A GENERAL SPATIAL DATA STRUCTURE

Linda G. Shapiro
Department of Computer Science, Kansas State University
Manhattan, Kansas 66506

Robert M. Haralick
Departments of Electrical Engineering and Computer Science, University of Kansas
Lawrence, Kansas 66045

Abstract

The processing of images to extract regions, boundaries, and objects creates a spatial database which contains large quantities of information consisting of objects, their attributes, their locations, and spatial relationships. In this paper we deal with the problem of representing such spatial data in a uniform structure from which queries may be answered, commands may be carried out, and matching may be performed. We define a spatial data structure and illustrate its use in representing map data. We discuss the kinds of manipulations required in a spatial data base to answer queries about spatial data. We then show that matching of spatial data structures is a mathematical problem of finding homomorphisms from one spatial data structure to another.

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 partially or completely segmenting an image, associating attributes with the image segments, and determining relationships between segments. In this paper we are not concerned with how any of this processing takes place. 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 been camouflaged. 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.

Whether the form of interaction is a question or a command, finding the answer or returning the required structure involves searching the spatial 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.

In this paper we discuss map data and a formal representation structure which we call a spatial data structure. The structure is rich, flexible, and efficient enough to logically store any of the spatial information in maps, line drawings, region adjacency graphs, etc., that we might desire to represent.

In Section II we define the spatial data structure and give some specific examples which illustrate the use of the structure to represent spatial information. In Section III we discuss the manipulation of a spatial data base for answering queries. In Section IV we discuss the mathematical nature of spatial data matching problems.

II. Maps and the Spatial Data Structure

There is a variety of information that can appear on a map, and our discussion here is intended to be representative but not inclusive. First we will give definitions of some frequently used concepts, and then we will define the spatial data structure and illustrate the use of this structure to represent these concepts. The following definitions are our data structure definitions for basic concepts relevant to spatial data. They are not necessarily the same definitions a cartographer would use. See Robinson[13].

A point is an ordered pair (x,y) where x represents the longitude and y the latitude of a physical point on a map. A node is a point together with an attribute-value table. Thus a node is a point that carries more information than just its coordinates. A city or road junction can be a node. A chain is an ordered set of points where the first and last points are nodes called the start node and the end node. A chain represents a directed curve line segment from its start node to its end node passing through each of the given points. Hence the chain must implicitly contain information about the intended interpolation function that will be used to fill in points between adjacent pairs of the given points. A line is an ordered set of chains such that the last point of each chain is the first point of the next chain. Figure 1 illustrates these concepts.

CH1318-5/78/0000-0238$00.75

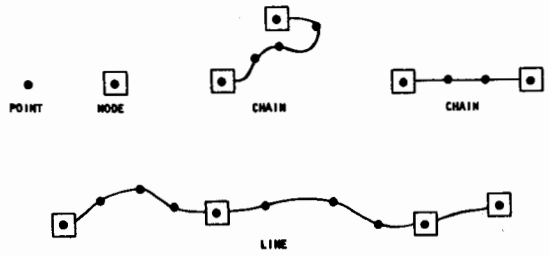


Figure 1 illustrates the concepts of point, node, chain, and line.

A simple boundary is a line where the start node of the first chain coincides with the end node of the last chain. A boundary is either a simple boundary or a simple boundary inside of which is a set of boundaries. This recursive definition allows us to have boundaries within boundaries within boundaries, and so on. Figure 2 gives some examples of boundaries. A boundary set is a set of boundaries. Generally the boundaries of a boundary set will be independent of each other (one boundary in the set will not be a part of another boundary of the set). The three boundaries of Figure 2 constitute a boundary set.

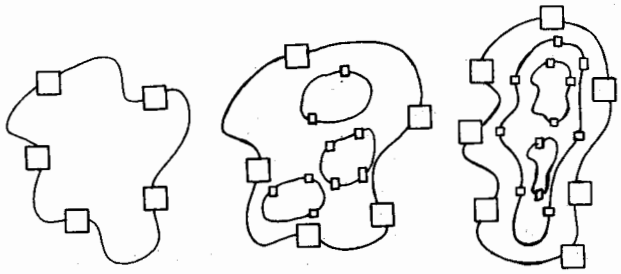


Figure 2 illustrates the concepts of boundary and boundary set.

A region is an entity having a boundary and usually containing other entities of interest such as nodes or chains of various sorts. A region adjacency relation is a binary relation associating each region with every other region which neighbors it. A chain region adjacency relation is a ternary relation which associates with every chain the region to its left and the region to its right as one travels along the chain from its start node to its end node.

As can be seen from these examples, there are many different types of information in maps: points, sequences of points, sequences of chains, tables, binary relations, ternary relations, and so on. In order to facilitate the discussion of the ways in which these different kinds of information relate to one another, we wish to adopt a unified representation. We will call our representation structure a spatial data structure. In order to define the spatial data structure, we first define an atom and an attribute-value table.

An atom is a unit of data that is not to be broken down further. For example, integers and character strings are commonly used atoms.

An attribute-value table is a set of pairs of the form (a,v) where a is an attribute and v is its value. 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 is an ordered pair $S = (A,R)$. A is an attribute-value table and R is a set $R = \{R_1,\ldots,R_K\}$ where for each $k = 1,\ldots,K$, there exists a positive integer N_k such that R_k is an N_k-ary relation on a set S_k. If there is a set of labels L_k which is associated with R_k, then $R_k \subseteq S_k^{N_k} \times L_k$. Otherwise $R_k \subseteq S_k^{N_k}$. Both the elements of S_k and the labels of L_k may be atoms or spatial data structures.

Notice that if we wish a spatial data structure to contain a set, then we can set $R_k \in R$ to be equal to S_k and not have any associated label set L_k. If we wish a spatial data structure to contain an ordered set or sequence, then we can set $R_k \in R$ to be $R_k \subseteq S_k \times L_k$ where L_k is a subset of the integers and R_k assigns to each integer in L_k an element of S_k.

