A Semantic Data Model for Multithematic Spatial Databases

A Dissertation submitted in partial fulfilment of the requirements for the M. Tech. (Computer Science) degree of the Indian Statistical Institute
by

Deboprotim Ghosh

under the supervision of

Amarnath Gupta

Contents

Acknowledgement

I would like to express my deep gratitude to my supervisor for his all round guidance and supervision. Further I would also like to thank Mr. Subhash Nandy and Dr. Aditya Bagchi for some very useful suggestion during the course of this dissertation. Lastly I deeply acknowledge all the staff of KBCS for lending me all possible help.

1 Introduction

Several models [1,2,3,4,5] for spatial information management have been proposed. Most of these systems offer a definition and query language which declares a spatial object in terms of a set of geometric base objects, mainly point, line and polygon. Some of them such as GeoSAL [2] offer more complex objects such as chain of lines and tessellation of polygons. Geometric declarations have some clear advantages.

- The spatial objects are directly representable in geometric data structures.
- Definite spatial access mechanisms and pre-processing techniques can be directly applied to the user defined objects.
- Since the user defined objects are geometric, computational geometric algorithms can be applied on them.

There are some disadvantages of direct geometric representation of all user objects.

1. Many spatial objects (such as a river system) that should have simple one-stroke conceptual definitions result in complicated potentially incomplete definition in terms of geometric entities.
2. Many conceptual objects can be defined not only by spatial objects but by some topological restrictions on such objects. A political map may need the inherent semantics of political system.
3. Operations semantically associated with spatial objects do not have obvious geometric counterparts; for example, roads which are polylines can be traversed, but boundaries which are also representable by (closed) polylines are not traversable. As a matter of fact, traversal has more to do with user's perception than geometry. Such restrictions are not apparent in geometric representation.

There has been object-oriented approaches involving proper inheritance and operation encapsulation. But that they are far from an adequate representation follows from these observations.

- They maintain the basic premise that the user needs to explicitly define the geometric character of the objects.
- The acknowledgment of the fact that only inheritance and encapsulation are not sufficient tools to model space. Relations other than IS_A occur in space and comes out from the mathematical structure of spatial domains. So, a semantically stronger model is necessary to abstract space.

In this dissertation, we present a more conceptual data model developed in course of our work on an Archaeological Information System. This model provides a topological abstraction of geometric information and is borne out of a necessity to disassociate the semantic representation from the internal representation. Our query language allows processing of queries at this higher level of abstraction. The inner geometric structures

are transparent and would be used only if the user's query cannot be answered at the semantic level. The complimentary use of the logical structure provides a more efficient utilization of the user's conceptual model of space. A second significant aspect of the model is that it builds on theme integration - which is akin to view integration in relational models with an important difference. A set of conceptually connected spatial objects are grouped into a structural aggregate called theme. Different themes may have different models and integration is affected through a shared name space and appropriate geometric anchors. The integration helps to answer intertheme queries in which the nodes of a query graph and the response belong to all different themes.
The organization of the dissertation is as follows. In Section 2, we deal with the formal aspects of data model. Section 3 deals with the definition, specifications and operations on themes, our major focus of the dissertation. In Section 4 our query language and its processing are described. Section 5 illustrates strategies for processing inter-theme queries. In Section 6, we conclude with a discussion on our implementation and the scope for future work.

2 Basic Concepts

The formal model that we are going to present is a many sorted algebra, where the sorts are

Aspatial Atomic Sort Σ_A = (int, float, string, bool)
Spatial Atomic Sort Σ_S = (point, polyline, region)
Attribute Constructor C = (set, tuple)

In literature, some [1] have referred to Σ_A as the Standard Domain D or the Base Representation. Attribute, according to [1] specifies a private property having a value from some domain, or derived from an constructor operation

$$C_1(C_2(...(C_n(d \in \Sigma_A \cup \Sigma_S))...)$$

where $C_i \in C$, a non-empty set of constructors and () is the currying function. So the total set of sorts is this system is given by

$$\mathbf{S} = \Sigma_A \cup \Sigma_S \cup C(\Sigma_A \cup \Sigma_S)$$

An attribute is represented by A_k^i, the ith attribute of the kth sort.
Now we move over to some definitions.

Definition 1 *A Spatial Object $s \in \mathbf{O}_s$ is a 3-tuple (sid, sst, asts) where*
sid $\in$ *SID , the space of all object identities*
sst *is an attribute A^0 called extent.*
It belongs to a specific spatial sort of the form
$A^0 \in \Sigma_S \cup C(\Sigma_S)$
asts *denotes one or more aspatial attributes from*
$\Sigma_A \cup C(\Sigma_A) \cup M(\Sigma_S \Rightarrow \Sigma_A)$ *where*
M is a measure function on Σ_S

Informally speaking, a spatial object has an sid, which uniquely identifies it, should be of one and only one spatial sort and can have a set of other aspatial attributes, some of them dependent on the spatial sort of the object and defined by a measure function M. For example, if the spatial sort of a spatial object is region, an instance of M can be the area of the polygon defined as $area : region \Rightarrow float$. In course of this work we have made the following assumptions on the basic atomic sorts.

- A region is a simple closed polygon
- A region has an area, minimum bounding rectangle (MBR), maximally enclosed rectangle (MER) and optionally depth(s).
- A polyline has a length and a number of segments.
- Every segment of a polyline has a general orientation and width.
- A point can be a member of any spatial type.

Definition 2 *A Spatial Relationship* **R** *is a 3-tuple (relname, f, invrelname) where*

relname	*denoted by n, is the name of the relationship*
f	*is the associated function*
	$f : relname \Rightarrow \mathbf{O}_s \times \mathbf{O}_s$
invrelname	*denoted by n', is the name of the inverse relationship*

It is implied that if $s_1, s_2 \in \mathbf{O}_s$, and $\exists r \in \mathbf{R}$ such that $f(r.n) = \{s_1, s_2\}$, then $f(r.n') = \{s_2, s_1\}$.
For example, let A and B be two spatial objects and let there be a relation EAST_ADJACENT. Its inverse relation is WEST_ADJACENT. Then if there exists a tuple in **R** of the form (EAST_ADJACENT, {A, B}, WEST_ADJACENT)} then there also exists a tuple (WEST_ADJACENT, {B, A}, EAST_ADJACENT)} in **R**. Next we define a designated spatial object called the *universe* to specify the domain of interest of a specific spatial database.

Definition 3 Universe *is a spatial object of the spatial sort region such that the following are satisfied.*

1. *No defined spatial object is outside the universe.*
2. *The universe is closed with respect to spatial operations, i. e., no spatial operation can generate a spatial object lying outside the universe.*

However, there is one case where a spatial object can lie outside the universe. Such spatial objects are called *referred spatial objects* which can be accessed only *by name*. No spatial access is granted to these objects and they cannot be generated by any spatial operation. An example is a city that lies at the terminal point of a road passing through the universe.

Definition 4 *Theme is a 3-tuple (theme_name, msob, theme_structure) where*

theme_name	*is the name of the theme*
msob	*is a set of member spatial objects of the theme*
theme_structure	*is a representation of the theme structure which physically denotes the topological constraints of the msob s such as rules of containment, connectivity, coverage and overlap*

Note that such a declaration of theme is different from that employed in GIS where a theme is defined on different values of an attribute. For example, if the attribute chosen is population then a ranking of space based on population (usually depicted by assigning different colors to different population zones) may constitute a theme. The theme declaration used here defines a logically interconnected set of spatial objects as a collective object that can be accessed simultaneously. We believe that this declaration is a more meaningful subdomain representation of the universe much like a view in a relational framework.

3 Formal Data Model

3.1 Data Definition Language

3.1.1 Universe

In course of data definition the user first declares the universe.

```
< Universe Declaration > ::= %BEGIN UNIVERSE
                                  UNAME :         < UName >;
                                  [TESSELLATION : < TName >;
                                  EXTENT :        < UExtent >;]
                                  <ExtName Declarations >
                              %END UNIVERSE
< UName > ::= < identifier >
< ExtName Declarations > ::= EXTERNAL <ExtNames> | ∈
< ExtNames > ::= < identifier > | < identifier >, < ExtNames >
```

The EXTERNAL declaration specifies a *referred spatial object* explained before. The reserved word TESSELLATION refers to the nature of grid (square, hexagonal and so on). It is optional and a square grid is chosen by default. EXTENT refers to the physical limits of the universe in terms of the grid. If none is mentioned, it is taken as the MBR of the universe polygon. Next we will see how to declare themes and discuss the design advantages that are obtained out of such declaration.

