GéoGraph: A Topological Storage Model for Extensible GIS

K. Bennis (1), B. David (2), I. Morize-Quilio (1), J.M. Thévenin (3), Y. Viémont (1)

(1) MASI, Université Paris VI
45, Avenue des Etats Unis, 78000 Versailles. France
e-mail: viemont@sabre.ibp.fr

(2) IGN-France
2, Avenue Pasteur, BP 68, 94160 Saint Mandé.France
e-mail: david@ign.uucp

(3) INRIA-Rocquencourt
BP 105, 78153 Le Chesnay.France
e-mail: theven@madonna.inria.fr

Abstract

Topological data structures are useful for reducing the cost of geometrical operations within a Geographic Information System (GIS). Unfortunately, manipulating such data structures can be quite complex -- especially when supporting multiple, overlapping geographical maps. The GéoGraph storage model proposed in this paper solves this problem. It is implemented as a toolbox, and is used as a low-level system layer for support of a GIS. The GéoGraph storage model is based on a graph with the corresponding basic traversal primitives, and can be integrated within an extensible relational DBMS so that important spatial operations can be directly executed by graph traversals. Furthermore, the graph is decomposable so that only the useful subset of the database can be loaded from disk without format conversion.

1. Introduction

Geographic information systems (GIS) require storage and manipulation of both semantic and spatial data. Whereas conventional DBMS data models (e.g., the relational model) are well suited to representing and manipulating semantic data, queries concerned with spatial data imply the use of geometric operations not directly supported by these data models. Furthermore, because the processing performance of geometric operations is strongly influenced by data representation, systems supporting spatial data benefit greatly from a data model specially tailored for efficient support of these operations.

There are several ways to implement spatial data. A simple solution is to store each spatial object as a coordinate list. Although coordinate lists reflecting the position of objects are sufficient to perform geometric operations, inter-object spatial relationships that are obvious when seen on a map are complex and costly to capture when using this representation. To reduce the number of inter-object comparisons required by this

approach, it is possible to use spatial indices on coordinate lists. As a more fundamental attack on this problem, however, it is possible to enrich the geometrical description of objects with topological information. This information explicitly materializes the connectivity and contiguity relationships between spatial objects, and can be represented as a graph that provides direct and efficient support for adjacency operations.

Topological information has been used in several geographical information systems [Morehouse85, Herring87, Kinnear87, Spooner90]. The topological information is usually stored in a graph using an appropriate internal representation [White79, Peuquet84]. These representations can be complex to maintain and may require expensive verification of integrity constraints. In many cases, topological graphs have been used to store relationships between spatial objects of one geographical map at a time. For instance the topology of a road map is stored separatly from the topology of a land cover map. As a consequence operations involving several maps require the fusion of several graphs, which can be a complex and expensive procedure [Schaller87].

In this paper we present GéoGraph, a storage model for topological information that supports efficient operations involving several layers of maps. This model stores the topology of an internal map corresponding to the overlay of several geographical maps. Hence spatial objects of one geographical map are decomposed into collections of elementary spatial objects and the internal map materializes the relationships between the spatial objects. This principle has alrady been used in some GIS like TIGER [Meixler85] and TIGRIS [Herring90]. We focuse on a clean integration of topological information in a DBMS so that semantic and spatial data can be manipulated in a uniform way.

GéoGraph is implemented using a toolbox approach and constitutes a low-level system layer that can support general purpose GIS. This storage model is based on the topological map theory that guarantees coherent updates of topological information, and provides a minimal set of operations for navigation through a topology [Dufourd88]. In this paper, we demonstrate a straightforward integration with an extensible relational DBMS supporting a GIS. The integration is based on a single graph that incorporates both relational data and spatial data in order to precompute all operations, and is currently being investigated in the framework of the GéoTropics system [Bennis90], an extensible DBMS based on extensions of SQL.

The paper is organized as follows. Section 2 reviews the basic concepts of topological maps used in GéoGraph. Section 3 introduces the concept of map overlay and then provides a formal definition of GéoGraph in terms of a graph structure and primitive operations on the graph. Section 4 illustrates the use of the GéoGraph toolbox for implementation of geographical operators of the GéoTropics system. Section 5 argues for a specific implementation of GéoGraph when it is incorporated into a relational DBMS. Section 6 gives our conclusions.

2. Topological map concepts

Topological maps have been defined as an extension of combinatorial maps [Edmond60, Cori81]. They provide the necessary support for expressing relationships between spatial objects in a plan [Dufourd88, Dufourd89]. This section gives intuitive definitions of the basic concepts of topological maps in order to highlight their contribution for topological information management in cartography. To clarify the discussion we distinguish non fully-connected topological maps from topological maps.

2.1. Non fully-connected topological maps

A non fully-connected topological map defines a graph similar to the well known topological graph [White79, Peuquet84], and uses two basic functions -- α and σ (see figure 1). In general this graph is not fully-connected. Edges of this graph represent lines which correspond to the location of linear features (e.g., roads), or boundaries of surfacic features that we call faces. Nodes of this graph represent intersections of edges. Conceptually, each edge is decomposed into two blades labeled by integers (e.g., b and -b) corresponding to the two possible orientations of the edge. Function α applied to a blade label gives the label corresponding to the opposite blade orientation. Function σ is a permutation which orders blades around their end-node in a clockwise fashion. Thus, σ applied to blade b gives the next blade ending at the end-node of b. Any traversal of the graph can be expressed by a combination of the functions α and σ. For instance, the boundaries of one face can be traversed turning counterclockwise applying a loop on function $\varphi = \alpha \circ \sigma$ to any blade of that face. In the graph, a geometry is associated to each edge. For this purpose, a last function γ is defined such that γ applied to a blade gives a coordinate list corresponding to the geometry of the associated edge. A non fully-connected topological map can be defined more formally as follows:

Definition: non fully-connected topological map

A non fully-connected topological map is defined as a quadruplet $(B, \alpha, \sigma, \gamma)$ where B is a finite set of blades; $\alpha : B \rightarrow B$ is a permutation such that $\forall b \in B\ \alpha(b) = -b$; $\sigma : B \rightarrow B$ is a permutation such that $\forall b \in B\ \sigma(b)$ is the next blade ending at the end node of b, turning clockwise; and γ is a function which applied to any blade returns the geometry of the associate edge.

