A SPATIAL DATA STRUCTURE

by

Linda G. Shapiro
Robert M. Haralick
Department of Computer Science
Virginia Polytechnic Institute
and State University
Blacksburg, VA 24061

Technical Report #CS79005-R

August 1979

I. INTRODUCTION

Digital map data, line drawings, and region adjacency graphs are all instances of spatial data that is usually organized in a discrete structural form as opposed to the iconic form of the gray tone or color image. Such structural organizations can be derived by hand gathered data or by segmenting an image, associating attributes with the image segments, and determining relationships between segments. In this paper we are not concerned with the origin or generation of the spatial information. We are concerned with the representation of the spatial information once it is created and the kinds of interactions we may wish to have with it.

We pose our interaction as a sequence of questions and commands. We may wish to know whether a railroad yard is in the image that the spatial information was extracted from. We might ask where the industrial areas are or whether there is any evidence that a brush area between a forest and an urban area has ever been burned. We may wish to find the biggest body of water within twenty miles of a particular city. We may ask the system to construct a region consisting of all the irrigated cropland in a certain state or to construct a road network including all the roads that go through a given city. Or, we may simply indicate that a region's area has changed and then specify the new boundary of the region.

Whether the form of interaction is a question or a command, finding the answer, modifying or returning the required region(s) or line(s) or point(s) involves searching the data structure for one or more objects or distances that satisfy the conditions of the query. We will focus attention on those interactions that require the execution of procedures that rely heavily on the structural representation of the spatial information. We will not concern ourselves at this time with the problem of efficient geometric and distance algorithms.

We consider maps to be a visual representation of spatial data. We call the formal organizational structure by which we may represent spatial data in the computer a spatial data structure. In this paper we give a definition of spatial data structure and some examples illustrating its use in raster format data, in vector format data, in procedures which do region editing, and in procedures which make spatial references. The structure is rich, flexible, and efficient enough to logically store any of the spatial information in maps, line drawings, region adjacency graphs, and other geographic entities that we might desire to represent.

In Section II we define the spatial data structure, give some specific examples which illustrate the use of the structure to represent spatial information, and show how the geographic data structures used by other researchers can be

accomodated by the spatial data structure. In Section III we discuss the manipulation of a spatial database for answering queries. In Section IV we discuss the mathematical nature of spatial data matching problems.

II. A Spatial Data Structure

The basic kinds of data found in maps are points, lines and areas. This data can be stored in many forms; grid cells, lists of points, lists of line segments, and polygons are common examples. The best storage representation depends on the algorithms that need to access it. Unfortunately, if everyone uses a restricted data structure specifically tied to one application, very little sharing or unifying of software or data can take place. In this section we define a general spatial data structure that can be efficiently used to represent any spatial or relational data.

Definition of the Spatial Data Structure

An atom is a unit of data that will not be further broken down. Integers and character strings are common examples of atoms. An attribute-value table A/V is a set of pairs A/V = {(a,v) | a is an attribute and v is the value associated with attribute a}. Both a and v may be atoms or more complex structures. For example, in an attribute-value table associated with a structure representing a person, the

attribute AGE would have a numeric value, and the attribute MOTHER might have as its value a structure representing another person.

A spatial data structure D is a set $D = \{R1,\ldots,RK\}$ of relations. Each relation Rk has a dimension Nk and a sequence of domain sets $S(1,k),\ldots,S(Nk,k)$. That is for each $k = 1,\ldots,K$, $Rk \subseteq S(1,k) \times \ldots \times S(Nk,k)$. The elements of the domain sets may be atoms or spatial data structures. Since the spatial data structure is defined in terms of relations whose elements may themselves be spatial data structures, we call it a recursive structure. This indicates 1) that the spatial data structure is defined with a recursive definition (and not that the information stored in it is infinitely recursive), and 2) that it will often be possible to describe operations on the structure by simple recursive algorithms.

A spatial data structure represents a geographic entity. The entity might be as simple as a point or as complex as a whole map. An entity has global properties, component parts, and related geographic entities. Each spatial data structure will have one distinguished binary relation containing the global properties of the entity that the structure represents. The distinguished relation is an attribute-value table and will generally be referred to as the A/V relation. When a geographic entity is made up of parts, we may need to know how the parts are organized. Or, we may wish to store a list of other geographic entities

that are in a particular relation to the one we are describing. Such a list is just a unary relation, and the interrelationships among parts are n-ary relations.

For example, we may represent the state of Virginia by a spatial data structure. In this case, the A/V relation would contain global attributes of the state such as population, area, boundary, major crop, and so on. The values of most of these attributes (population, area, major crop) are atoms. The value of the boundary attribute is a spatial data structure defining the boundary.

One obvious division of the state is into counties. A list of counties could be included as one of the relations, or it might be more valuable to store the counties in a region adjacency relation, a binary relation associating each region (county) with every other region (county) that neighbors it. Counties, of course, would also be represented by spatial data structures.

Some other geographic entities that are related to a state are its highways, railroads, lakes, rivers, and mountains. Some of these entities will be wholly contained in the state and others will cross its boundaries. One way to represent this phenomenon is to use a binary relation where the first element of each pair is a geographic entity, and the second element is a code indicating whether the entity is wholly contained in the state. The spatial data

structures representing the geographic entities themselves would contain more specific information about their locations. Figure 1 illustrates a simplified spatial data structure containing an attribute-value table, a county adjacency relation, and a lakes relation for the state of Virginia.

Comparison to Other Geographic Data Structures

One of the earlier and successful geographic information systems is Tomlinson's Canadian Geographic Information System (Tomlinson, 1967; Switzer 1975). This system includes such features as a command language, an assessment language, the possibility of overlaying maps, interactive graphics, and input from a drum scanner or an X-Y digitizer. In this system, regions are represented by polygons. Two files of data are used: the image data set which contains the line segments that define the polygons and the descriptive data set which contains the user-assigned identifiers, centroid, and area for each polygon. In the image data set, a line segment points to its left and right polygons and to the next two boundary chains that continue bounding the polygons on the left and on the right. Each polygon in the image data set points to its representation in the descriptive data set and vice versa.

In the U.S. Census DIME files (Cooke and Maxfield, 1967), the basic element is a line segment. Line segments