A node N can be represented by the spatial data structure $N = (A,\emptyset)$ where the attribute-value table A contains information about the node such as its coordinates. A chain C can be represented by the spatial data structure $C = (A,R)$ where A contains attributes such as length, start node, and end node, and R consists of the labeled relation R which is the ordered set of points in the chain. A line $\mathcal{L}$ can be represented by the spatial data structure $\mathcal{L} = (A,R)$ where the attribute-value table A contains global attributes of the line such as length or number of chains, and the set of relations R contains the singleton relation R which indicates the linear ordering of the chains that compose the line. A boundary B can be represented by the spatial data structure $B = (A,R)$ where the attribute-value table A contains the global attributes of the boundary such as the name of the region it surrounds, perhaps the smallest rectangle enclosing the boundary, and a name or designation of the outermost simple boundary that contains all the inner boundaries. The set of relations R contains the singleton relation R which is just the set of boundaries interior to the outermost boundary.

An area containing many regions can be represented by a spatial data structure whose attribute-value table contains information such as

population, smallest enclosing rectangle, and a designation of the spatial data structure representing the boundary. The relation set $\mathcal{R}$ could contain the region adjacency relation, the chain region adjacency relation, and perhaps the set of cities in the area. If one of the purposes for the spatial data structure were automated cartography, then the sequence of chains that constitute all boundaries for the area would be a labeled unary relation in $\mathcal{R}$.

In the remainder of this section we will illustrate the use of the spatial data structure in representing the typical combination of items that can appear in a map. For this purpose we use diagrams which show the logical connection between the data items which are stored. We do not address the problems of physical implementation such as organization of tables or lists and format of scalar data. Figure 3 shows the logical storage organization for the spatial data structure.

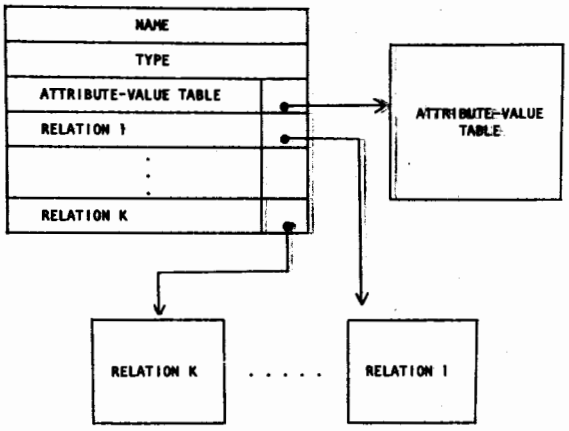


Figure 3 illustrates the logical storage organization for the spatial data structure.

Every spatial data structure has a header which has fields for name and type and perhaps other important identifying information. Below the header is a table having two columns. The first column is for names of relations which are the parts of the spatial data structure and the second column is for the pointers to them. The attribute-value table is considered to be a specially designated binary relation. Because of its importance, the first row in the table has the information about the attribute-value table. The attribute-value table and any of the relations associated with a spatial structure can have entries which are atoms or other spatial data structures.

As an example of the use of the spatial data structure, consider a map of Kansas containing cities, roads, rivers, and county boundaries. Figure 4 shows a portion of such a map. To represent this map, we need to represent the cities, important road junctions, road segments between cities or junctions, roads, county boundary segments, river segments, and rivers.

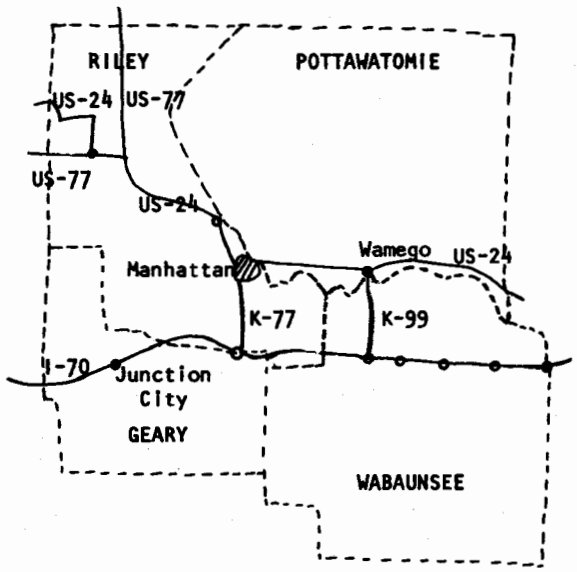


Figure 4 shows a portion of a road map of Kansas containing cities, roads, rivers, and county boundaries.

A city can be viewed as a node or as a bounded area. In this example we will use the first view and represent both cities and road junctions as spatial data structures of type NODE. Road segments, county boundary segments, and river segments will be represented by spatial data structures of type CHAIN, roads and rivers will be represented by spatial data structures of type LINE, and counties will be represented by spatial data structures of type REGION.

Figure 5 illustrates the spatial data structures representing cities, road junctions, and road segments for a portion of the map of Figure 4. The spatial data structure for each city is of type NODE and has an attribute-value table containing values for the attributes KIND, COORDINATES, POPULATION, and COUNTY. The value of the attribute COORDINATES is a point. The value of the attribute COUNTY is a pointer to the spatial data structure representing the county that the city lies in. The spatial data structure for a road junction is also of type NODE. The value of its KIND attribute specifies that it is a road junction and otherwise its attribute-value table is a subset of the city attribute-value table. The NODE spatial data structure contains a relation called ROAD SEGMENTS THROUGH. As shown in Figure 5, it is a unary, unordered relation (a set) whose elements are pointers to the spatial data structures representing road segments that pass through the city or junction represented by the node.

