing the desired element configuration and

the element instances found. For each binding, a unique instance of <landscape-def> is generated. The final list is returned. The details of this process will be covered later. For the moment, the algorithm is roughly as follows:

```
(begin
    (set! inst-list (make-all-instances (component-list <landscape-def>)))
    (set! bindings (satisfy inst-list constraint-list))
    (map
        (lambda (binding)
            (begin  (set! inst (make-inst <landscape-def>))
                    (send inst  set-binding binding)
                    inst))
        bindings))
```

Querying Instances

Class definition and instance generation are the core of the Landscape Reasoning Module. Without the ability to retrieve information from the instances, however, they are of little use. The send expression is the traditional mechanism in object oriented systems for interacting with objects. Instances of each class can accept a number of messages and return a variety of information. Currently, there is a minimal set of methods provided for retrieving information. The most useful of these is the generate-GIS-map method of the landscape class: (send <landscape-inst> generate-GIS-map). This produces a GIS data file which may subsequently be viewed within macGIS. Other methods of the landscape class return individual lists of the matrix, patch, and corridor instances found. Methods of the landscape-element classes return information such as area, width, shape index, etc. Methods of the data-layer class return information such as map cell width and cell units, etc.

Mapping from GIS Data to Landscape Elements

The two central computational tasks of our implementation are the generation of landscape element instances and the generation of landscape instances. This section describes the former in detail. In order to do this, the instance variables and methods of the data-layer class and landscape element classes must be explained in detail. While it is possible for a user to make instances of data-layer sub-classes, this is generally controlled by the process of landscape element instance generation. Taken as a whole, the generation of landscape element instances effects a mapping from GIS data to landscape elements. Below is a description of the process.

Each landscape element definition contains a reference to a previously defined data-layer. When an instance of that element is generated, an instance of the associated data-layer is generated first (if it has not been done already). The data-layer instance contains a list of all connected components (or clumps) from the GIS data layer. Each item in this list is a data structure with the following components: a two-dimensional array, or "bounding box", which is a subset of the full data layer, and which inscribes a single connected component; and four integers which supply the upper-left and lower-right coordinates of the bounding box's location in the full data layer.

In order to instantiate a landscape element, a pass through the list of clumps is made. The spatial characteristics of each clump are compared with the intra-element constraints supplied in the

landscape element definition. The first clump to satisfy these constraints is copied into the landscape element instance, marked as occupied in the data-layer instance, and the process terminates. Subsequent instantiations of the same landscape element sub-class examine unoccupied clumps in the same data-layer instance.

The process of generating landscape element instances described above is only a rough sketch of the method used for the patch class. There are several important differences between this and the methods for the corridor class and matrix class. When instantiating a corridor sub-class, the pass through the list of data-layer clumps must search within each clump for linear elements. A potential corridor may be connected with other non-linear clumps, and therefore be represented as a single clump in the data-layer instance. The instantiation method for corridors must attempt to separate out the linear portion(s). When instantiating a matrix sub-class, a potential matrix clump does not have to be fully connected, as is required for patches and corridors. These details will be addressed below.

Data-Layer Class

The data-layer class provides an interface between the macGIS data files and the needs of the landscape element classes. Two primary functions are served. First, the creation-script provided by the user is executed, resulting in a single macGIS data file. The map "image" is read in from this file along with its parameters: rows, columns, cell width, and cell units. Secondly, each connected component in this "image" is located, copied into a bounding box, and placed on the clump-info list. Figure 10 shows the instance variables and methods of the data-layer class.

Class: Data-Layer SuperClass: Object	
Instance Variables	Methods
creation-script cell-data rows cols cell-width cell-units clump-info occupied	initialize get / set display-layer

Figure 10. Data-Layer Class

When a user creates a sub-class of data-layer, it is only necessary to supply the creation-script. The creation-script, as shown in Chapter III, is a sequence of macGIS commands for producing a binary macGIS map data file. The initialize method executes the creation-script and reads the result into cell-data, a two-dimensional array of integers. Cells which contain the value one (black) represent land defined by the creation-script. Cells of value zero (white) are undefined background. The initialize method then proceeds to find all connected components in cell-data

using two procedures: get-bounding-boxes and connected-components. The map image parameters are then read into the instance variables rows, cols, cell-width, and cell-units. The cell-width and units are necessary later for computing spatial characteristics and converting between units.

The procedure connected-components locates and numbers each clump in cell-data. A global variable "directions" determines whether the procedure performs a search for four-connected or eight-connected components. The procedure get-bounding-boxes copies each uniquely numbered component into a minimal inscribing two-dimensional array, or bounding box. A list of the bounding boxes is returned and assigned to clump-info. The algorithm is given below in a scheme-like pseudo-code. The connected-components procedure is based on a depth-first algorithm presented in *The Elements of Artificial Intelligence* [Tanimoto 1987]. Figure 11 graphically represents the process.

```
(define get-bounding-boxes (lambda (cell-data rows cols)
    (let      ((clump-locations (connected-components cell-data rows cols)))
        (map
            (lambda (clump-loc)
                (get-bounding-box cell-data clump-loc))
            (clump-locations)))))

(define connected-components (lambda (cell-data rows cols)
    (scan (negate cell-data rows cols) rows cols)))

; Negates cells to facilitate keeping track of region growing.
(define negate (lambda (cell-data rows cols)
    (for-each-cell
        (lambda (cell)
            (set! cell (* cell -1)))
        cell-data)))

; Scan image row-by-row. For each new negative cell identified (eg. not in a
;    clump yet), do a depth-first search (dfs) for possible additional cells. Clumps
;    are numbered consecutively.
(define scan (lambda (cell-data rows cols)
    (let      ((count 0)
              (clump-locations '()))
        (for-each-cell
            (lambda (cell)
                (if  (< (value cell) 0)
                    (begin
                        (set! count (+ count 1))
                        (set! clump-locations (append clump-locations (loc cell)))
                        (dfs cell-data count cell))))
            cell-data))))
```

```
; If the value of cell is negative, set it to 'count'. Explore neighboring cells (based
;    on values in 'directions') via a depth-first search. Continue setting cell values to
;    'count' until no more negative neighboring cells are found.
(define dfs (lambda (cell-data count cell)
    (if  (< (value cell) 0)
        (begin
            (set! cell count)
            (for-each
                (lambda (next-direction)
                    (dfs cell-data count (next-cell-to-explore cell next-direction)))
                directions)))))

(define eight-connected     '((-1 -1) (-1 0) (-1 1) (0 -1) (0 1) (1 -1) (1 0) (1 1)))
(define four-connected      '((-1 0) (0 -1) (0 1) (1 0))
(define directions eight-connected)     ; Default mode
```

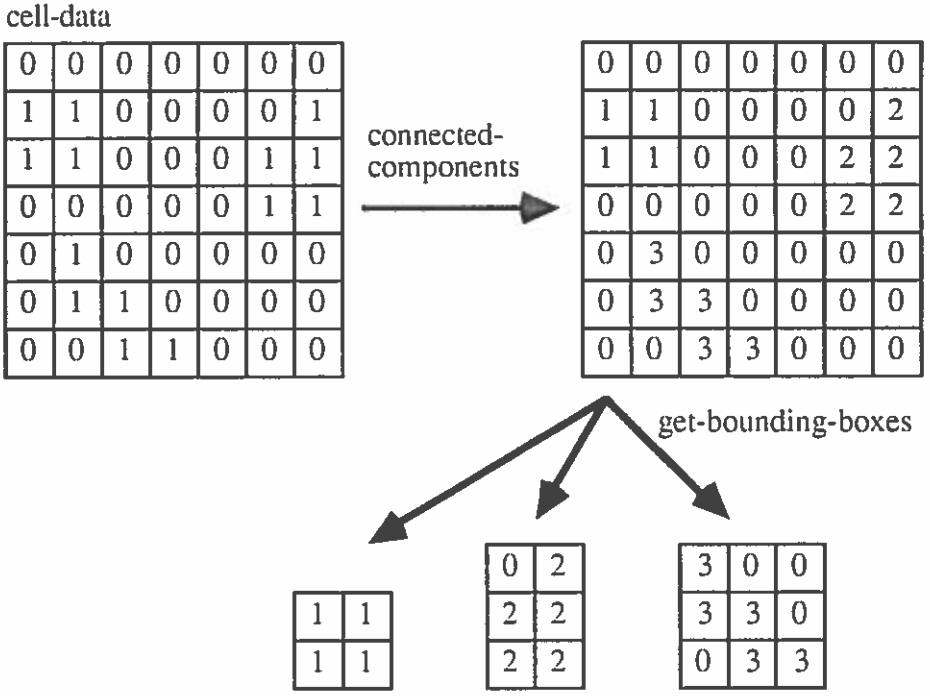


Figure 11. Finding Connected Components

All instance variables of the data-layer class, except occupied, are now initialized. The set of all connected components is available in the clump-info list. Each time a landscape element instance selects a clump, it is placed on the occupied list. In other words, there is a one-to-one relationship between landscape elements and clumps.

The methods get and set of the data-layer class provide a means to access the instance variables. These are used primarily by the algorithms for generating instances of landscape elements. A display-layer method is available for debugging purposes.

Landscape Element Classes

The three landscape element classes provide the functionality necessary for finding instances of themselves in the associated data-layers. The instantiation methods differ somewhat between classes and will therefore be examined individually. The first step is common to all three; an instance of the associated data-layer must be generated. If one has already been generated, eg. by another landscape element which references the same data-layer, the existing one will be used. The clumps of the data-layer are then subjected to the intra-element constraints and a suitable one is selected. It is this latter step which is unique for each element class.

