GROG : Geographical Queries using Graphs

MAINGUENAUD Michel
Institut National des Télécommunications
9 rue Charles FOURIER
91011 EVRY

33 1 60 76 40 40
email : MAINGUENAUD@FRINT51.BITNET

Abstract :

This paper describes a graphical Data Manipulation Language (DML), GROG. The central notion of GROG is that it allows recursive queries to be expressed very easily. GROG is a DML for a Geographical Information System (GIS).

This language is said to be graph-oriented by the way data are modelised, queries and the results of a query are expressed .

Keywords : Geographical Information System, Data Manipulation Language, Recursive Queries.

Advanced Database Symp
Kyoto Japan
7.8 Dec 89
Inf Proc. Soc. of Japan

GROG : Geographical Queries using Graphs

MAINGUENAUD Michel
Institut National des Télécommunications
9 rue Charles FOURIER
91011 EVRY
33 1 60 76 40 40
email : MAINGUENAUD@FRINT51.BITNET

1. Introduction :

In current research toward the design of more powerful Data Manipulation Language (DML), different research groups are simultaneously concentrating their work on logical-based query language. Because of the diversity of the Geographical Information System (GIS) applications [SMSE87] and users, there is a need to design a user-friendly query language.
Part of GIS applications is network management such as roads, telecommunications, railways, etc. These data can be easily represented using graphs. Manipulations of graphs are very often defined with a recursive (or logic-based) formalism. Starting with a graphical query language supporting recursion defined in [CMW87] we propose a DML for GIS applications relying on network management. Section 2 presents various queries adressed to such a GIS. Section 3 presents the syntax of semantics of [CMW87]. Section 4 presents the limitations of [CMW87]. Section 5 presents the extensions [M89]. Section 6 has the conclusions and discussion of futher work.

2. Network-oriented queries :

This section presents various typical queries adressed for example to a (road) Network Information Management System :

Query 1 : What are the paths from Nice to Paris ? This query can be evaluated with a graph traversal operator.
Query 2 : What are the paths from Nice to Paris with olny large-city stages ? This query can be evaluated with a graph traversal operator and constraints on the nodes of the graph.

Query 3 : What are the paths from Nice to Paris with AIR FRANCE (AF) company ? This query can be evaluated with a graph traversal operator and constraints on the edges of the graph.

Query 4 : What is the shortest path from Nice to Paris using motorway ? This query can be evaluated with a graph traversal operator, aggregates and constraints on the edges of the graph.

These queries can be written with Horn clauses [CGT88]. But Horn clauses are not well adapted as a user-interface : 1) Horn clauses are not a very user-friendly language for the one who is not used to recursive concepts, 2) Horn clauses can't modelise some typical queries such as :

Query 5 : What are the common parts between a path from Nice to Geneva (which costs less than 10 units) and a path from Paris to Vienna ?
Query 6 : What are the paths without any cycle from Nice to Paris ?

Therefore Network Information Management Systems need a user-friendly Data Manipulation Language which allows recursive queries to be expressed.

3. Starting point of GROG [CMW87]:

Data are modelised by a directed graph. A directed graph G is represented by G(N, E). N is a set of nodes, E is a set of edges between two elements of N. Nodes and edges are labeled. All the graphs which are used here will be multi-graphs : two edges with different labels can occur between two given nodes (to simply multi-graphs will be called graphs).

Definitions :

A graph G is defined by G (N_G, E_G, ψ_G, ν_G, ε_G) :

. N_G is a set of nodes $N_G = \{n_1, \ldots, n_p\}$

. E_G is a set of edges $E_G = \{e_1, \ldots, e_q\}$

. ψ_G is an incident function
$\psi_G : E_G \rightarrow N_G \times N_G$

. ν_G is a node labeling function :
$\nu_G : N_G \rightarrow I_n$ with $I_n = D_{n0} \times \ldots \times D_{nn}$
i.e. : Name x Population x Museum.

. ε_G is an edge labeling function :

ε_G : E_G -> I_e with $I_e = D_{e0}$ x ... x D_{ee}

i.e. : Company x Departure_time x Arrival_time

A set X $\{x_1, \ldots\}$ of variables and sets D_{ij} $\{d_{ij1}, \ldots\}$ (i = e, n) of constants for all the domains of the labeling functions are defined. A hyphen (-) will be interpreted in the labeling function as any value of domain D_{ij}.

Properties : A graph G is such as :

. $\forall n \in N_G \quad \exists e \in E_G / \psi_G(e) = (n, a) \vee \psi_G(e) = (a, n) / a \in N_G$
(There is no isolated nodes).