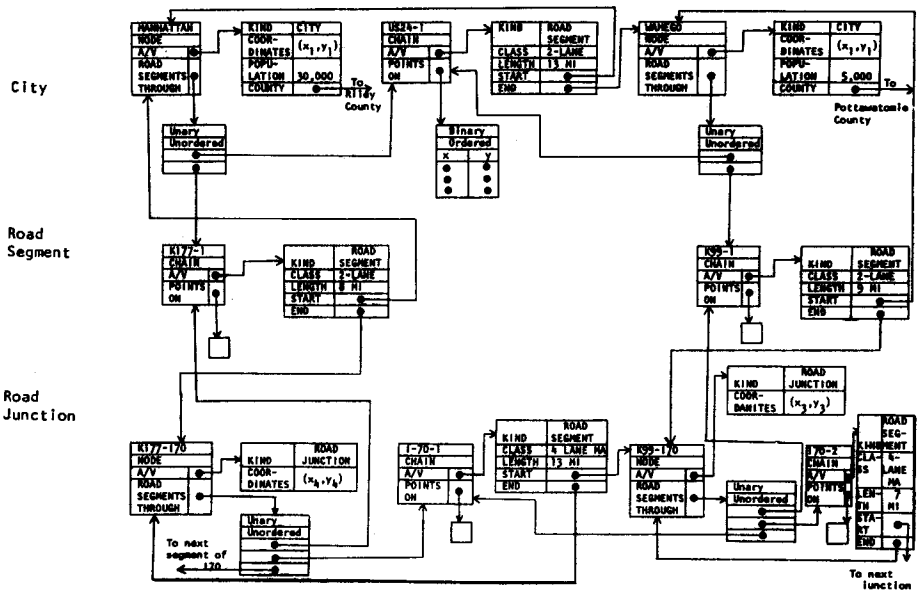


Figure 5 illustrates the spatial data structures representing cities, road junctions, and road segments for a portion of the map of Figure 4.

The spatial data structure for each road segment is of type CHAIN. The attribute-value table of this structure contains values for the attribute KIND (road segment), CLASS, LENGTH, START, and END (pointers to the start and end nodes of the chain). The relation POINTS ON is a binary, ordered relation consisting of the sequence of (x,y) coordinates of the points of the chain.

Figure 6 illustrates the spatial data structures for roads and counties. Roads are represented by structures of type LINE. Their attribute-value tables contain values for the attributes KIND (ROAD) and CLASS. The relation SEGMENTS OF is an ordered list of the road segments comprising the road. The county is represented by a spatial data structure of type REGION. Its attribute-value table contains values for the attributes KIND (COUNTY) and BOUNDARY. The value of BOUNDARY is a pointer to a structure of type SIMPLE BOUNDARY. This structure contains a relation SEGMENTS OF which is an ordered list of the boundary segments of the county. The segments are chains which can represent road segments, river segments, or just plain boundary segments. The CITIES IN relation in the region structure is an unordered list of cities in the county.

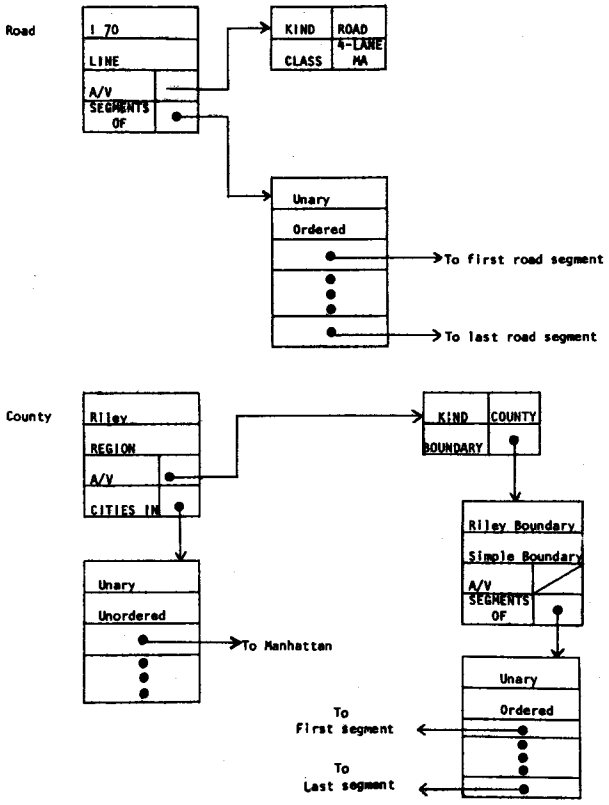


Figure 6 illustrates the spatial data structures representing roads and counties.

III. Manipulating the Spatial Data Base

The spatial data structure essentially allows certain kinds of information to be found quickly by following the pointers and traveling through the structure. For simple spatial data base manipulations, such pointer following may lead directly to the exact information desired. For more complex spatial data base manipulations, the pointer following in the data structure might only be able to specify a set which might have to be searched exhaustively or with some special procedure to select exactly the desired information. Hence, spatial data base manipulations may have two components: one concerned with the best way to travel through the structure and the other concerned with how to exhaustively search the set found by traveling through the structure.

In this section we discuss general methods for determining the different paths through the structure. We begin with an example query. Suppose we would like to find all the cities in Douglas County. Since Douglas County is a named region, we could look up Douglas County in a directory and see if there is a unary relation called "cities in" which is in the relation list for Douglas County. If so, then we follow the pointers to the cities in the list, copy the information and we are done. This would be an example of satisfying a query in a simple way.

If the unary relation called "cities in" is not in the relation list for Douglas County, then the information desired is either more difficult to obtain or perhaps not even obtainable. Suppose that the region Douglas County does have a relation which is called "river segments in" and suppose each river segment chain has a relation which is called "cities on." By the meaning of the words "in" and "on", if a city is on a river segment and the river segment is in a county, then the city is in the county. We will assume that this sort of knowledge about word meanings and relationships is available to the control processor that manipulates the spatial data system. For examples of the use of knowledge in intelligent systems, see Winograd[16] and Hewitt[10]. Thus by following the pointers through the structure, the set of all cities situated on river segments in Douglas County can be determined. This set is, of course, only a subset of the cities in Douglas County. However, if we also know as part of our global knowledge that all cities in Douglas County lie on river segments in Douglas County, then the control processor could determine that the subset of cities is, in fact, the entire set of cities in Douglas County. To be able to do this requires that the control processor be able to perform a limited amount of deductive reasoning. We illustrate this in the remainder of the section.

Suppose the control processor is given a query to the spatial data system and that the query has been suitably broken down and processed by a parser. For example, the query

FIND ALL CITIES IN DOUGLAS COUNTY

might be broken down as

```
OBJECT TO BE FOUND: CITY
NUMBER:             ALL
CONSTRAINTS:        IN DOUGLAS COUNTY
```

We would like the control processor to determine a path through the data structure resulting in the answer to the query. In fact, the control processor should be able to determine all such paths and then decide, possibly with some user interaction, on the best path to follow.