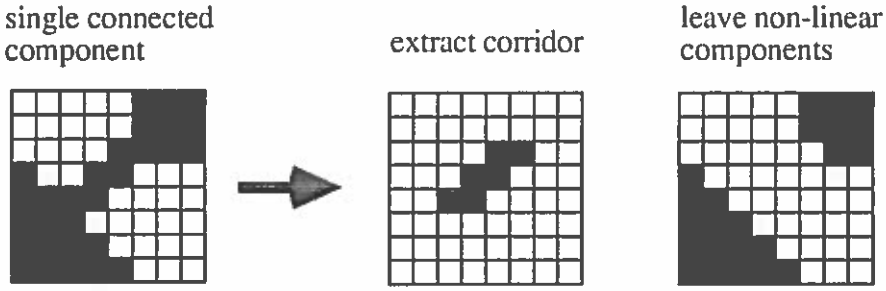


Figure 12. Extracting Corridors

The three classes will be examined in the following order: corridor, patch, and matrix. The reason for this is more than organizational. The corridors of a landscape must be identified before the patches or matrix. Locating an instance of a corridor may involve extracting a linear element from a single clump, as in Figure 12. This in turn may introduce new clumps to the data-layer. As such, the patches or matrix should not be identified until all instances of corridors are found.

Corridor Class

The instance variables and methods of the corridor class are shown in Figure 13. When the user creates a sub-class, it is only necessary to supply a data-layer name and constraint list. The constraint list must contain at least a minimum and maximum allowable corridor width, eg. ((> width (50 "feet")) (< width (200 "feet"))). This determines what is and is not considered a linear element of a data-layer clump. Beyond this, the user may specify additional intra-element constraints such as area, shape index, etc. The remaining instance variables are initialized when a suitable corridor clump is found.

Class: Corridor SuperClass: Landscape Element	
Instance Variables	Methods
data-layer constraint-list bound-box bound-box-info area edge-len shape-index i-e-ratio	initialize get / set

Figure 13. Corridor Class

The initialize method executes a pass through the clump-info list of data-layer. For each clump, a method known as shrink-propagate [Rosenfeld and Kak 1976] is applied to isolate linear elements within the specified width range. Shrink-propagate uniformly shrinks and re-expands a clump, the result being that linear elements do not survive. When used in conjunction with the image difference operator, linear and non-linear components may be separated.

Figure 14 shows the shrink-propagate method graphically. The first row shows the isolation of all corridors up to maxwidth by applying one shrink-propagate and one image subtraction. Essentially, the shrink-propagate produces a mask which covers the non-linear clumps. The second row performs another image subtraction in order to isolate the non-linear clumps (rest-box) which resulted from the first shrink-propagate. The third row shows the isolation of corridors which are more narrow than minwidth. The fourth row performs one final image subtraction in order to remove the narrow linear elements.

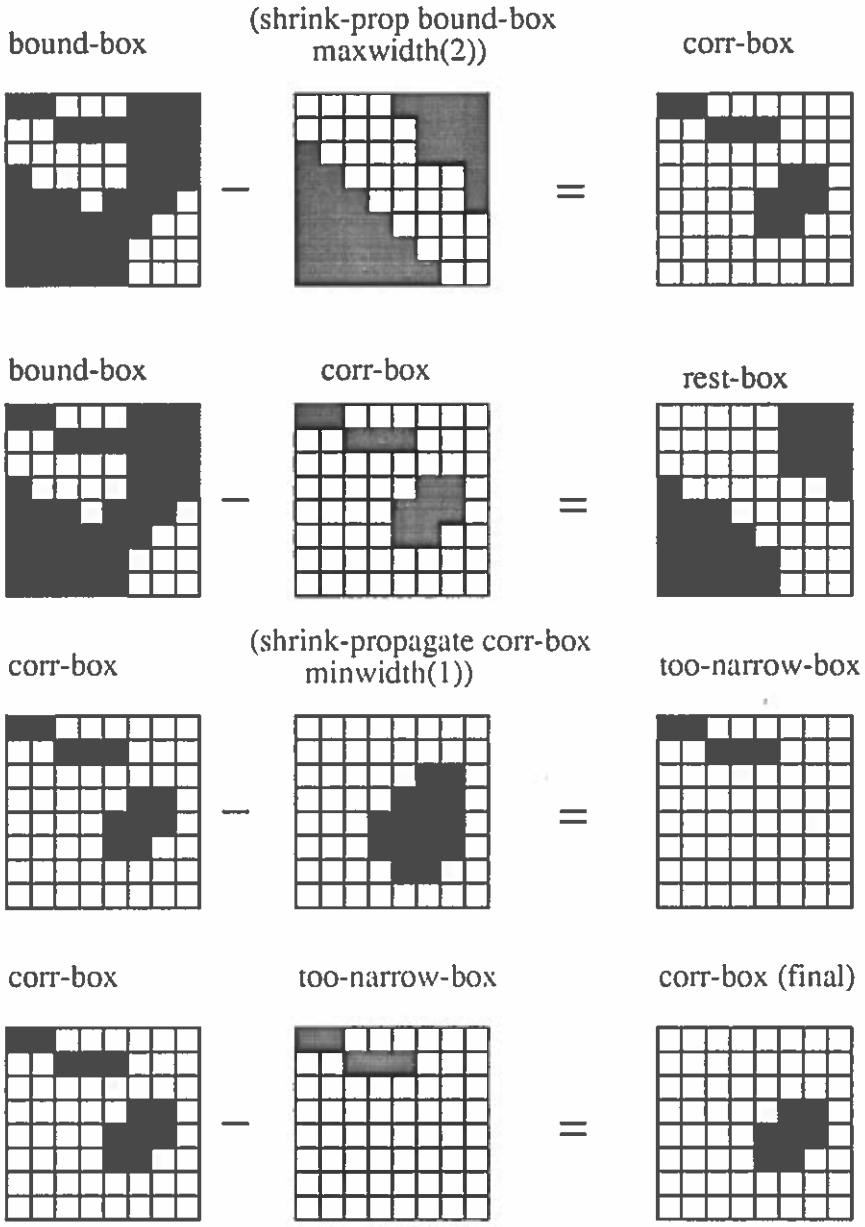


Figure 14. Shrink-Propagate Method for Extracting Linear Components

If the shrink-propagate process introduces new clumps (rest-box), they are added to the tail of the clump-info list. Any linear elements found are tested against the remaining constraints and the first one which satisfies them is copied into the bound-box variable and marked as occupied. The remaining are placed on the head of the clump-info list. The corridor class instance is then returned and the process terminates. The rough algorithm is given below.

```
(define initialize (lambda ()                    ; initialize method for corridor class
   (while  (and    (not found-valid-clump)
                   (not (null? clump-info)))
           (begin
           (set! bound-box (head clump-info))
           (set! sp-box (shrink-propagate bound-box maxwidth))
           (set! corr-box (image-difference bound-box sp-box))
           (set! rest-box (image-difference bound-box corr-box))
           (set! sp-box (shrink-propagate corr-box minwidth))
           (set! too-narrow-box (image-difference corr-box sp-box))
           (set! corr-box (image-difference corr-box too-narrow-box))
           (update-clump-info corr-box rest-box)
           (set! bound-box (head clump-info))
           (set! spatial-chars (compute-chars bound-box))
           (set! found-valid-clump (verify-constraints spatial-chars constraint-list))))

   (if found-valid-clump
       (begin
           (mark-as-occupied bound-box clump-info)
           (set-instance-variables)
           (return instance))
       #f)))
```

Patch Class

The instance variables and methods of the patch class are shown in Figure 15. When the user creates a sub-class, it is only necessary to supply a data-layer name and constraint list. The constraint list may have zero or more constraints. If no constraints are given, the first non-occupied clump of data-layer will be selected. The remaining instance variables are initialized when a suitable patch clump is found.

Class: Patch SuperClass: Landscape Element	
Instance Variables	Methods
data-layer constraint-list bound-box bound-box-info area edge-len shape-index i-e-ratio	initialize get / set

Figure 15. Patch Class

The initialize method executes a pass through the clump-info list of data-layer. Each clump is tested against the spatial constraints in the constraint-list and the first one which satisfies them is copied into the bound-box variable. The selected clump is marked as occupied in data-layer and the patch class instance is then returned. The algorithm is shown below.

```
(define initialize (lambda ()                    ; initialize method for patch class
   (while (and    (not found-valid-clump)
                  (not (null? clump-info)))
          (begin
          (set! bound-box (head clump-info))
          (set! spatial-chars (compute-chars bound-box))
          (set! found-valid-clump (verify-constraints spatial-chars constraint-list))))


   (if found-valid-clump
       (begin
           (mark-as-occupied bound-box clump-info)
           (set-instance-variables)
           (return instance))
       #f)))
```

Matrix Class

The matrix class is distinct from the patch and corridor classes in two aspects. One, when the data-layer associated with a matrix sub-class is made, the connectivity constraint, normally one, is temporarily extended according to the matrix definition. The constraint-list supplied by the user may specify a connectivity of two, for example. When the data-layer instance is generated, the connected-component algorithm will consider any cell up to a distance of two away as being part of a clump. The second major distinction is that there is only one matrix element per landscape. The clump of largest total area is selected as the matrix. As such, the user may not repeatedly generate distinct matrix instances as with corridors and patches.