. $\forall e_i \in E_G, e_j \in E_G \; \psi(e_i) = \psi(e_j) \Rightarrow \varepsilon(e_i) \neq \varepsilon(e_j)$
(two edges with the same extremities have different labels)

Queries :

A graphical query Q defined on a graph G is a set of labeled and oriented graphs $\{Q_1, \ldots, Q_p\}$. The labels of the nodes can be variables, or constants.

Graph Q $(N_Q, E_Q, \psi_Q, \nu_Q, \varepsilon_Q)$ is such as :

$\forall n \in N_Q$, $\nu_Q(n) \in D_{n0} \cup X$ x D_{n1}x...x D_{nn} (D_{n0} : set of node-constants)

$\forall e \in E_Q$, $\varepsilon_Q(e) \in D_{e0} \cup X$ x D_{e1} x...x D_{ee} (D_{e0} : set of edge-constants)

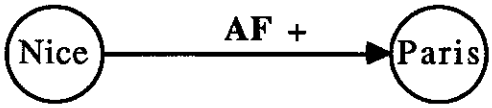
Edge-labeling :

A simple label is a tuple of constants ($\in D_{ij}$), variables or hyphens. (CMW87) generalises labels with a regular expression. These expressions are recursively defined as :

. A label $l \in I_e$, is such that l is a regular expression
. If S_1 and S_2 are regular expressions then $S_1 \mid S_2$ (S_1 or S_2) is a regular expression.
. If S_1 and S_2 are regular expressions then $<S_1, S_2>$ (S_1 then S_2) is a regular expression
. If S is a regular expression S^+ (Transitive Closure) is a regular expression.

Labeling examples : AF AF+ <AI, AF>+ <AI | AF>+ _+ ....

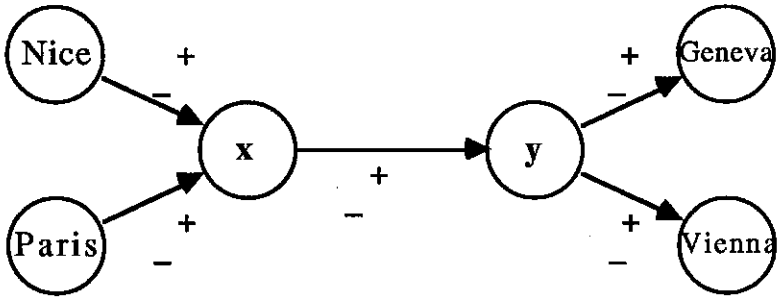
Query 3 can be expressed (with $I_e = De0$) by :



4. Limitations of [CMW87] :

Part of the limitations can be found in [CMW87]. To sum up the main limitations are : 1) type of recursive queries which can be expressed, 2) path management (aggregates, cycles, intersections, etc) 3) conditional expressions, 4) definition of the results of a query. These limitations are studied in detail in [M89].

This section olny presents the limitations of the intersection-typed queries. Query 5 cannot be expressed by the following graph (query) :



The reasons are :

1) It does not allow the whole path Nice-Geneva (resp. Paris-Vienna) to be part of the Paris-Vienna (resp. Nice-Geneva) path.
2) It does not allow to define any aggregate on the path Nice-Geneva.
3) Defining the result graph as the union of differents graphs leads to wrong results in case of aggregate in a multi-graph approach.
4) Defining the result graph to be an edge-independent subgraph homeomorphism between a query-graph Q and a data-graph G gives partial answers for variables x and y (out of the edge-matching between graph G and graph Q)

5. Extensions :

Limitations presented in section 4 lead us to define some extensions. The main extensions presented here are : 1) the manipulation of graph hierarchies, 2) the definition of the variables for path management 3) the definition of new edges 4) the definition of a "global query".
Let D be the set of convinient attributes on graphs, nodes, edges, let d_t (subset of D) be the set of attributes defined on type t (graph, node, edge).
Let ϑ be an alphabet to define names of variables.