3.1.2 Theme

In general a large spatial database would consist of a number of themes. Each theme, being a collective object must have a name. Thus the user declaration is given as shown in Figure 1

```
< Theme Declarations > ::= < Single Theme Declaration > |
                           < Single Theme Declaration >
                           < Theme Declarations >
< Single Theme Declaration > ::= % BEGIN THEME
                                     TNAME :    < ThName >;
                                     TTOP  :    < ThTop  >;
                                     < Theme Type Declarations >
                                   % END THEME
< ThName >  ::= < identifier >
```

Figure 1: User Declaration of Theme

```
< Theme Type Declarations > ::= < Single Theme Type Declaration > |
                                < Single Theme Type Declaration >
                                < Theme Type Declarations >
< Single Theme Type Declaration > ::=
        % BEGIN THEME  TYPE
          TTYPE : < ThObType >;
          < Theme Type Declarations >
          < Theme Object Declarations >
        % END THEME OBJECT TYPE
< Theme Type Declarations > ::=  < Region Declaration > |
                                 < Network Declaration >
```

Figure 2: Theme Type Declaration

For themes, we introduce a concept of *theme type*, which essentially constrains the operations allowed on the participant objects of the theme. These constraints are both topological, as well as functional. Through topological constraints the operations under a theme are forced to produce topologically meaningful spatial objects. For example, no operation on a political theme can result in a region which is unconnected to other regions, unless there is an intervening water body. A functional constraint, on the other hand restricts the object classes allowable under a certain theme. For example, it is illegal to declare a "region boundary" in a theme that accepts only polylines and points. For our purposes, two broad theme types have been considered, *region theme* and *network theme*. The first rests on notions of algebraic topology, while the second allows network analysis operations. These are referred by the keyword TTOP in the user declaration. Figure 2 shows the user syntax for theme type declaration.

Region Themes

A theme of the region type is characterized by four topological properties. The *decomposability* property asserts whether a region object declared under this theme is atomic or decomposable. In the latter case, the name of the decomposed object must be mentioned. For example, in a political theme, a state may be decomposed into districts. In case the largest decomposable object under a theme is the universe itself, the user has to state it explicitly. Following the example of the political theme, a declaration may be like

UNIVERSE IS COUNTRY
COUNTRY DECOMPOSES INTO STATE

Decomposable regions are further classified by their properties of *coveringness* and *disjointedness*. When a larger spatial object is decomposed into k smaller spatial objects, the set of decomposed objects produced may not spatially cover the parent object. A familiar example is a mineral map of a country, where there can always be a region with no mineral production. While this situation is easily managed by using null-valued regions, the user may choose to assign a non-null spatial constraint, whereby the theme enforces that the set of decomposed objects declared must obey the cover constraint. The constraint of disjointedness asserts that no two decomposed objects of a given parent spatial object in a theme can overlap in space. There are many cases where this is true, such as a political theme; on the other hand cases such as agricultural production would not be disjoint, because a given region can have many agricultural products. The property of *connectedness* is to specify that the spatial objects produced by decomposition are connected. If they are not, one may specify any constraints applicable to them. The constraints are specified by a set of predicates that must be satisfied by the objects in that theme. For example, an island state is declared by the following syntax.

ISLAND_STATE DECOMPOSED INTO ISLAND
NOT CONNECTED ISLAND
CONSTRAINT ((CONNECTING REGION IN ISLAND_STATE) IS WATER_BODY))

An object type called WATER_BODY has to be declared in order to have a valid system constraint.

While parsing a declaration, the system discovers recursive decompositions, as well as partitions which appear as a consecutive set of disjoint covering decompositions. A partition, can however be explicitly declared, as in Figure 3. The declaration parser can also check consistency by topological rules, such as a region declared non-disjoint must be connected, and hence the latter specification may be omitted in the user's declaration. A region can be either boundary-designated or non-boundary designated. If a region is declared to be of the former type, then the boundary of the region is explicitly stored. This enables sub-boundary operation of the region. For example finding the common boundary between two adjacent regions is meaningful only if both the regions are declared to be boundary designated. This also pertains to spatial adjacency between two regions. Two regions are spatially adjacent if and only if they share a common boundary. There exists a definite relation between boundary designation and field decomposibility, in that a decomposable field implies a boundary designated region. Note that in such the declaration parser checks for consistency of the boundary designation of the region and the boundaries of the field decomposition. A region is also qualified as a simple or complex type. All our discussion about region themes till now has been confined to simple type regions only. But sometimes it is necessary to call a set of (simple) regions collectively by a single identity. For example, such a

```
< Region Declaration > ::=  < Partition Declaration > |
                            < Other Region Declaration >
< Partition Declaration > ::= PARTITIONED INTO < PName >
< PName > ::= < identifier >
< Other Region Declaration > ::= < Boundary Declaration >
                                 < Field Declaration >
< Boundary Declaration > ::= BOUNDARY DESIGNATED ; <DType> |
                             NON-BOUNDARY DESIGNATED
< DType > ::= DECOMPOSABLE | ATOMIC
< Field Declaration > ::=  % BEGIN FIELD
                               FTYPE : <DType>;
                               <Field Attributes>
                           % END FIELD
< Field Attributes > ::= <Type1> <Type2> <Type3>

< Theme Object Declarations > ::= < Region Object Declarations > |
                                  < Network Object Declarations >

< Region Object Declaration > ::=
        < Simple Region Object Declarations > |
        < Complex Region Object Declarations >

< Simple Region Object Declarations > ::=
        < Single Region Declaration > |
        < Single Region Declaration >
        < Simple Region Object Declarations >

< Region Simple Single Theme Object Declaration > ::=

                    % BEGIN SIMPLE REGION THEME OBJECT
                      ONAME : < OName >;
                      <Polygon Declarations >
                    % END SIMPLE REGION THEME OBJECT

< Polygon Declarations > ::= < Single Polygon Declaration > |
                             < Single Polygon Declaration >
                             < Polygon Declarations >
```

Figure 3: Declaration of the region theme

```
< Complex Region Object Declarations > ::=
          % BEGIN COMPLEX THEME OBJECT
            CONSTITUTED OF < Simple Region Object Names >
          % END COMPLEX THEME OBJECT
 < Simple Region Object Names > ::=
            < OName > |
            < OName >, <Simple Region Object Names>
```

Figure 4: Declaration of a Complex Theme

collection can be THE FORESTS OF NORTH INDIA. This is a complex region object constituted of a number of regions, each of which are FORESTS and located in NORTH INDIA. It may be possible that NORTH INDIA is itself a complex object (in fact it is, composed of STATES in the northern part of INDIA) and proper constraints should be generated by the declaration parser to take care of such cases. Specifically if NORTH INDIA has not been defined then a proper reference should be kept of the fact with the constraints imposed by the spatial extent of the FORESTS. The declaration of such complex regions as shown in Figure 4.

Network Themes

A network theme is characterized by a collection of nodes, edges, and intersections. A theme object in a network is a sequence of segments which are an ordered node-pair with an edge between them. For example the river GANGA is a theme object in the theme RIVERS (say). The declaration parser expects the object GANGA to contain a sequence of segments. Besides this, the network theme also supports declaration of complex object, like the one in the Region theme which constitutes of a collection of simple objects in the same theme. The edge of a segment has several attributes, like width, general orientation, navigability, interpolability , edge costs etc. There are eight possible general orientation allowed viz. North-South, South-North, East-West, West-East, NE-SW, SW-NE, NW-SE, SE-NW. The user can associate one or more costs to the edges. These are used as static evaluation potential (s.e.p.) in the heuristic function to solve optimal path/route problems. Some of the attributes like navigability dynamically changes the cost of the edges in the query execution time. The concept of paths/routes introduces a new kind of object in the network realm. A path/route in such a domain is a collection of segments of different objects in the network theme. The path/route has an identity of its own and its specific declaration syntax. It is declared as shown in the Figure 5. A network object may join another network object or several other network objects may fork from one or more segments from a network object. This is called 'join' or 'fork' of network objects. A join or a fork imposes several constraints which are to be checked by the declaration parser in course of such declarations. The first of these constraints is the appreciation of the fact that a join or a fork declaration necessitates the declaration of the other. For example the river JAMUNA joining GANGA at segment number 2 implies a 'fork' on segment number 2 of GANGA by the name of JAMUNA. So the user may chose not to explicitly specify such inverse declarations and the system

```
< Composition > ::= % BEGIN ROUTE
                      RNAME : < RName >;
                      < Route Declarations >
                    % END ROUTE
< RName > ::= < identifier >
< Route Declarations > ::= < Single Segment Declaration > |
                           < Single Segment Declaration >;
                           < Route Declarations >
< Single Segment Declaration > ::=
          EXTENDING < RExtent > ON < NNames >
< NNames > ::= < NName > | < NName >, <NNames>
< NName > ::= < identifier >
< RExtent > ::= < StartSegmentno >, < EndSegmentno >
< StartSegmentno > ::= < integer >
< EndSegmentno > ::= < integer >
```

Figure 5: The Path Declaration Language

is bound to compute any such referential constraints. Furthermore, a mechanism should be provided to deal with false intersections. For example, a bridge over a road can lead to a false intersection due to the two-dimensional limitation of the viewing surface. Such intersections should be avoided by the query processor in path-finding problems. Finally we come to the 'interpolability' property of an segment-edge. When an edge is declared to be interpolable, then the points intervening the start and the end point of the segment becomes significant, in the sense that such an interpolated intervening point can be referred as a object type. This can be illustrated by an example. Consider a national highway covering a number of important cities. Of the several (hundreds) of segments in such a road, some of them actually passes through the cities. Such segments may be declared interpolable, so that an intervening interpolated point where the city is located can be referred. The declaration syntax of a network is shown in Figures 6 and 7.