A permutation φ can be deduced from α and σ such that $\varphi = \alpha \circ \sigma$.

According to this definition, the cycles $(b,\alpha(b))$ of α define edges, the cycles $(b,\sigma(b),\sigma\circ\sigma(b),\ldots,\sigma^{-1}(b))$ define nodes and the cycles $(b,\varphi(b),\varphi\circ\varphi(b),\ldots,\varphi^{-1}(b))$ of φ define faces. It is important to note that any blade defines one and only one edge, node, and face using respectively a cycle of α, σ or φ. The reverse is not true. From these properties we can deduce that, given a face defined by blade b and function φ, the adjacent area along blade b is defined by blade $\alpha(b)$ and function φ.

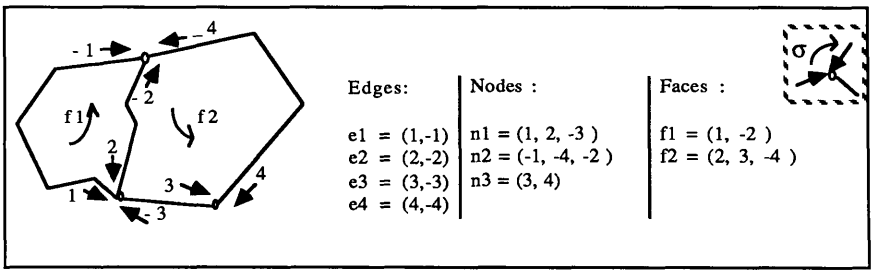

Figure 1: An example of map

Compared to the usual adjacency graphs, non fully-connected topological maps have the following advantages for geographic applications: (i) faces are easily enumerated; (ii) the planarity of a map can be checked very efficiently using α, σ, φ and γ [Dufourd89].

2.2. Topological maps

Applying the definition of a non fully-connected topological map, the graph resulting from a geographical map is in general not fully-connected, and useful relationships between different fully-connected components of this graph are not captured. In order to avoid this drawback, a topological map is defined below as an extension of a non fully-connected topological map, where a partial order among the fully-connected components is defined based on *geometric inclusion*. In support of this definition, we note that given any pair of fully-connected components (c1, c2), the following property is true: either c1 and c2 locations are disjoint (side by side), or one of c1 or c2 is fully included into one face of the other component. Otherwise (c1, c2) would form a single fully-connected component. In the second case, one component, say c1, constitutes a hole in a face of the other component (c2).

A hole is an external face which is defined by the external boundaries of a connected component. In order to distinguish internal faces from external faces a convention is introduced: an internal face is defined by its boundaries, turning counterclockwise; an external face is defined by its boundaries, turning clockwise. Figure 2 shows the graphic representation of a topological map.

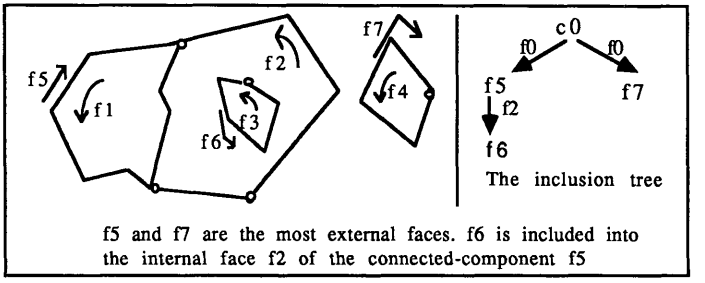

Figure 2: an example of topological map

A topological map can be defined formally as follows:

Definition: topological map

A topological map is defined by: (i) a quadruplet (B, α, σ, γ) which is a non fully-connected topological map; (ii) an inclusion tree τ, composed of nodes c_i representing the fully-connected components of the quadruplet (B, α, σ, γ) and arcs ($c_1 \rightarrow c_2$) connecting two components c_1 and c_2 iff c_2 is included in c_1. An arc ($c_1 \rightarrow c_2$) is labeled by the face of c_1 containing c_2. The root of τ is a virtual component c_0 containing all the components of the map.

3. The GéoGraph model

A topological map can efficiently handle all adjacency operations between objects of a map, but is restricted to operations applied to objects belonging to the same map. In cartography, however, the same area is often represented through several maps, each map representing spatial objects associated to a particular semantic point of view (e.g., road map, land cover, etc.), and users frequently apply complex operations involving several such maps. The GéoGraph model was therefore designed to efficiently deal with a topology involving several layers of maps.

The purpose of this section is to present an extension of topological maps supporting map overlay, and to then define the GéoGraph model. Our model is expressed in terms of a specific graph structure called GéoGraph (which represents an extended topological map), plus a set of basic traversal primitives for GéoGraphs.

3.1. Extending topological maps

Operations involving several layers of maps are based on costly geometrical intersection computations. Optimized computational geometry algorithms for computing shape intersections have been studied [Preparata85], but, even with the use of supporting index structures, they remain slow.

To avoid geometric intersection computations during query processing, intersections between spatial objects (regions, lines and points) of several (overlapping) maps can be pre-computed during the creation of the database. This technique has been exploited in the GEOQL system [Sack87], where geometric processing is reduced by adding to the object coordinate lists of several maps the points corresponding to inter-map object intersections. Checking for object intersections then consists of looking for explicit shared points.

The GéoGraph model is based on pre-computing a collection of elementary spatial objects (ESOs) that correspond to a decomposition of the spatial objects of the original maps. Figure 3 illustrates the result of this pre-computation step applied to the overlay of a map representing Paris's districts and a map representing the underground rail network of Paris. The ESO collection resulting from the pre-computation step constitutes a new map which can be stored as a topological map where the ESOs are represented as faces, edges, and nodes. To speed up operations defined on the original maps while actually

using the ESO map in computations, it is necessary to keep links between the original objects and the ESOs. These links materialize aggregations of ESOs which represent original elements. On the example of figure 3, district number 8 has been split in two faces which have to be aggregated to reconstruct the map of Paris's districts. Topological maps can be extended to maintain these aggregation links.