5.1 Hierarchies of graphs :

To manipulate a graph G (N, E), Γ a graph labeling function is defined :

$$\Gamma \quad : \quad G(N,E) \rightarrow \quad I_g \quad \text{with } I_g = D_{g0} \times \ldots \times D_g$$

Let G_1 (N_{G1}, E_{G1}, ψ_{G1}, ν_{G1}, ε_{G1}) and G_2 (N_{G2}, E_{G2}, ψ_{G2}, ν_{G2}, ε_{G2}) be two different graphs :

a) $\Pi_L : N \times N \rightarrow E_0$
$\Pi_L (n_1, n_2) = \{e_{Lk} / \Psi_L (e_{Lk}) = (n_1, n_2)\ n_1 \in N_{G1} \wedge n_2 \in N_{G2}\}$
$E_0 = \{e_{Lk} / \Psi_L(e_{Lk}) = (ni, nj)\ ni \in Ni, nj \in Nj \wedge Ni <> Nj\}$

b) $\Psi_L : E_0 \rightarrow N \times N$
$\Psi_L (e_{Lk}) = (n_1, n_2)$ such as $n_1 \in N_{G1} \wedge n_2 \in N_{G2}$

c) $\varepsilon_L : E_0 \rightarrow I_L$ such as $I_L = D_{L0} \times \ldots \times D_s$
$\varepsilon_L (e_{Lk}) = (i_{L0}, \ldots, i_{Ls})$

After extension of node n_i ($n_i \in N_1$) with the graph G_2 (N_2, E_2), the result graph G' is defined by G' ($N_{G'}$, $E_{G'}$, $\psi_{G'}$, $\nu_{G'}$, $\varepsilon_{G'}$) :

$$E_L = \{ e_k \in E_0 / \quad \exists n_1, n_2 , n_1 \in N_1 \wedge n_2 \in N_2 \wedge e_k \in \Pi_L (n_1, n_2) \cup \Pi_L (n_2, n_1)\}$$

$N_{G'}$: $N_1 \cup N_2$

$E_{G'}$: $E_1 \cup E_2 \cup E_L$

$\Psi_{G'}$: $\Psi_{G'} (e_k) = \Psi_1 (e_k)$ if $e_k \in E_1$
$\Psi_{G'} (e_k) = \Psi_2 (e_k)$ if $e_k \in E_2$
$\Psi_{G'} (e_k) = \Psi_L (e_k)$ if $e_k \in E_L$

$\nu_{G'}$: $\nu_{G'}(n_j) = \nu_1(n_j)$ if $n_j \in N_1$
$\nu_{G'}(n_j) = \nu_2(n_j)$ if $n_j \in N_2$

$\varepsilon_{G'}$: $\varepsilon_{G'}(e_k) = \varepsilon_1(e_k)$ if $e_k \in E_1$
$\varepsilon_{G'}(e_k) = \varepsilon_2(e_k)$ if $e_k \in E_2$
$\varepsilon_{G'}(e_k) = \varepsilon_L(e_k)$ if $e_k \in E_L$

5.2 Extension of variables :

Graph-typed variable (called g_i):

γ : ϑ -> $G(N, E, \Psi_G, \nu_G, \varepsilon_G)$
g_i -> $G_i(N_{Gi}, E_{Gi}, \Psi_{Gi}, \nu_{Gi}, \varepsilon_{Gi})$

A graph can be represented by a set of $[d_g, \{n_i, n_j, d_e\}]$. d_g is a set of data on the graph (history, ...). $\{n_i, n_j, d_e\}$ represents the set of edges.

Derived variables :

Node-typed variable (called v_i) :

ν' : ϑ -> $D_g \times N \cup X$ in fact $D_g \times N \cup X \times \emptyset \times \emptyset$
v_k -> $\{d_g, n_q\}$ $\{d_g, \{(n, \perp, \perp)\}\}$

where $\perp$ represents the undefined symbol
(representation of an "node-edge")

A node-edge is an edge with a unique known extremity (initial one). It does not exist any data about this edge. Function ε applied to this edge will respond the undefined symbol $\perp$. A node-typed variable is a set of graphs. These graphs can be represented by : $\{d_g, \{(n, \perp, \perp)\}\}$.

Path-typed variable (called V_i) :

ε' : ϑ -> $D_g \times N \cup X \times N \cup X \times D_e$
V_{ij} -> $(d_g, \{(n_1, n_q, d_{e1}), \ldots, (n_m, n_2, d_{em})\})$
V_i = $\{V_{ij}\}$

A user-hidden variable V_{ij} represents a unique path. Data are defined on this variable such as length or aggregates on the edges of the path (a path from n_1 to n_2 is a set defined as : $\{(n_1, n_q, d_{e1}), \ldots, (n_m, n_2, d_{em})\}$). A path-typed variable is a set of variables V_{ij}.

These variables allow to extend the definition of the language allowing :

1) to take into account not olny ε (e_i) where e_i is an edge of a path but the whole path (ε').
2) to define aggregate functions (such as Min, Length, Count, etc) defined on path-typed variables or/and recursively on these functions (i.e. : Min (Length (V_1))), and classical aggregate-typed variables to memorise the value of an aggregate.

