Graph Data Model Operations for Network Facilities in a GIS

Langou B.*,** Mainguenaud M.*

* France Telecom - Institut National des Télécommunications
9 rue Charles Fourier F91011 Evry - France
+ 33 1 60 76 47 14 + 33 1 60 76 47 80 (fax)
Email: langou@etna.int-evry.fr Michel.Mainguenaud@int-evry.fr

** ATARAXIE
Technoparc - 22 rue Gustave Eiffel
F78300 POISSY

Abstract:

In this paper, we present graph operators defined on a graph data model. A graph data model is well adapted to model information such as public transportation, water pipes… This model is designed to handle network facilities in a Geographical Information System. A graph data model must provide a coordinate-free notion of abstraction for a node (or an edge).
Graph manipulations are divided into three classes. Basic operators manage the notion of abstraction (i.e., the DEVELOP and UNDEVELOP operators). Elementary operators manage the notion of graph and sub-graph (i.e., UNION, CONCATENATION, SELECTION and DIFFERENCE operators). High level operators manage GIS user interface operators (i.e., PATHS, INCLUSIONS and INTERSECTIONS operators).

I. INTRODUCTION

A Geographical Information System (GIS) should provide the opportunity to manage network facilities (i.e., railway, electricity, telephone). A graph [2] (i.e., a set of labeled nodes and a set labeled edges) is the major conceptual notion used to model network-oriented data. To capture more meaning, a graph data model must provide a mechanism of abstraction. A node (i.e., a town) may represent the abstraction of one (or several) network (i.e., a local area transportation network). An edge may also represent the abstraction of a network (i.e., a railway line).

[14] presents a graph data model. This paper presents the operators defined on this model. The data model allies graph theory concepts (i.e., node, edge, graph) and an object oriented database paradigm (i.e., abstraction). This data model is defined at a logical level (i.e., the same level as [5]). The notion of coordinates is not present. The notion of abstraction is not provided from a physical representation (i.e., spatial representation) to an object level [6,12] but within an object level. Since the data model is richer than a flat graph, specifications of database operators are more complex. Several levels of interaction are defined to manipulate network-oriented data. Basic operators correspond to the first level (i.e., directly linked to the concept of abstraction in the graph data model). Elementary operators correspond to the second level (i.e., the manipulations of graphs and sub-graphs). High level operators correspond to the third level (i.e., GIS user interface operators). We retained three classes of high level operators. They correspond to basic operations to manage network facilities [4]: the evaluation of a path between an origin and a destination, the intersection of paths and the inclusion of paths. Results of a query are influenced by the fact that the graph data model presents a higher level of abstraction than a flat graph.

Part II presents a brief overview of the graph data model used in this paper; Part III presents the basic operators; Part IV presents the elementary operators; Part V presents the high level operators; Part VI presents a conclusion and future work.

II. BRIEF OVERVIEW OF THE GRAPH DATA MODEL

The graph data model allies notions of graph theory such as vertices (here called nodes), edges and graphs with notions of an object-oriented paradigm such as abstraction. A node or an edge may represent the abstraction of sub-networks. We do not provide any constraint on the topology of this sub-network (except that loops are forbidden). In this paper, we use the concept of logical OID (Object IDentifier) to define the operators. This notion is similar to the notion of OID in object oriented databases since the semantics is the same. Nevertheless, the OID defined in the graph data model is completely system independent.

II.1. Basic concepts

The graph data model is based on the concept of nodes, edges and graphs. These notions are similar to these notions in graph theory [2]. A graph is a couple (N, E) where N is a set of nodes and E is a sub-set of the Cartesian product N x N. Figure II.1 presents the formal definition.

$$N = \{n_1, \ldots, n_p\}$$
$$E = \{ (n_i, n_j) / n_i \in N, n_j \in N \}$$
(n_i is said to be the initial node and n_j is said to be the end node for an edge (n_i, n_j))

Figure II.1 - Formal definition of a graph

In this paper, a graph is considered as orientated (i.e., the order, initial node / end node, is relevant). Several edges may be defined with the same initial and end nodes. We do not limit this number but we do not allow an edge (n_i, n_i). A node is used to model for example a town. An edge is used to model for example a link between two towns. Nodes and edges are labeled. We do not consider in this paper the label associated with a node or an edge. Let real life data (i.e., towns) modelled by a node or an edge be named objects.