3.2 The role of Geometric Objects

Though we have mentioned in the introduction that the user query language in our model is not explicitly based on geometric objects, this does not mean that the geometric aspect of space can be ignored. The phrase in our context means that the underlying geometric representation of the objects is hidden from the user once it is declared. The access to the internal representation then becomes an exclusive privilege of the Query Processor. The data structures and algorithms for such geometric representation have not been dealt with in our work. For an excellent survey on such topics, the reader is referred to [9]. In query processing we will show how to use the underlining geometry of space.

```
< Network Declaration > ::= < Nodes >
                            < Edges >
                            < Intersections >
< Nodes > ::= % BEGIN NODES
                < Nodelist >
              % END NODES
< Nodelist > ::= < Node > |
                 < Node >, < Nodelist >
< Edges > ::= % BEGIN EDGES
                < Edgelist >
              % END EDGES
< Edgelist > ::= < Edge > |
                 < Edge > < Edgelist >
< Edge > ::= < CO > : < CO >;
             ENO    : < integer >;
             % BEGIN ATTRIB
               <Attributelist>
             % END ATTRIB
< Attriblist > ::= < attribute > |
                   < attribute > < Attributelist >
< attribute > ::= < Nav attribute >;
                  < Direction attribute >;
                  < Orientation attribute >;
                  < Width attribute >;
                  < Interpolable attribute >;
                  < Cost attribute Declaration >
< Cost Attribute Declaration > ::= % BEGIN EDGE COST
                                     < EdgecostList>
                                   % END EDGE COST
< EdgecostList > ::= < SingleEdgeCost > |
                     < SingleEdgeCost >< EdgecostList >
< EdgecostList > ::= < CostName > : < Cost >;
< CostName > ::= < identifier >
< Cost > ::= < integer >
< Intersections > ::= % BEGIN INTERSECTION
                        < Intersectionlist>
                      % END INTERSECTION    | ∈
< Intersectionlist > ::= < Intersection > |
                         < Intersection >
                         < Intersectionlist >
```

Figure 6: The Network Theme Declaration Language

```
< Intersection > ::= < Intersection Type >
                     < Intersection edges Declaration >
< Intersection Type > ::= < Junction Type > | INCIDENTAL
< Junction Type > ::= FORK | JOIN | CROSSROAD
< Intersection edges Declaration > ::= < Edge1 >, < Edge2 >
< Edge1 > ::= < integer >
< Edge2 > ::= < integer >
< Network Theme Object Declaration > ::=
     < Network Simple Theme Object Declaration > |
     < Network Complex Theme Object Declaration > |
     < Composition >
< Network Simple Theme Object Declaration > ::=
     % BEGIN SIMPLE NETWORK THEME OBJECT
       ONAME : < OName >;
       <EdgeNoList>
     % END SIMPLE NETWORK THEME OBJECT
< Network Complex Theme Object Declaration > ::=
     % BEGIN COMPLEX NETWORK THEME OBJECT
       ONAME : < OName >;
       CONSTITUTED OF <ONames>
     % END COMPLEX NETWORK THEME OBJECT
< ONames > ::= < OName > | <ONames>, <ONames>
```

Figure 7: The Network Theme Declaration Language(contd..)

3.3 Logical Models of Theme

A logical model consists of a specific set of objects and operations, designed in such a fashion that conceptual models of many applications can map onto it. In course of this work, we have identified three logical models to capture the theme semantics outlined so far. These models are hierarchical graphs, networks and interval multigraphs. They are, in fact, complementary to each other, and in the general case the user's data definition would partially map into each. Out of these, the basic structure of the network model is practically the same as the network theme outlined earlier. Properties of the other two are described in the following subsections, while operations related to them will be discussed alongwith the query language.

3.3.1 Hierarchical Graph Model

A hierarchical graph is a structure having the following properties.

1. It consists of an m-ary tree called the *p-tree* containing objects called *planes*, rooted at the designated node called *universe*.

2. A plane consists of a **connected multigraph** of *node entities*. One node entity can be connected to another by at most k directed edges. Each edge has a label $e_i, i = 1..k$. Two nodes connected by an edge with a label e_i are said to be adjacent in the direction e_i.

3. A node entity can be either a **single node**, or **a set of nodes**. Participant nodes of a set entity need not be connected. However, no participant node can be isolated from all other nodes in the plane.

4. The parent of the $(i+1)$-th plane of the hierarchical graph is one and and only one entity of the i-th plane. If there is a node entity in plane i without any children, it is *atomic* else it is *decomposable*.

5. All planes of a level of the p-tree is called a resolution level. Two node entities can also be edge connected if they belong to different planes, but to the same resolution level.

Next we specify the semantics of a hierarchical graph in terms of the underlying computational geometry.

1. To every singleton node in the leaf level of a hierarchical graph, there is a polygon in absolute space, defined in terms of the declared tessellation.

2. For each decomposition in the hierarchical graph, there is a polygon fully contained within the polygon corresponding to its parent.

3. For two adjacent nodes in the hierarchical graph, connected by an edge label e, there exists common sides between their corresponding polygon, such that the *average* orientation of the common sides is e. Here average orientation is derived from arctan $\frac{\Sigma y_i}{\Sigma x_i}$, discretized to the appropriate angular resolution level.

4. **A nonleaf node of the hierarchical graph may explicitly refer to a polygon. In this case, all the declared topological properties will be obeyed by the polygon. There is one exception to this explicit reference. An i-th level node of the p-tree explicitly refers to a polygon p^i, and all nodes of a plane π at the 2nd descendant level refer** to explicit polygons $p_0^{(}i + 2)...p_k^{)}i + 2)$. However parent_of($\pi$), belonging to the i-th level of the p-tree <u>does not</u> refer to an explicit polygon. In this case, $p^{(}i + 1)$ is called a <u>dummy node</u> and is defined to represent $\bigcup p_j^{(}i + 2)$. An illustration of this situation appears at a political theme if a country decomposes directly into districts without going through the intermediary level of states, albeit state has been defined in the theme.

3.4 Interval Multigraph Model

An 2D interval multigraph is a structure with the following properties

1. It consists of k sets of typical nodes $N^1..N^k$, where i indicates the type of node $n_j^i \in N^i$
2. No two nodes n_j^i and n_l^i of the same set are connected.
3. Nodes $n_j^i \in N^i$ and $n_z^y \in N^i$ (i $\neq$ y) may be connected by a labelled "directed edge" e, or its dual e' with the direction reversed. The number of connecting edges will not exceed an upper limit. Two connected nodes are said to be "overlapping" along the direction e, if their connecting edge has label e.
4. Isolated nodes <u>are</u> allowed to appear in the graph.

So far we have described the multigraph characteristics of our model. Next, the interval semantics would be explained in terms of the underlying computational geometry.

1. For each node in the multigraph there exists one and only one polygon in absolute space.
2. If two nodes in the multigraph are connected by an edge, then the corresponding polygons overlap in absolute space.
3. The label of the edge connecting two polygons represents either a complete containment of one in the other, or denotes the direction of the half-plane in which one polygon extends with respect to the other. Naturally, if there are k cardinal directions defined in the system, only $k/2$ need be specified as edge labels.

It is simple to establish that the semantics above does follow the properties of an interval graph in 2D space. For simplicity consider, there are only four cardinal directions, east, west, north and south. Therefore, there are two edge labels. Let us say, they are "overlapping and projecting eastwards" and "overlapping and projecting southwards", an equivalent of the raster representation of 2D space. The specification of four cardinal directions is, in fact, a restatement of the assumption that a square grid tessellation constitutes the underlying partition of absolute space. Hence, any pair of polygons that overlap and lie to the east (west) of the other, will be represented by the first edge label. Similarly, a pair of polygons overlapping and extending to the south (north) of one another will be represented by the second edge label. Since, each of these two labels constitute an interval graph in the appropriate direction, their simultaneous use, leads to an interval (multi-)graph in two dimensions.

Next, let us look at a more pragmatic method of representing the interval multigraph in two dimensions. Continuing with our assumption of four cardinal directions, it is possible to restructure our node-edge-node representation of the graph such that we first denote all the squares of the grid as distinct tuples. For each such tuple, we record all the polygons that overlap with it. The interval information can then be directly obtained by inspecting the tuples necessary for any search. As a further pragmatic modification, we observe that we could construct the alternate representation of the interval multigraph simply because *the square grid tessellation provides a* partition *of absolute space*. Hence, any *other* structure with the same property could be used in place of the explicit use of the tessellation. Therefore any level of all *themes* with the partition property can be used to represent this alternate interval representation. We call this representation as the **interval dictionary** and the theme used for this purpose as the **base theme**.

4 Query Processing

4.1 User's Query Language

The basic structure of a query expression consists of three clauses, much like the ones used in SQL; select, from, and where.

1. The select clause can be thought of as akin to the select clause in in normal SQL statement. It lists the attributes desired in the results of a query.

2. The from clause lists the theme-names and/or theme objects in theme-names to be consulted in the evaluation of the query.

3. The where clause corresponds to the selection predicate(s). It consists of predicates involving attributes of the theme objects used in the from clause.

A typical query has the from
select $A_1, A_2, ...A_n$
from $T_1, T_2, ...T_n$
where P
Here each A_i represents an attribute. The list $A_1, A_2, ...A_n$ may be replaced with a * to select all attributes of all theme objects appearing in the from clause. Each Ti is either a theme-name or a theme-object, specified directly or as a function of another theme object. The functions are dependent on the underlining theme-structure. For example, a decomposition of a non-leaf node in a hierarchical graph may either be enumerated explicitly by citing all the names, but a simpler way to specify it is in declaring the set as a descendant function of the parent i.e., if our need is to specify all the districts of WESTBENGAL, either explicit declarations of the districts can be done or else the user can simply refer to the set as STATES OF WESTBENGAL. The internal translation of such statements will be discussed later. The query structure pertaining to each of the three basic structures are shown in Figures 8, 9, 10 and 11.