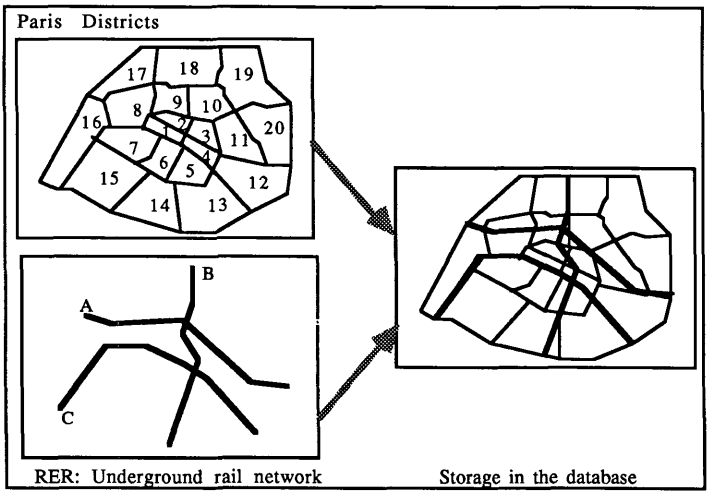

Figure 3: The corresponding storage of two themes.

A topological map extended with aggregation links maintains adjacency relationships between ESOs as usual, plus intersection and inclusion relationships between aggregations of ESOs. Using an extended topological map, most of the operators involving intersection and inclusion relationships on several maps can be evaluated using graph-based processing instead of geometric processing.

3.2. GéoGraph definition

We first introduce a few notations appropriate in the context of a collection of geographical maps. In most of our examples M will denote a particular geographical map. A geographical map M is described by semantic data and spatial data. The spatial data of a geographical map are called regions, lines, or points, and represent, respectively, surfacic features, linear features or ponctual features. We denote by R, L, or P, sets of regions, lines, and points, and by r, l, or p, the elements of these sets. The spatial data of a particular geographical map constitute a subset of R, L or P which is identified by a unique name. We use S to denote one of these subsets. The ESO resulting from the pre-computation step applied to the collection of geographical maps are called faces (inner faces), holes (outer faces), blades and nodes. We denote by F, H, B, N, or C, respectively, the sets of faces, holes, blades, nodes and coordinate lists, and by f, h, b, n, or c, the elements of these sets. A region is an aggregation of faces

and a line is an aggregation of blades. A point is associated to a node.

The GéoGraph graph is a specitic representation of the extended topological map described in section 3.1. This graph is illustrated in figure 4. Nodes of this graph are elements of R, L, P, F, B, and N. Nodes and faces are explicitly represented in this graph (unlike the representation of topological maps given in section 2) in order to more directly associate spatial data with semantic data in the database. Edges of the GéoGraph represent three kinds of functions: functions connecting elements of the original maps to ESOs; functions on the topological maps; and functions connecting each blade to the nodes representing the face and the node defined by the blade. Edge functions connecting elements of an original map to ESOs are identified using the name associated by the map to these elements. This allows retrieving from ESOs the original elements of a particular map. The graph of GéoGraph can be defined more formally as follows.

Definition:

GéoGraph is a graph (X, A) where $X = R \cup L \cup P \cup F \cup H \cup B \cup N \cup C$ is a set of vertices of G and A is the set of edges defined below:

(i) Links between spatial objects and ESO:

- $(r, S, f) \in A$ iff $r \in S$, $S \subset R$, $f \in F$ and f is a component of r,
- $(l, S, b) \in A$ iff $l \in S$, $S \subset L$, $b \in B$ and b is a component of l,
- $(p, S, n) \in A$ iff $p \in S$, $S \subset P$, $n \in N$ and n correspond to p,

(ii) Topological links:

- $(b, \alpha, b') \in A$ iff $b \in B$, $b' \in B$ and $\alpha(b) = b'$,
- $(b, \sigma, b') \in A$ iff $b \in B$, $b' \in B$ and $\sigma(b) = b'$,
- $(b, \varphi, b') \in A$ iff $b \in B$, $b' \in B$ and $\varphi(b) = b'$,
- $(b, \gamma, c) \in A$ iff $b \in B$, $c \in C$ and c is the coordinate list associated with b,
- $(f, h) \in A$ iff $f \in F$, $h \in H$ and h is a hole into the internal face f,

(iii) Correspondence between faces, nodes and blades:

- $(b, f) \in A$ iff $b \in B$, $f \in F$ and f is the left face of b,
- $(b, h) \in A$ iff $b \in B$, $h \in H$ and h is the left hole of b,
- $(b, n) \in A$ iff $b \in B$, $n \in N$ and n is the end node of b.

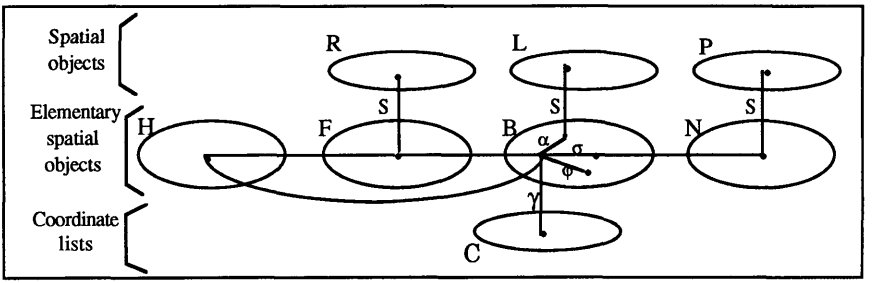

Figure 4: Links between objects in GéoGraph

3.3. Primitive operations

The GéoGraph model provides a set of basic primitives to traverse a GéoGraph. These primitives constitute a toolbox dedicated to the implementation of efficient geometric operations involving several maps via graph traversal operations. This set of primitives is detailed below:

- traversals of aggregation links between ESO and regions, lines or points are supported by the following primitives :
  - (i) MemberFaces (r, S) = {f ∈ F / r ∈ R and (r, S, f) ∈ A},
  - (ii) OwnerRegions (f, S) = {r ∈ R / f ∈ F and (r, S, f) ∈ A},
  - (iii) MemberBlades (l, S), OwnerLines (b, S), MemberNode (p, S), Ownerpoint (n, S) are defined similarly. Note that MemberNode and Ownerpoint are singletons;
- traversals of the topological map defined on ESO are supported by :
  - (i) α, σ and φ which are the conventional permutations of topological maps,
  - (ii) γ gives the coordinate list,
  - (iii) The primitives necessary to traverse the inclusion tree of topological maps :
    ContainingFace (h) = {f∈ F / h∈ H and (f, h) ∈ A},
    ContainedFaces (f) = {h ∈ H / f∈ F and (f, h) ∈ A};
- traversals between faces holes and nodes defined by blades:
  (since faces and nodes are described by a list of blades, the inter-connections of blades, faces and nodes are expressed in the next primitives)
  - (i) Left (b) = {f∈ F / b∈ B and (b, f) ∈ A} (Left (b) is a singleton),
    BoundingBlades (f) = {b ∈ B / f∈ F and (b, f) ∈ A},
  - (ii) Left (b) = {h∈ H / b∈ B and (b, h) ∈ A} (Left (b) is a singleton),
    BoundingBlades (h) = {b ∈ B / h∈ H and (b, h) ∈ A},
  - (iii) EndNode (b) = {n∈ N / b∈ B and (b, n) ∈ A} (EndNode (b) is a singleton),
    ArrivingBlades (n) = {b ∈ B / n∈ N and (b, n) ∈ A}.

GéoGraph primitives allow traversal of all edges of a GéoGraph in both directions, so any traversal of a GéoGraph can be expressed by a combination of these primitives. Within a GéoGraph, adjacency relationships between objects of several geographical maps can be deduced from traversals along aggregation edges and along edges materializing the topological map defined on the ESOs. Containment relationships between objects of the same type (region/region or line/line) belonging to different geographical maps can be deduced directly from traversals along aggregation edges. Containment relationships between objects of different types (region/line, region/point or line/point) belonging to different geographical maps can be deduced from traversals along

aggregation edges and along edges materializing the topological map defined on the ESO. GéoGraph primitives are thus sufficient to carry out all the operations based on the inter-object relationships.

To ease the implementation of these operations it is possible to deduce functions useful as construction blocks in the design of a GIS. For example, the function AdjFaces(f) = {Left (b) / b ∈ BoundingBlades (f)} can be defined to retrieve the adjacent faces of face f; the function Right(b) = Left (α(b)) can be defined to retrieve the right face of blade b. In summary, GéoGraph supports all the links between spatial objects which are important to efficient geometric operators. We leave unspecified the details concerning links between spatial objects and semantical objects. These links are not included in the GéoGraph model for they depend on the data model used in the GIS.

4. Using the GéoGraph model

This section illustrates the implementation of spatial operators as an extention of a DBMS using the GéoGraph model. For the sake of clarity, the spatial operators are presented in a relational context, but it is important to note that these operators and their implementation can be generalized to other data models.

First, we present a possible integration of the GéoGraph data in a database using a relational DBMS which can be extended with abstract data types [Stonebraker83]. Then, classical spatial predicates and spatial functions are enumerated. Spatial predicates are applied to a couple of spatial objects to check some spatial properties. The definition of the main spatial operators required in a relational DBMS extended toward geography is then introduced. These operators are based on spatial predicates and spatial functions. They work on sets of spatial objects and return the combination of spatial objects which satisfy a given predicate. Finally the implementation of these operators with the GéoGraph model is detailed. The architecture and the query langage of such a DBMS extended toward geography can be found in [Bennis90].

4.1. Connection of GéoGraph with an extended relational database

A cartographic object is composed of semantic and spatial data. In order to support the spatial data, the spatial domains Regions, Lines and Points are added to the conventional domains of values used in relational DBMSs. A spatial relation is then defined as a relation containing at least one attribute which takes its values in a spatial domain. A map is a spatial relation with exactly one spatial attribute which is a key of the relation. The conventional relational operators (i.e. selection, join and projection) are augmented with spatial operators for spatial attribute manipulations.

To clarify the discussion, a few notations are introduced below. We use two spatial relations named R and S. We denote by R.k (resp S.l) the spatial attribute of R (resp S). We denote by r (resp s) a tuple of the R (resp S) spatial relation. We denote by ΔV a set of values varying on the same domain and by v a particular value of this set. The result of a spatial operation is a relation denoted by RES.

In the sequel of the paper we assume for clarity that the database is stored according to the DBGraph storage model introduced in [Pucheral90]. This model stores a relational database as a bipartite graph composed of a set of tuples named T, a set of values named V and edges connecting each tuple to each of its attribute values (see figure 5). The purpose of this storage model is to precompute all conventinal relational operations, which is complementary with the objective of the GéoGraph. Two basic traversal primitives are provided: succ_tup(t, R.k) is a function from T to V that delivers the value of attribute R.k of tuple t and succ_val(v, R.k) is a function from V to T that delivers the set of tuples whose th attribute value is v.

In a DBGraph, tuple vertices (resp. value vertices) may be grouped on a relation basis (resp. domain basis) since the relations form a partition of T (resp. V). Spatial domains constitute three subsets of V which comprise the sets R, L, P used as entry points in the GéoGraph.