We define the notion of Master_node and Master_edge. They represent an abstraction of a sub-network (i.e., the level of abstraction of a Master_node or a Master_edge is higher than the level of abstraction of the sub-network they represent). Such a sub-network is called an Associated_network. An Associated_network is defined as a graph G (N, E). Elements of N are called specialized nodes. Elements of E are called specialized edges. To connect this network to the different levels of abstraction, we define the concept In_edges and Out_edges. The In_edges (respectively Out_edges) of an Associated_network of a Master_node represent the set of edges "arriving to" (respectively "leaving") this graph. Figure II.2 presents the definition of In_edges and Out_edges.

$$In\_edges = \{ (n_i, n_j) / n_i \notin N \wedge n_j \in N \}$$
$$Out\_edges = \{ (n_i, n_j) / n_i \in N \wedge n_j \notin N \}$$

Figure II.2 - Formal definition of In_edges and Out_edges for a Master_node

The In_edges (respectively Out_edges) of an Associated_network of a Master_edge (e_i) represent the set of edges "leaving" (respectively "arriving to") the initial node (respectively end node) of a Master_edge. Let us define the function Initial_node (respectively End_node) of a Master_edge (e_i) such as Initial_node (e_i) (respectively End_node) returns the initial (respectively end) node of an edge (e_i). Figure II.3 presents the definition of In_edges and Out_edges for the Associated_network G□(N,□E) of a Master_edge, e_i.

$$In\_edges = \{ (n_i, n_j) / n_i = Initial\_node (e_i) \wedge n_j \in N \}$$
$$Out\_edges = \{ (n_i, n_j) / n_i \in N \wedge n_j = End\_node (e_i) \}$$

Figure II.3 - Formal definition of In_edges and Out_edges for a Master_edge

An empty graph, G∅, is a graph with no node (i.e., a graph G (N, E) such as N = ∅). This graph is the neutral element of a set of graphs.

II.2. Abstraction

The notion of abstraction [3,13,17] is modelled by the concept of OID. Each basic component (i.e., node, Master_node, edge, Master_edge, network, Associated_network) of the graph data model has an OID. This OID is structured in several layers from L to 1. Each layer corresponds to a level of abstraction. To simplify the presentation, we present in this part the structuring of a node OID to take into account the notion of abstraction. The other OIDs are defined on the same principles. The structure of the OID will evolve along this paper to provide a complete structure whenever all the operators are defined. The very beginning structure of an OID (i.e., the abstraction part) is defined by two rules. Figure II.4 presents with the database tuple constructor [] these two rules.

The abstraction part of an OID is defined by a list of integers. An element of this list is called a layer. An object (o) in the layer k of abstraction is defined by:

Rule 1 - o is located at the top level of abstraction (i.e., k = L). The first layer has the local identifier (i.e., an integer) and the other layers are null.

Rule 2 - o is not located at the top level ($1 \leq k < L$). The (L - k) first layers contain each local identifier of the higher levels of abstraction. The layer k contains the local identifier. The (k - 1) last layers are null.

OID : [hierarchy_of_abstraction : list of integer]

Figure II.4 - Rules to build an OID

II.3. Conclusion

The graph data model provides a structure to handle networks in a GIS database context. The central notion is the abstraction of sub-networks. The notion is valid for nodes and for edges. The Associated_networks, In_edges and Out_edges allow to model several levels of abstraction. The following sections present the various operators defined on the graph data model.

III. BASIC OPERATORS

In this section, we present the operators directly linked to the concepts of the graph data model (i.e., abstraction). Two basic operators are defined: the DEVELOP operator and the UNDEVELOP operator. The DEVELOP operator provides more specific details by merging a sub-network associated with a Master_node or with a Master_edge into the studied graph. The converse operation, the UNDEVELOP operator, provides a more restricted graph by the replacement of a sub-network by a Master_node or a Master_edge.