4.2 Theme Structure Operations

Hierarchical Graph

The following operations are supported in the hierarchical structure :

1. ANCESTOR (sid [,lvl]) ⇒ { sid }
   Purpose : returns a set of ancestors of a spatial object. The optional level 'lvl' acts as a filter of ancestors at level 'lvl'. So,

   FATHER (sid) == ANCESTOR (sid, 1)
   GRANDFATHER (sid) == ANCESTOR (sid, 2)

2. DESCENDANTS (sid [,lvl]) ⇒ { sid }
   Purpose : returns a set of descendants of a spatial object. The optional level 'lvl' acts as a filter of descendants at level 'lvl'. So,

   CHILDREN (sid) == DESCENDANTS (sid, 1)
   GRANDCHILDREN (sid) == DESCENDANTS (sid, 2)

3. NEIGHBOR (sid [,Dir]) ⇒ { sid }
   Purpose : returns a set of neighbors of a spatial object. The optional direction 'Dir' acts as a filter of neighbors in appropriate direction, presently taken as any of 8 cardinal directions.

4. LCA ({sid}) ⇒ sid
   Purpose : returns the least common ancestor of a set of spatial objects at same or different level.

Network

The following operations are supported in the network structure :

```
< Hierarchical Query Statement > ::=
             < Set Statement > |
             < General Query Statement > |
             < Navigational Query Statement > |
             < Aggregate Query Statement >

 < General-Select-clause > ::= SELECT < attributes >
                               FROM < plane-denoter >
                               [ AT < level-denoter > ]
                               WHERE < predicate-list >
 < Set-statement > ::= SET CURRENT NODE = < select-set-clause >
 < Select-set-clause > ::= SELECT < plane-denoter >
                           FROM < level-denoter >
                           WHERE < predicate-list >
 < Navigational Query Statement > ::= SELECT < attributes >
                                      FROM < neighbor >
                                      FOLLOWING < edge-label >
                                      WHERE < predicate-list >
 < neighbor > ::= NEXT   [ <neighborno> ] NODE |
                  UNION  ( NEXT [ <neighborno> ] ) |
                  NEXT [ < neighborno > ] IN < plane-denoter > |
                  UNION ( NEXT [neighborno] IN < plane-denoter >
 < plane-denoters > ::= < plane-denoter > |
                        < plane-denoter >, < plane-denoters >
 < plane-denoter > ::=
         < plane-id > |
         ANCESTOR ( < ancestor-type > ) OF < plane-denoter > |
         DESCEDANT ( < descendant-type > ) OF < plane-denoter > |
         LCA  < plane-denoters > |
         JOIN ( plane-denoters >
 < ancestor-type > ::= < integer > | *
 < descendant-type > ::= < integer > | *
 < level-denoter > ::=
         < level-name > |
         ANCESTOR < ancestor-type > OF < level-name > |
         DESCENDANT < descendant-type >  OF < level-name >
 < Aggregate Query Statement > ::= FIND < agg-attributes >
                                   IN < select-set-clause >
                                   <  space >
 < space > ::= OVER < abs-space > | ε
```

Figure 8: User's Query Language for Hierarchical Graphs

```
< Network Query Statements > ::=
        < Node Set Statements > |
        < General Network Query statement> |
        < Network aggregate Query statement> |
        < Explorative Query statement>
< Node Set statement > ::=
             SET CURRENT_NETWORK_NODE =
             < network-select-set-clause >
< network-select-set-clause > ::= SELECT < Network_Node >
                                  WHERE < predicate-list >
< General Network Query statement > ::=
             SELECT < attribute-list >
             FROM < Network_Node >
             WHERE < predicate-list >
< Network_Node > ::= <Network_Object> |
                     < segno-clause > OF < Network_Node > |
                     FORKING FROM < Network_Node > |
                     JOINING OF < Network_Node > |
                     REACHABLE FROM < Network_Node > |
                     COMMON OF < Network_Nodes >
< Network_Nodes > ::= < Network_Node > |
                      < Network_Node >, < Network_Nodes >
< segno-clause > ::= SEGNO < Segments >
< Segments > ::= < Enumerable Segments > |
                 < Indexed Segments > | *
< Enumerable Segments > ::= < segno > |
                            < segno >, <Enumerable Segments>
< Indexed Segments > ::= <segno1>..<segno2>
< segno > ::= < integer >
< segno1> ::= < integer >
< segno2> ::= < integer >
< attribute-list > ::= < attributes > |
                       < attributes >, < attribute-list >
< attributes > ::= SP | EP | NODES | <SegmentAttributes>
< SegmentAttributes > ::= LENGTH | WIDTH | NAVIGABILITY |
                          DIRECTION | ORIENTATION | INTERPOLABILITY
```

Figure 9: User's Query Language for Network

```
< Network_Object > ::=
        <Network Theme Object> |
        <Network_Composition>
< Network aggregate query statement > ::=
        FIND < aggregate-attributes > IN
        < network-select-set-clause >
        < space >
< space > ::= OVER < abs-space > | ε
< abs-space > ::= < Extent >
< aggregate-attributes > ::= SUM | AVG | COUNT | MIN | MAX
< Explorative Query Statement > ::=
         FIND < Aggregate of Exploration >
         < Route attributes > OF ROUTE
         FROM < Network_Node > TO < Network_Node >
< Route attributes > ::= LENGTH | WIDTH
```

Figure 10: User's Query Language for Network(contd..)

```
<Interval Query Statement> ::= <Interval Set Statement> |
                               <General Query Statement>|
                               <Aggregate Query Statement>

<Interval Set Statement> ::=
                         SET CURRENT REGION_NODE =
                         <Region-select-set-clause>
<Region-select-set-clause> ::= SELECT <RObject-denoter-clause>
                               [ IN <Base-object-denoter> ]
                               WHERE <Predicate-list>
<RObject-denoter-clause> ::=
              RObject |
              OVERLAP OF <RObject-denoter-clause> |
              MINIMAL COVER OF <RObject-denoter-clause>

<Base-object-denoter> ::=
              * | Plane-id | Object |
              ANCESTOR OF <Base-object-denoter> |
              DESCENDANTS OF <Base-object-denoter> |
              LCA OF <Base-object-denoter>
```

Figure 11: User's Query Language for Interval Graphs

1. FORKING_FROM (sid [,segno]) ⇒ { sid }
   Purpose : returns a set of network spatial objects forking from a network spatial object. The optional field segno acts as a filter of Segment Number = segno.

2. JOINING_OF (sid) ⇒ { (sid,segno) }
   Purpose : returns a set of network spatial objects, segment number pairs. Normally a network spatial object joins only 1 network spatial object, but in some cases the function returns a set. For example, a confluence of more than two nodes in a network.

3. COMMON_OF ({ sid }) ⇒ (sid, { segno })
   Purpose : returns a common subsequence of a set of network spatial objects.

4. REACHABLE (sid) ⇒ { sid }
   Purpose : returns a set of spatial objects which are reachable from a spatial object.

5. SP (sid [,segno]) ⇒ CO
   Purpose : returns the starting point(co-ordinate pair) of a network spatial object of optional Segment number segno. If no segno is specified, then by default the first segment is chosen.

6. EP (sid [,segno]) ⇒ CO
   Purpose : returns the ending point(co-ordinate pair) of a network spatial object of optional Segment number segno. If no segno is specified, then by default the last segment is chosen.

7. NODES (sid [,{ segno }]) ⇒ { CO }
   Purpose : returns a set of coordinate pairs corresponding to the start and end points of the segments denoted by the set of segments segno. If no such set is specified, then by default all the segments of the spatial object is taken as the set.

8. SEGMENT_ATTR (sid [,{segno}] [,{attrlist}]) ⇒ { attributes }
   Purpose : returns a set of attributes of a spatial object of segment numbers chosen by the set segno. The attributes desired are chosen by a set attrlist.

9. LENGTH (sid [,{segno}]) ⇒ float
   Purpose : returns the length of the set of segments segno of a spatial object. If no such set is specified, all the segments are chosen by default. In that case the function returns the total of the spatial object.

10. LENGTH_COST (sid, CostType [,{segno}]) ⇒ integer
    Purpose : returns the cost-length of type CostType of the set of segments segno of a spatial object. If no such set is specified, all the segments are chosen by default. In that case the function returns the total of the spatial object.

11. AVG_WIDTH (sid, {segno}) ⇒ float
    Purpose : returns the average width of the set of segments segno. If no such set is specified, all the segments are chosen by default. In that case the function returns the average width of the spatial object.

12. LENGTH_COMPOSITION (sid [, {sid,{segno}}]) ⇒ float
    Purpose : returns the length of a composition (route) spatial object. The filtering construct can be the length of the route on a set of segment numbers of a set of network spatial objects. For example, the length of that part of a route R that stretches from Segment Number #Segno1 to #Segno2 of Road R1 and Segment Numbers #Segno3 to #Segno4 of Road R2.

13. LENGTHCOST_COMPOSITION (sid, CostType [, {sid,{segno}}]) ⇒ integer
    Purpose : analogous to the previous function, only that the cost length of a composition is returned.

14. FIND_PATH (sid1 [,segno1],sid2 [,segno2]) ⇒ { sid }
    Purpose : returns a set of compositions if a path exist between Segment Number segno1 of the source spatial object and Segment number segno2 of the destination spatial object. If any of the the Segment numbers are not specified, by default all segment numbers are taken.