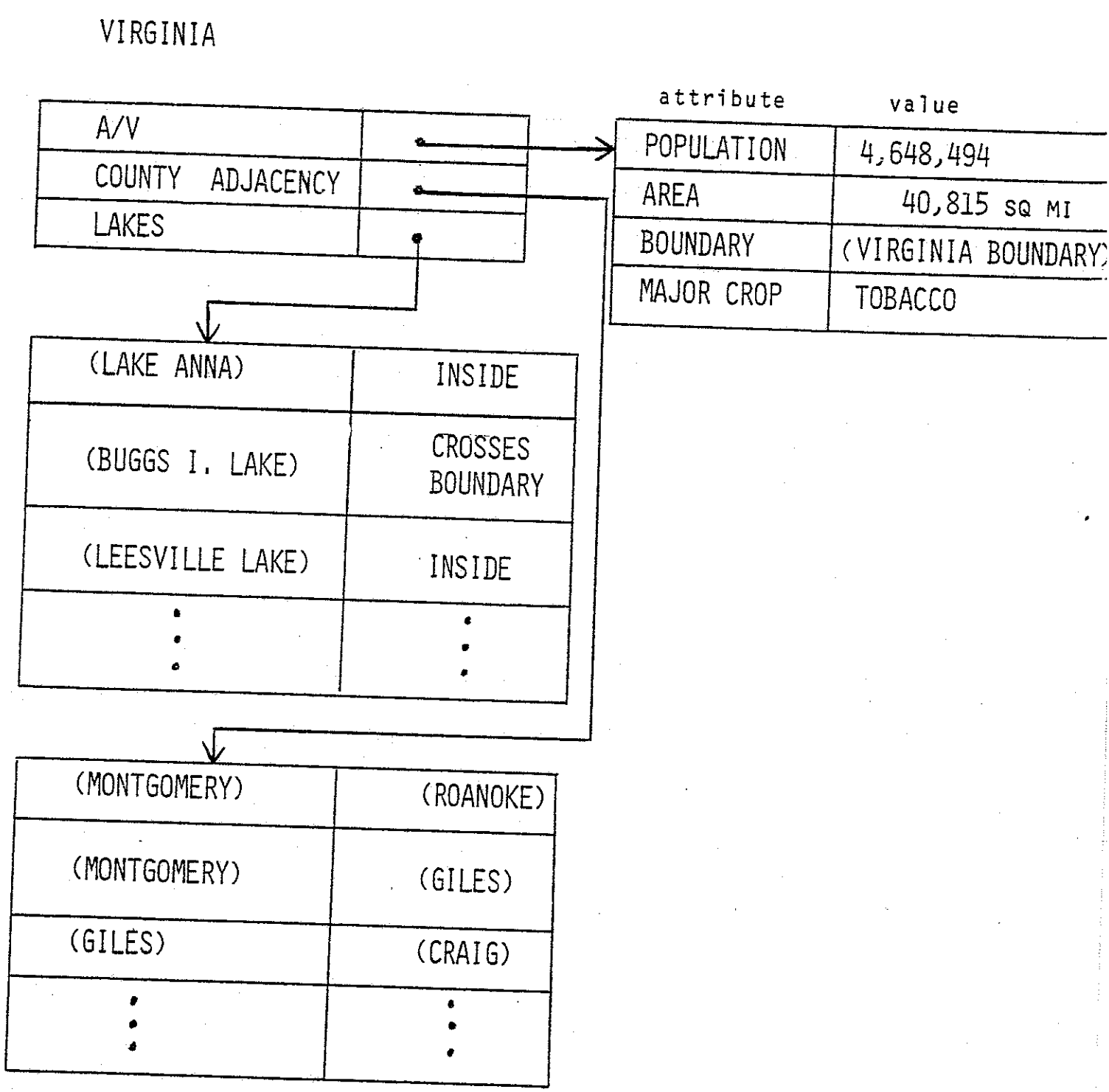


Figure 1 illustrates a spatial data structure representing the state of Virginia. The A/V relation is an attribute-value table. The COUNTY ADJACENCY relation contains pairs of adjacent counties. The LAKES relation contains pairs consisting of a lake and an indication of whether it lies inside or on the boundary of the state.

are defined by two end nodes plus codes for the polygon on the right side and the polygon on the left side of the segment. In this system, the data structure has been kept simple at a cost of extra CPU time for certain operations. For instance, determining what line segments share a node or finding the whole outline of a polygon requires searching the database. This structure is sufficient to represent all topological spatial relations between regions, and is also used by Hanson and Riseman (1976) in their VISIONS system.

The POLYVRT system (Harvard Laboratory for Computer Graphics and Spatial Analysis, 1974) and the LUDA system (Fegeas, 1977) are similar to the DIME system, except that the basic unit is a chain---a directed list of points that can be used to define an entire polygon. In POLYVRT, searching can take place in two directions: from chain to polygon and from polygon to chain.

The spatial data structure defined in this paper can easily store the spatial data in all of the above systems in exactly the same logical manner. We illustrate this by defining a system with similar characteristics. In our system, a point is an atom consisting of an ordered pair (X,Y) where X represents latitude and Y longitude. A chain C is a spatial data structure C = (A/VC, LP). LP is an ordered list (unary relation) of points that define the chain. The attribute-value table A/VC of a chain contains the attributes LEFT_POLYGON, RIGHT_POLYGON, NEXT_CHAIN_LEFT,

and NEXT_CHAIN_RIGHT whose values correspond to the pointers in Tomlinson's system. The attribute-value table can also contain such global information as the length of the chain or a function to be used to interpolate between the points.

A polygon P is a relatively simple spatial data structure P = {A/VP}. In this case, the attribute-value table contains the attributes FIRST_CHAIN and POSITION. The value of FIRST_CHAIN is the first chain of the polygon, and the value of POSITION is LEFT or RIGHT depending on whether the polygon lies to the left or to the right of the first chain. Such global attributes as AREA and CENTROID can also be stored in the attribute-value table.

Figure 2 illustrates this structure for a simple 'map' of two regions P1 and P2. In this example, we have chosen the chains to be the longest sequence of points that have exactly one region to their right and one region to their left. The directions of the chains were chosen arbitrarily. We do not mean to suggest that the points in a chain be stored sequentially as (X,Y) condinates. Instead, we are leaving the physical storage mechanism open. In some applications, storing differences or using Freeman chain codes (Freeman, 1974) might be appropriate. In other applications, storing the chain in a parametric functional form might be appropriate. Regardless of the physical form of storage, the structural aspect of the representation is the same.

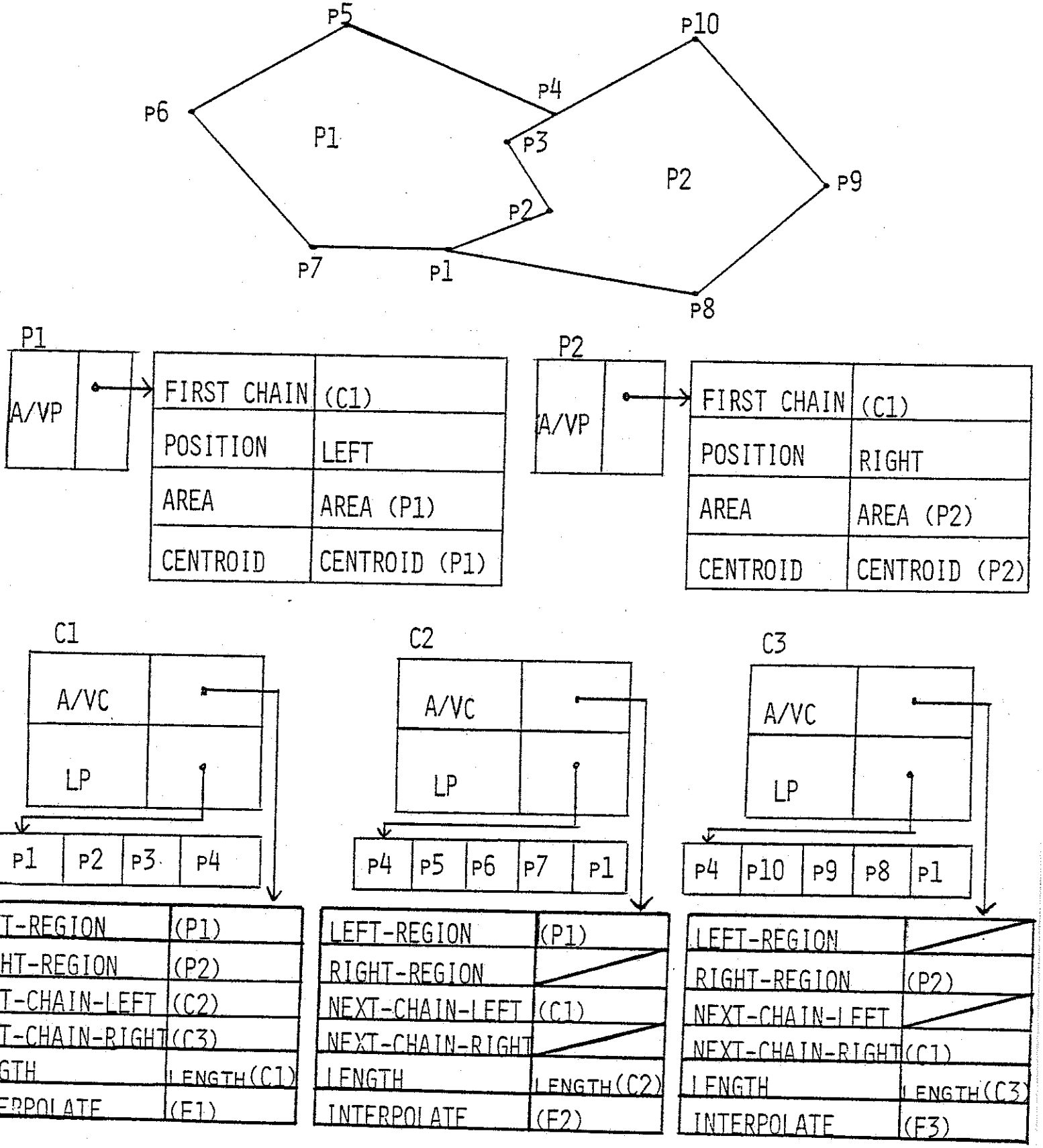


Figure 2 illustrates a spatial data structure for a simple map that encompasses the structures used in the Canada Geographic Information System, the DIME system, the POLYVRT system, and the LUDA system.

Edwards, Durfee, and Coleman (1977) use a hierarchical polygonal data structure to represent regions that can have holes, which in turn can have holes of their own, to any level of nesting. Because the spatial data structure is a recursive structure, it can naturally handle such a hierarchy. We define the boundary of a region as follows. A boundary is a polygon plus a (possibly empty) list of boundaries of interior polygons. Thus a boundary can be represented by a spatial data structure B = {A/VB, LB} where A/VB contains the attributes FIRST_CHAIN and POSITION, and the unary relation LB is a list of boundaries. As before, FIRST_CHAIN is the first chain of the polygon, and POSITION indicates whether the bounded region lies to the left or the right of the first chain.

When LB is empty, B is a polygon or simple boundary. When LB is not empty, then B has holes in it. Each of these holes is also a boundary, so it may also have holes. Thus this spatial data structure handles the hierarchical polygonal data structures. Other hierachic structures such as Brassel's hierarchically organized spatial data base of Thiessen polygons can be handled similarly by our spatial data structure.