Class: Matrix SuperClass: Landscape Element	
Instance Variables	Methods
data-layer constraint-list bound-box bound-box-info area	initialize get / set

Figure 16. Matrix Class

The instance variables and methods of the matrix class are shown in Figure 16. As before, the user supplies the data-layer name and constraint-list. The initialize method executes a complete pass through the clump-info list of data-layer. The clump of largest area is copied into bound-box. The matrix class instance is then returned. The algorithm is shown below.

```
(define initialize (lambda ()                    ; initialize method for matrix class
   (set! max-area -1)
   (while  (not (null? clump-info))
           (begin
           (set! temp-bound-box (head clump-info))
           (set! temp-area (area temp-bound-box))
           (if  (> area max-area)
               (begin
               (set! max-area temp-area)
               (set! area temp-area)
               (set! bound-box temp-bound-box)))))
   (return instance)))
```

Landscape Class

The landscape class provides a means for defining a desired landscape configuration, and a means for holding the results of finding an instance of that configuration. The instance variables and methods are shown in Figure 17. The user, when creating a sub-class, supplies the desired configuration in the component-list and constraint-list. The generate-landscapes procedure utilizes these lists to generate instances, of the landscape sub-class, which contain landscape elements meeting inter-element constraints. The landscape elements of a landscape instance are stored in the element-list variable. The generate-GIS-map method produces an output map which is viewable from macGIS. In this fashion, the landscape instances produced by generate-landscapes may be examined visually. Alternately, a user may retrieve the actual landscape element instances via the get method.

Class: Landscape SuperClass: Object	
Instance Variables	Methods
component-list constraint-list element-list	generate-GIS-map get / set

Figure 17. Landscape Class

The "analysis" of GIS data is carried out mainly by the generate-landscapes procedure. The specification as to what to do is given by the component-list and constraint-list. The landscape instances hold the results of this "analysis". Ideally, we would have preferred bundling the work of generate-landscapes into an initialize method of the landscape class, much like the initialize method of the landscape element classes. The lack of class variables and class methods in

SCOOPS, however, precluded this. We will therefore focus on the algorithms of the generate-landscapes procedure, keeping in mind that the result is a set of one or more instances of the landscape class.

The generate-landscapes procedure begins by generating all possible instances of landscape element types specified in the component-list. An algorithm known as satisfy is then applied to the set of landscape element instances and the constraint-list (see Figure 18). The constraint-list is essentially a pattern of variables and constraints which describes a spatial configuration of landscape elements. This pattern is matched against the set of instances. Satisfy finds all possible bindings of variables over the instances such that the pattern is consistent. The result is a list of lists. Each list contains bindings between the variables of the pattern and the landscape element instances. A landscape instance is then generated for each set of bindings. The landscape element instances of the binding set are copied into the landscape instance.

Given:

set-of-instances = ((lake-patch inst1) (lake-patch inst2)
(lake-patch inst3) (river inst4) (river inst5))

where:

inst1 and inst2 are both adjacent to inst4 and
inst2 and inst3 are both adjacent to inst5

constraint-list = ((lake-patch ?p1) (lake-patch ?p2) (river ?r)
(! (adjacent ?p1 ?r)) (! (adjacent ?p2 ?r)))

meaning:

find instances of connected lake-river-lake structures

Execute:

(satisfy set-of-instances constraint-list)

which returns:

(((?p1 inst1) (?p2 inst2) (?r inst4))
((?p2 inst1) (?p1 inst2) (?r inst4))
((?p1 inst2) (?p2 inst3) (?r inst5))
((?p2 inst2) (?p1 inst3) (?r inst5)))

Figure 18. Application of Satisfy

The algorithm for generate-landscapes is as follows:

```
(set! first-inst (make-instance <landscape-def>))
(set! component-list (send first-inst get-component-list))
(set! constraint-list (send first-inst get-constraint-list))
(set! element-list (find-all-instances component-list))
```

```
; First instance will always contain full set of landscape element instances.
(send first-inst set-element-list element-list)

(set! bindings (satisfy element-list constraint-list))

(set! landscape-instances (map (lambda (binding)
                                    (begin
                                    (set! inst (make-inst <landscape-def>))
                                    (send inst set-element-list binding)
                                    inst))
                               bindings))

(return (cons first-inst landscape-instances))
```

LRM - A Tool for Spatial Analysis of Landscapes

The Landscape Reasoning Module (LRM) is a prototype Macintosh application built around the mechanisms described in this chapter. Since the ultimate goal of this thesis is to facilitate spatial landscape analysis for the average GIS user, a menu-driven application was developed to isolate the user from some of the bookkeeping details of defining, instantiating, and querying landscape objects. The user-interface is still quite primitive, but does demonstrate the potential for a complete, robust implementation. This section continues by describing the architecture and user-interface of LRM. Chapter V provides solutions to two analysis tasks and demonstrates the operation of this application.

Figure 19 shows the general architecture of LRM. A standard Macintosh event-loop interface is built which associates menu items with the language expressions for defining, instantiating, and querying landscape objects. In addition, a means to enter, store, edit, and retrieve landscape definitions is provided. Communication with macGIS is implemented via shared data files for the creation-scripts, GIS map input, and GIS map output. LRM is designed to be co-executed with macGIS using MultiFinder. Ideally the two applications should be integrated, however time constraints precluded this. This places the burden on the user of switching activity between applications according to instructions given by LRM dialog boxes.

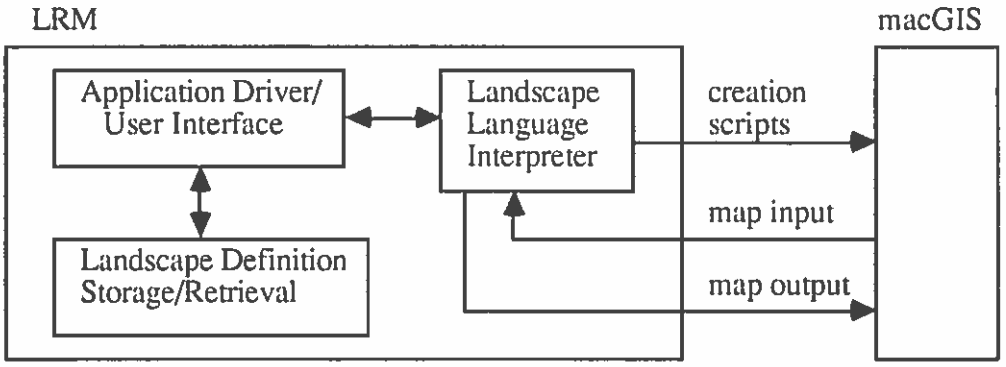


Figure 19. LRM Architecture

Launching LRM brings up a menu bar and status window as shown in Figure 20. The File and Edit menu entries allow the user to maintain files of landscape definitions. The def-data-layer, def-element, and def-landscape expressions for a particular analysis task are typically entered and stored in one file. The user may also build up a library of landscape definitions in a master file, and copy the definitions as needed into a task specific file. The Define menu entry allows the user to load a selected file of landscape definitions into LRM. In effect, each expression of the file is sent to the landscape language interpreter. The Generate menu entry has several items for invoking the generate-landscapes expression. The Examine menu entry allows the user to write out the landscape instances as macGIS map data files. Options currently allows the user to specify whether the connected-component procedure will generate four-connected or eight-connected clumps. The status window keeps the user apprised of progress, since some activities require considerable computing time.

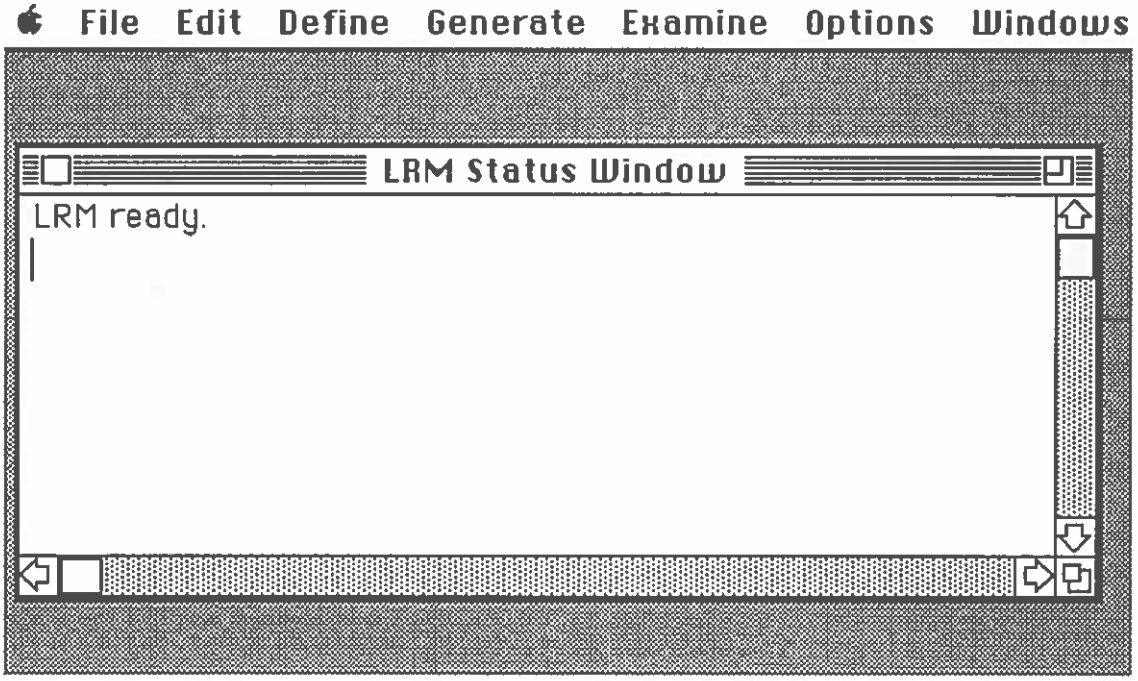


Figure 20. LRM Application Screen

The prototype implementation of LRM requires that it be co-executed with macGIS. Future implementations should fully integrate the two applications. Currently, the user must switch between applications during the course of analysis. When the user loads a set of definitions using the Define menu entry, the creation-scripts of the data-layer definitions are extracted and written to a file. This file contains a sequence of macGIS commands. The user must switch to macGIS via MultiFinder and execute these commands using the "Read Commands" menu item. The user may then switch back to LRM and proceed with generating instances. If the Examine menu entry is used to produce macGIS map data files, the user must again switch to macGIS to view them.