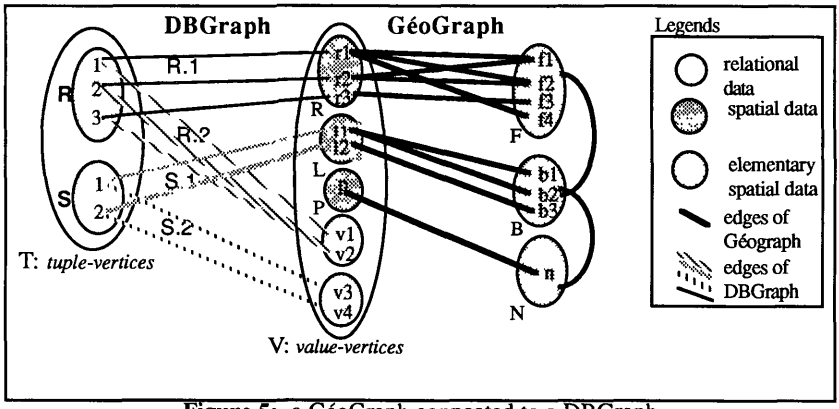

Figure 5: a GéoGraph connected to a DBGraph

4.2. Spatial Predicates and Spatial Functions.

A spatial predicate takes two input spatial attribute values, and checks whether a given spatial property is satisfied by this pair. The two input values can be of different domains, although there are some restrictions depending on the predicate. Figure 6 summarizes the classical spatial predicates and the domains of the allowed input values. Spatial predicates fall into two categories: one for checking neighborhood relationships such as adjacency of regions; the other for checking containment relationships such as inclusion or overlap of spatial objects. For detailed information on these predicates see [David89]. Each spatial predicate can be evaluated by a traversal of the GéoGraph. This traversal can be expressed by a sequence of primitive operations of the GéoGraph model. One part of this traversal corresponds to a translation of the regions, lines or points given in entry into ESO (faces, blades and nodes), while the other part selects elementary components which satisfy the predicate. The first part uses operations on aggregation links and the second part uses operations of topological maps. Examples of such

traversals can be found in section 4.4.

Predicate	Region	Line
Region	Adjacent Overlap Inclusion	Border Overlap
Line	Right Left Overlap Inclusion	Connected Overlap Inclusion
Point	Inclusion	Ends Inclusion

Figure 6: The spatial predicates

Spatial functions are useful to calculate new values which are either numerical or geometrical (coordinates lists). Some are also used to compute new values of type region, line or point based on new aggregations of ESO. These functions can be classified into two categories: those requiring a geometrical computation on spatial values, and those which can be sped up by the topological map included in the GéoGraph model. Figure 7 summarizes the main spatial functions.

Functions	Unary	Binary
Speed up by GéoGraph		**Result ∈ R or L or P** Intersection(O1,O2) with O1 ∈ R or L and O2 ∈ R or L Fusion(O1,O2) with O1 ∈ R andO2 ∈ R or O1 ∈ L andO2 ∈ L Difference(O1,O2) with O1 ∈ R andO2 ∈ R or O1 ∈ L andO2 ∈ L
Involve a geometrical computation	**Result is numerical** Area(O1) with O1 ∈ R Perimeter(O1) with O1 ∈ R Length(O1) with O1∈ L **Result is a coordinate list** Geometry(O1) with O1 ∈ R or L	**Result is numerical** Distance(O1,O2) with O1 ∈ R or L or P and O2∈ R or L or P

Figure 7: The spatial functions

4.3. Spatial Operators

The data model of GéoTropics uses three basic spatial operators: the spatial selection, the spatial join, and the calculation operators. The first two of these correspond to extensions of the conventional selection and join operations of the relational model. The spatial operators take as arguments one or two spatial relations and return a spatial relation, and are now explained

The **Spatial Selection** operator, denoted by Sel(S, Q), is applied to a relation S, and determines the subset ΔS of the tuples of Swhose spatial attribute satisfy a qualification Q. The qualification Q is a simple comparison S.l Θ const where const is a spatial constant of the domain region or point and where Θ is a spatial predicate. The spatial predicate used is generally inclusion or overlap. Selections involving these predicates correspond to the usual geographical operations of clipping and windowing. The selection operator is expressed as:

Sel(S, Q) = {s∈ S/ (s.l Θ c) is true}

The **Spatial Join** operator denoted by Join(R, S, Θ), is applied to the spatial relations R and S, and determines a set of tuples composed of all possible combinations of a tuple r ∈ R concatenated to a tuple s ∈ S, such that the spatial attributes R.k and S.l of r and s satisfy the join condition R.kΘS.l, where Θ is a spatial predicate. Most of the spatial operations based on relationships between spatial objects can be expressed by a join operation involving a specific spatial predicate. The join operator is expressed as follows:

Join (R, S, Θ) = {(r, s) / r ∈ R, s ∈ S and (r.k Θ s.l) is true}

The **Spatial Calculation** operator, denoted by Calc (R, f) or Calc (R, S, f) can take one or two relations as arguments. It is similar to the spatial join, but extends the concatenation of the entry relations attributes with the result of a spatial function f applied to the spatial attributes. Note that the join predicate is replaced by function f. This operation can be expressed as follows :

Calc (R, f) = { (r, f(r.k) / r ∈ R }

Calc (R, S, f) = { (r, s, f(r.k, s.l)) / r ∈ R, s ∈ S}

The widely used overlay operation can be translated as follows: projection (Calc (R, S, ∩)) where ∩ involve R.k, S.l and the projection discards these two attributes.

4.4. Spatial Operator Implementation.

The GéoGraph model is currently being used to support the three spatial set-oriented operators presented in section 4.3. Numerous versions of these operators can be deduced depending on the spatial predicate used. It is not possible to give three general algorithms that support all versions of these operators. For illustration, this section focuses on two versions of the join operator and one version of the calculation operator which correspond to the more commonly used geometric operations and illustrate well the functionnality of the GéoGraph model. The join operation is first applied to retrieve all couples of adjacent regions of two maps (spatial relations). Then it is applied to retrieve all couples of overlaping regions and lines of two maps. The joins involves the adjacency and the containment relationships, which belong to the two classes of spatial predicates. The classical operation of overlay is then expressed in primitives of the GéoGraph model. The execution of the selection operator is not detailed in this section because it is efficiently handled by algorithms involving geometrical indices. For this operation the GéoGraph model had to be augmented with geometric indices for elementary components (Face, Blade and Node) and spatial attribute values (region, line or point) (see section 5).