If the control processor is to determine one or more paths through the data structure, it needs some knowledge of the structure and of the spatial relationships stored in the structure. One kind of knowledge we can provide is a prototype of each kind of spatial data structure. (This is similar to a "schema" in database management terminology). A prototype of a spatial data structure includes the type of the structure, a prototype of the attribute-value table for that structure, and a prototype of each relation in the structure.

Figure 7 shows sample prototypes for the NODE and REGION structures that were used in Section II to represent cities and counties. The attribute-value table prototype of the NODE prototype tells the system that the attribute-value table of each NODE will contain a KIND attribute whose value is CITY or ROAD JUNCTION, a COORDINATES attribute whose value is a point, a POPULATION attribute whose value is an integer, and a COUNTY attribute whose value is a pointer to a spatial data structure of type REGION. The relation ROAD SEGMENTS THROUGH also found in the NODE prototype is indicated as a unary, unordered relation with a prototype element containing a pointer to a CHAIN structure.

The REGION attribute-value prototype tells the system that the KIND attribute will have the value COUNTY (until we add some more kinds of regions to our system) and the value of the BOUNDARY attribute will point to a spatial data structure of type SIMPLE BOUNDARY. The CITIES IN relation will be unary, unordered, and each element will contain a pointer to a NODE structure.

In order to answer a query concerning spatial relationships, the system must have some built-in understanding of those relationships. Some of the common spatial relationships that we might expect the system to know are IN, ON, CONTAINS, SURROUNDS, IS SURROUNDED BY, THROUGH, ADJACENT TO, and PART OF. One kind of knowledge of these spatial relationships is how they can be put together. Suppose, as in the example at the beginning of this section, that the REGION structure representing counties had a relation called RIVER SEGMENTS IN instead of the relation CITIES IN, and each CHAIN structure had a relation called CITIES ON. The system needs to know that "city on river segment" and "river segment in county" implies "city in county." More generally, we can state a spatial axiom concerning the relationships IN and ON:

a ON b and b IN c → a IN c

This axiom represents a kind of transitivity stated more generally by

on ∘ in → in

This transitivity knowledge might be stored as a table T where the rows and columns represent relations and $T(R_1,R_2) = R_3$ if for every a, b, and c, aR_1b and bR_2c imply aR_3c. Figure 8 illustrates such a table for the relations IN, ON, THROUGH, and ADJACENT TO. The table form of storing such knowledge is compact for binary relations. More general methods include production rules, Shortclife[14] and procedural knowledge, Hewitt[10] and Winograd[16].

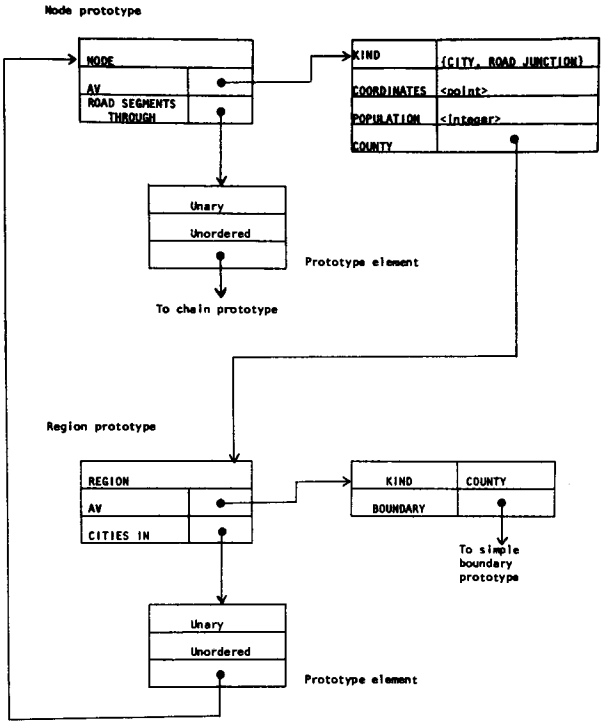


Figure 7 illustrates sample prototypes for the NODE and REGION structures.

T

	IN	ON	THROUGH	ADJACENT TO
IN	IN	ON	-------	ADJACENT TO
ON	IN	ON	-------	ADJACENT TO
THROUGH	IN	ADJACENT TO	THROUGH	ADJACENT TO
ADJACENT TO	ADJACENT TO	ADJACENT TO	-------	----------

Figure 8 illustrates a transitivity table T where $T(R_1,R_2) = R_3$ if aR_1b and bR_2c implies aR_3c.

Now suppose that the system is given the modified prototypes shown in Figure 9, the transitivity knowledge shown in Figure 8, the specific knowledge that all cities in Douglas County lie on river segments in Douglas County, and the query

FIND ALL CITIES IN DOUGLAS COUNTY

The system can generate two paths to the solution. To generate the first path, it first determines (by table look-up) that a CITY is represented by a spatial data structure of type NODE. Examining the NODE prototype, it finds the COUNTY attribute which points to a REGION structure representing a county. The system needs to know that the REGION structure, like all spatial data structures, contains a character string name which can be compared to the name DOUGLAS and must be able to deduce that the COUNTY attribute refers to the county that the city is in. Then the system can generate a procedure for answering the query. A high-level version of such a procedure is given below.

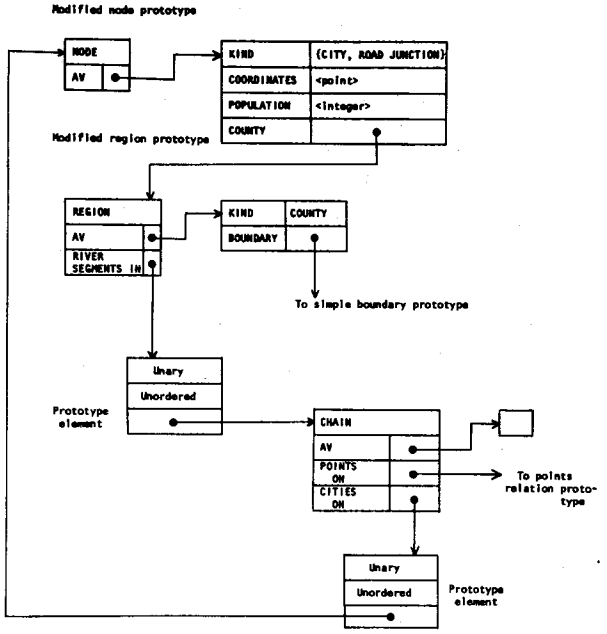


Figure 9 shows a modified set of prototypes to be used in answering the query "Find all cities in Douglas County."

```
procedure CITIES_IN_DOUGLAS_COUNTY_1;
ANSWER_LIST = the empty list;
do for each spatial data structure D of type
          NODE
  if the KIND attribute of D is CITY
  then if the name of the REGION pointed to
          by the COUNTY attribute is DOUGLAS
       then add D to ANSWER_LIST
       else continue;
end CITIES_IN_DOUGLAS_COUNTY_1;
```