III.1. DEVELOP operator

The DEVELOP operator has two arguments. The first argument is a graph (i.e., G_0). This graph contains Master_nodes and/or Master_edges to be developed. The second argument is a set of graphs. Each graph represents a path (i.e., only a unique Associated_network by Master_node and only a unique In_edge and Out_edge by Associated_network). The result is a set of graphs. They are as many graphs as in the second argument. For each graph, Master_nodes and Master_edges also present in the first argument (i.e., G_0) are developed. Associated_networks are merged into the studied graph. Figure III.1 presents the signature of this operator.

	DEVELOP (G_o, {G_i}) : {G'_i}
where	G_o : is a graph with the set of nodes and /or a set of edges to be developed
	G_i : a graph to be expanded ($i = 1, ..., n$)
	G'_i : a graph after being expanded ($i = 1, ..., n$)

Figure III.1 - The signature of the DEVELOP operator

Let n be the number of graphs in the second argument. The number of graphs in the result is also n since no deletion of graph is involved in this operator.

A difference is introduced between the development of a Master_edge and a Master_node. The development of a Master_edge implies the deletion of this Master_edge in the result since the Associated_network is merged into the studied graph. The development of a Master_node does not imply the deletion of this node since it may be the initial node (or the end node) of an edge (or a Master_edge) [14].

As a convention, the DEVELOP operator provides a single development of Master_nodes and/or Master_edges (i.e., only one level of abstraction). To provide a recursive application of the DEVELOP operator, a similar operator, DEVELOP*, is available. This operator has a unique argument, a set of graphs, since each Master_node and each Master_edge are recursively developed.

A graph obtained by a DEVELOP* operator does not contain any more Master_node or Master_edge. Figure III.2 presents the signature of this operator.

	DEVELOP* ({G_i}) : {G'_i}
where	G_i : a graph to be expanded ($i = 1, ..., n$)
	G'_i : a graph after being expanded ($i = 1, ..., n$)

Figure III.2 - The signature of the DEVELOP* operator

Let n be the number of graphs in the argument. The number of graphs in the result is also n since no deletion of graph is involved in this operator.

III.2. UNDEVELOP operator

The UNDEVELOP operator is applied to a set of graphs. For each graph with nodes and edges defined on two different levels of abstraction (at least), a level of abstraction (i.e., the lower level) is transformed into Associated_networks. The Master_nodes and/or Master_edges are introduced into the studied graph.
Figure III.3 presents the signature of the UNDEVELOP operator.

	UNDEVELOP ({G_i}) : {G'_i}
where	G_i : a graph to be reduced ($i = 1, ..., n$)
	G'_i : a graph after being reduced ($i = 1, ..., n$)

Figure III.3 - The signature of the UNDEVELOP operator

Let n be the number of graphs in the argument. The number of graphs in the result is also n since no deletion of graph is involved in this operator.

To be able to reconstruct a sub-network as an Associated_network, the initial OID of this sub-network must be present in the OID of nodes and/or edges. A hierarchy of network is introduced. The definition of the OID is therefore augmented with the abstraction of dependent networks. The semantics of this hierarchy is the same as the hierarchy of abstraction defined on nodes and edges. Figure III.4 presents the new structure of an OID.

OID :	[hierachy_of_abstraction : list of integer
	hierarchy_of_network : list of integer]

Figure III.4 - The new structure of an OID

As a convention, the UNDEVELOP operator provides a single reduction of sub-networks into Master_nodes and/or Master_edges. To provide a recursive application of the UNDEVELOP operator, a similar operator, UNDEVELOP*, is available. The UNDEVELOP* operator is also applied to a set of graphs. The result is a set of graphs. Figure III.5 presents the signature of this operator.

	UNDEVELOP* ({G_i}) : {G'_i}
where	G_i : a graph to be reduced (i = 1, ..., n)
	G'_i : a graph after being reduced (i = 1, ..., n)

Figure III.5 - The signature of the UNDEVELOP* operator

Let n be the number of graphs in the argument. The number of graphs in the result is also n since no deletion of graph is involved in this operator.

The nodes (respectively the edges) of a graph obtained by an UNDEVELOP* operator belong to the same level of abstraction.