CHAPTER V

SOLUTION TO ANALYSIS TASKS

This chapter demonstrates the use of LRM in the context of the two analysis tasks presented in Chapter III. The interaction with LRM will be described briefly and the generated solution presented for each task. The full landscape definitions will not be repeated here, so it would be helpful to review the last section of Chapter III.

Rivers and Lakes

The first analysis task is to locate all lake-river-lake instances in a forested area such that one lake is the source of water, and the other lake the sink. The landscape class definition is as follows:

```
(def-landscape lake-river-lake-system
   component-list     =  ((corridor river-corridor)
                          (patch lake-patch)
                          (matrix forest-matrix))
   constraint-list    =  ((lake-patch ?l1) (lake-patch ?l2)
                          (river-corridor ?r) (forest-matrix ?fm)
                          (! (adjacent ?l1 ?r))
                          (! (adjacent ?l2 ?r))
                          (! (> (area ?l1) (area ?l2))))
)
```

Solution begins by launching both the macGIS and LRM applications. The New item of the File menu of LRM is selected in order to bring up an editing window for entering the landscape definitions. The definitions for forest-land, water, river-corridor, lake-patch, forest-matrix, and lake-river-lake-system are entered here. The contents of the window are then saved to a file.

The next step is to load the file of definitions. The "Load Definitions..." item of the Define menu allows selection of the file. Once the definitions are loaded, it is necessary to transfer to macGIS in order to execute the data-layer creation-scripts. MacGIS has a "Read Commands" menu item which allows execution of the creation scripts residing in the file "READ THESE COMMANDS." Once this is complete, one transfers back to LRM.

The landscape definitions are now loaded, and macGIS has prepared the binary data layers for LRM. The next step is to request that LRM generate all possible instances of the lake-river-lake-system landscape definition. The "Generate [full]..." item of the Generate menu entry initiates this process. When complete, the status window displays the number of instances that were generated. The "Generate GIS Map..." item of the Examine menu entry allows you to dump a macGIS map file for each instance generated.

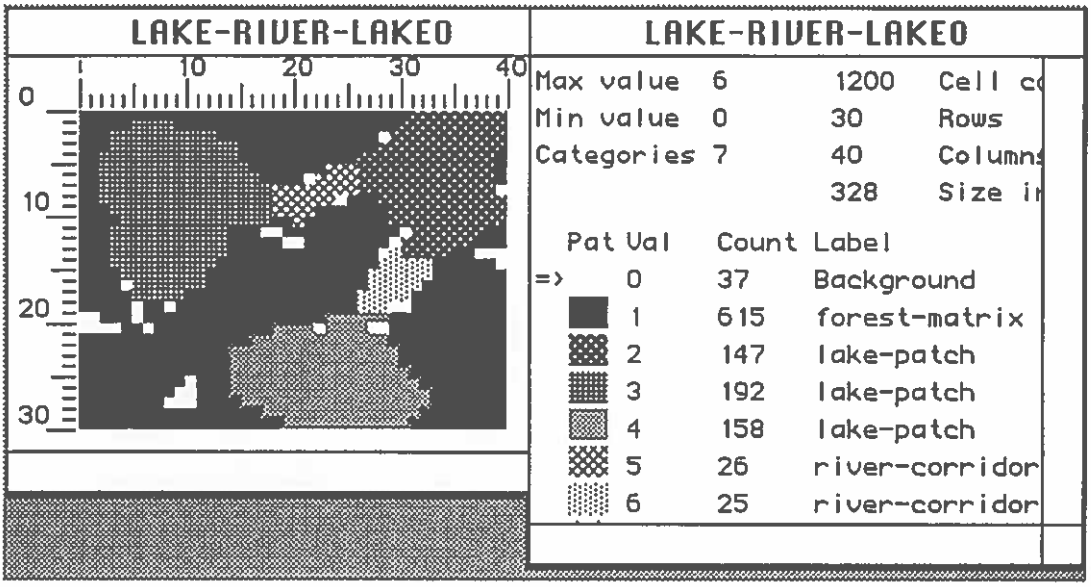


Figure 21. Landscape Elements Prior to Inter-Element Constraints

The map of Figure 21 shows all the landscape element instances found prior to applying the inter-element constraints of the landscape definition. Three lakes, two rivers, and a forest matrix have been found. Note that the smaller streams of the original waterbodies map have been eliminated by the intra-element constraint on river width (see Figure 7). Figure 22 shows the two maps produced which represent the two distinct instances of lake-river-lake systems which meet the inter-element constraints.

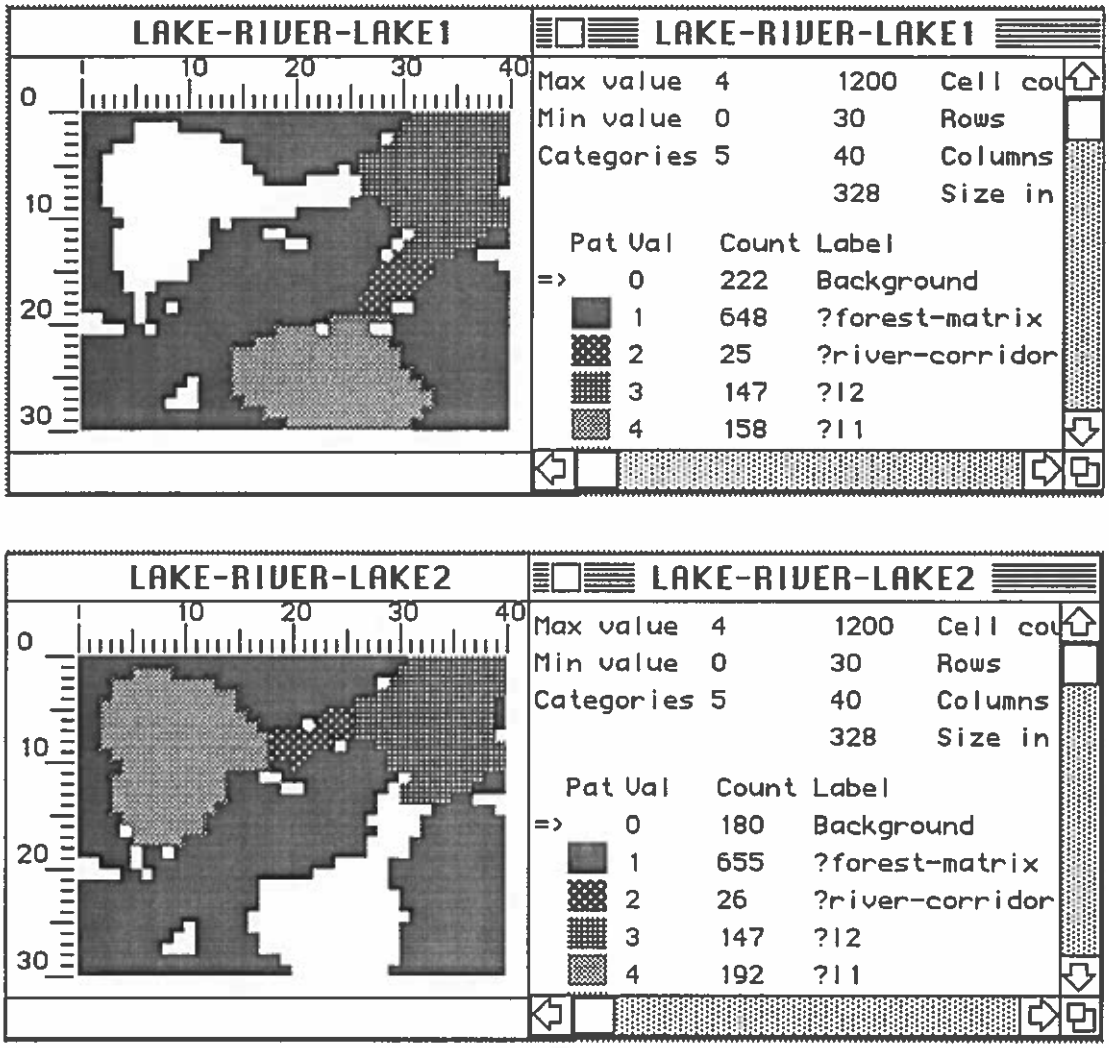


Figure 22. Lake-River-Lake: Two Solutions

The two solutions show a larger lake ?11 feeding into a smaller lake ?12. Had we removed the constraint that one lake be larger than the other, four solutions would have been generated. The two additional solutions would correspond to the permutations of the labels ?11 and ?12 on the two lake instances. With our constraint that ?11 be larger than ?12, this alternate labelling becomes inconsistent. While the generation of four instances would not pose a problem here, in general the permutations of element labelling can result in a large number of instances. This may be desired in some cases, but in others the alternate labelling provides no useful information.

Pest Control

The second analysis task is to locate two orchards suitable for an experiment in pest control. A natural predator will be introduced into the "test" orchard and the other orchard will serve as a control. A number of restrictions apply to the suitability of the orchards. First, they must be at least 500,000 square feet in size. Second, they must be at least 10,000 feet apart to avoid the possibility of the natural predator migrating from the test orchard to the control orchard. Third, the

orchard chosen for the test must be at least 2,000 feet from any source of noise such as mining, timber harvest, or heavy industry. Finally, since the natural predator nests at night on river snags and feeds elsewhere during the day, the test orchard must be within 2,000 feet of a river . The landscape definition, as shown in Chapter III, is as follows:

```
(def-landscape pest-control-test
   component-list     =  ((corridor river)
                          (patch orchard-patch)
                          (patch noise-source))
   constraint-list    =  ((orchard-patch ?op1)          ; test orchard
                          (orchard-patch ?op2)          ; control orchard
                          (river ?r)
                          (! (> (min-dist ?op1 ?op2) (10000 "feet")))
                          (! (< (min-dist ?op1 ?r) (2000 "feet")))
                          (! (for-all noise-source
                                 (> (min-dist ?op1 noise-source) (2000 "feet"))))))
```

As with the first task, the definitions are entered and loaded. The Generate menu item is used to initiate the process of generating landscape instances. Some preliminary discussion follows before presenting the solutions.