5.3 New edges :

The homeomorphism defined in [CMW87] has to be changed for the language to be able to take into account aggregates. The constraint on the paths such as they have to be disjoint node-to-node has to desappear. An answer is now a set of graphs $\{R_i\}$ and each element of this set is a result graph (a possible answer to the query). This graph has not to be connexe. The final answer is a logical OR of all the result graphs. But the union of all the result graphs is not a result graph (out of the aggregate functions).

Result graph :

A result graph R (graph R_i) is defined as a graph G (see section 3)

$$R\ (N_R, E_R, \psi_R, \nu_R, \varepsilon_R)$$

but the constraint :

. $\forall\ n \in N_R \quad \exists\ e \in E_R / \psi_R\ (e) = (n, a) \vee \psi_R\ (e) = (a, n)$ où $a \in N_R$
(there is no isolated node).

desappears. An answer can be an intersection such as a node-edge $(n, \perp)$.

The constraint becomes :

. $\forall\ n \in N_R \quad \exists\ e \in E_R / \psi_R\ (e) = (n, a) \vee \psi_R\ (e) = (b, n)$ with $a \in N_R \cup \{\perp\}$, $b \in N_R$

Edges :

Three types of edges are available :

Link Edge 

direct/transitive

Inclusion Edge 

Intersection Edge 

General properties :

These edges follow the same rules : 1) these edges are oriented, 2) these edges represent the results of sub-queries, 3) these edges are a binary operator (one initial point, one end point), 4) these edges represent paths without cycle.

General definitions :

An edge is defined by Π_T such as : $\Pi_T : \alpha \times \beta \rightarrow \gamma$

Let α, β, γ be the type of the initial and end points of an edge.
Let T represents the type of an edge : $T \in \{D, T, Is\_e, Ic\_e, Ic\_ne, Is\_n, Ic\_n\}$

Manipulation of edges :

Link : 1)- Direct Link	Π_D
2)- Transitive Link	Π_T
3)- Intersection of paths (edges)	Π_{Is_e}
4)- Inclusion of paths (edges)	Π_{Ic_e}

Manipulation of nodes and edges :

5)- Inclusion	Π_{Ic_ne}

Manipulation of nodes :

6)- Intersection of nodes	Π_{Is_n}
7)- Inclusion of nodes	Π_{Ic_n}

Definition of the operators :

To define the operators of GROG, a formal system is thus presented : F (G, M, D_0) such as :

Let G be a graph (with the previous definitions)
Let M be a set of manipulations $M = \{M_1, ..., M_7\}$

Function Γ is defined by : $\Gamma : G \rightarrow D_{g0} \times ... \times D_g$
Function ε_G is defined by : $\varepsilon_G : E \rightarrow D_{e0} \times ... \times D_e$
Function ν_G is defined by : $\nu_G : N \rightarrow D_{n0} \times ... \times D_n$

Definitions :

Let OID_ε, OID_n, OID_Γ be bijective applications such as :

$OID_\varepsilon : E \rightarrow D_{e0}$ (therefore OID_ε^{-1} exists)
$OID_\nu : N \rightarrow D_{n0}$ (therefore OID_ν^{-1} exists)

with $D_{e0} = \{e_1, ..., e_e\}$ and $D_{n0} = \{n_1, ..., n_n\}$

Let P_{Ri} be a set of graphs $\{g_1, ..., g_g\}$ obtained by the decomposition of a graph R_i (path of a result graph R_i). Edges of graph g_i belong to a sub-set of E_{Ri} (path or non-connexe edges). g_i is a sub-graph of graph R_i. Each g_i represents an instanciation of an edge of a query-graph Q.

$OID_\Gamma : P_{Ri} \rightarrow D_{g0}$ (therefore OID_Γ^{-1} exists)

with $D_{g0} = \{g_1, ..., g_g\}$

From now let D_0 be $D_{e0} \cup D_{n0} \cup D_{g0}$

Axiom 1 : Edge identifier

OID_ε allows to associate to each edge of graph G ($e_i \in E_G$), a symbol ($e_i \in D_0$). $OID_\varepsilon(e_i) = e_i$

Axiom 2 : Node identifier

OID_V allows to associate to each node of graph G ($n_j \in N_G$), a symbol ($n_j \in D_0$). $OID_V(n_j) = n_j$

Axiom 3 : Path identifier

To each path, p, of a sub-query (an edge of graph Q) a symbol ($g_k \in D_0$) and a word, M(p), are defined. M(p) is obtained by concaténation (+ or Σ) of the symbols ($e_i \in D_0$) of the edges respecting the order within the path.