III.3. Conclusion

The DEVELOP and UNDEVELOP operators are the core of graph manipulations. They take full advantage of the abstraction defined in the graph data model. They introduce the notion of multi-scale network from a logical point of view.
They are very similar to the NEST and UNEST operators in the algebra for complex objects [18]. Let R be a relation in Non First Normal Form. Let G be a graph structured with the graph data model. Let G_0 be the graph containing the nodes (and/or the edges) to develop. Let G' be the graph G after applying DEVELOP (G_0, {G}). Figure III.6 presents the properties.

UNNEST (NEST (R)) = R
NEST (UNNEST (R)) # R

DEVELOP (G_0, UNDEVELOP ({G'})) = {G'}
UNDEVELOP (DEVELOP (G_0, {G})) = {G} (from the graph data model point of view)
UNDEVELOP (DEVELOP (G_0, {G})) # {G} (with the introduction of alphanumerical data)

Figure III.6 - Properties of DEVELOP and UNDEVELOP operators

The DEVELOP operator may lead to deletion of Master_edges. The UNDEVELOP operator is able to reconstruct these edges. Unfortunately, alphanumerical data associated with these edges are no longer available. So, from the graph data model point of view, the topology of the graph has been re-built. From the alphanumerical data point of view, information has been lost (as it is for the NEST/UNNEST configuration).

IV. ELEMENTARY OPERATORS

The elementary operators correspond to the second level operators. They define operations on graphs and sub-graphs. They are similar to graph theory operators. The notion of abstraction introduces a distinction between nodes and Master_nodes (respectively edges and Master_edges). Therefore, elementary operators must take into account this new properties. Graph theory operators cannot be directly applied. Four operators are defined: the UNION, CONCATENATION, SELECTION and DIFFERENCE operators.

IV.1. UNION operator

The semantics of the UNION operator is the same as the semantics of the union operator in graph theory. The main difference is introduced by the notion of abstraction. A union between two graphs must take into account Associated_networks defined for Master_nodes and/or Master_edges. The union between a node and a Master_node modelling the same object is defined as the Master_node. The union between an edge and a Master_edge modelling the same object is defined as the Master_edge. The union between two Master_edges provides a unique Associated_network since the initial and end nodes of the two Master_edges belong to the Associated_network. The union is recursively applied on sub-networks as soon as the two graphs have a common Master_node and/or Master_edge.
This operator has two arguments. The first and the second arguments are sets of graphs. The result is a set of graphs. Figure IV.1 presents the signature of this operator.

$$\cup\ (\{G_i\},\ \{G'_i\}\ ) : \{G''_i\}$$

where G_i : is a graph $(i = 1, ..., n)$

G'_i : is a graph $(i = 1, ..., p)$

G''_i : is a graph after the application of the UNION on graph G_i and G'_i $(i = 1,..., n \times p)$

Figure IV.1 - The signature of the UNION operator

Let n be the number of graphs in the first argument. Let p be the number of graphs in the second argument. The number of graphs in the result is n times p (i.e., a graph for each element of the cross product).

IV.2. CONCATENATION operator

The CONCATENATION operator is very similar to the UNION operator. To provide a result, the CONCATENATION operator requires a common object in the two graphs. The notion of abstraction introduces several configurations. Two graphs may have a common node without having common edges. Two graphs may have a common edge without having common nodes (i.e., the cross product of Master_node and node as initial and end nodes). Two graphs may have no common node and no common edge but a common object (i.e., an edge defined between two nodes and a Master_edge defined between two Master_nodes representing the same objects as the edges and the nodes do). The concatenation is recursively applied on sub-networks.
Whenever two graphs have no common object, the UNION operator provides as a result, two disjoint sub-graphs in the same graph. The CONCATENATION operator provides an empty graph in such a configuration.
The CONCATENATION operator has two arguments. The first and the second arguments are sets of graphs. The result is a set of graphs. Figure IV.2 presents the signature of this operator.

$$\oplus\ (\{G_i\},\ \{G'_i\}\ ) : \{G''_i\}$$

where G_i : is a graph $(i = 1, ..., n)$

G'_i : is a graph $(i = 1, ..., p)$

G''_i : is a graph after the application of the CONCATENATION on graphs G_i and G'_i

$(i = 1, ..., k \quad / \quad k \leq n \times p)$

Figure IV.2 - The signature of the CONCATENATION operator

Let n be the number of graphs in the first argument. Let p be the number of graphs in the second argument. The number of graphs in the result may be less than n times p since a graph of the first argument may have no common object with any other graph of the second argument.