Figure 23 shows the candidate orchards prior to applying any constraints. The orchards have been numbered for discussion purposes. The intra-element constraint on minimum orchard size results in orchards 1, 4, 6, 7, and 9 being eliminated by the process of landscape element instantiation. Orchards 2, 3, 5, and 8 are the remaining patches to be considered by the inter-element constraints of the landscape definition. Figure 24 shows the various sources of man-made noise in the region. Here also, the smaller patches are eliminated by an intra-element area constraint.

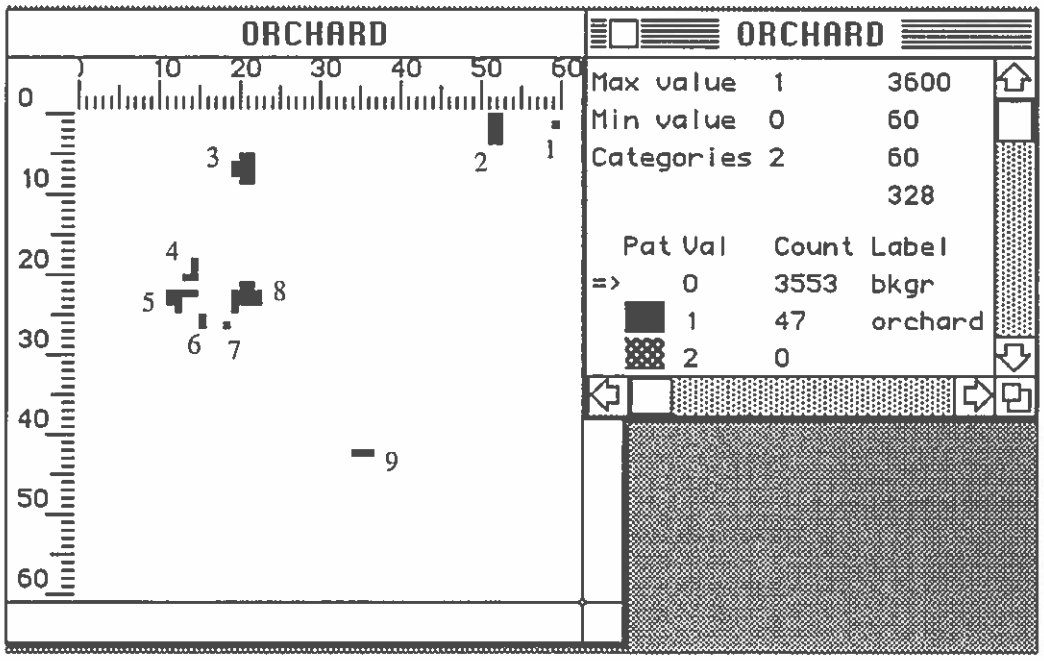


Figure 23. Map of All Orchards

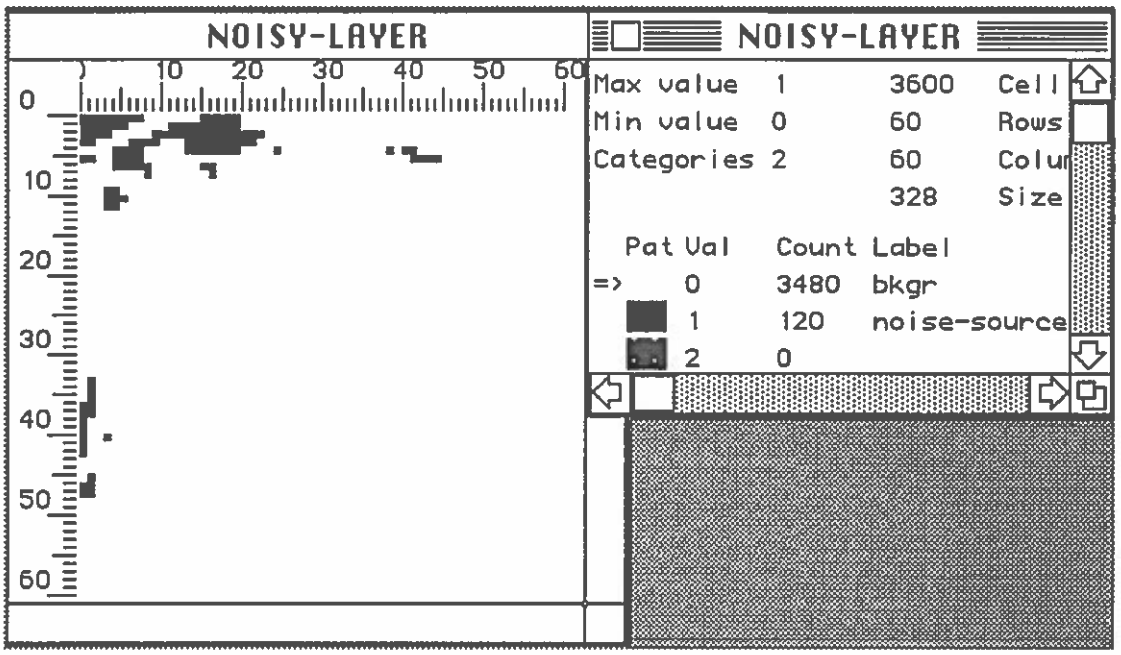


Figure 24. Map of Noise Sources

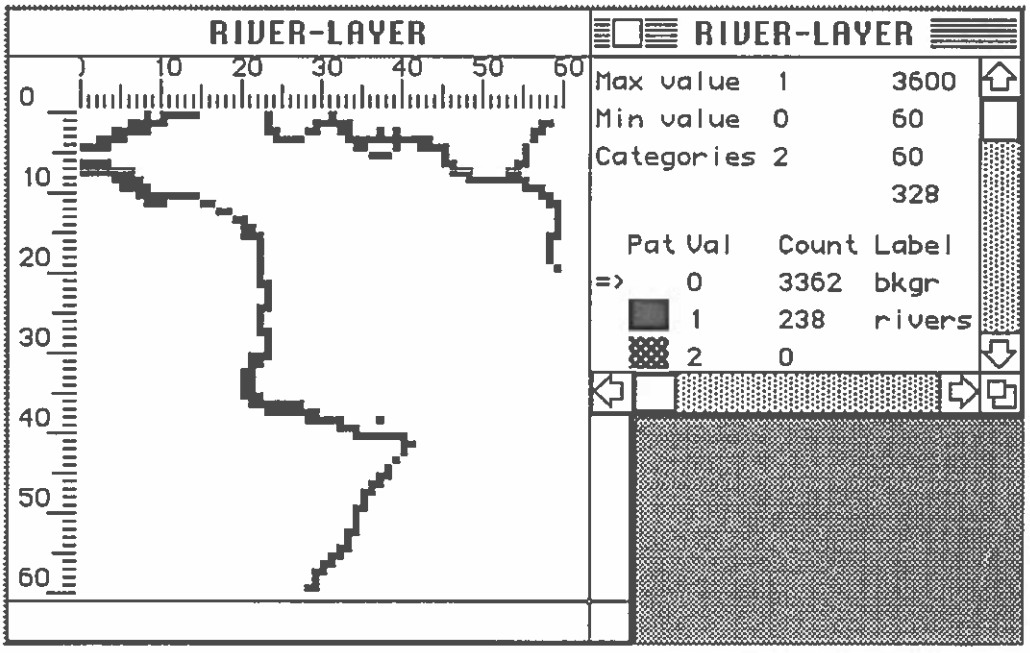


Figure 25. Map of Rivers

Figure 25 shows the rivers of the region. There are four separate rivers, one of them a single cell of the map. The shrink-propagate algorithm used for instantiating corridor elements breaks the rivers into six separate runs and eliminates some short segments. This occurs even though the

constraints on river width should result in the four rivers of Figure 25 being accepted "as-is". The utility of the shrink-propagate algorithm, as implemented here, degrades as the width of rivers in a map approaches the grid resolution. Here we have several segments of the river with a cell width of one.

The "Generate [full]..." item of the Generate menu entry produced four unique solutions. Figures 26 through 29 show the four landscape instances generated. The first solution, Figure 26, shows the test orchard patch (?op1) adjacent to a river with the control orchard patch (?op2) in the upper right. The two orchards are at least 10,000 feet apart and orchard ?op1 is distant from any noise-source, as stipulated by the constraints.