Burton's polygonal representation (Burton, 1977) allows quick solutions of the point in polygon and polygon intersection problems. It is based on breaking up a polygon into basic sections that are maximal length chains, monotonic in both coordinates. The chain spatial data

structure as described above may be modified so that the list of points are monotonic in both coordinates. Thus Burton's algorithms may be applied to our chain structure.

The triangle data structure has been used for representing surface data. A triangle represents a piece of a three-dimensional surface. The vertices of the triangle are nodes containing information such as elevation and slope. To obtain information about points of the surface interior to a triangle, an interpolation may be used with a homogeneous coordinate system based on the three nodes' sampled values (Gold, 1976; Males, 1977). In the triangle data structure, each triangle points to its vertices and adjacent triangles. Thus a triangle is a spatial data structure T = {LV, AT} where LV is a list of three vertices, and AT is a list of adjacent triangles. Each vertex V is a spatial data structure V = {A/VV} where the attribute-value table A/VV contains the attributes SLOPE, ELEVATION, and other information.

GEOGRAF is a system proposed by Peucher and Chrisman (1975) to handle both planar data and surface data. The system includes the concepts of 1) a least common geographic unit (an area that will not be partitioned further) 2) a chain group (a set of chains that form a boundary of two areal units of a given polygon class, and 3) an attribute cross-reference table. To handle surface data the system has a two-part data base including both a

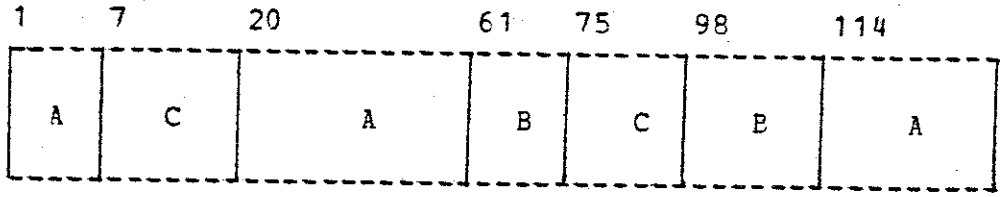
triangle structure and a set of points that lie along lines of high information content. The triangle structure is a low level structure comparable to the image data set of the Canada Geographic Information System. The points in the second set consist of peaks, pits, and passes. This set in some ways corresponds to the descriptive data set of Tomlinson. The GEOGRAF system is a movement toward unification of geographic data structures.

Go, Stonebraker, and Williams (1975) describe a relational database approach to implementing a geographic data system. The basic entity in their system is a map, a collection of lines, points, line groups (polygons), and zones (collections of polygons) represented by a 9-ary relation. A query language QUEL which is similar to SEQUEL (Chamberlin and Boyce, 1974) is used to interrogate the system. The system can handle simple queries about a map and can display information from a map. Because of its relational nature, this system is related to the more sophisticated system we discuss in this paper.

The systems just described all store their geographic data in vector form. Vector form is only one form of spatial data. It represents areas by their boundaries and has the advantage of a very compact representation. Raster or grid form is another form of spatial data. In this form areas are represented by the grid cells that cover them. The advantage of raster format data is the simplicity of

performing certain tasks such as map overlay. Our spatial data structure can handle raster form data just as easily as it handles the vector form.

Consider, for example, storing an entire map of regions in a run length encoded form of raster grid cell data. Shown below is one row of such map data.



In the row shown, there are seven intervals, each one of which belongs to one of three regions: A, B, or C. Each interval is specified by a beginning pixel, an ending pixel, and an interval label. By grouping together all intervals of the same label we may represent this row by the following table.

		Internal List Name	
A:	(1,6),(20,60),(114,130)	IL47A	Partition List PL81B
B:	(61,74),(98,113)	IL52D	
C:	(7,19),(75,97)	IL36E	

In this case, the row points to the partition list (Merrill, 1973) PL81B which contains the interval lists IL47A, IL52D, and IL36E, each of which contains a set of intervals for some region in the row.

Grid cell data structures explicitely represent areas. We will call the entities employing this representaion 'map areas'. Thus, the entity 'map area' is a spatial data structure MA = {A/VMA,PLR}, where the attribute-value table A/VMA has the attribute THEME with values such as 'soil type' or 'land use'. PLR is the partition list binary relation. It consists of a set of ordered (row,partition list) pairs. The entity 'partition list' is a spatial data structure PL = {A/VPL,ILS}, where the attribute-value table A/VPL has the attribute ROW whose value is the number of the row being divided up by the partition list. ILS is the set of interval lists composing the partition PL. Finally, the entity 'interval list' is a spatial data structure IL = {A/VIL,HS}, where the attribute-value table A/VIL contains the attributes NAME and ROW. The attribute NAME takes on a value which is the name of the region to which the intervals in IL belong. The attribute ROW has as its value the row number. HS is the ordered list of horizontal strips (intervals) in the interval list. Each strip in HS is an ordered pair whose first component is the beginning pixel and whose second component is the ending pixel of the strip.

III. Design of a Spatial Information System

We are currently involved in the design and implementation of an experimental spatial information system using the spatial data structures concept of Section II. The system will answer user queries and solve problems

presented to it in a subset of English. In this section, we describe some of the important features of the proposed system.

Major Data Structures

The spatial data structure is the primitive or building block of the system. A finite number of spatial data structure types will be allowed. For instance, the system might include spatial data structures representing the high-level entities states, cities, counties, highways, rivers, lakes, and mountains and the lower-level entities boundaries, simple boundaries, and chains. Thus the system might contain a spatial data structure whose name is MONTGOMERY and whose type is COUNTY.

For each type of spatial data structure, the system will keep a prototype structure. The prototype will indicate what attributes are found in the attribute-value relation of this type of spatial data structures and what relations besides the A/V relation comprise the data structure. Similarly a finite number of relation types will be allowed, and the system will keep prototypes of the allowable relations. Thus the STATE prototype might indicate that all spatial data structures of type STATE have a COUNTY_ADJACENCY relation. The COUNTY_ADJACENCY prototype would indicate that this is a binary relation and that both components of each pair in the relation are spatial data

structures of type COUNTY. (See Figure 4 in the section entitled Primitive Operations.) A user query might involve a specific spatial data structure or a specific type of spatial data structure. For fast access in either case, the system will include a spatial data structure name dictionary that maps a name to a spatial data structure and a spatial data structure type dictionary that maps a type to a list of all spatial data structures of that type. Similarly a relation type dictionary will map a relation type to a list of all relations of that type. We are planning to implement relations as relational trees (Shapiro, 1979). Note that all of these structures can be represented by spatial data structures, unifying the whole system.

Program Structure

We envision an intelligent system that can handle a variety of queries. For example,

1) What is the population of Richmond?

2) What cities with population greater than 20,000 lie in Montgomery County?

3) What crops are grown within a radius of one hundred miles from Charlottesville?

4) In what counties is 20 per cent of the area mountainous?

5) What is the shortest route from Alexandria to Blacksburg?

are all reasonable queries. Some of the questions require quick table-look-ups to produce the answer and some require searching through the database and using special purpose

algorithms. It is also possible that there are several different procedures or several different paths through the database to find an answer to a query. In this case, the system should be able to choose the most efficient way to solve the problem. We propose the following program organization.

The user queries or commands are processed by a PARSER routine which translates the query into a component command and a list of qualifiers. The PARSER will detect any sytax errors and try to rectify the mistakes through dialog with the user. The output of the PARSER is accepted by the TREE-GENERATION routine which creates a tree of possible solution paths through the database. The most efficient solution path is determined by the TREE-EVALUTATION routine which produces, as its output, a procedure of primitive database functions which when executed will satisfy the user query. Finally, the EXECUTOR executes the procedure to complete the command. Figure 3 depicts the flow of a user query from input to answer.