IV.3. SELECTION operator

A labelling function applied to nodes, edges and networks allows to define alphanumeric properties. The semantics of the SELECTION operator is to determine relevant nodes, edges or

networks according to selection criteria. As an example "A town having a number of inhabitant greater than 100,000" corresponds to a selection criterion associated with a node; "A cost less than 100 units" corresponds to a selection criterion associated with an edge; "Inter-city railway lines" corresponds to a selection criterion associated with a network.
The semantics of the SELECTION operator is similar to the semantics of the selection operator in graph theory. The main difference is introduced by the notion of abstraction. The selection criteria are recursively applied to specialized nodes (or edges). A Master_node (or a Master_edge) may be transformed into a node (or an edge) whenever Associated_networks become empty (i.e., no node and no edge are relevant according to the selection criteria). A Master_node may be removed and replaced by the result of the DEVELOP operator whenever it does not respect the selection criteria and at least one node of an Associated_network respects the selection criteria. A Master_edge may be transformed into an edge whenever at least one edge of its Associated_network does not respect the selection criteria. Its Associated_network is merged (i.e., the DEVELOP operator and the UNION operator) into the studied graph.
The SELECTION operator applied on a graph provides a sub-graph from this graph. This operator has two arguments. The first argument is the selection criteria. We do not consider here the modelling of such criteria [15]. The second argument is a set of graphs on which the selection criteria are applied. The result of the SELECTION operator is a set of graphs.
Figure IV.3 presents the signature of this operator.

$$\sigma \text{ (Criteria, } \{G_i\} \text{) : } \{G'_i\}$$

where Criteria : represents the modelling of the selection criteria

G_i : is a graph $(i = 1, ..., n)$

G'_i : is a graph after the application of the SELECTION on graph G_i

$(i = 1, ..., k \quad / \quad k \leq n)$

Figure IV.3 - The signature of the SELECTION operator

Let n be the number of graphs in the second argument. The number of graphs in the result may be less than n since a graph of the second argument may have no relevant node and edge.

IV.4. DIFFERENCE operator

The semantics of the DIFFERENCE operator is the same as the semantics of the graph theory operator of difference. The DIFFERENCE operator applied between two graphs requires a common object into these two graphs (i.e., a node, Master_node, edge or Master_edge) to introduce a removal in the first graph. This operator is not symmetrical. The difference is recursively applied on sub-

networks. A Master_node (or a Master_edge) may be transformed into a node (or an edge) whenever Associated_networks become empty (i.e., they belong to the two graphs).
Whenever a Master_node (or a Master_edge) with a non-empty Associated_network has to be removed, the Associated_network is merged (i.e., with the DEVELOP and UNION operators) into the studied graph.
The DIFFERENCE operator has two arguments. The first is a set of graphs. The second argument is a graph. This graph contains nodes and edges to be removed from graphs of the first argument. The result is a set of graphs. Figure IV.4 presents the signature of this operator.

$$\#\ (\{G_i\},\ G_0\ ) : \{G'_i\}$$

where G_i : is a graph $(i = 1, ..., n)$

G_0 : is a graph

G'_i : is a graph after the application of the DIFFERENCE on graph G_i and G_0

$(i = 1, ..., k \quad / \quad k \leq n)$

Figure IV.4 - The signature of the DIFFERENCE operator

Let n be the number of graphs in the first argument. The number of graphs in the result (i.e., k) may be less than n since the graph G_0 may belong to the first argument.

IV.5. Conclusion

The merge of basic operators and elementary operators provides a tool box to manipulate graphs. This tool box is very similar to the operators to the operators defined in the relational algebra (i.e., selection, union, difference). A functional approach (i.e., a combination of operators) seems to be a relevant modelling of a query. An execution plan of a relational query is transposed here to build an execution plan of a query defined on graphs.