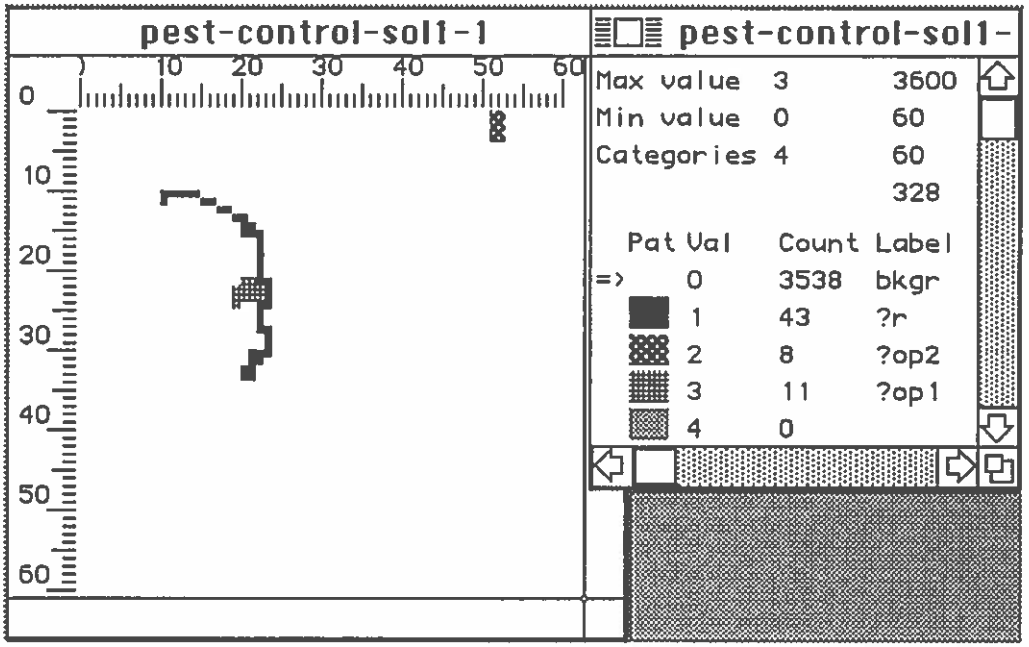


Figure 26. Pest-Control: Solution One

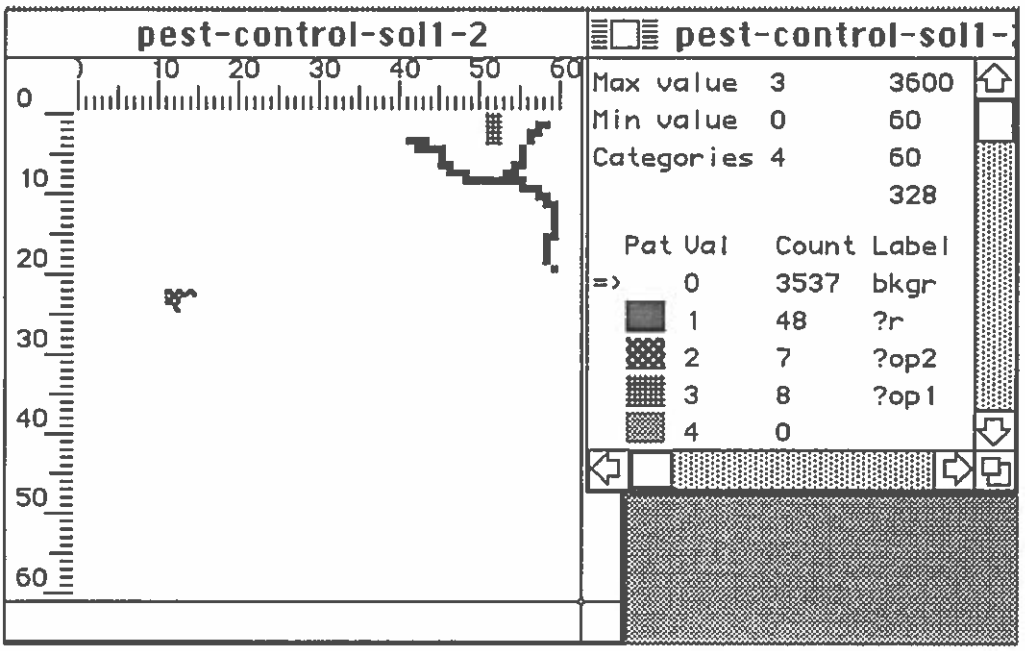


Figure 27. Pest-Control: Solution Two

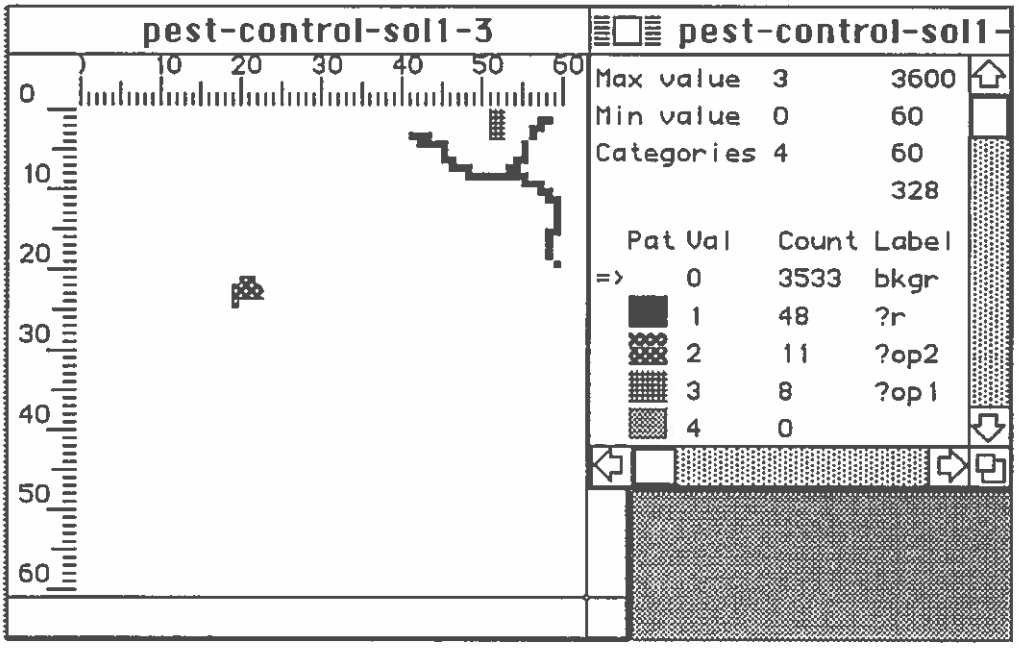


Figure 28. Pest-Control: Solution Three

The second solution, Figure 27, labels the orchard in the upper right as the test orchard ?op1 and selects another orchard as the control ?op2. Again, ?op1 is near a river as stipulated by the constraints.

The third solution, Figure 28, contains the same orchards as the first solution. The labels for ?op1 and ?op2 were switched, another river was selected, and the constraints remained satisfied.

The fourth and final solution, Figure 29, shows the upper right orchard as the test orchard ?op1 again. Another orchard has been chosen as the control ?op2. While ?op2 is close to a number of noise sources, the constraint on noise-proximity only applies to ?op1. The constraint on noise-proximity has prevented the inverse of this solution from being generated, as occurred with solutions one and three.

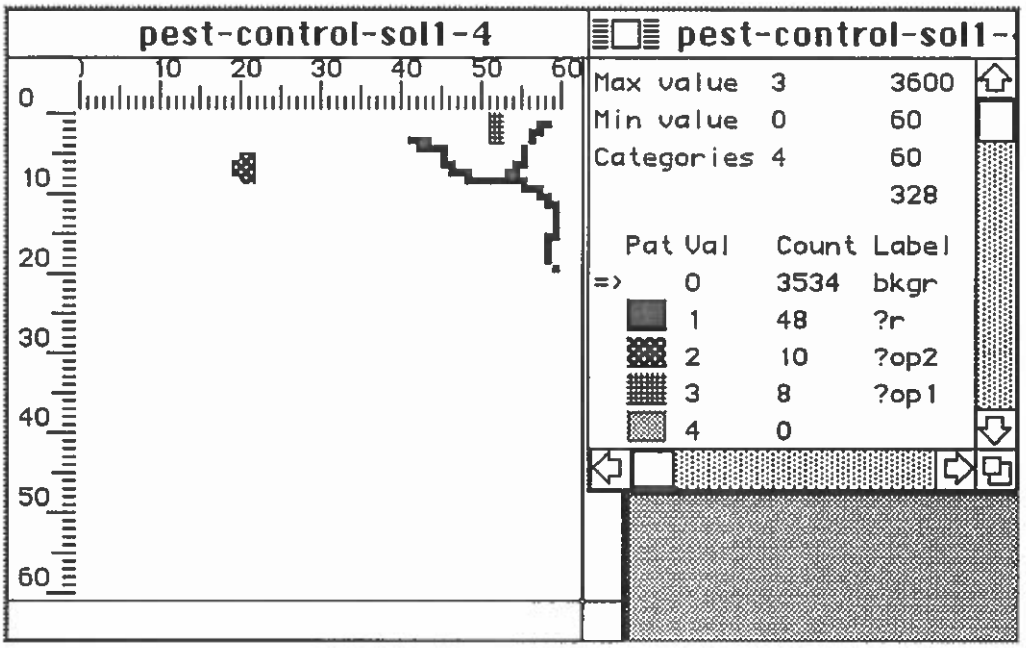


Figure 29. Pest-Control: Solution Four

The inter-element constraints were applied to four candidate orchards and eight candidate river segments (due to the behavior of the shrink-propagate algorithm). There are twelve possible ways to label the four orchards with ?op1 and ?op2, corresponding to P(4,2), or number of 2-permutations of 4 elements. There are eight possible ways to label the eight river segments with ?r. The noise sources, while a part of the constraints, are not included (labelled) in the final map. This means that 96 uniquely labelled landscapes (12*8) were tested against the inter-element constraints, of which only four survived.

CHAPTER VI

EVALUATION

The research presented here was initiated by a simple question: What is the nature of user interaction with GIS? The question was specialized towards asking what support, if any, do various GIS give for the spatial analysis of landscapes? This specialization was motivated by two factors. One, through the examination of various literature on landscape ecology and land use planning, the significance of landscape structure to landscape analysis became apparent [Forman 1986; Li 1989; Lounsberry et. al. 1981]. Two, numerous publications espouse the potential of GIS in landscape ecology and land use planning [Burrough 1986; Peuquet and Marble 1990].