Each of the major modules must access information structures as part of its input. The PARSER requires a grammar that describes the syntax of the query language. Since the query language should be English-like, the PARSER will also need some world knowledge to help disambiguate some of the phrases used. (See Winograd, 1972). The TREE-GENERATION routine uses world knowledge, spatial knowledge,

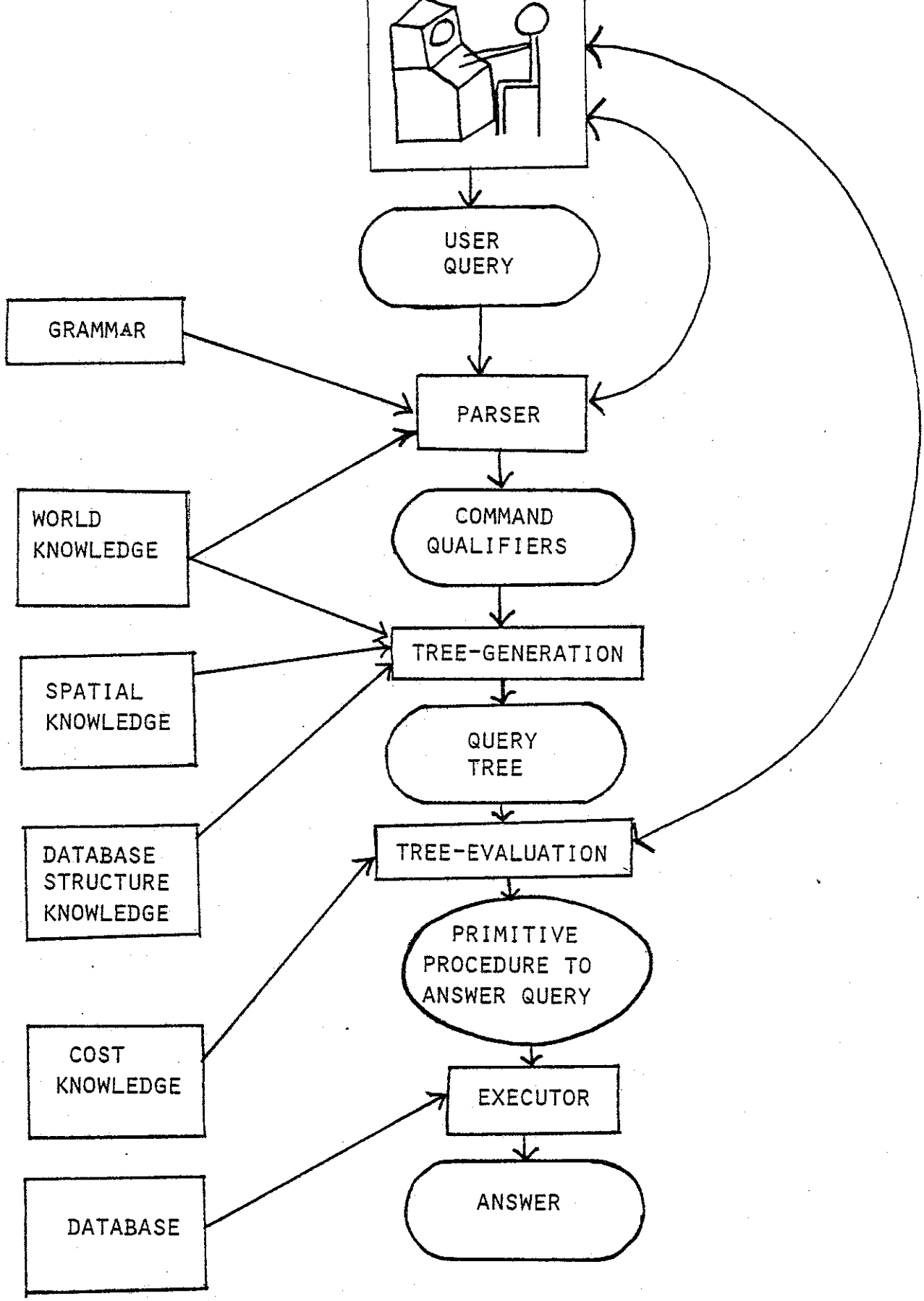


Figure 3 illustrates the flow of a user query through the spatial information system.

and knowledge of the database structure. The world knowledge and spatial knowledge can be represented in several different ways including relational tables (Codd, 1970), production rules (Shortcliffe, 1970), and procedures (Hewitt, 1972). Again each of these knowledge representations can be represented by spatial data structures. The database structure knowledge should include the spatial data structure name dictionary, the spatial data structure type dictionary, the relation type dictionary, and all the prototypes.

The TREE-EVALUATION routine uses cost knowledge to choose a path from the set of paths produced by the TREE-GENERATION routine. The cost knowledge may be stored in tabular form or built into the TREE-EVALUATION routine in the form of procedures. The TREE-EVALUATION routine may be offered a choice between a high cost for an exact answer or a lower cost for a less than perfect answer.

The input to the EXECUTOR is a procedure of primitive functions that operate on the database. The EXECUTOR has access to the entire database of information. The EXECUTOR can also access the primitive functions and special purpose high-level algorithms that are built into the system.

Primitive Operations

The spatial data structure is a relational structure. Because of this, the spatial database system shares many characteristics of relational database systems. In particular, all of the primitive operations used in relational database systems are applicable to the relations of a spatial data structure. We will use a small example database to motivate the use of these and other primitive operations.

Figure 4 illustrates a set of prototypes for spatial data structures and their relations that might be found in a spatial information system. The STATE prototype indicates that STATE is a type of spatial data structure having an A/V_STATE relation, a COUNTY_ADJACENCY relation, and a RIVERS relation. The A/V_STATE relation has four attributes: NAME, whose value is a character string, POPULATION and AREA whose values are numbers, and BOUNDARY whose value is a spatial data structure of type POLYGON.

The COUNTY_ADJACENCY relation is a binary relation, and each member of each pair is a spatial data structure of type COUNTY. The RIVERS relation is a unary relation, and each element is a spatial data structure of type RIVER. The other prototypes convey similar information.

The following questions are possible queries to a

PROTOTYPES

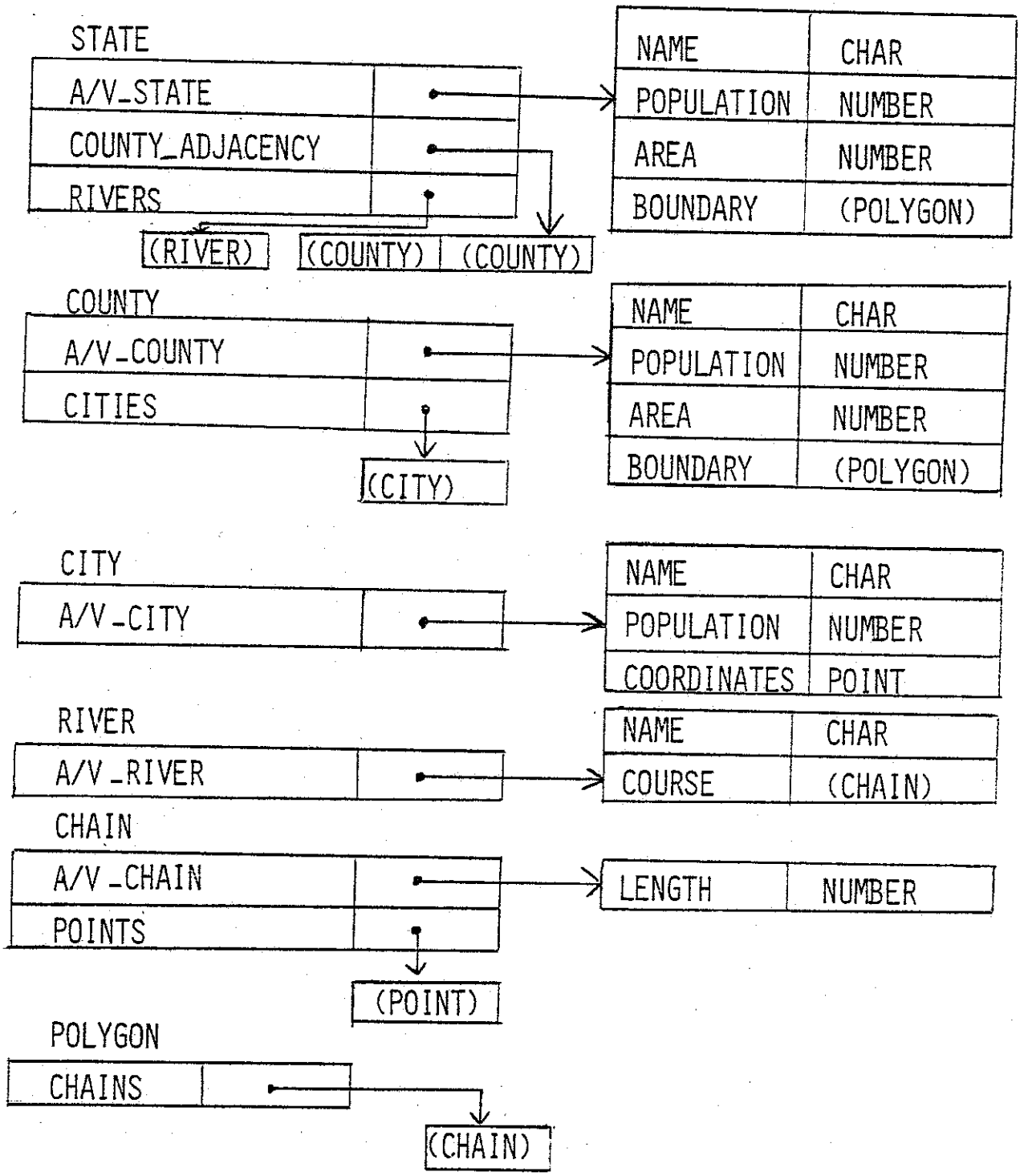


Figure 4 illustrates a set of prototypes for the spatial data structure types STATE, COUNTY, CITY, RIVER, CHAIN, and POLYGON and the relation types COUNTY ADJACENCY, RIVERS, CITIES, POINTS, and CHAINS.

spatial information system having the prototypes of Figure 4. Under each question, we suggest a sequence of operations that might be performed to answer the query.