V. HIGH LEVEL OPERATORS

In this part, three classes of operators based on classical graph manipulations are presented. Nevertheless, they are defined in the context of the graph data model. They are designed to manage major part of GIS network-oriented queries. The PATHS operator is the evaluation of paths between an origin and a destination [1,7,8,10,11]. The INCLUSION operators are defined by an operator of extraction, the inclusion of nodes in a path and the inclusion of a path in a path. The INTERSECTION operators present a similar division: the intersection of two sets of nodes provided by two paths and the intersection of two sets of edges provided by two paths.

V.1. PATHS operator

Network management is far from being completely handled by commercial GIS products. Generally, network processing queries are solved by a specific module. Path evaluation operator (often restricted to the shortest path) is not integrated into the database operators. This weakness [9] is also present in the database query language since no extended SQL has a correct management of the path operator (even those dedicated to the transitive closure management). Defining a complete framework to express network-oriented queries is not so simple. [4] presents a set of "path" queries. These queries involve a transitive closure query with fifteen classes of interactions (such as constraints on nodes, edges, nodes and edges).

Furthermore, the definition of the result as a unique result graph is not relevant for the GIS context. The result of a path query must be a set of paths. As an example in airplane applications, the shortest way in time is generally far from being one of the less expensive (but what is the correct cost function?). The cheapest way is most part of the time far from being one of the most convenient and so on. Unfortunately, a graph defined by the union of individual solutions is not relevant as a result as soon as aggregate constraints are defined in the query (which is generally the case) [16].

The "classical" DBMS are based on the principle of the Closed World Assumption: "If a datum does not logically follow from a set of database operations, then we conclude that the negation of this datum is valid." Therefore, the path operator provides an answer only if the complete (from the initial node to the end node) path exists. This option is coherent in the case of a flat network. The graph data model introduces several levels of definition. Therefore, we introduce the principle of the Open Graph Assumption: "While evaluating a path operator at a level of abstraction i, if a datum does not logically follow from a set of graph data model operations, then the same evaluation is performed at level i+1." The recursive application of this mechanism provides different levels of abstraction in a result. Level i+1 is always available since the path is first evaluated at the upper level of abstraction. The Open Graph Assumption introduces the notion of a global direction since the level i+1 is a more general edge. Therefore, the path operator (->) presents two options: the exact operator and the approximated operator. The exact operator is the equivalent of a classical path operator. It requires the existence of a complete path. The approximated operator takes full advantage of (1) the node hierarchy and (2) the network topology to provide a multi-level solution.

Figure V.1 presents the signature of this operator.

-> (G, G', Criteria) : {G''$_i$}
where G : is a graph (N,E) such as E = ∅ and N is reduced to a node modelling the origin
G' : is a graph (N,E) such as E = ∅ and N is reduced to a node modelling the destination
Criteria : the path criteria and constraints (modelled as defined in [15])
G''$_i$: is a graph after the application of the PATHS operator on graph G and G'

Figure V.1 - The signature of the PATHS operator.

The number of paths in the result is undefined. For the time, we consider paths defined between a unique origin and destination. Nevertheless, these nodes may be obtained by an application of elementary and basic operators.

V.2. INCLUSION operators

High level operators of inclusion provide three kinds of results. These distinctions are linked to the construction of a graph (i.e., a set of nodes and a set of edges). Π_{e_ne} represents the operator for the extraction of nodes in a path; Π_{ic_n} represents the set operator for the inclusion of nodes; and Π_{ic_e} represents the set operator for the inclusion of edges.

V.2.1. EXTRACTION of nodes from a path

The Π_{e_ne} operator is applied to a set of graphs. The result is a set of graphs. This operator provides for each graph the set of nodes involved in the edge definition (i.e., there is no isolated node). Figure V.2 presents the signature of this operator.

$$\Pi_{e\_ne}\ (\{G_i\}\ ) : \{G'_i\}$$

where G_i : is a graph $(i = 1, ..., n)$

G'_i : is a graph, $E = \varnothing$, after the application of the Π_{e_ne} on graph G_i $(i = 1,...,k / k \leq n)$

Figure V.2 - The signature of the Π_{e_ne} operator.

Let n be the number of graphs in the argument. The number of graphs in the result may be less than n since a graph in the argument may have only isolated nodes.