Example : Let $p = [e_1\ e_2\ e_3\ e_4]$ be a path

$$M(p) = e_1 + e_2 + e_3 + e_4 = \sum_{k=1}^{4} e_k = e_1e_2e_3e_4$$

Axiom 4 : Node-of-a-path identifier

To each node identified by ($n_j \in D_0$) belonging to a path identified by g_k ($g_k \in D_0$) is defined a couple of symbols (g_k, n_j)

$$OID_{\Gamma V} : P_{Ri} \times N_{Ri} \rightarrow D_{g0} \times D_{n0}$$

Axiom 5 : Definition of a word and sub-word

Let *a* and *b* be two words of the paths p_1, p_2 and e_k be the symbols of the edges of the paths.

$a = M(p_1) = e_i + ... + e_j$
$b = M(p_2) = e_1 + ... + e_p$

a is a sub-word of *b* $\Leftrightarrow$ $b = e_1 + ... + e_i + ... + e_j + ... + e_p$
(written $a \angle b$)

All the definitions here will refer to a unique result-graph R_i. $P_{Ri} = \{g_i\}$ represents the instanciations of each edge of the graph Q. Without lost of generality, in the following we olny consider a graph g_i (the instanciation of an edge of the graph Q). P_i will refer to an edge, NP_i will refer to a set of nodes for a path (the elements of NP_i look like (g_k, n_j) and all g_k are equal).

Definitions for edge-manipulations :

To each edge definition α, β, and γ will take their value in $\{\nu', \varepsilon'\}$. They represent the starting-point, the end-point and the result of a query. An example of a query will be given for each definitions.

Direct Link edge : Edge between two nodes n_1 and n_2

Π_D and Π_T are obtained using $\alpha = \beta = \nu'$ and $\gamma = \varepsilon'$

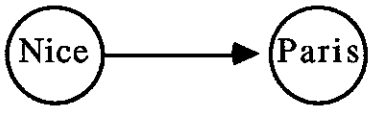
Direct :

The goal is to find the nodes j such as j is a successor of a node i. ({i and j / $j \in \Gamma^+(i)$})

Π_D : $(D_g \times (N \cup \vartheta)) \times (D_g \times (N \cup \vartheta)) \rightarrow D_g \times N \times N \times D_e$

Example :

Give all the one-edge paths (direct links) between Nice and Paris



Manipulation 1 :

After $\Pi_D(ni, nj)$:

Π_D (ni, nj) $\leftarrow$ {p / M(p) = ek $\wedge$ Ψ_G (OID_ε^{-1} (ek)) = (ni, nj)}

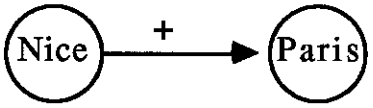
(p is a one-edge-path)

Transitive Closure form :

The goal is to find all nodes j such as there exists at least one path between i and j ($\{i, j / j \in \Gamma^+(i) \vee \exists$ a path from i to j$\}$)

Π_T : $(D_g \times (N \cup \vartheta)) \times (D_g \times (N \cup \vartheta)) \rightarrow D_g \times N \times N \times D_e$

Example : Query 1 is defined by



Manipulation 2 :

After $\Pi_T(ni, nj)$:

$\Pi_T (ni, nj) \leftarrow \{g / M(g) = e1 + ... + ek \wedge \Psi_G (OID_{\varepsilon^{-1}} (e1)) = (ni, *) \wedge \Psi_G (OID_{\varepsilon^{-1}} (ek)) = (*, nj)\}$

(g is a one-path-graph)

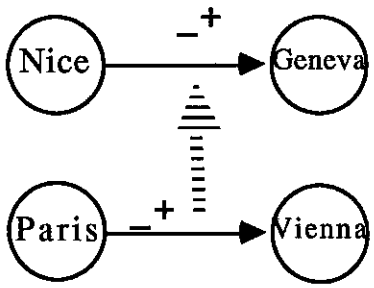
Intersection edge : The goal is to find the common parts between two paths

Π_{Is_e} is obtained using $\alpha = \beta = \gamma = \varepsilon'$

Π_{Is_e} : $(D_g \times N \times N \times D_e) \times (D_g \times N \times N \times D_e) \rightarrow D_g \times N \times N \times D_e$

Even if the intersection is a symetric manipulation, the edge is still oriented to be as close as possible to the natural language (which requires an order in a sentence).