Let us consider the adjacent-join, a **spatial join** operation involving an adjacent predicate. Two regions are adjacent if they have adjacent faces and if they do not have

common faces. The algorithm performing the operation is given figure 8. This algorithm may be decomposed into four steps, each of which is a traversal of a subpart of the GéoGraph. For each tuple r of R, the first step decomposes the region-value of attribute R.k into faces. The second step executes a sequence of topological operations on ESOs in order to get the set of faces which are adjacent to some faces obtained in the first step. The third step converts the faces obtained in step2 into region-value of attribute S.l. All these regions contain at least a face adjacent to one face of the region-value of r. The fourth step checks that region-values selected in the 3rd step do not have common faces with the region-value of r.

```
Function Join (R, S, Adjacent)
/* R.k and S.l take their values in R */
/* we assume card(R) < card(S) */
begin
/*.....1st step */
for each r ∈ R do
    ΔV3 = ∅;
    ΔV1 := MemberFaces(r.k, R.k);          /* decomposition into elementary faces */
    /*.....2nd step */
    for each v1 ∈ ΔV1
        ΔV2 := BoundingBlades (v1);          /* give all blades bounding v1 */
        for each v2 ∈ ΔV2
            ΔV3 := ΔV3 ∪ Left (v2);          /* give adjacent faces of v1 */
        endfor
    endfor
    ΔV3 := ΔV3 - ΔV1;                        /* discard faces belonging to r.k */
    /*.....3rd step */
    for each v3 ∈ ΔV3
        ΔV4 := ΔV4 ∪ OwnerRegions(v3, S.l);  /* retrieve regions of S.l including face v3 */
    endfor
    /*.....4th step */
    for each v4 ∈ ΔV4
        if ΔV1 ∩ MemberFaces(v4, S.l) ≠ ∅     /* discard the regions which have */
                                              /* common faces with region r.k */
        then RES := RES + (r , succ_val(v4, S.l) );   /* build the result made of tuple r */
                                              /* and the tuple owning the region v4 */
endfor
end
```

Figure 8: Join operator involving the adjacency predicate

Consider now the overlap-join operator, a join operation involving a spatial predicate checking the overlap between a line and a region. The algorithm performing this operation is given figure 9 which starts from the lines to reach the overlaping regions. A line and a region overlap if at least one blade of the line defines two faces belonging to the region. This algorithm is also based on four steps which are similar to the four steps of the previous algorithm. Nevertheless, these steps cannot be expressed in exactly the same way. The first three steps determine the set of regions on the left of the line and the set of regions on the right of the line. The fourth step performs the intersection of these two sets.

```
Function Join (R, S, Overlap)
/* R.k takes its value in L
and S.l takes its value in R  */
/* we assume card(R) < card(S) */
begin
/*.....1st step */
for each r∈ R do
      ΔV4:= ∅;
      ΔV1 := MemberBlades(r.k, R.k)          /* decomposition into elementary blades */
      for each v1 ∈ ΔV1
            /*.....2ndstep */
            vl= Left(v1);                    /* give the face on the left of blade v1 */
            vr:= Left(α(v1));                /* give the face on the right of blade v1 */
            /*.....3rd step */
            ΔV2 := OwnerRegions(vr, S.l);    /* retrieve regions of S.l including face Vr */
            ΔV3:= OwnerRegions(vl, S.l);     /* retrieve regions of S.l including face Vl */
            /*.....4th step */
            ΔV4:= ΔV4 ∪ (ΔV2 ∩ ΔV3);         /* discard the regions which border line r.k */
      endfor
      for each v4 ∈ ΔV4
            RES = RES ∪ (r , succ_val(v4, S.l) );   /* build the result made of tuple r */
                                                    /* and the tuple owning the region v4 */
endfor
end
```

Figure 9: Join operator involving the overlap predicate

Finally, let us consider a version of the spatial **Calculation operator** involving the function of intersection of two regions. This operation is applied to two maps and produces the overlay of the maps. The corresponding algorithm is given figure 10. If we compare this algorithm to the previous ones, it can be decomposed into four steps with an empty second step, since containment relatioships between objects of the same type can be deduced from traversals along aggregation edges without the use of the topology. In this algorithm, the computation of ∩ (r.k, s.l) is reduced to a simple set-intersection between two sets of faces with no access to the coordinates of the spatial objects geometry .

In our experience, most of the algorithms of binary geometrical operations can be decomposed into four steps. The first step decomposes the spatial attribute values of the first map into a set ESO1, using aggregation edges. The second step performs topological operations starting from ESO1 to reach a set ESO2 satisfying a topological predicate. The third step retrieves spatial attribute values of the second map which aggregate elements of ESO2, and the fourth step performs some supplementary verifications. For operations involving containment relationships on objects of the same type, the second step is not required. An important contribution of the extended topological map is that algorithms based on these four steps always access ESOs of the same spatial location.

```
Function Calc (R, S, Intersection)
/* we assume the R.k and S.l take their values in R */
begin
 /*.....1st step */
for each r ∈ R do
    ΔV1 := MemberFaces(r.k, R.k);          /* decomposition into elementary faces */
    ΔV2:= ∅;
    /*.....3rd step */
    for each v1 ∈ ΔV1
        ΔV2=ΔV2∪OwnerRegions(v1, S.l);     /* retrieve regions of S.l including face v1 */
    endfor
    /*.....4th step */
    for each v2 ∈ ΔV2
        ΔV3 := MemberFaces(v2, S.l);       /* decomposition into elementary faces */
        RES = RES + (r, succ_val(v2, S.l), ΔV1 ∩ ΔV3);   /* build the result made */
                                           /* of tuple r and of the tuple owning region */
                                           /* v2 and of the intersection of r.k and v2 */
    endfor
endfor
end
```

Figure 10: Calculation operator involving the intersection function

5. Implementation

There are many ways to implement the GéoGraph graph. It could be implemented within a Network DBMS, an Object Oriented DBMS or as an extension of a Relational DBMS. We detail below a particular implementation based on the third approach, and on the connection of GéoGraph with the DBGraph model [Pucheral90] (as detailed in section 4.1). The key point of this implementation is a good data clustering. The objective is to partition the two graphs of figure 5 into separate segments which can be loaded separatly according to the needs of operations being executed.