V.2.2. INCLUSION of nodes

The Π_{ic_n} operator is applied to two graphs. The nodes of these graphs are involved in this operator. The result is a graph. This operator is not symmetrical. It requires the first set of nodes to be included into the second set of nodes. The result of this operator is the first set of nodes whenever it fulfills the requirements. Figure V.3 presents the signature of this operator.

$$\Pi_{ic\_n}\ (G, G'\ ) : G''$$

where G : is a graph G(N,E) such as $E = \varnothing$

G' : is a graph G(N,E) such as $E = \varnothing$

G" : G or $G\varnothing$

Figure V.3 - The signature of the Π_{ic_n} operator.

V.2.3. INCLUSION of edges

The Π_{ic_e} operator is applied to two graphs. The edges of these graphs are involved in this operator. The result is a graph. This operator is not symmetrical. It requires the first set of edges to be included into the second set of edges. The result of this operator is the first set of edges whenever it fulfills the requirements. Figure V.4 presents the signature of this operator.

	Π_{ic_e} (G, G') : G"
where	G : is a graph G(N,E)
	G' : is a graph G(N,E)
	G" : G or G$\varnothing$

Figure V.4 - The signature of the Π_{ic_e} operator.

V.3. INTERSECTION operators

High level operators of intersection provide two kinds of results. This dichotomy is linked to the construction of a graph (i.e., a set of nodes and a set of edges). Π_{is_n} represents the operator for the set intersection of nodes and Π_{is_e} represents the operator for the set intersection of edges.

V.3.1. INTERSECTION of nodes

The Π_{is_n} operator is applied to two graphs. The nodes of these graphs are involved in this operator. The result is a graph. The intersection operator is recursively applied to the sub-networks. A Master_node may be transformed into a node whenever Associated_networks become empty. The intersection between a node and a Master_node representing the same object is defined as the node. Figure V.5 presents the signature of this operator.

	Π_{is_n} (G, G') : G"
where	G : is a graph G(N,E) such as E = $\varnothing$
	G' : is a graph G(N,E) such as E = $\varnothing$
	G" : is a graph after the application of the Π_{is_n} on graph G and G' such as E = $\varnothing$

Figure V.5 - The signature of the Π_{is_n} operator.

V.3.2. INTERSECTION of edges

The Π_{is_e} operator is applied to two sets of edges. The intersection operator is recursively applied to the sub-networks. A Master_edge may be transformed into an edge whenever the Associated_network becomes empty. The intersection between an edge and a Master_edge representing the same object is defined as the edge.
Figure V.6 presents the signature of this operator.

	Π_{is_e} (G, G') : G"
where	G : is a graph
	G' : is a graph
	G" : is a graph after the application of the Π_{is_e} on graph G and G'

Figure V.6 - The signature of the Π_{is_e} operator.

V.4. Conclusion

The main difference between graph theory and GIS network-oriented operators is based on the cost evaluation function (i.e., deciding whether a node or an edge is relevant). To simplify, the time to evaluate a cost function of a node (or an edge) is considered as negligible in graph theory during the detection of a path from one node to another. Unfortunately in a GIS context, these functions are far from being negligible since they involve several database operations. GIS queries are more interested in the properties defined on the nodes or on the edges than on the topology of an answer. Nevertheless, the notion of abstraction and the notion of global direction are very important. The graph data model provides the topology of a graph. This assumption is based on the fact that topologies of geographical networks (i.e., railway lines, rivers, roads) are not modified very frequently (although alphanumerical data may be frequently updated). A pre-compiled structure becomes realistic. The data model takes full advantage of this condition. The PATHS operator offers therefore a higher expressive power (i.e., abstraction and direction).

VI. CONCLUSION

Geographical Information Systems are now widely used for several applications (i.e., transport analysis, urban planning). Network analysis operators are not directly integrated into the database system (i.e., an Extended SQL statement to manage path operators). This weakness is due to the lack of common data structures between thematically-oriented data (i.e., town, forest) and network-oriented data (i.e., railways, electricity).