Example : Query 5 (without aggregates) is defined by



Manipulation 3: :

After Π_{is_e} (P_1, P_2)

1 / P_1 is unchanged

2 / P_2 is unchanged

3 / Π_{is_e} (P_1, P_2) $\leftarrow$ P_3

$$P_3 = \{ p / M(p) \angle M(P_1) \wedge M(p) \angle M(P_2) \}$$

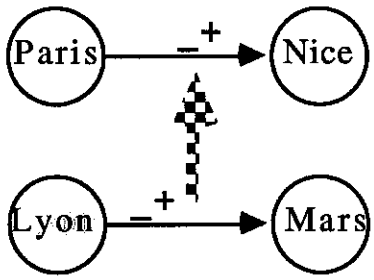
In this example, P_1 represents for a result graph, R_i, a path from Paris to Vienna, P_2 a path from Nice to Geneva and P_3 the set of sub-paths (p) which belong to the path P_1 ($M(p) \angle M(P_1)$) and to the path P_2 ($M(p) \angle M(P_2)$). P_3 is a set of graph (sub-path). Each graph may not be connexe and represents the intersection of two given paths. The notion of set is defined for P_3 out of the fact that the answer may not be connexe.

Inclusion Edge : The goal is to find a path which is contained in an other path.

Π_{Ic_e} is obtained using $\alpha = \beta = \gamma = \varepsilon'$

Π_{Ic_e} : $(D_g \times N \times N \times D_e) \times (D_g \times N \times N \times D_e) \rightarrow D_g \times N \times N \times D_e$

Example : What are the paths from Paris to Nice and from Lyon to Marseille such as the path Lyon-Marseille is included in the path from Paris to Nice ?



Manipulation 4 :

After Π_{ic_e} (P_1, P_2) :

1 / $P_1 \leftarrow$ $\{p_1 / p_1$ path (of P_1), p_2 path (of P_2) $M(p_1) \angle M(p_2)\}$

2 / $P_2 \leftarrow$ $\{p_2 / p_2$ path (of P_2), p_1 path (of P_1) $M(p_1) \angle M(p_2)\}$

3 / $\Pi_{ic\_e}(P_1, P_2) \leftarrow P_1$

Let PN be the word of a path from Paris to Nice and LM be the word of the path from Lyon to Marseille :

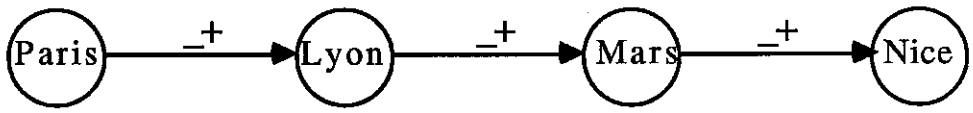
(1) The edge defined with the nodes Lyon and Marseille symbolises a path p_1 from Lyon to Marseille ($M(p_1)$ = LM). This path is included in PN ($M(p_2)$ = PN)(with the definition of the inclusion $M(p_1) \angle M(p_2)$).

(2) The edge defined with the nodes Nice and Paris (p_2) symbolises a path p_2 from Paris to Nice such as the word (path) containts a sub-word (path) LM (with the definition of the inclusion $M(p_1) \angle M(p_2)$) after application of point (1).

(3) The edge defined with point (2) and point (1) symbolises the same path as in point (1).

Cycles :

In the previous query the cycles are forbidden (by definition of the edges). But the query defined here which may seem similar authorizes cycles to appear :



Each edge is a path without cycle (by definition), but there could exist a node in common between a path Paris-Lyon and Marseille-Nice.

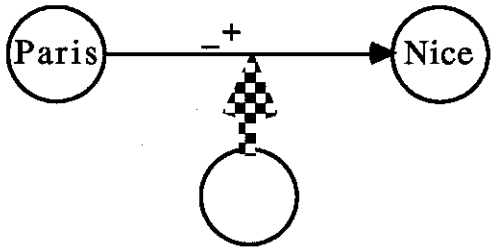
Definitions for edges and nodes manipulations :

Inclusion Edge : The goal is to find a node (or a set of node) which belongs to a path.

Π_{Ic_ne} is obtained using $\alpha = \gamma = \nu'$ and $\beta = \varepsilon'$

Π_{Ic_ne} : $(D_g \times (N \cup \vartheta)) \times (D_g \times N \times N \times D_e)$ -> $D_g \times N$

Example : What are the stage-cities of a path Paris-Nice ?