1) What cities are in county X?

A. Locate county X.
B. For each CITY C in CITIES(X).
    1. Look up N = NAME(C).
    2. Add N to the relation being created.

2) What cities are in state X ?

A. Locate state X.
B. Perform a projection operation on COUNTY ADJACENCY(X) to obtain a list of counties.
C. For each county Y in the list
    For each city C in CITIES(Y)
        1. Look up N = NAME(C).
        2. Add N to the relation being created.

3) What is the length of river R?

A. Locate river R.
B. Return value of LENGTH (COURSE(R)).

4) What cities lie on rivers in state X?

A. Locate state X.
B. Perform a projection operation on COUNTY ADJACENCY(X) to obtain a list of counties.
C. For each county Y in the list
    For each city C in CITIES(Y)
        For each river R in RIVERS(x)
        if POINT_CHAIN_DISTANCE(COORDINATES(C), COURSE(R))=0
        then add C to the relation being created.

5) What cities are within 15 miles of a river in state X?

same as 5) except change '=0' to '≤15'.

6) What counties in state X does river R flow through?

A. Locate state X.
B. Perform a projection operation on COUNTY ADJACENCY(X) to obtain a list of counties.

C. For each county Y in the list
if CHAIN_INTERSECTS_POLYGON(COURSE(R), BOUNDARY(Y))
then add Y to the relation being created.

7) What states does river R flow in?

For every state S
if R is an element of RIVERS(S) then add S to the relation being created.

8) What points do river X and river Y share?

A. Locate river X.
B. Locate river Y.
C. Construct the intersection of POINTS(COURSE(X)) with POINTS(COURSE(Y)).

9) Answer the following questions about the region R defined by the states X, Y, and Z.
A. Locate state X.
B. Locate state Y.
C. Locate state Z.
D. Create a new temporary spatial data structure R from the information in X, Y, and Z and dependent on the prototype for spatial data structures of type REGION. This will include such operations as finding sums of populations and areas and constructing a new boundary based on three existant boundaries.

From these sample queries, we find the following operations are necessary.

1. projection in the relational database sense
2. selection in the relational database sense
3. intersection or join in the relational databases sense
4. look up the value of an attribute
5. call on geometric or distance functions
6. create a list
7. add elements to a list

8. comparison
9. determine if an N-tuple is a member of a relation
10. create a new relation
11. create a new spatial data structure

The system will also require a number of relation utility functions. To give the reader some feeling for these, we list several of the utility access operations that are required.

- return the i'th N-tuple in a relation
- return the position in a relation at which a given N-tuple is found
- return the next N-tuple in a relation
- insert an N-tuple in a relation
- delete an N-tuple from a relation
- delete a relation from a spatial data structure
- add a relation to a spatial data structure
- delete a spatial data structure from the system
- copy a relation
- copy a spatial data structure
- catalogue a new relation
- catalogue a new spatial data structure
- construct the union of two relations
- construct the intersection of two relations

Geometric Operations

Thus far we have emphasized operations which explicitely depend on the relational kind of information the system might have stored In this section, we illustrate how a basic geometric operation such as interactive region editing might be done. For this example, we use the run length encoded raster format data. The algorithm sketched here is more general than the one given in Peguet (1979).

In the region editing situation, a connected region is given and an operator desires to change the boundary of the region. The change can consist of adding area to the region and/or subtracting area from the region. To specify the required change, the operator can draw an arc overlaying a picture of the given region and can designate which side of the arc is associated with area to be inside the region and which side of the arc is associated with area tc be outside the region.

Figure 5 illustrates an example region having a boundary shown as a solid line and the operatcr-specified arc shown as a dashed line. Notice that the arc crosses the boundary. We view this situation as one in which the arc is partitioned into two pieces by the boundary which crosses it. The arc segment which lies entirely outside the region can have its sides designated as inside and outside in two possible ways. These are show in Figure 6. Likewise, the

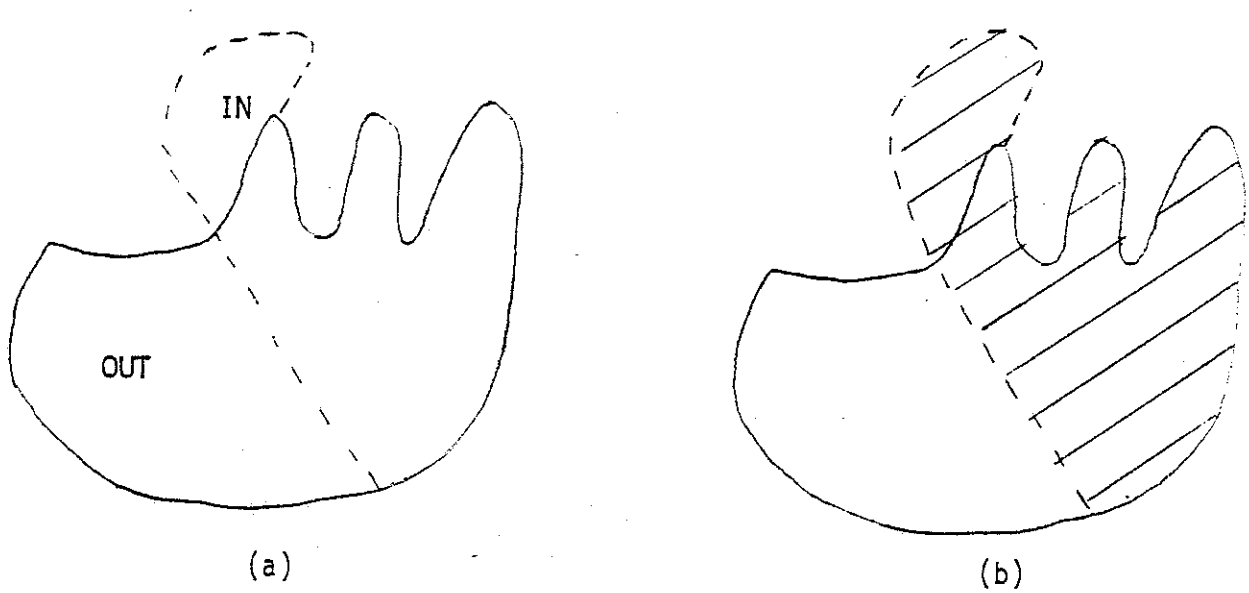


Figure 5(a) shows a region enclosed by the solid boundary and an operator drawn arc which designates the way in which the region is to be modified. Figure 5(b) shows the new region after modification as the hatched area.

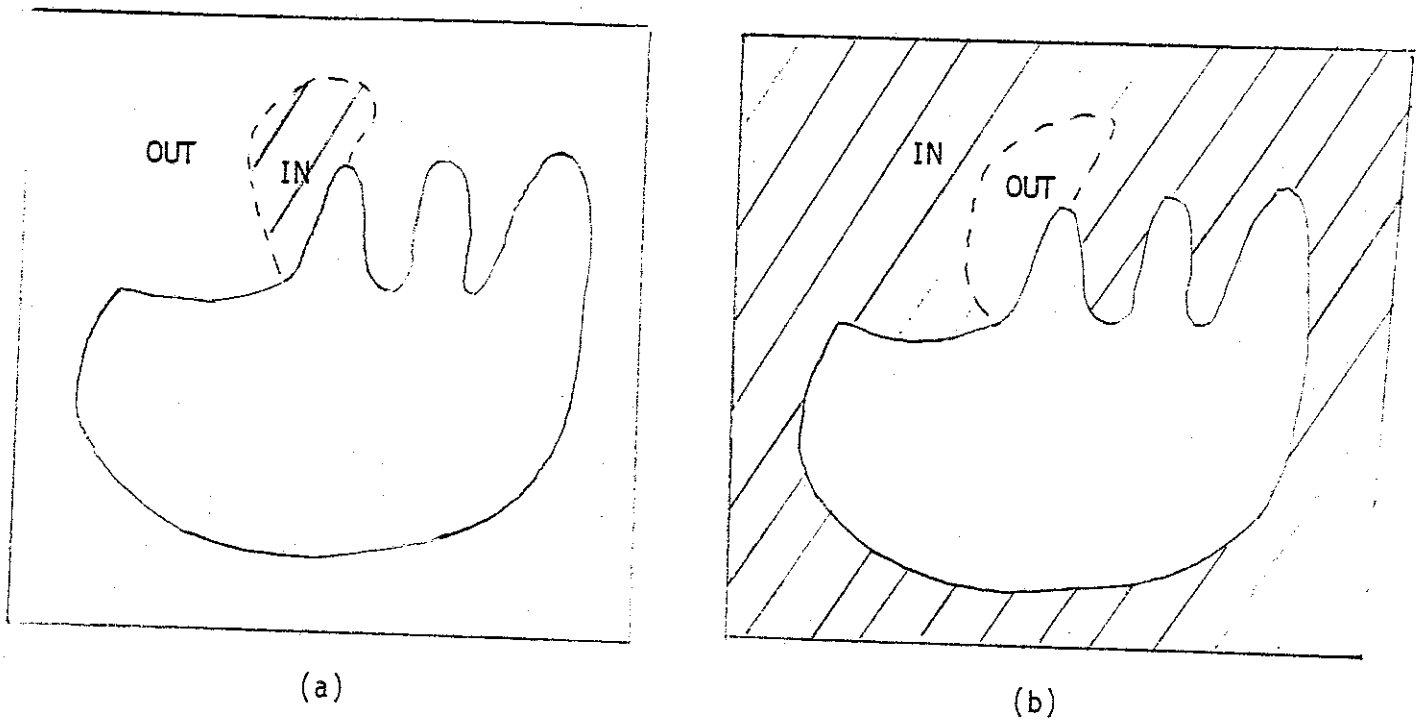


Figure 6 illustrates how an arc lying outside the region can have its sides designated in two possible ways, thereby defining two different areas (shown as hatched) which may be added to the area of the original region.