In order to ease the data partitionning, tuples and values are stored separately (see figure 11). Since the domains form a partition of the set of values V, all the values varying over the same domain can be clustered in a separate segment. Taking advantage of vertical partitioning, the values of one domain can be loaded independently of the others. Similarly, since the relations form a partition of the set of tuples T, the tuples of one relation can be stored in one segment. Each object stored in a segment has a unique and invariant identifier (OID). Thus, tuples and values can be referenced by OID's.

In the definition of a DBGraph, an edge between a tuple and a value can be traversed in both directions. Consequently an edge is represented with two physical arcs: one from the tuple to the value, and also a reverse arc. A tuple is implemented as an array of OIDs, each referencing its attributes values, which are stored in separate domains. These OIDs materialize arcs from the tuples to the values. Reverse arcs are materialized by inverted lists attached to the values. Each inverted list is divided into a set of sublists so that there

is one sublist per attribute varying on the domain of the value. All these sublists are referenced by an array attached to the corresponding value. Because of vertical partionning, it is possible to cluster all the inverted sublists of one attribute into one segment. Indices may be added on domain values to speed up selections on all attributes varying on the same domain.

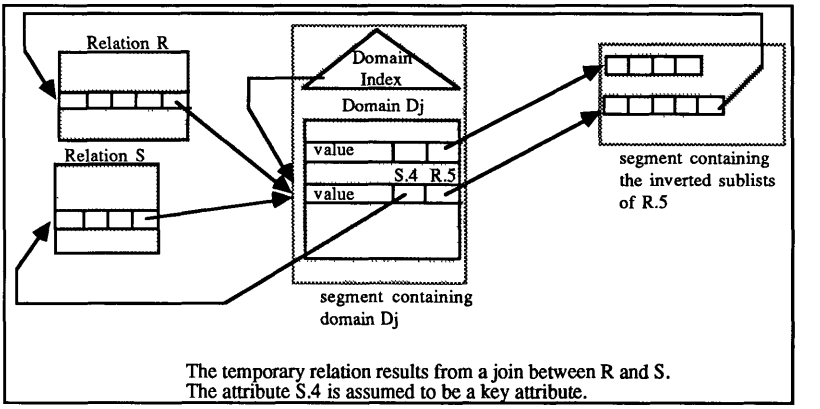

Figure 11: implementation of the DBGraph part

Consider now the implementation of the GéoGraph part. There is a direct mapping between a value of the spatial domain point and a node. Thus attributes of type point directly reference node values stored in a node domain. Values of the spatial domain region (resp. line) are stored as set of OIDs referencing face (resp. blade) values stored in a face (resp. blade) domain. The sets of OIDs materialize aggregation links from spatial objects to ESOs. Reverse links from blades and faces to the tuples, materilize reverse aggregation links from ESOs to tuples containing spatial objects. Face and blade domains are stored like DBGraph domains with inverted lists in order to materialize these links. It is not necessary to maintain inverted lists for region and line values. For some attributes it may be inefficient to store the values separatly from the tuples because they are accessed each time the tuple is accessed, and graph traversals of costly operations don't use links between the corresponding attribute values and the tuples. Values of these attributes can be stored directly in the tuples. For example, region and line attributes fall in this category. Spatial indices are maintained for the three domains node, blade and face. Values of these domains are clustured with the spatial indices which reinforce the contribution of the vertical partitionning, since algorithms based on extended topological maps favors access to ESOs of the same spatial location (see section 4.4).

Values ot the three domains face, blade and node have a complex structure that stores the topology of the ESO map (see figure 12). The main part of this topology is supported by the blade values. A blade value is represented by a record containing five fields: (i) the OID of the opposite blade, which materializes the α function; (ii) the OID of the next blade of the end-node, which materializes the σ function; (iii) the OID of the left face of

the blade, which, used in conjunction with the α function, allows retrieval of the two faces bordering the blade; (iv) the OID of the end-node of the blade, which is necessary to access to the inverted list of the node; and (v) the OID of a coordinate list materializing the geometry of the blade. Coordinate lists are stored in a separate domain without inverted list. This domain is clustered with a geometrical index. A face value is the OID of any blade of the face. The value of one node contains only the OID of one blade reaching it. This information is sufficient for a node since the σ function gives the complete cycle of blades reaching it. Furthermore its geometry can be extracted from the geometry of one of its blade. In a similar way, the value of one face is composed of a record containing the OID of one blade of its boundary and a set of OID corresponding to the set of holes contained in this face. Applying a succession of α and σ functions to the blade referenced by the face gives the complete cycle of blades of its boundary. Its geometry can be obtained from the geometry of all of these blades.

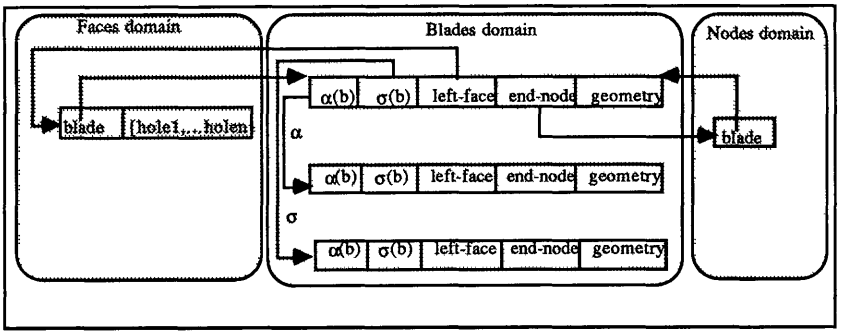

Figure 12: Spatial values representation

5. Summary and futur work

In this paper, we have presented the GéoGraph storage model, a toolbox supporting low layers of GIS in an extensible way. The definition of this model was given independently of implementation detail, in terms of a graph structure and primitive operations on that structure. This facilitates the description of the toolbox functionality, and supports our argument of general utility. Topological information and geometric information of several maps are incorporated in a single graph that directly supports geometric operations based on adjacency and containment relationships. This graph is based on the topological map theory guaranteeing that all updates on topological information are coherent, and providing a minimal set of operations to navigate through the graph.