Manipulation 5:

After Π_{ic_ne} (N_1, P_1) :

1 / $N_1 \leftarrow$ {ni / OID_v(ni) is the origin or the extremity of an edge ek such as ek $\angle$ M(P_1) (Ψ (OID_{ε}^{-1} (ek)) = (ni, *) $\vee$ Ψ (OID_{ε}^{-1} (ek)) = (*, ni))}

2 / $P_1 \leftarrow$ {p_1} if $N_1 \neq \varnothing$
$\varnothing$ otherwise

3 / Π_{is_ne} (N_1, P_1) $\leftarrow$ N_1

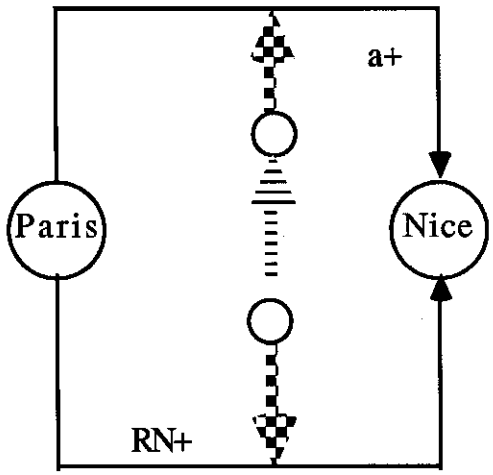
Definitions for node-manipulations :

Intersection edge : The goal is to find nodes which belong to two paths

Π_{Is_n} is obtained using $\alpha = \beta = \gamma = \nu'$

Π_{Is_n} : $(D_g \times (N \cup \vartheta)) \times (D_g \times (N \cup \vartheta)) \rightarrow D_g \times N$

Example : What are the common stage-cities of two paths Nice-Paris using tool-roads (a) and using major-roads (RN) ?



Manipulation 6:

After Π_{is_n} (NP1, NP2) :

1 / NP1 is unchanged

2 / NP2 is unchanged

3 / Π_{is_n} (NP1, NP2) $\leftarrow$ {ni / ni $\in$ NP1 $\wedge$ ni $\in$ NP2}

In this example, intersection edge represents the set of nodes (ni) which are stage-cities of two given paths Paris-Nice.

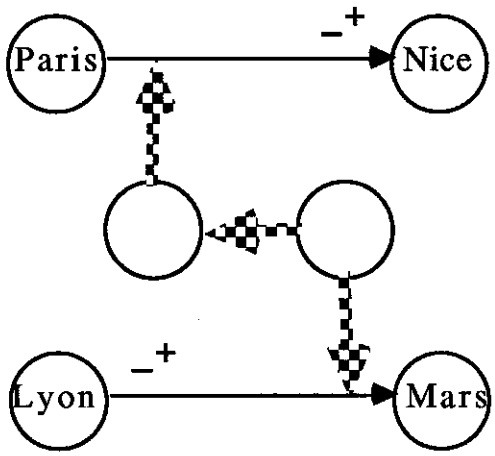
Inclusion Edge : The goal is to find the set of nodes included in an other set of nodes.

Π_{Ic_n} is obtained using $\alpha = \beta = \gamma = \nu'$

Π_{Ic_n} : $(D_g \times \vartheta) \times (D_g \times \vartheta) \rightarrow D_g \times N$

Example :

What is the set of stage-cities of a path Lyon-Marseille such as each of them is also a stage-citie of a path Paris-Nice ?



Manipulation 7:

After Π_{ic_n} (NP1, NP2) :

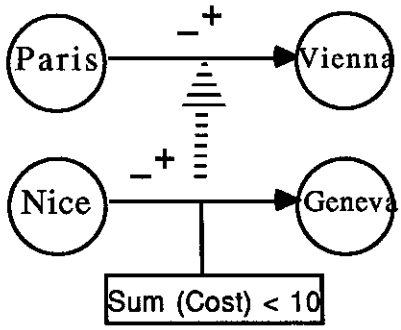
1 / NP1 ← {ni ∈ NP1 / ni ∈ NP2}

2 / NP2 is unchanged

3 / Π_{ic_n} (NP1, NP2) ← NP1

The difference between the intersection and inclusion notions is such that the inclusion leads to a reduction of the cardinality of set v1 because set v1 has to be included in set v2.
A node which is a stage-city in a path between Lyon and Marseille but not a stage-city between Paris and Nice will not be considered (even if it is a stage-city of a path Lyon-Marseille).

By the end query 5 can be expressed like :



5.4 Global queries :

The juxtaposition operator // does not generate a graph but a global query (by definition). This allows to defined queries with a logical OR such as i.e. What are the direct paths (without any stages) from Nice to Paris with AF company or if we have to stop, we must have a stop at Marseille whatever the company is. A global query Q is a set of sub-queries Q_1, ... , Q_p :

. Q_i (N_i, E_i, Ψ_i, ν_i, ε_i) and N_i, E_i, Ψ_i, ν_i, ε_i are defined as previous.

$$Q = Q_1 (N_1, E_1) // \ldots // Q_p (N_P, E_P)$$

The juxtaposition operator // allows queries to be defined in parallel.