This procedure represents the first path to the solution. To generate the second path, the system first determines that a COUNTY is represented by a REGION structure. The REGION prototype has no attribute or relation whose name contains the word CITY. On searching further, the system finds that the relation RIVER SEGMENTS IN contains elements which point to chains representing river segments and that the CHAIN prototype contains the relation CITIES ON whose elements point to NODE structures. Now the deductive portion of the system uses the transitivity table T to determine that RIVER SEGMENTS IN a county and CITIES ON a river segment together can provide a list of CITIES IN a county. It then uses the further knowledge that all cities in Douglas County lie on river segments in Douglas County to determine that the above list is equivalent to the list of all cities in Douglas County. This gives us the second path to the solution represented by the following procedure.

```
procedure CITIES_IN_DOUGLAS_COUNTY_2;
DOUG = the spatial data structure of type
       REGION and KIND COUNTY representing
       DOUGLAS COUNTY;
ANSWER_LIST = empty list;
do for each river segment S in the RIVER
       SEGMENTS IN relation of DOUG;
       add all the elements of the CITIES ON
       relation of S to ANSWER_LIST;
end CITIES_IN_DOUGLAS_COUNTY_2;
```

After the system has generated the two (or more) paths to the answer, it must evaluate these paths using a cost function and basing its evaluation on certain properties of the data base. For instance, in procedure CITIES_IN_DOUGLAS_COUNTY_1, the cost of executing the nested if-then-else statement must be multiplied by the number of nodes in the database. In procedure CITIES_IN_DOUGLAS_COUNTY_2, the cost of finding the structures representing Douglas County must be added to the product of the number of river segments in Douglas County and the cost of accessing and copying the CITIES ON relation of each river segment. With most map data we would expect the second procedure to be the less costly.

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. (For graph homomorphisms, see Harary[9]). If there is a homomorphism from one structure to a part of another structure, 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 $R_1 \subseteq S_1^N$ and $R_2 \subseteq S_2^N$ be two N-ary relations and let h be a function from S_1 to S_2. The composition of R_1 with h, denoted by $R_1 \circ h$, is defined by

$$R_1 \circ h = \{(s_1,\ldots,s_N) \in S_2^N \mid \text{there exists } (s_1',\ldots,s_N') \in S_1^N \text{ such that } h(s_i') = s_i,\ i = 1,\ldots,N\}.$$

Thus the composition of an N-ary relation with a function is another N-ary relation. If $R_1 \circ h = R_2$, then R_1 and R_2 have the same structure. If $R_1 \circ h \subseteq R_2$, then R_1 has the same structure as a subset of R_2. If $R_1 \subseteq S_1^N \times L_1$ and $R_2 \subseteq S_2^N \times L_2$ are two labeled N-ary relations, h is a function from S_1 to S_2, and h_L is a function from L_1 to L_2, the composition of R_1 with the pair (h,h_L) denoted by $R_1 \circ (h,h_L)$ is defined similarly by

$$R_1 \circ (h,h_L) = \{(s_1,\ldots,s_N,\ell) \in S_2^N \times L_2 \mid \text{there exists } (s_1',\ldots,s_N',\ell') \in S_1^N \times L_1 \text{ such that } h(s_i') = s_i,\ i = 1,\ldots,N \text{ and } h_L(\ell') = \ell\}.$$

A spatial data structure contains an attribute-value table and a set of relations. Two spatial data structures can be considered similar if

(1) Each of their common attributes have similar values

and

(2) Each of their common relations have similar structures

However, we may wish to compare two structures with respect to a subset of their common attributes and relations. This motivates the following definition.

Let $D_1 = (A_1,R_1)$ and $D_2 = (A_2,R_2)$ be two spatial data structures. Let A_1 be the set of attributes of A_1, A_2 be the set of attributes of A_2, and $\alpha \subseteq A_1$. For each $R \in R_1 \cup R_2$, assume there is a set S_R and an integer N_R such that $R \subseteq S_R^{N_R}$. Let $\rho \subseteq R_1$.

A homomorphism from D_1 to D_2 with respect to (α,ρ) is a 4-tuple (f,F,h,H) where

(1) f is a function from α to A_2 satisfying that if the value v_a of attribute a is a spatial data structure, then so is the value $v_{f(a)}$ of f(a).

(2) F is a set $\{f_a \mid a \in \alpha\}$ satisfy that if the value v_a of a is a spatial data structure, then f_a is a homomorphism from v_a to $v_{f(a)}$.

(3) h is a function from ρ to R_2 satisfying for every $R \in \rho$, $N_R = N_{h(R)}$.

(4) $H = \{(h_R, H_R) \mid R \in \rho\}$ where

(i) h_R is a function from S_R to $S_{h(R)}$ satisfying:

(a) if $s \in S_R$ is a spatial data structure, so is $h_R(s)$
and
(b) $R \circ h_R \subseteq h(R)$

(ii) H_R is a set $\{g_s \mid s \in S_R\}$ satisfying that if s is a spatial data structure, g_s is a homomorphism from s to $h_R(s)$.

Intuitively, if D_1 and D_2 are two spatial data structures and α is a subset of the attributes of D_1, then each attribute in α must map to an attribute of D_2. If the value of an attribute is a spatial data structure, then the value of the attribute it maps to must be a spatial data structure and there must be a homomorphism between these two structures. If ρ is a subset of the relations, there must be a function that maps each relation in ρ to a relation of D_2 of the same order. For each such pair of relations (R_1, R_2), there must be a function that maps elements of the set S_{R_1} to elements of the set S_{R_2}. This function must map spatial data structures to spatial data structures and its composition with R_1 must be a subset of R_2. Furthermore, if this function maps a spatial data structure to another, there must be a homomorphism from the first structure to the second.