arc segment which lies entirely inside the region can have its sides designated as inside and outside in two possible ways. These are shown in Figure 7.

One algorithm which can do the region editing begins with the original arc divided into segments, each of which is entirely inside or outside the region and each of which touches the region boundary only at its beginning or ending point. The algorithm can be explained in terms of coloring. For an arc segment which lies outside the region, locate any point on the "in" labeled side of the arc. Color this point. Then color all points reachable from this point without crossing either the arc segment or region boundary. All such colored points must be subtracted from the region.

Coloring regions requires a connected components labeling algorithm (Rosenfeld and Kak, 1976). First the vertical boundaries of the operator-specified arc are used to modify the interval lists of the region to be edited. Then each interval is considered as a node in a graph. If a pair of intervals or successive rows has overlapping columns and the pair of rows for the duration of the overlapping columns is not separated by a horizontal segment of the operator-specified arc, then the corresponding nodes in the graph are linked together. Each connected component of the resulting graph corresponds to a connected region whose intervals are defined by the nodes of the graph component. The edited region will correspond to exactly one of the

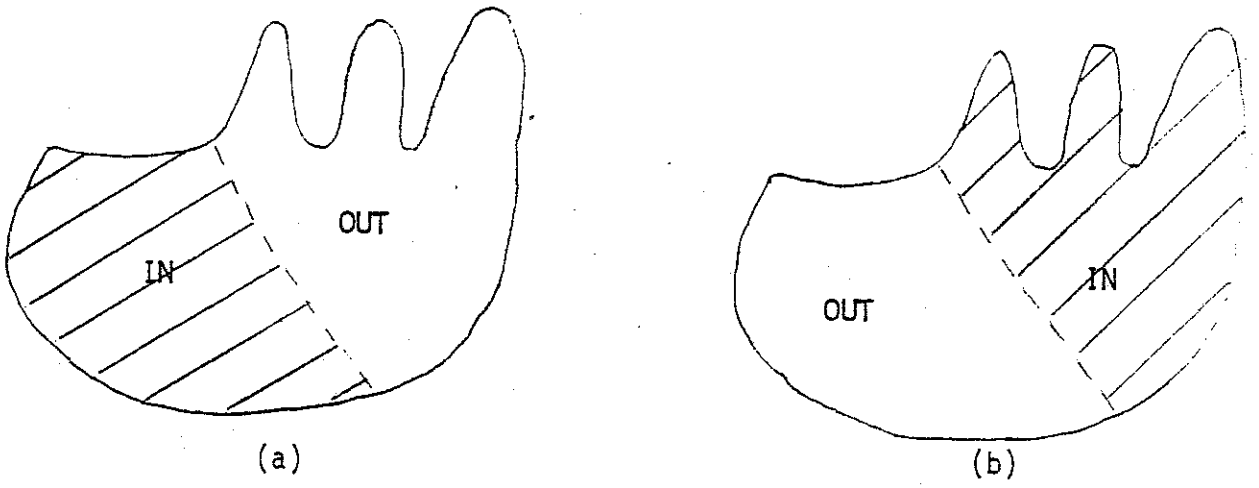


Figure 7 illustrates how an arc lying inside the region can have its sides designated in two possible ways thereby defining two different areas (shown as hatched) which may become the new region.

components determined by the labeling algorithm. Which component it is can be determined by locating any part of its boundary which is coincident with some part of the operator specified arc and seeing if the inside of the region corresponds to the "inside" label of that boundary segment.

To carry out this kind of operation the following two geometric primitives are required.

1) Search through all the horizontal strips in the interval lists specified by the region and/or its complement to determine that strip containing a given (row,column) pair.

2) Given the ordered interval lists for a pair of rows, determine all intervals having overlapping columns.

In addition, the relational primitive of transitive closure of a binary relation is required in order to determine the connected components of the associated graph.

Of course for other basic queries, the system will require a number of geometric utility functions. Included are

1) interval list intersection,

2) interval list union,

3) interval list complement, and

4) interval list growing by a specified distance both horizontally and vertically.

IV. HOMOMORPHISMS ON SPATIAL DATA STRUCTURES

A different kind of question that can be asked about map data is whether two entities have similar structures. For example, it might be interesting to compare the road network structures around two cities. A function that preserves structure is called a homomorphism. If there is a homomorphism from one structure to a part of another then we have a basis for considering the two structures similar and comparing them further. Since the spatial data structure is a recursive structure, we will define a homomorphism for this structure with a recursive definition. First we define the composition of a function with a relation.

Let $R1 \subseteq S1 \times S2 \times \ldots \times SN$ and $R2 \subseteq T1 \times T2 \times \ldots \times TN$ be two N-ary relations, and let h be a function from $S = S1 \cup S2 \cup \ldots \cup SN$ to $T = T1 \cup T2 \cup \ldots \cup TN$. The composition of R1 with h is defined by $R1 \circ h = \{(t1,\ldots,tN) \in T1 \times \ldots \times TN \mid$ there exists $(s1,\ldots,sN) \in R1$ with $h(si) = ti,\ i=1,\ldots,N\}$.

Thus the composition of an N-ary relation with a function is another N-ary relation. If $R1 \circ h = R2$, then R1 and R2 have the same structure. If $R1 \circ h \subseteq R2$, then R1 has the same structure as a subset of R2. A spatial data

structure is a set of relations. Two spatial data structures can be considered similar if each of their common relations have similar structures. However, we may wish to compare two structures on the basis of only some of their common relations. This motivates the following definition of homomorphism.

Let D1 and D2 be two spatial data structures. For each relation R in D1 U D2, there is an integer N(R) and a sequence of sets S(1,R),..., S(N(R),R) such that $R \subseteq S(1,R) X \ldots X S(N(R),R)$. A homomorphism from D1 to D2 is a pair (f,F) where

1) f is a function from a subset D of D1 to D2 satisfying for every $R \in D$, $N(R) = N(f(R))$.

2) $F = \{(h_R, H_R) \mid R \in D\}$ where

A) h_R is a function from
$S(R)=S(1,R) U.....U S(N(R),R)$ to
$S(f(R))=S(1,f(R)) U.....U S(N(f(R)),f(R))$
satisfying

a) $s \in S(R)$ is a spatial data structure iff $h_R(s) \in S(f(R))$ is a spatial data structure

b) $R \circ h_R \subseteq f(R)$

B) H_R is a set of homomorphisms, $H_R = \{G_S \mid s \in S(R)\}$ satisfying that if s is a spatial data structure then G_S is a homomorphism from s to $h_R(s)$.

Intuitively, if D1 and D2 are two spatial data structures and D is a subset of the relations of D1, then f is a function that maps each relation R in D to a relation f(R) of D2 of the same order. For each such pair of relations (R,f(R)), h_R is a function that maps each element of the domain of R to an element of the domain of f(R). Since these elements can be either atoms or themselves spatial data structures, h_R is restricted to map atoms to atoms and spatial data structures to spatial data structures. Furthermore, if h_R maps a spatial data structure s to another spatial data structure $h_R(s)$, then there must be a homomorphism from s to $h_R(s)$. H_R is the set of such homomophisms.

To illustrate the recursive homomorphism, we will look at a simple, abstract example. Figure 8 shows two spatial data structures SDS1 and SDS2. A homomorphism (f,F) can be defined as follows. Let D be the subset of SDS1 defined by D = {A/V1,R1} and define f:D -> SDS2 by f(A/V1) = A/V2 and f(R1) = R3. This function f satisfies our restriction in that it maps order 2 relations to order 2 relations.