Although GéoGraph is intended for various higher level data models, we illustrated the utilization of this storage model in the context of an extensible relational DBMS. In this context, the resulting GIS is itself extensible and can exploit fully the toolbox aspect of GéoGraph. We showed that the GéoGraph graph can be integrated with relational data in a straightforward fashion. Algorithms of the main geographical operations have been

given in an abstract form using the basic primitives of GéoGraph. These algorithms, based on graph traversals, are simple and exibit desirable locality properties.

A specific implementation of the GéoGraph model has been proposed. This implementation avoids data duplication and shows that the GéoGraph graph can be partitioned to minimize disk trafic. Spatial data are clustered with spatial indices. This, combined with the space locality properties of the algorithms, reinforce the contribution of vertical partitionning. This implementation is currently being experimented in the framework of the GéoTropics system, an extensible GIS based on extensions of SQL [Bennis90].

Additional research will be useful in enhancing the GéoGraph model. For example, the decision to always overlap geographical maps has some drawbacks: operations involving only one map can be slowed down, since the number of elementary storage elements can be unnecessarily large. It may prove more efficient to selectively overlap layers based on the frequency of their joint use in queries[David90].

References

[Ansaldi85] Ansaldi S., De Floriani L., Falcidieno B., "Geometric Modeling of Solid Objects by Using Face Adjacency Graph Representation", ACM SIGGRAPH'85 Conf., San Francisco, USA, 1985.

[Bennis90] Bennis K., David B., Quilio I., Viémont Y., "GéoTROPICS: Database Support Alternatives for Geographic Applications", 4th Int. Symposium on Spatial Data Handling, Zurich, Switzerland, July 1990.

[Cori81] Cori R. & Vauquelin B., "Planar maps are well labeled trees", Can. J. Math., Vol. XXXIII, 1981.

[David89] David B., "External Specifications for the Cartographic DBMS", ESPRIT-TR 2427-0022 TROPICS Project, June 1989.

[David90] David B., Viémont Y., "Data Structure Alternatives for Very Large Spatial Databases", Sorsa colloquium 90, Fribourg, Deutshland, July 1990.

[Dufourd88] Dufourd J.F., "Algebraïc Specification and Implementation of the Topological Combinatorial Maps", PIXIM'88 proc., Paris, France, 1988.

[Dufourd89] Dufourd J.F, Gross C. et Spehner J.C., "A Digitizing Algorithm for the Entry of Planar Maps", Computer Graphics International'89, Leeds, Spring-Verlag, June 1989.

[Edmonds60] Edmonds J., "A Combinatorial Representation of Polyhedral Surfaces", Notices Amer. Math. soc. n°7, 1960.

[Herring87] Herring J.R., "TIGRIS: Topologically integrated geographic information system", Auto-Carto'8 proc., Baltimore, Maryland, USA, March 1987.

[Herring90] Herring J., "The definition and development of a Topologically Spatial

Data System", Photogrametry and Land Information Systems, Lausanne, Suisse, March 1989 (Published in 1990).

[Lienhardt89] Lienhardt P., "Subdivision of N-Dimensional Spaces and N-Dimensional Generalized Maps", 5th ACM Symposium on Computational Geometry, Sarbrüken, RFA, June 1989.

[Kinnea87] Kinnea C., "The TIGER Structure", Auto-Carto'8 proc., Baltimore, USA, March 1987.

[Meier82] Meier A., "A Graph Grammar Approach to Geographic Databases", Proc. of 2nd Int. Work. on Graph Grammars and their Application of Computer Science, october 1982, Lecture Notes in Computer Science, Springer Verlay, 1982.

[Meixler82] Meixler D., Sadlfeld A., "Storing, Retrieving and Maintaining Informations On Geographic Structures", Auto-Carto'7 Proc., Wachington, USA, March 1985.

[Morehouse85] Morehouse S., "A Geo-Relational Model for Spatial Informations", Auto-Carto'7 Proc. , Washington D.C., USA, 1985.

[Peuquet84] Peuquet Donna J., " A Conceptual Framework and Comparison of Spatial Data Models", Cartographica vol21 n°4, 1984.

[Pucheral90] Pucheral P., Thévenin J.M., Valduriez P., "Efficient main memory Data Management Using the DBGRAPH Storage Model", VLDB90 proc., Brisbane, Australia, August 1990.

[Sack87] Sack-davis R., Mcdonell K.J., "GEOQL - A Query Language for Geographic Information Systems", Australian and New-Zeland Association for the Advancement of Science Congress Townsville, Australie, August 1987.

[Samet85] Samet H., Webber E., "Storing a collection of Polygons Using Quadtree", ACM Transaction on Computer Graphics 3 (4), July 1985.

[Schaller87] Schaller J., 3The Geographical Information System (GIS) ARC/INFO", EuroCarto VI proceedings, Czechoslovakia, April 1987.

[Spooner90] Spooner R. "Advantages and Problems in the Creation and Use of Topologically Structured Database", Photogrametry and Land Information Systems, Lausanne, Suisse, March 1989 (Published in 1990).

[Stonebraker83] Stonebraker M., Rubenstein B., Guttman A., "Application of Abstract Data Types and Abstract Indices to CAD Databases", ACM Sigmod, San-Jose, 1983.

[Waugh87] Waugh T.C., Healey R.G., "The GEOVIEW Design, a Relational Approach to Geographical Data Handling", Int. J. Geographical Information Systems", 1(2), 1987.

[White79] White M. "A Survey of the Mathematics of Maps", Auto-Carto'4 proc., 1979.