The definition can be easily extended to labeled relations using the extended definition of composition defined earlier in this section. We will illustrate the concept of a homomorphism with an example.

Consider the two spatial data structures NAME1 and NAME2 shown in Figure 10. Let α be the set of attributes {a1,a2} and let f(a1) = a5 and f(a2) = a4. Since the value of a2 is an atom "at2", we do not have to make any further checks on a2. Since the value of a1 is a spatial data structure, we must check that the value of a5 is also a spatial data structure. Since this condition is met, the function f satisfies condition (1) of the homomorphism definition.

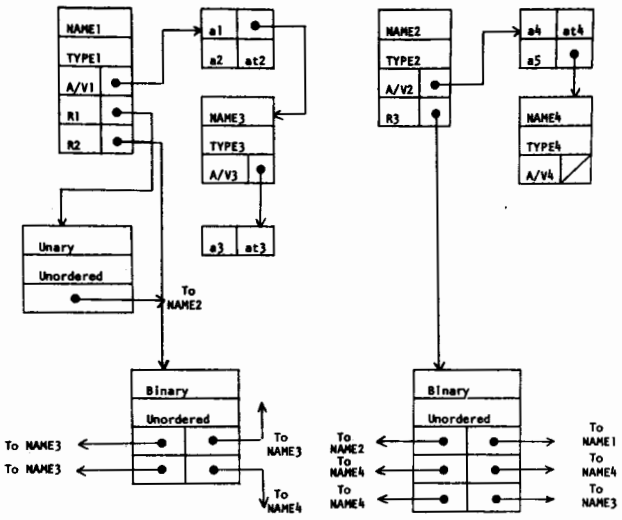


Figure 10 illustrates two spatial data structures NAME1 and NAME2 where there is a homomorphism from NAME1 to NAME2 with respect to ({a1,a2},{R2}).

Now let $F = \{f_{a1}, f_{a2}\}$. f_{a2} can be $\emptyset$ since the value of a2 is not a spatial data structure and we are not concerned with f_{a2}. f_{a1} must be a homomorphism from the spatial data structure NAME3 to the spatial data structure NAME4. In this case if we choose $\alpha' = \emptyset$, $\rho' = \emptyset$, and $f_{a1} = (\emptyset,\emptyset,\emptyset,\emptyset)$, f_{a1} is trivially a homomorphism from NAME3 to NAME4 with respect to $(\emptyset,\emptyset)$. Thus F satisfies condition (2) of the homomorphism definition.

Next, let $\rho = \{R2\}$ and let h(R2) = R3. Since both R2 and R3 are binary, unordered relations, this satisfies condition (3) of the homomorphism definition. It remains to define $H = \{(h_{R2}, H_{R2})\}$ where h_{R2} is a function from S_{R2} to S_{R3} and H_{R2} is a set $\{g_s \mid s \in S_{R2}\}$ of homomorphisms.

Now S_{R2} = {NAME3, NAME4} and S_{R3} = {NAME1, NAME2, NAME3, NAME4}.

Let h_{R2}(NAME3) = NAME4 and h_{R2}(NAME4) = NAME3.

Then $R2 \circ h_{R2}$ = {(NAME4, NAME4), (NAME4, NAME3)} $\subseteq$ R3 = h(R2). With this and the fact that h_{R2} maps spatial data structures to spatial data structures, condition (4) (i) is satisfied.

Finally, let $H_{R2} = g_{NAME3}, g_{NAME4}$ where $g_{NAME3} = (\emptyset,\emptyset,\emptyset,\emptyset)$ and $g_{NAME4} = (\emptyset,\emptyset,\emptyset,\emptyset)$. Again, g_{NAME3} is trivially a homomorphism from NAME3 to h_{R2}(NAME3) = NAME4 with respect to $(\emptyset,\emptyset)$, and similarly g_{NAME4} is a homomorphism from NAME4 to h_{R2}(NAME4) = NAME3 with respect to $(\emptyset,\emptyset)$. Thus the

condition (4) (ii) is satisfied and (f,F,h,H) is a homomorphism from spatial data structure NAME1 to spatial data structure NAME2 with respect to ({a1,a2},{R2}). Figure 11 illustrates the homomorphism pictorially.

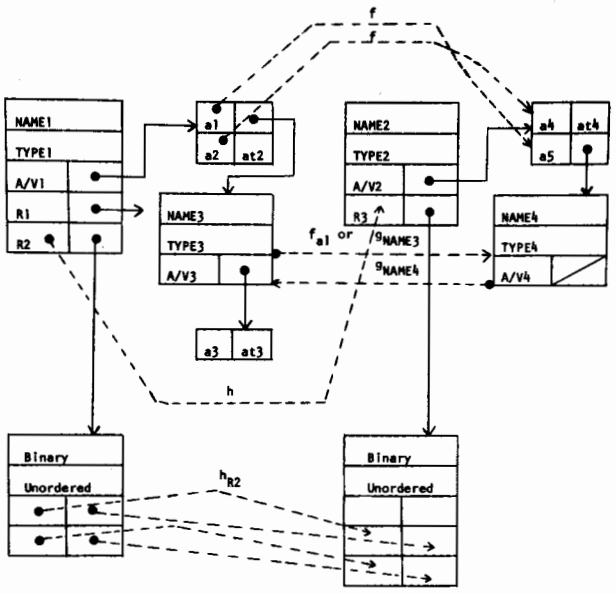
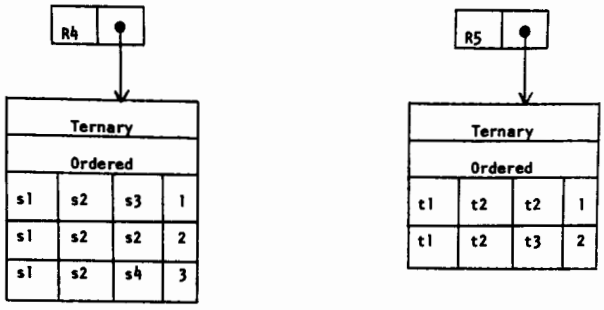


Figure 11 pictorially illustrates the homomorphism $(f, F = \{f_{a1},\phi\}, h, (h_{R2},H_{R2} = \{g_{NAME3},g_{NAME4}\}))$ from spatial data structure NAME1 to spatial data structure NAME2.