The set F consists of the pair $(h_{A/V1}, H_{A/V1})$ and (h_{R1}, H_{R1}). Let $h_{A/V1}(a1)=a5$, $h_{A/V1}(a2)=a4$,

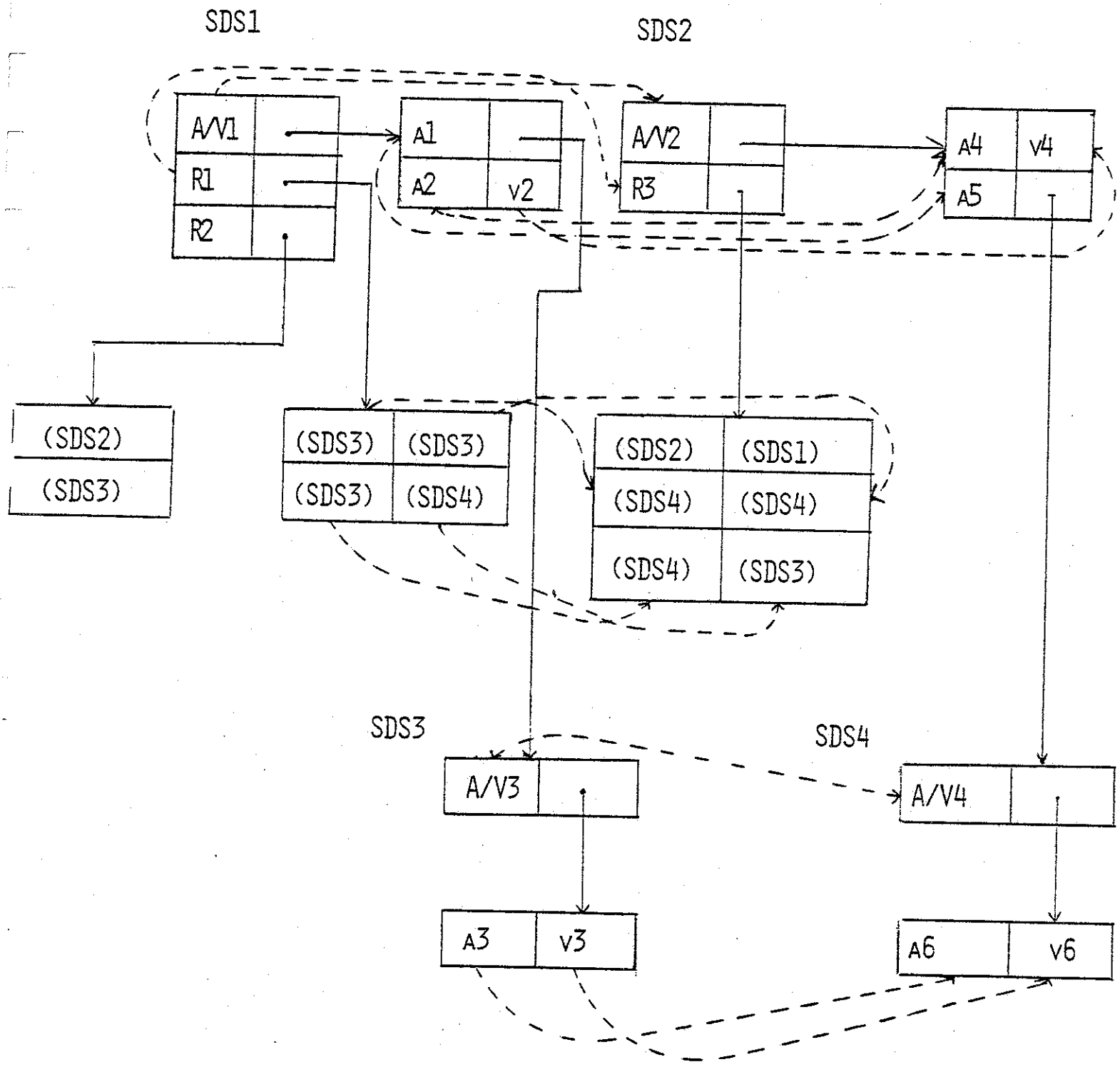


Figure 8 illustrates a homomorphism from spatial data structure SDS1 to spatial data structure SDS2.

$h_{A/V1}(SDS3)=SDS4$, and $h_{A/V1}(v2)=v4$. Then $h_{A/V1}$ maps atoms to atoms and spatial data structures to spatial data structures. (It also maps attributes to attributes and their values to the corresonding values, which is a desirable requirement for attribute-value table relations.) Since $h_{A/V1}$ maps spatial data structure SDS3 to spatial data structure SDS4, the set $H_{A/V1}$ must contain a homomorphism G_{SDS3} from SDS3 to SDS4. Let $G_{SDS3}=(g,G)$ where $g(A/V3) = A/V4$ and $G=\{(d_{A/V3},\emptyset)\}$. Let $d_{A/V3}(a3)=a6$ and $d_{A/V3}(a6)=v6$. Since a3, a6, v3, and v6 are atoms, no further levels of homomorphisms are needed.

Now, with respect to R1 and R3, let $h_{R1}(SDS3)= SDS4$ and $h_{R1}(SDS4)= SDS3$. Then $R1 \circ h_{R1} \subseteq R3$ as required in 2.A.b above, and we have already given a homomorphism G_{SDS3} from SDS3 to SDS4. In a similar manner we can construct a homomophism G_{SDS4} from SDS4 to SDS3. Thus 2.B is also satisfied, and we have a two-level homomophism from SDS1 to SDS2. The dashed arrows in Figure 8 illustrate this homomorphism pictorially.

Some discussion of the construction of this homomorphism is important. First, the function f was defined on the set of relations D={A/V1,R1}. The relation R2, although a part of SDS1, was left out of D since there was nothing in SDS2 of the same order that R2 could map to. In general, a measure of similarily must be defined which takes into account the percentage of the relations included

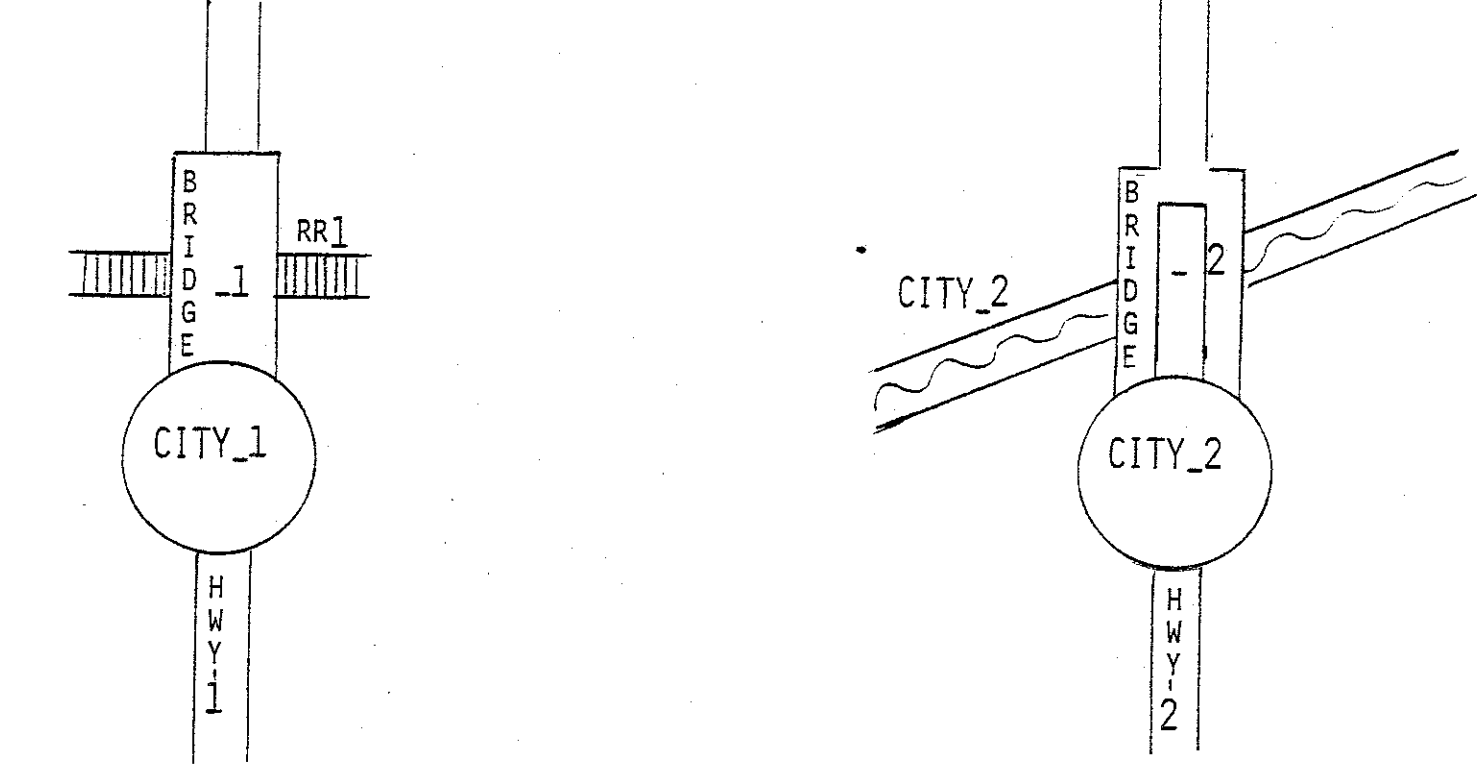
in each level of homomorphism. Clearly we can always define a null homomophism that uses relations at <u>every</u> level, but this would not be a very interesting mapping.