Wherever possible complex functions are executed by using basic functions; for example, to find out the length of a path between a source and a destination spatial object the following algorithm can be followed :

```
Step 1. FIND_PATH (sid1 [,segno1],sid2 [,segno2] );
Step 2. LENGTH_PATH = 0
Step 3. WHILE NOT end-of-set
LENGTH_PATH = LENGTH_PATH + LENGTH_COMPOSITION (sid);
Step 4. return LENGTH_PATH
```

Interval Multigraph

1. OVERLAP_OF ({sid}) ⇒ { sid }
   Purpose : returns the spatial objects in the resolution of the base theme of the region of overlap of several spatial objects (in different theme types). Its inverse function is OVERLAP_IN.

2. MINIMAL_COVER ({sid}) ⇒ { sid }
   Purpose : returns the minimal cover of a set of spatial objects.

3. Any valid boolean combination using AND, OR, NOT expressed in the conjunctive normal form.

4.3 Query Processing Mechanism

In this section we shall deal with processing of queries in the three basic structures. We will use a PASCAL like psuedo language to present our algorithms. It is assumed that some functions are already available. Such functions are marked with a *.

Hierarchical Graph

The following are some procedures which are used in query processing

1. Procedure Name : GetPlaneDenoters
   Purpose : returns all the plane-denoters specified in the plane-denoter and level-denoter clauses in a global set B.
   Global Variables Used : a set of plane denoters B
   Algorithm :

```
Procedure Getplanedenoters( plane-denoter clause, level-denoter-clause )    BEGIN
   { The function ATOMIC_PDC tests if the plane-denoter clause evaluates to a
single plane-id. If yes it returns TRUE else it returns FALSE }
   IF *ATOMIC_PDC ( plane-denoter clause ) = TRUE THEN
      B := *SETUNION ( B, plane-denoter-clause )
      { The function SETUNION returns the union of its argument sets }
   ELSE
      BEGIN
      { divide the plane-denoter clause into two parts PRECEDENT and ANTECEDENT parts of plane-denoter clause and recursively apply the PRECEDENT function on the set collected by the ANTECEDENT }
         f := *PRECEDENT ( plane-denoter-clause, level-denoter-clause);
         g := *DESCENDANT ( plane-denoter-clause, level-denoter-clause);
         GetPlaneDenoters(g);
         B := f(B);
         { f can be of the following forms :
         *ANCESTOR returns the ancestor(s) of objects at plane denoter
         *DESCENDANT returns the descendant(s) of objects at plane denoter
         *LCA returns the least common ancestor of objects at plane denoter
         *JOIN returns the join of objects at plane denoter
         }
      END;
   END; { Procedure Getplanedenoters }
```

2. Procedure Name : GetPlaneIds
   Purpose : returns all the plane-ids specified
   in the global set B satisfying predicates
   in a global set A
   Global Vbls Used : a set of plane ids A
   Algorithm :

```
Procedure GetPlaneIds ( B, predicate-clause )
   BEGIN
      WHILE *CARDINALITY(B) ≻ 0 DO
      BEGIN
         PD := *ENTRY(B);
         { The function ENTRY returns the first element in its argument set }
         WHILE *CARDINALITY(PD) ≻ 0 DO
         { The function CARDINALITY returns the number of elements in its
argument set }
            BEGIN
               POID := ENTRY(PD); { Plane Object Id }
               IF PREDICATE(POID, predicate-clause) = TRUE THEN
                  A := *SETUNION (A, POID);
               { The function PREDICATE returns TRUE if the object satisfies
all the predicates in its argument }
               PD := *SETDIFFERENCE(PD, POID);
               { The function SETDIFFERENCE returns the difference set of
its arguments }
            END;
         B := *SETDIFFERENCE(B, PD);
      END;
   END; { Procedure GetPlaneIds }
```

3. Procedure Name : GetNeighborNodes
   PURPOSE : returns all the neighbors specified in the
   neighbor-clause and edge-label-clause of an
   object satisfying the predicates in the
   global set N.
   GLOBAL VBLS USED : a set of neighbors N
   ALGORITHM :

```
Procedure GetNeighborNodes ( neighbor-clause, edge-label-clause, Object)
   BEGIN
      IF *EXIST(UNION, neighbor-clause) = TRUE THEN
         UNION_ON = TRUE ELSE
         UNION_ON = FALSE;
      { The function EXIST returns TRUE if the word UNION occurs in the
neighbor clause }
      IF *EXITS([K], neighbor-clause) = FALSE THEN k=1;
      IF *EXIST(IN, neighbor-clause) = TRUE THEN
         BEGIN
            IN_ON = TRUE;
            Getplanedenoters(plane-denoter-clause of neighbor-clause,NULL);
            GetPlaneIds(B);
         END
         ELSE
            IN_ON = FALSE;
         Node := Object;
         Tempnode := Object;
         i = 0;
         REPEAT
            IF UNION_ON = TRUE THEN
            BEGIN
```

```
                Tempnode := *NEIGHBOR ( Node, edge-lebel-clause);
                { The function NEIGHBOR returns the immediate neighbor of
Node in direction of edge-label }                IF ( IN_ON = FALSE ) OR (
*INSET(A, Tempnode) = TRUE ) THEN
                BEGIN
                   N := *UNION ( N, Tempnode );
                   i := i + 1;
                END;
                Node := Tempnode;
             END;
          UNTIL i¡k;
          IF UNION_ON = FALSE THEN
             N := *SETUNION(N, Tempnode);
   END; { Procedure GetNeighborNodes }
```

Set Statement Processing :

```
Step 1: GetPlaneDenoters (plane-denoter-clause, level-denoter-clause)
Step 2: GetPlaneIds ( B, predicate-clause )
Step 3: IF ( *SINGLETON(A) = FALSE ) return ERROR
ELSE return *ENTRY(A);
```

General Query Statement Processing :

```
   BEGIN
   { check for single POID }
   IF ATOMIC_POID(plane-denoter-clause) = TRUE AND
   PREDICATE ( POID ) = TRUE THEN
      A := SETUNION(A, POID);
      ELSE
      BEGIN
         GetPlaneDenoters(plane-denoter-clause, level-denoter-clause);
         GetPlaneIds(B);
      END;
      *GetAttributes(A, attribute-list);
      { This function selects out the attributes of the plane-ids }
   END.
```

Navigational Query Statement Processing :

```
   BEGIN
      Getplanedenoters(plane-denoter-clause,NULL);
      GetPlaneIds(B);
      C := A;
      N := NULL;
      WHILE *CARDINALITY(C) ≻ 0 DO
      BEGIN
         GetNeighborNodes(neighbor-nodes, predicate-clause, edge-label-clause,
*ENTRY(C) );
         C := *SETDIFFERENCE(A, *ENTRY(A) );
```

```
        END;
        GetAttributes(N, attribute-clause);
    END.
```

Network :
The following are some of the procedures used in network query processing

1. Procedure Name : GetNObjects
   Purpose : returns all the qualifying Objects and their segments in a global set N.A element of N is of the form (Object, {Segment Numbers})
   Global Vbls Used : the set N.
   Algorithm :

```
Procedure GetNObjects( Network-node-clause )
    BEGIN
        IF *ATOMIC_NO( Network-node-clause ) = TRUE THEN
            N := *SETUNION ( N, Network-node-clause, All segments of Network-node-clause );
        ELSE
            BEGIN
                *GetNObjects( *ANTECEDENT(Network-node-clause) );
                N := *PRECEDENT ( N );
                { The PRECEDENT function can be one of the following
                    SEGMENT_FILTER(NObject,{Segment Numbers})
                    selects only the specified segments of the NObect
                    FORKING_FROM (NObject, {Segment Numbers} )
                    returns the objects which fork from the specified Segment Numbers of the argument Network Object
                    JOINING_OF(NObject)
                    returns the objects to which the argument Network Object joins
                    REACHABLE_FROM(NObject)
                    returns the objects to which are reachable from the argument Network Object
                    COMMON_OF( NObjects )
                    returns the common subsequence of a set of Network Objects.
                }           END;
        END; { Procedure GetNObjects }
```

Node Set Statement Processing :

```
    Step 1: GetNObjects ( Network-node-clause );
    Step 2: IF ( *SINGLETON(N) = FALSE ) THEN return ERROR
        ELSE return *ENTRY(N);
```

General Network Query Statement Processing :

```
    BEGIN
        GetNObjects( Network-node-clause );
        *GetAttributes (N, attribute-clause);
    END.
```

Interval Multigraph :
The following procedures are used in the processing of interval multigraph queries.