This chapter evaluates LRM with respect to the overall goal of facilitating landscape analysis and briefly introduces a number of issues encountered during research. First, the operational landscape language will be evaluated in terms of its syntax, semantics, and completeness. Second, the implementation will be examined in terms of complexity, integration with macGIS, and the user-interface. In other words, our efforts will be examined first in terms of descriptive adequacy, then procedural adequacy.

First we consider the syntax of the landscape language. A major advantage of the syntax, which is driven by the underlying object oriented approach, is the modularity of definitions. Since definitions of data-layers, landscape elements, and landscapes are modular, libraries of these expressions may be built up in files and drawn upon as needed. The definitions can be loaded in any order, with all resolution of references taking place at the time of instance generation.

The current syntax is quite crude. The expressions for defining landscapes, namely def-data-layer, def-element, and def-landscape, provide only a slight advantage over the actual declaration of sub-classes in the SCOOPS object oriented scheme extension. The expressions make-inst and send are, in fact, SCOOPS syntax. The remaining expressions are simply scheme procedures and special forms. The expression syntax for intra-element constraints is also inconsistent with that of the inter-element constraints. For example, the inter-element constraints must be prefixed with '!'.

Much of the syntax was driven by the needs of the underlying interpreter. It was also unclear initially as to how much responsibility for handling interaction should be within the language, and how much should be a part of the user interface of the implementation. Given that our purpose is to facilitate interaction, a more consistent and readable syntax needs to be developed.

The semantics of the various object classes in the landscape class hierarchy are based primarily on the definitions provided by Forman in *Landscape Ecology* [1986]. Here he defines the matrix, patch, corridor, and landscape in terms of structure, function, and dynamics. Portions of the structural definitions were extracted and used as a basis for implementing the operational semantics.

The semantics of the patch were straightforward. A patch can be instantiated by any connected component in the data. In our language, a corridor is made distinct from a patch only by width. A more precise semantics should include other properties such as extent and curvilinearity. The main difficulties here are the development of precise quantitative definitions of the concepts and algorithms for computing these characteristics from the raster representation of the elements.

The semantics of the matrix are somewhat crude. The connectivity requirements of the connected components are temporarily relaxed, if specified, and the largest single component is selected as the matrix. While Forman does point out that the matrix is generally the component of largest area in a landscape, it is unclear as to whether the connectivity relaxation is appropriate.

The implementation of a landscape as a collection of landscape elements parallels the definitions well.

The class hierarchy of our landscape language currently represents a minimal, high level taxonomy of landscape objects. The hierarchy could be extended with additional pre-defined specializations of the matrix, patch, corridor, and landscape. The corridor class, for example, could be refined to include sub-classes for closed and open curves, eg. "ring" and "open-ended" corridor types. Another specialization, the peninsula, could potentially become a sub-class of patch. Specializations of the landscape class might include sub-classes such as "network" and "checkerboard". The class hierarchy provides a natural structure for incorporating these additional specializations. For each addition, algorithms for instantiation must be developed.

The inter-element constraint expressions of the landscape class currently limit the analysis capabilities of LRM in two ways. One, the constraint list allows only a conjunction of simple constraints. This limits the ability to describe certain configurations. For example, find a set of agricultural patches such that each is at least 200 feet from any other or within 50 feet of a river. It is not possible to express this with the current constraint language, since it involves a variable number of elements and a disjunction of constraints. Two, there is no mechanism by which compound landscape objects may be defined. In other words, it may be useful to define a structure composed of several landscape elements for some problems. For example, to define a recreation area, we may want to specify a lake-river-lake element, and subsequently express constraints in terms of this structured element as to its proximity to a small town. We will examine these two limitations in turn.

The constraint list represents a conjunction of simple constraints. The constraints are simple in that the syntax of each constraint is limited to a predefined form with no support for composition. The bulk of the constraints are of the form: (! (<relation> <function-of-elements> <value>)) or (! <boolean-function-of-elements>). The one exception is the for-all expression which represents a form of universal quantification over an element type, eg. (! (for-all high-crime-patches (> (min-dist ?my-home high-crime-patches) (2 "miles")))). A more consistent and expressive constraint language is needed, perhaps more like first order logic, which allows existential and universal quantification as well as disjunctions of constraints.

The second addition to the constraint language would be the ability to define structured objects composed of several landscape elements. Constraints could then be applied to the structured object as a unit. For example, one could define a potential agricultural development zone as a structured object composed of an undeveloped land patch within 50 feet of a river corridor. A desired landscape configuration could then be described as the set of all potential agricultural development zones greater than 2 miles from a residential patch. The ability to define structured objects could be implemented within the constraint language of the landscape class, or even at the same level as the definition of data layers and landscape elements, eg. a def-compound-element expression.

The ability to define structured objects raises some complications for the constraint satisfaction process, however. When a landscape instance is generated, a complete set of landscape element instances is generated first. The satisfy algorithm is then applied to the landscape element instances and the constraint list. If some of the landscape element instances are compound elements, the existing satisfy algorithm may not generate a complete set of consistent bindings. For example, a residential patch and an agricultural development zone may together satisfy the constraints and result in a landscape instance. However, if the agricultural development zone were broken up into its primitive landscape elements, then possibly a disjunct of the constraint expression could also be satisfied, resulting in another landscape instance.

This leads us to a consideration of the complexity of computing landscape instances from their definitions. The computing time required for locating landscape instances is occupied primarily by the connected-component algorithm, the shrink-propagate algorithm, algorithms that evaluate spatial characteristics, and the satisfy algorithm. The connected-component algorithm is linear in the number of cells of the GIS data layer. Although it utilizes a depth-first search for connected

regions, a region is marked when found and will be avoided by future searches. The number of cells of a data layer, however, can be quite large. The connected-component algorithm is invoked once for each data-layer-def in the set of definitions for an analysis task. Once computed, we can save and reuse a data layer for several problems.

The connected-component algorithm may also be invoked when the shrink-propagate algorithm breaks up a single component. The shrink-propagate algorithm is O(kn) where k is the number of cells by which the clump is to be uniformly contracted and expanded, and n is the number of cells in the bounding box. Again, the number of cells may be quite large.

The intra-element spatial characteristics are computed for every connected component in every data layer. The characteristics used in this thesis are all derived from area and perimeter which are both linear in the number of cells of the bounding box. The inter-element spatial characteristics are computed on demand when a constraint expression is encountered by the satisfy algorithm. Computing whether two components are adjacent is linear. Computing minimum distance between components is O(nm), where n and m are the number of cells in each bounding box respectively. Each cell of one component must be compared with every cell of the other.

Satisfy is an exponential algorithm in general. As used in this thesis, we can more precisely characterize the complexity. Consider a generalized version of the inter-element constraint expression: ((type1 ?a) (type1 ?b) (type2 ?c) (! (constraint ?a ?b)) (! (constraint ?b ?c))). We are assuming that the list of desired types and variables, eg. (type1 ?a), will precede all constraints. The constraint expression is matched against a list of landscape element instances of the various types, eg. ((type1 inst1) (type1 inst2) (type1 inst3) (type2 inst1) (type2 inst2)). All possible bindings of variables to instances which result in a consistent constraint expression are computed.

The process of finding all bindings can be broken into two steps. First, find all bindings of variables to instances prior to applying constraints. Second, apply the constraints to each binding set. The number of binding sets produced by the first step corresponds to the product:

$$\prod_{i=1}^{\#\text{ of types}} P(n_i, r_i)$$

There will be one term in the product for each type in the constraint expression. Each term is as follows: given r expressions of a single type, eg. (type1 ?a), and n instances of that type, eg. (type1 inst1), there will be P(n,r) possible bindings. The second step tests each binding set against all the constraints (worst case). The result is:

$$\prod_{i=1}^{\#\text{ of types}} P(n_i, r_i) \quad * \quad \#\text{ of constraints}$$

The first factor can grow to be quite large. The run time of LRM on a given analysis task was dominated by the satisfy algorithm.

Considering system integration, LRM is currently implemented as a separate program which interacts with macGIS through shared data files. This imposes on the user an inconvenient protocol for switching between applications. Given that LRM is written in Scheme and macGIS is written in C, integrating the two would involve either the use of calling conventions or the recoding of LRM in C.

The user interface component of LRM is currently quite minimal. The basic capabilities of the landscape language are represented, but there is a great deal of room for extensions. The user can freely enter and load landscape definitions, but the ability to generate instances is restricted to the

landscape class only. The status window provides only the most general information on the progress of execution. It would be useful to have a display of current landscape definitions, instances generated, and detailed execution state. The user could examine the spatial characteristics of instances by selecting them from a list, or even by graphically selecting them from a display of the map.

CHAPTER VII

CONCLUSION

The focus of our research has been almost exclusively on the spatial structure of landscapes. In terms of the operational landscape language, the definition of a data-layer with a creation script is the only point at which non-spatial information is considered. The remainder of the process, namely landscape and landscape element instantiation, is based purely on spatial characteristics. A means by which to incorporate non-spatial data would allow analysis to consider the function and dynamics of a landscape, not just the structure. This section will briefly examine this issue. We will then close by relating our efforts to other recent research.

When an instance of a data-layer definition is generated, the result is binary array. Cells with the value one represent land of a specific type, and cells with the value zero represent the absence of that type. The information in the GIS data from which this array was derived, using the creation-script, is now abstracted away. If the attributes and cell values were to be maintained, a number of extensions to subsequent analysis could be realized.