Second, the definition of homomorphism requires no special handling of the attribute-value table. In a previous paper (Shapiro and Haralick, 1978), we singled out the attribute-value table relations for special restrictions. The function mapping elements of pairs in one table to elements of pairs in a second table was required to map an attribute to an attribute and in this case, a homomorphism had to exist between their (spatial data structure) values. A stricter requirement might force an attribute in the first table to map to the same attribute in the second table.

Figure 9 shows two geometric spatial data structures GSDS1 and GSDS2. GSDS1 consists of a bridge, a city, a railroad, and a highway in a specified geometric relationship. GSDS2 consists of a bridge, a city, a river, and a highway in a similar relationship.

At the top level, the pair $(f,(h_{R1},H_{R1}))$ where

$f(R1) = R2$
$h_{R1}(BRIDGE\_1) = BRIDGE\_2$
$h_{R1}(CITY\_1) = CITY\_2$
$h_{R1}(HWY\_1) = HWY\_2$
$h_{R1}(RR1) = RIVER\_2$
$h_{R1}(entrance\text{-}to) = entrance\text{-}to$
$h_{R1}(routed\text{-}over) = routed\text{-}over$
$h_{R1}(crosses) = crosses$
$H_{R1} = \{G_{BRIDGE\_1}, G_{CITY\_1}, G_{HWY\_1}, G_{RR1}\}$
$G_{BRIDGE\_1} = G_{CITY\_1} = G_{HWY\_1} = G_{RR1} = \emptyset$



GSDS1

R1	

BRIDGE_1	CITY_1	ENTRANCE-TO
HWY_1	BRIDGE_1	ROUTED-OVER
BRIDGE_1	RR1	CROSSES

GSDS2

R2	

BRIDGE_2	CITY_2	ENTRANCE-TO
HWY_2	BRIDGE_2	ROUTED-OVER
BRIDGE_2	RIVER_2	CROSSES

Figure 9 illustrates two geographic spatial data structures that are similar at least at the top level.

is a homomorphism from GSDS1 to GSDS2. In order to extend the homomorphism to another level, we must define non-null homomorphisms from BRIDGE_1 to BRIDGE_2, CITY_1 to CITY_2, HWY_1 to HWY_2, and RR1 to RIVER_2. At each level, the homomorphisms may be weak or strong, depending on how many relations are compared and how much collapsing takes place. For instance, BRIDGE_1 and BRIDGE_2 have different physical attributes, but in some ways are still similar.

Finding homomorphisms in map data can provide interesting information about the structure of the data. We envision an interactive system where the user may specify which spatial data structures, which of their relations, and how many levels to compare. However, finding even one-level homomorphisms has been shown to be an NP-complete problem, although look-ahead operators have been proposed to speed up the search. Finding these multi-level homomorphisms is an interesting problem that we will be investigating in the near future.

V. SUMMARY

We have defined a spatial data structure that can be used to represent spatial objects. The structure consists of a set of N-ary relations often including an attribute-value table. The entries in the table and the objects on which the relations are defined may also be spatial data structures. Thus the spatial data structure is a recursive

structure.

The use of the spatial data structure was illustrated by a representation of a portion of a map of Virginia including counties and lakes. The spatial data structure was shown to be able to handle all the geographic data structures proposed by other researchers. A discussion of the manipulations required to answer queries about such a structure suggested that the database system should contain a control processor which when given a query whould determine all possible paths through the structure to answer the query and select the best path with the use of a cost function. The control processor would need a prototype of each kind of spatial data structure in the system and must have some knowledge of the semantics of the relations in the spatial data structures. The control processor might also possess some special purpose knowledge about particular objects in the systems. A study of the operations needed to answer queries concerning spatial data structures led to a list of suggested primitive operations in the system.

One high-level operation of interest in a system of spatial data structures is the matching of two structures. Since the spatial data structure is a recursive structure, the function mapping one spatial data structure to another can also be defined recursively. The definition of a spatial data structure homomorphism allows us to measure the similarity of two spatial data structures at one or more levels of the structures. The problem of finding these multi-level homomorphisms is the subject of our future work

in this area.

REFERENCES

1. Brassel, K., "A Topological Data Structure for Multi-Element Map Processing," An Advanced Study Symposium on Topological Data Structure for Geographic Information Systems, Harvard University, Cambridge, Massachusetts, October 1977.

2. Burton, W., "Representation of Many-Sided Polygons and Polygonal Lines for Rapid Processing," Communications of the ACM, Vol. 20, No. 3, March 1977, pp. 166-170.

3. Chamberlin, D. D. and R. F. Boyce, "SEQUEL: A Structured English Query Language", Proc. 1974 ACM SIGMOD Workshop on Data Description, Access, and Control.

4. Codd, E.F., "A Relational Model of Data for Large Shared Data Bases", CACM, Vol. 13, No. 6, June 1970, pp. 377-389.

5. Cook, D. and W. Maxfield, "The Development of a Geographic Base File and Its Uses for Mapping," Proceedings of URISA, Garden City, Long Island, September 1967.

6. Edwards, R.L., R. Durfee, and P. Coleman, "Definition of a Hierarchical Polygonal Data Structure and the Associated Conversion of a Georgraphic Base File from Boundary Segment Format," An Advanced Study Symposium on Topological Data Structure for Geographic Information Systems, Harvard University, Cambridge, Massachusetts, October 1977.

7. Fegaes, R., "The Graphic Input Procedure - An Operational Line Segment (Polygon Graphic to Digital Conversion)," An Advanced Study Sylmposium on Topological Data Structure fofr Geographic Information Systems, Harvard University, Cambridge, Massachusetts, October 1977.

8. Freeman, H., "Computer Processing of Line Drawing Images," Computing Surveys, Vol. 6, No. 1, March 1974, pp. 57-97.

9. Go, A., Stonebraker, M., and Williams, C., An Approach to Implementing a Geo-Data System, Memo No. ERL-M529, Electronics Research Laboratory, College of Engineering, University of California at Berkeley, 1975.

10. Gold, C., "Triangular Element Data Structures," Users Applications Symposium Proceedings, The

University of Alberta Computing Services, Edmonton, Alberta, Canada, 1976.

11. Hanson, A. and E. Riseman, A Progress Report on Vision: Representation and Control in the Construction of Visual Models, COINS TR 76-9, University of Massachusetts, Amherst, Massachusetts, July 1976.

12. Hewitt, C., "Procedural Embedding of Knowledge in PLANNER", Proceedings of the Second Joint Conference on Artificial Intelligence, London: British Computer Society, pp. 167-182.

13. Laboratory for Computer Graphics, "POLYVRT: A Program to Convert Geographic Base Files," Harvard University, Cambridge, Massachusetts, 1974.

14. Males, R., "ADAPT - A Spatial Data Structure for Use with Planning and Design Models," An Advanced Study Symposium of Topological Data Structures for Geographic Information Systems, Harvard University, Cambridge, Massachusetts, October 1977.

15. Merrill, R., "Representation of Contours and Regions for Efficient Computer Search", CACM, Vol. 16, 1973, pp. 69-82.

15. Pequet, D. J., "A Raster-Mode Algorithm for Interactive Modification of Line Drawing Data", Computer Graphics and Image Processing, Vol. 10, 1979, pp. 142-158.

16. Peucher, T.K. and N. Chrisman, "Cartographic Data Structures", The American Cartographer, Vol. 2, No. 1, April 1975, pp. 55-69.

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

18. Shapiro, L. G., "Data Structures for Picture Processing" to appear in Computer Graphics and Image Processing, 1979.

19. Shapiro, L. G. and R. M. Haralick, "A General Spatial Data Structure", Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, May 31-June 2, 1978, Chicago, pp. 238-249.

20. Shortcliffe, E. H., Computer-Based Medical Consultations, MYCIN, Elsevior, New York, 1976.

21. Switzer, W. A., "The Canada Geographic Information System", Automation in Cartography, eds. J. M. Wilford-Brickwood, R. Bertrand, and L. van Zuylen,

International Cartographic Association, The Netherlands, 1975.

22. Tomlinson, R., "A Geographic Information System for Regional Planning," Land Evaluation (Stewart, ed.), McMillian of Australia, Sydney, Australia, 1968.

23. Winograd, T. Understanding Natural Language, Academic Press, New York, 1972.