In this paper, we present the operators defined on the graph data model allowing the notion of abstraction. We define three levels of operators. Basic level operators, the DEVELOP and UNDEVELOP operators, are directly linked to the concept of abstraction present in the data model. The DEVELOP operator provides more details for a graph. The converse operation, the UNDEVELOP operator, reduces the data set. Elementary operators manage the notion of graphs and sub-graphs. The SELECTION, UNION, CONCATENATION, DIFFERENCE operators, are similar to the basic operators of graph theory (i.e., extended to a data model allowing abstraction for nodes and edges). High level operators, the PATHS, INCLUSIONS and INTERSECTIONS operators, correspond to the most common classes of queries addressed to a GIS.

The high levels operators introduce a new conception in graph operators for GIS since the result is defined with several levels of abstraction. This notion changes the visualization of a result. Next work is to define a formal Data Definition Language for the graph data model. The notion of view can now be integrated since the data manipulation operators are available.

References

[1] Becker RA, Eick SG, Miller EO, Wilks AR: Network Visualization, 4th International Symposium on Spatial Data Handling, Zurich, Switzerland, 23-27 July 1990, Colombus OH:International Geographical Union

[2] Berge C. : Graphs and Hypergraphs, North Holland, Amsterdam, 1973

[3] Borgida A, Myropoulos J, Wong HK, Generalization/Specialization as a Basis for Software Specification, Brodie, Myropoulos, Schmidt Eds "On Conceptual Modelling perspectives from Artificial Intelligence, Database and Programming Languages, Springer Verlag, 1984

[4] Boursier P., Mainguenaud M.: Spatial Query Languages: Extended SQL vs. Visual Languages vs. Hypermaps, 5th International Symposium on Spatial Data Handling, Charleston, SC, USA, 3-7 August 1992, Colombus OH:International Geographical Union

[5] Cruz IF, Mendelzon AO, Wood PT: A Graphical Query Language Supporting Recursion, ACM SIGMOD Conference, San-Fransisco, USA, May 1987

[6] David B., Raynal L., Schorter G., Mansart V. : GéO2: Why objects in a geographical DBMS?, Advances in Spatial databases, D. Abel, BC Ooi (Eds), Lecture Notes in Computer Science n°692

[7] Dayal U. et al : PROBE - A research Project in Knowledge-Oriented Database Systems: Preliminary Analysis, Technical Report CCA-85-03, July 1985

[8] Eck (van) JR, De Jong T.: Adapting Datastructures and Algorithms for Faster Transport Network Computations, 4th International Spatial Data Handling Symposium, Zurich, Switzerland, July 24-27 1990, Colombus OH:International Geographical Union

[9] Garey MR, Johnson DS: Computers and Intractability: A Guide to the Theory of NP-Completeness, WH Freeman and Co Eds, New York, 1979

[10] Güting RH: Extending a Spatial Database System by Graphs and Object Class Hierarchies, International Workshop on Database Management System for Geographical Applications, Capri, Italy, May 1991

[11] Haas LM, Cody WF: Exploiting Extensible DBMS in Integrated Geographical Information Systems, 2nd Symposium on Large Spatial Databases, Zurich, Switzerland, August 1991, Lecture Notes in Computer Science n° 525, Springer Verlag

[12] Heres L., Lahaije P. , Claussen H., Lichtner W., Sielbold J.: GDF A porposed Standard for Digital Road Maps to be Used in Car Navigation Systems, Technical paper, Philips CE/Cis-Lab, Eindhoven, 1993

[13] Hull R, King R: Semantic Database Modelling: Surveys, Applications and Research Issues, ACM Computing Surveys, Vol 19, September 1987

[14] Mainguenaud M, Simatic XT: A Data Model to Deal with Multi-scaled Networks, Computer, Environment and Urban Systems, Vol 16, p 281-288, 1992

[15] Mainguenaud M.: From the User Interface to the Data Base Management System: Application to a GIS, 5th International Conference on Human Computer Interaction, Orlando, USA, Aug. 1993

[16] Mainguenaud M.: The Results of GIS Queries, IEEE/CS Visual Languages'93, Bergen, Norway, Aug. 1993

[17] Smith JM, Smith DC: Database Abstraction: Aggregation and Generalization, ACM Transaction On Database System, Vol 2, n°2, 1987

[18] Ullman JD: Principles of Database and Knowledge-base Systems, Computer Science Press,1988