Some discussion of the construction of this homomorphism is important. The function f was defined on the set $\alpha = \{a1,a2\}$ which contains all of the attributes of A/V1. The function could also have been defined on {a1}, {a2}, or $\emptyset$. Of course, the more attributes there are in α, the stronger is the homomorphism. The homomorphisms f_{a1} and g_{NAME3} from NAME3 to NAME4 and the homomorphism g_{NAME4} from NAME4 to NAME3 were defined as the null homomorphism $(\emptyset,\emptyset,\emptyset,\emptyset)$ with respect to $(\emptyset,\emptyset)$. Clearly, the null homomorphism maps any spatial data structure to any other spatial data structure. In this case, the null homomorphism was the only homomorphism possible from NAME3 to NAME4 or from NAME4 to NAME3, since A/V4, R3, and R4 are all empty.

The function h was defined on the set $\rho = \{R2\}$. It was not possible to make ρ any larger since R1 is a binary relation and the spatial data structure NAME2 contains no corresponding binary relation. The function h_{R2} was the only possible function from S_{R2} to S_{R3} that satisfied $R2 \circ h_{R2} \subseteq R3$. Finally, the relations R2 and R3 are both unlabeled binary relations. If R2 were a labeled relation, then R3 would also be a labeled relation and the function h_{R2} would be an ordered pair (h_{R2},h_L) where h_L would map labels of L2 to labels of L3 and the extended definition of composition would be used.

For example, consider the labeled relations R4 and R5 of Figure 12. R4 and R5 are both labeled ternary relations where the labels specify an ordering on the triples. If h_{R4} and h_L are defined as shown in Figure 12, then $R4 \circ (h_{R4},h_L) \subseteq R5$. In fact, $R4 \circ (h_{R4},h_L) = R5$.



$h_{R4}(s1) = t1$
$h_{R4}(s2) = t2$
$h_{R4}(s3) = t2$
$h_{R4}(s4) = t3$

$h_L(1) = 1$
$h_L(2) = 1$
$h_L(3) = 2$

Figure 12 illustrates two labeled ternary relations R4 and R5 and a function (h_{R4},h_L) such that $R4 \circ (h_{R4},h_L) \subseteq R5$.

A spatial data structure is a recursive structure since both its attribute-value table and its relations may contain spatial data structures. A homomorphism from one spatial data structure to another is also recursive since if the data structure has several levels, so may the homomorphism. It may be that we are only interested in the homomorphism at the top level and can define all lower homomorphisms as the null homomorphism, or it may be that we want to compare the structure all the way to the bottom level.

Consider the two structures of Figure 13. ARRANGEMENT1 consists of a bridge, a city, a railroad, and a highway in a specified geometric relationship. ARRANGEMENT2 consists of a bridge, a city, a river, and a highway in a similar relationship. At the top level, the 4-tuple (f,F,h,H) where

f(COUNTY) = COUNTY

F = {the null homomorphism}

h(R) = R

h_R(BRIDGE1) = BRIDGE2

h_R(CITY1) = CITY2

h_R(HWY1) = HWY2

h_R(RR1) = RIVER2

h_L(entrance-to) = entrance-to

h_L(routed-over) = routed-over

h_L(crosses, 90^o) = (crosses, 45^o)

$H = \{g_{BRIDGE1}, g_{HWY1}, g_{CITY1}, g_{RR1}\}$

$g_{BRIDGE1} = g_{HWY1} = g_{CITY1} = g_{RR1}$ = the null homomorphism

is a homomorphism from ARRANGEMENT1 to ARRANGEMENT2 with respect to ({COUNTY},{R}). If we want the homomorphism to extend to another level, we must define non-null homomorphisms from the value of COUNTY in ARRANGEMENT1 to the value of COUNTY in ARRANGEMENT2 and from BRIDGE1 to BRIDGE2, CITY1 to CITY2, HWY1 to HWY2, and RR1 to RIVER2. At each level the homomorphisms defined may be weak or strong, depending on how many attributes and relations are compared and how much collapsing takes place. For instance, BRIDGE1 and BRIDGE2 have different physical attributes, but in some ways are still similar.

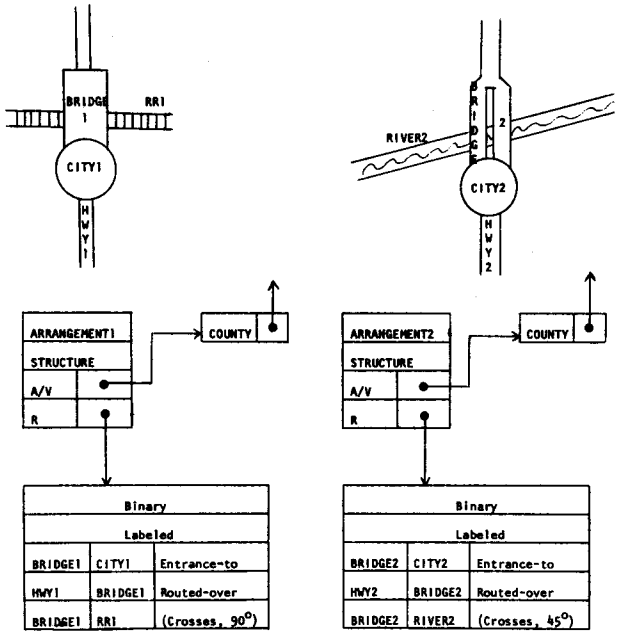


Figure 13 illustrates two geographic spatial data structures which are similar at least at the top level.

Finding homomorphisms in map data can provide interesting information about the structure of the data. 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. Related Literature

Computerized spatial data representation is a requirement for all scene analysis systems, automated cartographic systems, and geographic information systems. Because in this paper our examples have been in terms of map data, we will briefly review related ideas mostly in the geographic information system literature. We will show that each of the data structures used by previous researchers can be accommodated by the spatial data structure defined in this paper.