1. Procedure Name : GetBaseObjects
   Purpose : returns the base objects specified in the argument clause in a global set B
   Global Vbls Used: a global set of base objects B
   Algorithm:

```
Procedure GetBaseObjects( base-object-denoter-clause )
    BEGIN
    IF *ATOMIC(base-object-denoter-clause) = TRUE THEN
        B := *SETUNION(B, base-object-denoter-clause);
        ELSE
            BEGIN
            { Divide and recursively apply the PRECEDENT
            and ANTECEDENT functions }
                *ANTECEDENT(base-object-denoter-clause);
                B := *PRECEDENT(B);
            END;
    END; {Procedure GetBaseObjects }
```

2. Procedure Name : GetRegionObjects
   Purpose : returns the region specified in the argument clause in a global set R
   Global Vbls Used: a global set of RObjects R
   Algorithm:

```
Procedure GetRegionObjects(RObject-denoter-clause)
    BEGIN
        IF *ATOMIC(RObject-denoter-clause) = TRUE THEN
            R := *SETUNION(R, RObject-denoter-clause);
            ELSE
            BEGIN
            { Divide and recursively apply the PRECEDENT and
            ANTECEDENT functions }
                *ANTECEDENT(RObject-denoter-clause);
                R := *PRECEDENT(B);
            END;
    END; {Procedure GetRegionObjects }
```

3. Procedure Name : GetRegionObjectsInBase
   Purpose : returns the region specified in R which lies in the set of base objects B

```
in a global set R
Global Vbls Used: a global set of RObjects R
Algorithm:
Procedure GetRegionObjectsInBase(RObject-denoter-clause, R, B)
   BEGIN
      R' := *OVERLAPPING(B);
      { The function OVERLAP_IN returns the region objects
      which falls and/or overlaps in the region of B }
      R := *SETINTERSECTION(R, R');
   END; {Procedure GetRegionObjectInBase}
```

4.4 Query Formulation in the basic theme structures

In this section, we show how a query can be written in the query language pertaining to a theme structure.

Hierarchical Graph

The queries can be confined to objects in the same plane (Example 1 and 2) or can involve objects across planes (Example 3) User Query 1:

Let us consider the simple query.

FIND OUT ALL THE STATES OF INDIA.

This can be written in the query language as : SELECT NAME FROM STATES OF INDIA

A variation of this query can be

FIND OUT ALL THE ATTRIBUTES FROM ALL STATES OF INDIA.

This can be written as :

SELECT * FROM STATES OF INDIA

User Query 2:

This is a example of a query where attributes are desired from more than one object.

FIND OUT THE POPULATION OF DISTRICTS OF BIHAR AND WEST BENGAL

This can be written as :

SELECT POPULATION FROM DISTRICTS OF BIHAR, WESTBENGAL

User Query 3:

Now for a more complex query involving navigation between planes.

FIND OUT THE SITES IN THE DISTRICTS OF THE STATES ADJACENT TO WEST BENGAL

This can be written as :

SET CURRENT HIERARCHY NODE = "WESTBENGAL"

SELECT NAME FROM SITES OF NEIGHBOR *

Explanation : In order to navigate between planes one node in the graph has to be made the current node. It is from this node any adjacency is calculated. Since we are interested to get all neighbors of WESTBENGAL, we have specified NEIGHBOR *. Since an adjacency relation preserves planes (i.e., the answers to the sub-clause gives objects in the same plane), the neighbors of WESTBENGAL also belongs to the same level as WESTBENGAL. Since we are interested in sites of the states we have specified

SITES OF NEIGHBOR.
Now consider that a further condition is imposed in the above query.
FIND OUT THE SITES IN THE DISTRICTS OF THE STATES ADJACENT TO WEST BENGAL AND THE AREA OF THE DESTRUCTS IS MORE THAN 100 UNITS.
The query is modified as :
SET CURRENT HIERARCHY NODE = "WESTBENGAL"
SELECT NAME FROM SITES OF NEIGHBOR *
WHERE AREA(DISTRICTS) > 100
Note that the DISTRICTS will correspond to that of the current node pointer i.e., WESTBENGAL.
If any other attribute of the resultant objects are wanted, say the artifact distribution then the query becomes :
SET CURRENT HIERARCHY NODE = "WESTBENGAL"
SELECT NAME, ARTIFACT_DIST FROM SITES OF NEIGHBOR *
Network
User Query 1
REPORT ALL ATTRIBUTES FROM ALL SEGMENTS OF GANGA
This can be written as :
SELECT * FROM SEGNO * OF GANGA
A particular attribute can be chosen by
SELECT WIDTH FROM SEGNO 2..5 OF GANGA
gives the width of the segments 2 through 5.
User Query 2
Consider the query
FIND ALL/SOME ATTRIBUTES OF THE SEGMENT NUMBER(S) FROM A RIVER

WHICH FORKS FROM SEGMENT NUMBER 3 OF A RIVER TO WHICH THE RIVER JAMUNA JOINS.
This can be written as :
SELECT * FROM SEGNO 3..7 OF FORKING FROM SEGNO 3 OF JOINING OF JAMUNA
In case more than one fork exists, all the network objects are chosen.
User Query 3
Consider the query :
FIND THE TOTAL LENGTH OF PATH FROM SEGMENT NUMBER 5 OF GANGA TO SEGMENT NUMBER 2 OF DAMODAR
This can be written as :
SELECT LENGTH OF PATH FROM SEGNO 5 OF GANGA TO SEGNO 2 OF DAMODAR
In general any form of specification of a Network Object is possible. For example,
SELECT LENGTH OF PATH FROM SEGNO 5 OF FORKING FROM SEGNO 2 OF GANGA TO SEGNO 2 OF DAMODAR
finds the length of the path from segment number 5 of the river which forks from Seg-

ment Number 2 of GANGA to Segment Number 2 of DAMODAR. In case more than one fork exists, all the path attributes are reported.
Interval Multigraph
User Query 1
FIND ALL RICE PRODUCING REGIONS IN THE BANKURA DISTRICT OF WEST-BENGAL
This can be written as :
SELECT NAME FROM RICE-PRODUCING IN DESCENDANTS(1) OF WESTBENGAL WHERE DESCENDANTS(1) = "BANKURA"
A variation of this query is :
FIND ALL DISTRICTS IN WESTBENGAL WHICH BOTH PRODUCES RICE AND WHEAT
This can be written as :
SELECT NAME FROM OVERLAP OF RICE-PRODUCING, WHEAT-PRODUCING IN WESTBENGAL
User Query 2
Consider the query :
FIND ALL DISTRICTS WHICH PRODUCES RICE IN WESTBENGAL
This can be written as :
SELECT NAME FROM OVERLAP IN RICE-PRODUCING, WHEAT-PRODUCING WHERE NAME IN DESCENDANTS(1) OF WESTBENGAL.

4.5 Query Evaluation

This section focuses on the actual query evaluation. Three representative queries are selected, one from each theme structure. The queries chosen are fairly complex to demonstrate the various aspects of query processing.
Hierarchical Graph
Consider this query :
FIND OUT THE SITES IN THE DISTRICTS OF THE STATES ADJACENT TO WEST BENGAL
The steps of Query processing are : Step 1: GetNeighborNodes("WESTBENGAL");
returns : BIHAR, ORISSA, ASSAM (say)
Step 2: Descendants(2) returns sites of BIHAR, ASSAM and ORISSA.
Network
Consider this query :
FIND ALL/SOME ATTRIBUTES OF THE SEGMENT NUMBER(S) FROM A RIVER WHICH FORKS FROM SEGMENT NUMBER 3 OF A RIVER TO WHICH THE RIVER JAMUNA JOINS.
The steps of processing involves the following :
Step 1: GetNObjects(JOINING OF JAMUNA);
returns GANGA (say)
Step 2: GetNObjects(FORKING FROM SEGNO 3 OF GANGA)
returns BHAGIRATHI (say)

Step 3: GetAttributes(SEGNO 3..7 OF BHAGIRATHI)
Interval graph
Consider this query :
FIND ALL DISTRICTS IN WESTBENGAL WHICH BOTH PRODUCES RICE AND WHEAT IN THE DISTRICTS
The steps involved in the processing are :
Step 1: For each name in OVERLAP_OF(WHEAT-PRODUCING, WHEAT-PRODUCING) do
Step 2: if ANCESTOR(name, 1) = WESTBENGAL THEN report name

5 Intertheme Query

5.1 Navigation as a basic means of intertheme query

Note that a significant feature of our model is to offer a mechanism for navigation, not only between spatially related entities of the same theme, but between entities of different themes. Hence spatial adjacency and representation of inter- theme overlaps are vital to our system. In order to achieve this objective, a concept of absolute space has been proposed as a means of communication between 2 themes. This absolute space used in our system is a square grid tessellation.

5.2 Intertheme Operations

Hierarchical-Network

1. NNODES_FLOWING_THROUGH ({sid}[,[Type]) ⇒ ({sid})
   Purpose : returns the set of network objects, optionally of type Type flowing through the set of objects(nodes) in the hierarchical graph. For example, NNODES_FLOWING_THROUGH("WESTBENGAL",RIVERS) gives all the rivers flowing through WESTBENGAL. Its inverse function is NNODES_FLOWS_THROUGH.
2. PATH_FLOWS_THROUGH (sid[,{segno}][,true constraints][,false constraints]) ⇒ ({sid})
   Purpose : this is analogous to the function NNODES_FLOWS_THROUGH.

Network-Interval

1. REGION_IN (sid , {segno}) ⇒ {sid}
   Purpose : returns the set of regions through which a set of segment numbers of a network object passes. A * in the segment number set means all the segment numbers. By default also, all the segment numbers are chosen.
2. NNODES_IN (sid) ⇒ ({sid,{segno}})
   Purpose : returns the set of network nodes and the segments through which a

network object passes.