The def-data-layer expression would still be used to define types of land, but the definitions for landscape elements could place further constraints on the attribute values of the connected components. For example, a waterbody data-layer may be created from the non-zero values of a GIS data layer which give the depth of water over a region. In addition to producing a binary array, the water depth values could be retained as well. Connected components would still be located based on the binary array, however the landscape element definitions would have access to the depth values of each connected component. The intra-element constraints on a lake-patch definition might read: ((< area (2000 "sq feet")) (< (max water-depth) (10 "meters"))). This, of course, begs the question as to what constraints representing the definition of a landscape element should go into the data-layer creation script, and what should be in the landscape element definition.

The non-spatial data of landscape elements could also be utilized by the inter-element constraints of the landscape class. Evaluation metrics for the landscape could be defined in terms of attribute values of landscape elements. Constraints may then be placed on these metrics as well as the spatial relations. Alternately, landscape instances may be "rated" by these metrics for suitability in the context of some land use planning task.

Rapid advances in software and hardware technology will continue to push the evolution of Geographic Information Systems forward. Some particular advances to note are in the areas of: user interfaces, object oriented software design, image processing, and artificial intelligence. A number of recent publications point to the expanding role of artificial intelligence techniques in GIS [Webster 1990; Albert 1988]. This thesis has successfully integrated concepts from the above areas as well as landscape ecology and landscape architecture.

We have taken a first pass at defining an operational language for reasoning about landscape structure. The primitives of this language derive their semantics from the fields of landscape ecology and landscape architecture. The implementation of the language primitives is made possible by data structures and algorithms from geographic information systems, artificial intelligence, image processing, and object oriented design. The guiding principle is that problem solving should take place in the task domain, not the domain of the underlying data representation.

A similar effort is the KBGIS system developed by the U.S. Geological Survey [Albert 1988]. KBGIS allows the definition of spatial objects using a spatial object language (SOL). A

query describing a desired spatial configuration of objects may be formed, also in the SOL, and the system will search the GIS database for instances. A Spatial Object Knowledge Base stores information about objects and object definitions. A Function Knowledge Base stores information used in the search for spatial objects and for computing spatial properties. A "learning" component was also implemented in order to reduce search time for future queries.

The main difference between KBGIS and LRM is in the definition of spatial objects. LRM provides a set of spatial objects with pre-defined semantics, eg. matrix, patch, and corridor. KBGIS provides basic pixel, pixel group, and relational properties by which object definitions may be built up. KBGIS also addresses additional issues such as query efficiency and data compression via quadtrees.

The notion of reasoning with a "scene" of objects instead of an image (data-layer) is given a precise treatment in *A Logical Framework for Depiction and Image Interpretation* by Reiter and Mackworth [1989]. Reiter and Mackworth address the problem of mapping from image to scene specifically in the context of map interpretation. Here they define a taxonomy of map image objects, the two major components being chains and regions. This corresponds closely to the vector representation of GIS. Also defined is a taxonomy of scene objects, the two major components being linear-scene-objects and areas. Subtypes of these components are entities such as road, river, land, and water. This corresponds closely to the language of landscape structure presented here. The map image is represented as a set of formulas in first order logic. Knowledge about spatial relations in the scene domain is also cast into formulas of first order logic. Finally, a set of depiction axioms defines a mapping between map image objects and scene objects. An interpretation of the map is then formally defined as a model of the map image, scene, and depiction axioms.

In terms of GIS, each data layer can be considered an image. The data layer is a static representation which uniformly records the source data. No emphasis is placed on particular regions or components of the source data. An exception to this would be the relational records associated with points, lines, and polygons in the vector representation. It is assumed, however, that this data is entered without prior knowledge of specific analysis tasks.

A scene is defined to be an interpretation of the images, or data layers, in the context of a specific analysis task. Here one focuses on the content of the images. In the case of landscape ecology, one is interested in the composition and spatial configuration of landscape elements such as patches, corridors, and the matrix. Similarly with land-use planning. Essentially, one is isolating components of the image and reasoning about the inter-relationships.

An important observation of many GIS data representations and transformation operations is that they treat the map as an image. Granted, the attribute values of the data layer represent higher level properties of the region, but they are associated directly with components of an image, not components of a scene. As such, deriving new information requires operations on the image components. In this case, the cells and/or vectors of the GIS representation. Analysis in this situation is a process of mapping a problem onto operations on images. A landscape is a structure of interacting elements, a scene if you will. Representing it only as a collection of cells, or vectors, results in problem solving taking place in the domain of the image, not the domain of the scene.

In closing, we have taken a first pass at developing a high level tool for landscape analysis with Geographic Information Systems. Further refinements of the implementation and landscape language semantics should result in a valuable system for landscape ecologists and land-use planners. Further extensions to our implementation, such as the incorporation of non-spatial data, may enable support for more advanced analysis tasks, and even simulation. Research into integrating advanced computing techniques with Geographic Information Systems and landscape analysis has proven to be an area rich with avenues for further exploration.

APPENDIX

BNF SPECIFICATION OF LANDSCAPE LANGUAGE

```
<valid-exp>  ::=   <def-data-layer-exp> | <def-elt-exp> | <def-lndscp-exp> |
                   <gen-inst-exp> | <send-exp> | <scheme-exp>

<def-data-layer-exp>   ::= (def-data-layer <name>
                               creation-script = (( <macGIS-commands> )
                                                   (load <filename> )))

<def-elt-exp>          ::= (def-element <element-type> <name>
                               data-layer = <name>
                               constraint-list = ( <intra-elt-constraint-exps> ))

<def-lndscp-exp>       ::= (def-landscape <name>
                               component-list = ( <components> )
                               constraint-list  = ( <inter-elt-constraint-exps> ))

<gen-inst-exp>  ::= (make-inst <name> ) | (generate-landscapes <name> <mode>)
<mode>          ::= 'full | 'non-data-layer | 'satisfy-only

<send-exp>      ::= (send <inst> <message> <arguments> )

<macGIS-commands>      ::= <macGIS-command> <macGIS-commands> |
                           <macGIS-command>

<element-type>         ::= matrix | corridor | patch

<intra-elt-constraint-exps>  ::= <intra-elt-constraint> <intra-elt-constraint-exps> |
                                 <intra-elt-constraint>
<intra-elt-constraint>       ::= Ø | (<relation> <spatial-characteristic> <value>)
<relation>                   ::= < | <= | > | >= | =
<spatial-characteristic>     ::= width | area | i-e-ratio | connectivity | shape-index
<value>                      ::= (<number> <units>)
<units>                      ::= "nounits" | "feet" | "meters" | "sq feet" | "sq meters"

<components>           ::= <components>  <component> | <component>
<component>            ::= Ø | (<element-type> <name>)

<inter-elt-constraint-exps>  ::= <inter-elt-constraint>  <inter-elt-constraints> |
                                 <inter-elt-constraint>
<inter-elt-constraint>       ::= Ø | (<name> <variable>) | (! <test>) |
                                 (! (for-all <name> <test>))
```

<test> ::= (<relation> <function> <value>) |
(<relation> <value> <function>) |
(<relation> <function> <function>) |
<function>

<function> ::= any scheme procedure with <variable>'s or <name>'s as arguments which returns boolean or numeric

<variable> ::= any scheme symbol preceded by a '?'

<name> ::= 'any scheme symbol'
<filename> ::= 'any valid macintosh filename in double quotes'
<macGIS-command> ::= Ø | macGIS command in double quotes
<inst> ::= SCOOPS object instance returned by <gen-inst-exp>
<message> ::= SCOOPS object method name
<arguments> ::= SCOOPS object method arguments

BIBLIOGRAPHY

Albert, T. M. 1988. *Knowledge-Based Geographic Information Systems (KBGIS): New Analytic and Data Management Tools*. Mathematical Geology. Vol. 20, No. 8, pp. 1021-1035.

Ballard, D. H. and Brown, C. M. 1982. *Computer Vision*. Prentice-Hall, Inc., New Jersey.

Burrough, P. A. 1986. *Principles of Geographical Information Systems for Land Resources Assesment*. Clarendon Press, Oxford.

Cowen, D.J. 1988. GIS Versus CAD versus DBMS: What are the Differences? In *Introductory Readings in Geographic Information Systems*. Taylor and Francis, New York, pp. 52-61.

Dangermond, J. 1983. A Classification of Software Components Commonly Used in Geographic Information Systems. In *Introductory Readings in Geographic Information Systems*. Taylor and Francis, New York, pp. 30-51.

Forman, R. T. T., and Godron, M. 1986. *Landscape Ecology*. John Wiley and Sons, New York.

Larsen, K. and Hulse, D. 1989. *macGIS - A Geographic Information System for the Macintosh: The User's Guide*. University of Oregon, Eugene, Oregon.

Li, H. 1989. *Spatio-temporal Pattern Analysis of Managed Forest Landscapes: A Simulation Approach*. Doctoral Dissertation, Oregon State University.

Lounsbury, J. F., Sommers, L. M., and Fernald, E. A. 1981. *Land Use: A Spatial Approach*. Kendall/Hunt Publishing Company, Dubuque, Iowa.

Peuquet, D. J. 1984. A Conceptual Framework and Comparison of Spatial Data Models. In *Introductory Readings in Geographic Information Systems*. Taylor and Francis, New York, pp. 250-285.

Reiter, R. and Mackworth, A. K. 1989. *A Logical Framework for Depiction and Image Interpretation*. Artificial Intelligence 41, pp. 125-155.

Rosenfeld, A. and Kak, A. C. 1976. *Digital Picture Processing*. Academic Press, New York.

Tanimoto, S. L. 1987. *The Elements of Artificial Intelligence*. Computer Science Press, Inc., Maryland.

Webster, C. 1990. *Rule-based Spatial Search*. International Journal of Geographical Information Systems. Vol. 4, No. 3, pp. 241-259.