One of the earlier and successful geographic information systems is Tomlinson's Canadian Geographic Information System[15]. The basic spatial data structure used in this system consists of a directed boundary chain which points to its left and right regions. Each boundary chain also points to the two next boundary chains which continue bounding the region on the left and on the right. This structure is basically the simple boundary data structure described in this paper implemented as a linked list of boundary chains.

It is important for any spatial data structure to allow the representation of areas which are nested or which have holes. Those systems, like Tomlinson's which emphasize boundary connections to the next continuing boundary chain, usually detour a traversal of the exterior boundary of a region to the interior boundary of the region. When the traversal of the interior boundary is finished, another virtual chain detours the traversal back to the exterior boundary. Although such a system permits retrieving boundaries and keeping track of whether the current boundary includes or excludes areas in it, the mechanism is not elegant and perhaps even inefficient algorithmically.

The U.S. Census DIME files[3] has a basic structure consisting of straight line boundary chains which point to their left and right regions and which point to their beginning and ending nodes. This structure is the first fully topological structure and is the one which Hanson and Riseman[8] use in their VISIONS system. It is a structure efficient to represent all topological spatial relations between regions. Beginning with any directed boundary chain for a specified region on its left, the next boundary chain can be located by searching the DIME file for any boundary chain having the specified region to its left and having its starting node be the ending node of the given boundary chain. To find all regions adjacent to a given region, we need to search the DIME file for all boundary chains having for either their left region or right region the specified region. If the specified region is found on the right, an adjacent region will be found on the left and vice-versa. To find all the holes in a region, we need to begin with any boundary chain on the exterior of the region. We traverse this boundary until the entire exterior boundary has been found. Then we locate any boundary chain within this region having

one of its region pointers pointing to the initially specified region. A traversal using each such boundary chain will generate a hole in the region. Although the DIME file representation is sufficient to permit the finding of holes, it is not algorithmically efficient. In terms of the spatial data structure suggested in this paper, the basic DIME file structure for any region can be implemented as a quintary relation associated with the spatial data structure for that region.

The chain structure of POLYVRT[11] and LUDA[5] are logically similar to the chain spatial structure discussed in this paper. We have, however, not suggested that the points in a chain be stored sequentially as (x,y)-coordinates. We have left the physical storage mechanism open. In some applications storing differences or using a Freeman chain code[6] might be appropriate. In other applications storing the chain in more of a parametric functional form might be appropriate. Regardless of the physical form of the storage, the structural aspect of the representation is the same.

The hierarchical polygonal data structure of Edwards, Durfee, and Coleman[4] allows the same efficient retrieval of a region's interior boundaries (or nested zones) as the boundary data structure defined in this paper. However, the logical convention employed in the implementation differs from the ones suggested here.

Because of its computational efficiency, Brassel[1] suggests a hierarchically organized spatial data base of Thiessen polygons for determining the node which is closest to a given spatial position. The implementation of this structural organization easily fits into the spatial data structure we have defined in this paper. At each level of the hierarchy, the node defining each Thiessen polygon points to the nodes of neighboring Thiessen polygons. The distance between a given point and a starting Thiessen polygon node may be computed. Then using this connectedness structure, the neighboring Thiessen polygon nodes can be found, the distances between the given point to these nodes can be computed, and the closest of these nodes located. The node network can be traversed in this manner locating that Thiessen polygon node closest to the given spatial point. Then using the pointers from the closest node at the current level to the neighboring Thiessen polygon nodes at the next level the searching can be continued until the closest node at the lowest level has been found. A pointer to the set of Thiessen polygon nodes at the current level of the hierarchy as well as a pointer to the set of Thiessen polygon nodes at the next level of the hierarchy are easily stored in a unary relation (a set) of a node in the spatial data structure defined in this paper.

A spatial data structure sometimes used for representing surfaces on the basis of sampled points is the triangular data structure. Here a two-dimensional area is divided into a set of mutually exclusive triangles. The triangular partition is constrained in the following manner: if any non-zero length of one side of a triangle is adjacent to any other triangle, then the entire side of the one triangle must be an entire side of the adjacent triangle. The vertices of the triangles are nodes. For digital terrain models such nodes would contain elevation, slope, or other important samples surface information. To obtain the elevation or slope information at any place in a triangle, an interpolation may be used with a homogeneous coordinate system based on the three nodes sampled values[7,12].

In the triangular data structure, each triangle points to its adjacent triangle and to its vertices. Thus, for the spatial data structure defined in this paper to represent a triangulation we need only add a node list and an adjacent triangle list to the unary relations in each triangular region. The information of these lists can be easily obtained by traversing the boundary of a triangular region and picking up the required information from the nodes associated with each boundary chain.

Burton's polygonal representation[2] allows quick solution of point in a polygon and polygon intersection problems. It is based on breaking up a polygon into basic sections which are maximal length chains, monotonic in both x and y coordinates. The maximal length is not a requirement for the algorithm, although it will speed up the search. Burton's polygonal representation is easily put in the spatial data structure defined here: just make the chains used in the spatial data structure monotonic in both x and y coordinates. Hence, all that is required is either an explicit representation of the minimum and maximum x-y coordinates of each chain or a way to obtain the minimum and maximum from the beginning and ending nodes of each chain.

VI. Summary

We have defined a spatial data structure that can be used to represent spatial objects. The structure consists of an attribute-value table and a set of relations. 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 Kansas including cities, road junctions, road segments, roads, county boundary segments, counties, river segments, and rivers. 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 would 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 system. An example of finding all cities in a county illustrated some of manipulations necessary to answer queries concerning spatial data structures.

One 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. 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.

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

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

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

7. Gold, C., "Triangular Element Data Structures," Users Applications Symposium Proceedings, The University of Alberta Computing Services, Edmonton, Alberta, Canada, 1976.

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

9. Harary, F., Graph Theory, Addison-Wesley Publishers, Reading, Massachusetts, 1969.

10. Hewitt, C., Description and Theoretical Analyses (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models on a Robot, Ph.D. Thesis, M.I.T., April 1972.

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

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

13. Robinson, Arthur and Randall Sale, Elements of Cartography, John Wiley and Sons, Inc., New York, 1969.

14. Shortclife, Edward H., Computer-Based Medical Consultations, MYCIN, Elsevier, New York, 1976.

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

16. Winograd, T., Understanding Natural Languages, Academic Press, 1972.