5.3 Grid

The tessellation of the absolute space provides a means of navigating between themes. The tessellation is user-configurable and can be rectangular or hexagonal. However in the present implementation we have resorted to the rectangular grid. So, the universe has been divided into a rectangular grid much like our familiar cartesian co-ordinate system. Some points in the grid (from now on referred to as 'grid-points') are chosen to serve as anchors by which navigation from one object to another, either in the same or different theme is done. The choice of grid-points is done like :

1. The corners of all the polygonal regions are grid-points
2. The start and end points of all segments of polylines are grid-points
3. All point type spatial sorts are grid-points
4. The centroid of a polygonal region is a grid-point. This is called the representative grid-point of the polygon. All references to this polygon in absolute space will be based from this point.
5. The points of polygon-polygon intersection and polygon-polyline intersection are grid-points.

Note that the prospective grid-points are determined during insertion and not during computation. The efficiency in the two cases is an optimization issue and have not been dealt with.

5.4 Grid Operations

1. CONSTRUCT_SEGMENT_RECTANGLE(sid, segno, distance)
   Purpose : constructs a rectangular region with the segno of a network object as the halfway dividing line parallel to the segment.
2. ENUMERATE_RECTANGLE(rectangular region)
   Purpose : reports all the grid-points in the rectangular region
3. ENUMERATE_CIRCLE(circular-region)
   Purpose : reports all grid-points in a circular region
4. ENUMERATE_CONICAL_REGION(conic-region)
   Purpose : reports all grid-points in a conical region

5.5 Anchors

The processing of inter-theme queries involves dealing with spatial objects coming from more than one theme i.e., such queries crosses theme boundaries. This necessitates a means of navigation between themes. Such navigation is realized through constructs known as anchors. In our system, we have conceived of the following anchors :

1. Base Theme Anchor (BSA)
2. Geometric Anchors
    - Minimum Bounding Rectangle Anchor (MBRA)
    - Maximally Enclosed Rectangle Anchor (MERA)
    - Grid Anchor (GA)

The choice of an anchor will depend on the nature of the query and determined by the Query Processor. The general guideline is that as far as possible queries are answered in the semantic level and should navigation be necessary BSA is preferred if the resolution is within tolerable limits. More precise answers are offered through Geometric anchors but with the added overhead of computation. A query region reference is resolved in the following manner : If the query region is resolvable by a BSA then we are done. Otherwise we adopt the following strategy-

1. If the query region is outside the MBR of a polygon, then the polygon is ignored (as it do not fall in the query region)
2. If the query region is contained within the MER of a polygon, then the polygon is selected as an area-of-interest.
3. If the query region falls outside the MER but within the MBR, then the query region falls the rectangular ring between the MBR and MER.

So, by the above method we can substantially reduce our search area and ignore some regions beyond our area-of-interest. This is significant when we consider the fact that polygon manipulation in absolute space is costly.

5.6 Intertheme Query formulation

The following examples demonstrate how to formulate intertheme queries The examples have been grouped as involving the following theme combinations :

1. Hierarchical-Network
2. Hierarchical-Interval
3. Network-Interval
4. Hierarchical-Network-Interval

Hierarchical-Network
User Query 1
FIND ALL RIVERS FLOWING THROUGH WEST BENGAL
This can be written as :
SELECT NAME FROM SEGNO * OF RIVER WHERE RIVER FLOWING THROUGH WESTBENGAL
User Query 2
FIND ALL SITES WITHIN A DISTANCE OF 10 MILES FROM OF THE PORTION OF GANGA THAT PASSES THROUGH UTTAR PRADESH
This can be written as :
SELECT NAME FROM DESCENDANTS(2) OF UTTARPRADESH WHERE NAME WITHIN 10 OF SEGNO * OF GANGA FLOWING THROUGH UTTARPRADESH
Hierarchical-Interval
User Query 1
FIND ALL SITES IN THE HILLY REGION OF WEST BENGAL
This can be written as :
SELECT NAME FROM DESCENDANTS(1) OF OVERLAP OF HILLY-REGION IN WESTBENGAL
Interval-Network
FIND A PATH BETWEEN SEGMENT NUMBER 3 OF GANGA TO SEGMENT NUMBER 2 OF MATLA AVOIDING ANY FOREST
This can be written as :
FIND PATH FROM SEGNO 3 OF GANGA TO SEGNO 2 OF MATLA WHERE PATH NOT FLOWING THROUGH FOREST-REGION
Hierarchical-Network-Interval
User Query 1
FIND ALL SITES IN THOSE DISTRICTS IN WHICH THOSE FORESTS LIE THROUGH WHICH THE RIVER BRAHMAPUTRA FLOWS
This can be written as :
SELECT NAME FROM DESCENDANTS(1) OF OVERLAP OF FOREST-REGION WHERE BRAHMAPUTRA FLOWING THROUGH FOREST-REGION

5.7 Query Processing of Intertheme queries

5.7.1 Local message passing and intertheme objects

An intertheme object is a object generated out of intertheme operation(s) on one or more theme objects. The process of such intertheme object generation can be abstracted to be a "local message passing" mechanism. Based on the query at hand, the operand theme objects send out messages and <u>listens</u> for responses from other theme objects. As far as possible, an intertheme query is broken into a number of sub-queries each of which can be evaluated by intra-theme operations. Proper intertheme operations are then applied to these partial results(the answer of the sub-queries). An example of the "local message passing" process is finding out all theme objects within a particular distance in a specific direction of an operand theme object. Such a query is answered by constructing the region of interest with respect to the operand theme object, sending messages from the operand theme object in the region of interest and selecting theme objects which respond to this message.

5.7.2 Examples

Consider Query 1 from Hierarchical-Network.

Step 1: Get a River from the list of Rivers. If none left stop

Step 2: If WESTBENGAL in ANCESTOR(nnode_flows_through(River,*,*,*),1)
then report River

Step 3: Goto Step 1

Consider Query number 2

FIND ALL SITES WITHIN A DISTANCE OF 10 MILES FROM OF THE PORTION OF GANGA THAT PASSES THROUGH UTTAR PRADESH

This is an example of a query which cannot be answered in the semantic level and we have to resort to the underlining absolute space. The steps involved in the processing of this query are the following :

```
Step 1: If UTTARPRADESH not in nnode_flows_thru("GANGA",*,*,*)
then stop
Step 2: For each segment of segment number Segno
reported by flows_thru do
      a. ConstructSegmentRectangle ("GANGA",Segno,10)
      b. For each grid-point in the Rectangle do
         b1. if SITE in GridTag(grid-point) then report grid-name.
```

Let us consider the Hierarchical-Interval query.

FIND ALL SITES IN THE HILLY REGION OF WEST BENGAL

The steps for the processing of the query are the following :

Step 1: Find the districts from OVERLAP OF HILLY-REGION

Step 2: If a ANCESTOR(district, 1) = WESTBENGAL then
report DESCENDANTS(district, 1)

Let us now consider the Network-Interval query.

FIND A PATH BETWEEN SEGMENT NUMBER 3 OF GANGA TO SEGMENT NUMBER 2 OF MATLA AVOIDING ANY FOREST

The steps involved in the processing are the following :

Step 1: Get a path between source and destination by
FIND_PATH(GANGA, 3, MATLA, 2). If no path exits stop

Step 2: If path_flows_thru(path,*,*,FOREST_REGION)=TRUE then
report path

Step 3: Goto step 1

Finally we consider the Hierarchical-Network-Interval query.

FIND ALL SITES IN THOSE DISTRICTS IN WHICH THOSE FORESTS LIE THROUGH WHICH THE RIVER BRAHMAPUTRA FLOWS

This can be answered by the following steps :

Step 1: For each district reported in nnode_flows_thru("BRAHMAPUTRA",*,FOREST-

REGION,*)
report DESCENDANTS(district, 1)

6 Conclusion

To sum up, we have achieved a semantic disassociation between user's perception of space and its internal representation. This is achieved by a semantically stronger data modelling. A prototype system has been built based on the concepts discussed in the dissertation. The prototype system is configured into four subsystems or themes. They are as follows :

1. Political System with Topology Region
2. Physiographic System with Topology Region
3. Hydrographic System with Topology Network
4. Transportation System with Topology Network

The UNIVERSE is declared to be INDIA, the Base theme as POLITICAL with the **resolution** as the districts of a state. As we have said before, this declaration enforces a complete partition of INDIA into a set of districts. The file structure is organized as a collection of B-trees. The B-trees used in the system are the following

1. Political Country Btree - indexes all the political countries
2. Political State Btree - indexes all the political state
3. Political District Btree - indexes all the political districts
4. Political Site/City Btree - indexes all the political sites and cities
5. Physiographic Forest Btree - indexes all the forests
6. Physiographic Hilly Btree - indexes all the hilly regions
7. Physiographic Plain Btree - indexes all the plain regions
8. Physiographic Marshy Btree - indexes all the marshy regions
9. Physiographic Plateau Btree - indexes all the plateau regions
10. Physiographic Mountain Btree - indexes all the mountaineous regions
11. Hydrographic Lake Btree - indexes all the lakes
12. Hydrographic Sea Btree - indexes all the seas
13. Hydrographic River Btree - indexes all the rivers

14. Transportation Road Btree - indexes all the roads

In addition there is a Header Btree which maintains a roster of all the objects defined in a system. Every object has a 2 level entry into the file structure :

1. One in the Header Btree
2. One in the Btree of its proper type and theme.

For example, the river GANGA has its entries into the Header Btree and Hydrographic River Btree. All the defined objects of the system are kept in a dictionary called the Btree Dictionary. For a particular object, the Header Btree node keeps the offset of the object into this Dictionary. The dictionary file keeps for every object the following informations in addition to others :

1. offset into the data file for detailed information about the object
2. root of the Btree of the children of this object if the object happens to be in a theme having a hierarchical structure.

The access to any object is done in the following manner : The Header Btree is searched to obtain a reference to the object. The Btree Dictionary file is searched to get informations about the object. Many aspects of the problem has not been dealt with. While most of them concerns with query processing, others concern efficient data structure designs and the third involves efficient algorithms design. Let us first consider the issues in query optimization [10]. A particular query can be answered in a number of ways, some in the semantic level and the others using the geometric data structure. It depends on the nature of query to decide what mechanism to use. This Query Plan, a typical part of the Query Optimizer has not been considered. The algorithm and data structure design are far from optimized. This is particularly true in the realm of the computational geometry part of the work. Further, issues in spatial reasoning has not been adequately addressed. For a good account of topological and spatial reasoning, the reader is referred to [6,7,8]. Nevertheless, we believe that this work has addressed the major aspects of the functional specifications and operations of spatial structures.

References

[1] A. Gupta et al., "An Extended Object Oriented Data Model for Large Image Bases", *Proceedings of the SSD91* LNCS 525, pp. 45-61, 1991.

[2] P. Svensson et al., "Geo-SAL : A Query Language for Spatial Data Analysis", *Proceedings of the SSD91* LNCS 525, pp. 119-140, 1991.

[3] M.Scholl, "Thematic Map Modelling", *Proceedings of the SSD89* LNCS 409,pp. 45-61, 1989.

[4] P.van Oosterom et al., "An Object Oriented Approach to the Design of Geographical Information Systems", *Proceedings of the SSD89* LNCS 409,pp. 45-61, 1989.

[5] M.J Egenhofer et al., "A Topological Data Model for Spatial Databases", *Proceedings of the SSD89* LNCS 409, pp. 45-61, 1989.

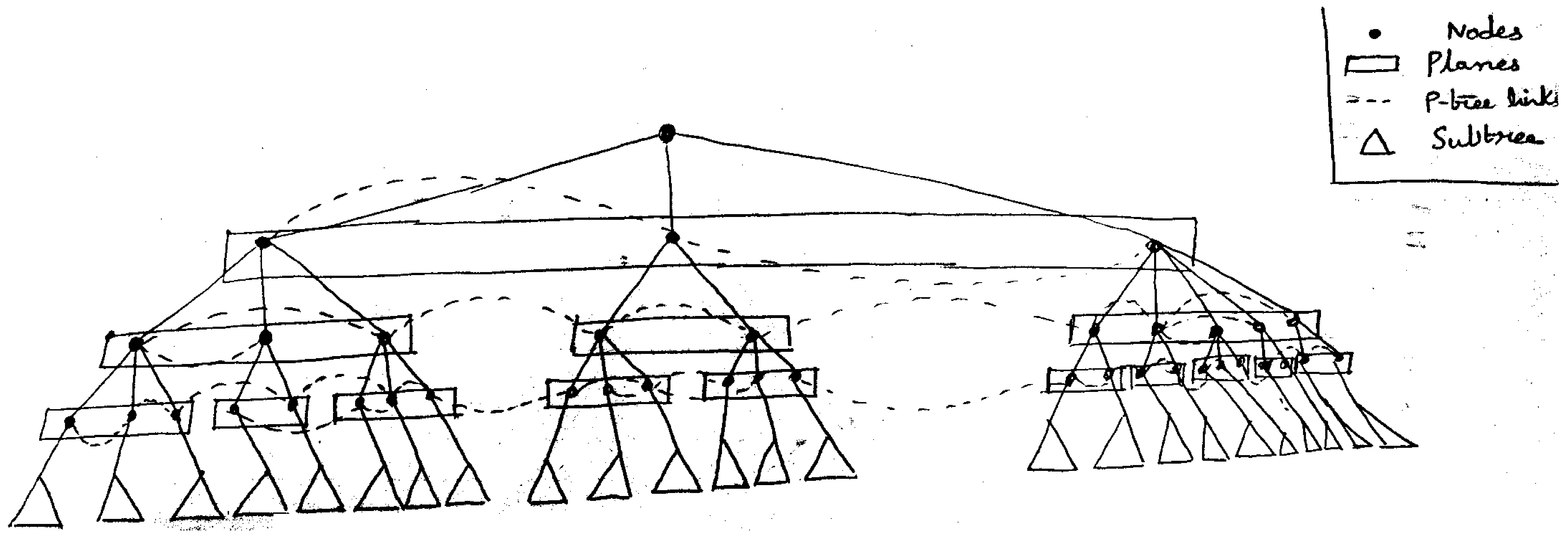
[6] M.J. Egenhofer, "Reasoning about Binary Topological Reasoning", *Proceedings of the SSD91* LNCS 525, pp. 143-160, 1991.

[7] S.Dutta, "A Representational Framework for Approximate Spatial and Temporal Reasoning", *Proceedings of the SSD91* LNCS 525, pp. 161-182, 1991.

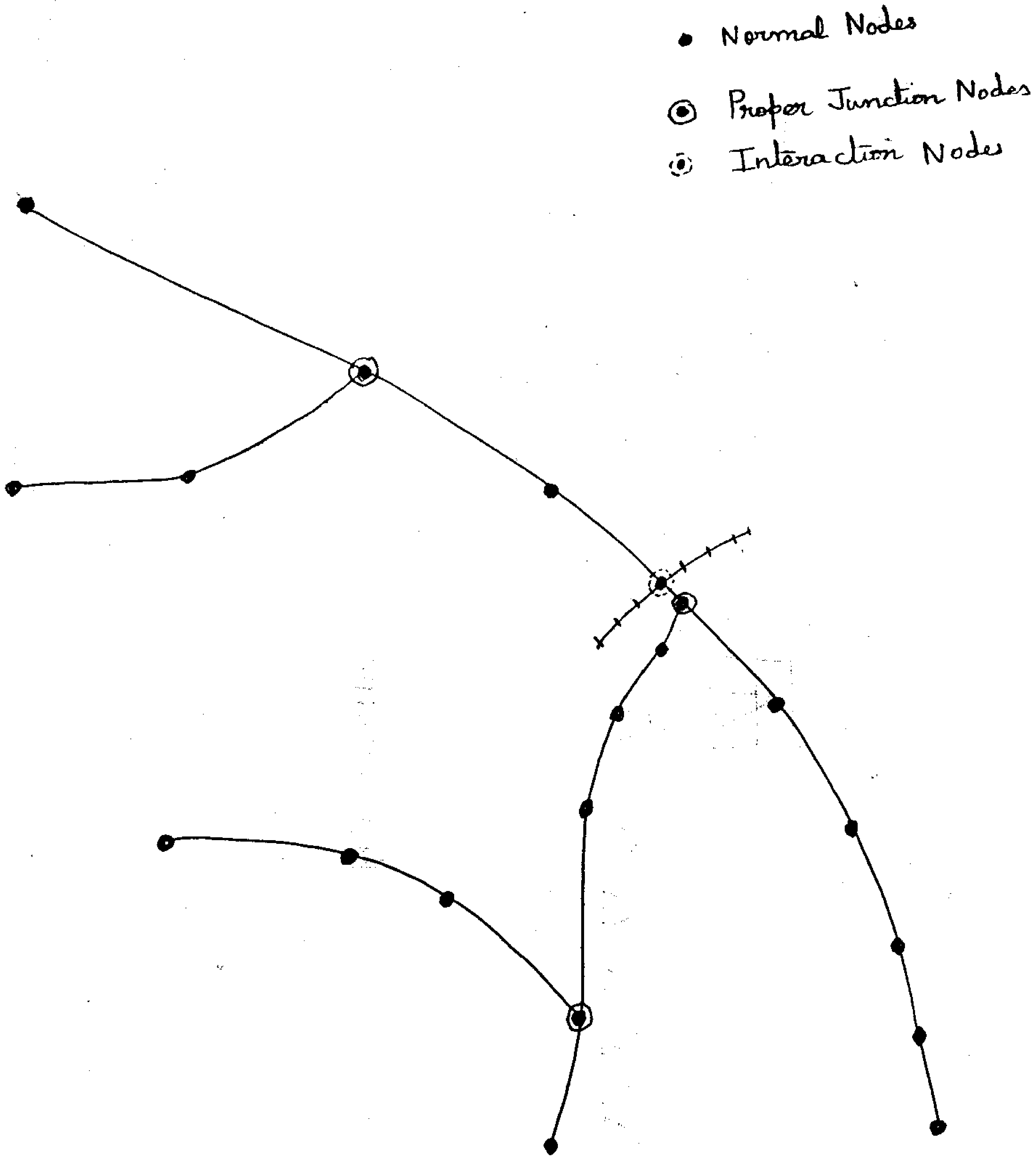
[8] S.Dutta, "Qualitative Spatial Reasoning : A Semi-Quantitative Approach using Fuzzy Logic", *Proceedings of the SSD89* LNCS 409, pp. 345-364, 1989.

[9] D.T. Lee, F.P. Preparta, "Computational Geometry - A Survey", *IEEE Transactions On Computers*, Vol. 33, No. 12, pp. 1072-1101, 1984.

[10] L. Becker and R.H. Guting, "Rule Based Query Optimization in an Extensible Geometric Database System", *ACM Transactions On Database Systems*, Vol. 17, No. 2, pp. 247-303, 1992.



HIERARCHICAL GRAPH STRUCTURE



NETWORK STRUCTURE