Properties :

The juxtaposition (//) and the fusion (&&) are associative and fusion (&&) is distributive on the juxtaposition (//). The results in case 2 and 3 are global queries.

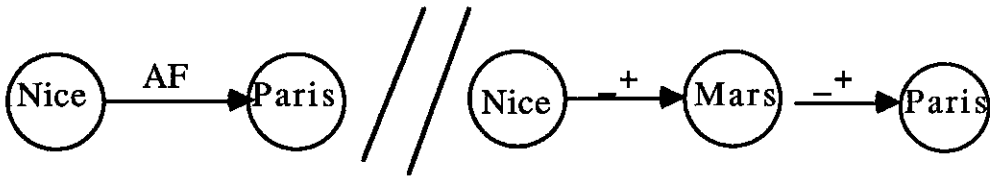
1) $G_1 (N_1, E_1)$ && $(G_2 (N_2, E_2)$ && $G_3 (N_3, E_3))$ =
$(G_1 (N_1, E_1)$ && $G_2 (N_2, E_2))$ && $G_3 (N_3, E_3)$

2) $G_1 (N_1, E_1)$ // $(G_2 (N_2, E_2)$ // $G_3 (N_3, E_3))$ =
$(G_1 (N_1, E_1)$ // $G_2 (N_2, E_2))$ // $G_3 (N_3, E_3)$

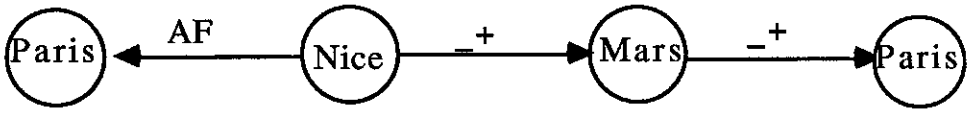
3) $G_1 (N_1, E_1)$ && $(G_2 (N_2, E_2)$ // $G_3 (N_3, E_3))$ =
$(G_1 (N_1, E_1)$ && $G_2 (N_2, E_2))$ // $(G_1 (N_1, E_1)$ && $G_3 (N_3, E_3))$

The juxtaposition operator (//) can be seen as a logical "OR" between two queries :

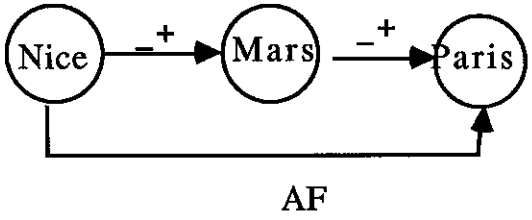
Example :



is by definition different from the fusion (&&), which can be seen as a logical "AND". Therefore this graph is different from the previous graph :



An other expression for the fusion (&&) is :



Results :

A global query is such as :

$$Q = Q_1 // \ldots // Q_p = // Q_i \quad (i = 1, \ldots, p)$$

Q has no solution if :

$\forall i \in \{1, \ldots, p\}$, Q_i has no solution

Q_i has no solution if :

. An edge of graph Q_i has no instanciation or

. A variable has no instanciation (value equal to $\varnothing$) unless if it is required. Asking a value to be equal to $\varnothing$ is a way to express the negation concept.

6. Conclusion :

Part of Geographical Information Systems (GIS) is network-oriented information management applications. This paper presents a Data Manipulation Language based on [CMW87]. This language allows the user to define easily graph-manipulation-oriented queries. The implementation of this language is realistic [P87, M89]. Graphical screens and Extended Data Base Mangement System are very convinient tools. Transitive closure algorithms in a Data Base context have been widely studied [BR86].
The partial implementation (recursive queries without variable) was done two years ago. It was based on a weak couplage between a non-first normal form RDBMS [S88] and Prolog. Transitive Closure operators with aggregates were taken into account by Prolog and paths were stored in the RDBMS. A macintosh-like interface allowed the user to define a simple recursive query (are very complex queries realistic?) and to get information from an edge of the result graph.
Future work will be oriented toward the definition of a thematic Data Manipulation Language and a study of the links between the network component approach and the thematic component approach to define a complete Geographical Data Manipulation Language.

REFERENCES

BR 86 — F. Bancilhon R. Ramakrishnan
An amateur's Introduction to Recursive Query Processing Strategies
Proceedings of the ACM SIGMOD conference
Washington May 1986

CGT88 — S. Ceri G. Gottlob L. Tanca
Logic Programming and Databases
Springer Verlag 1988

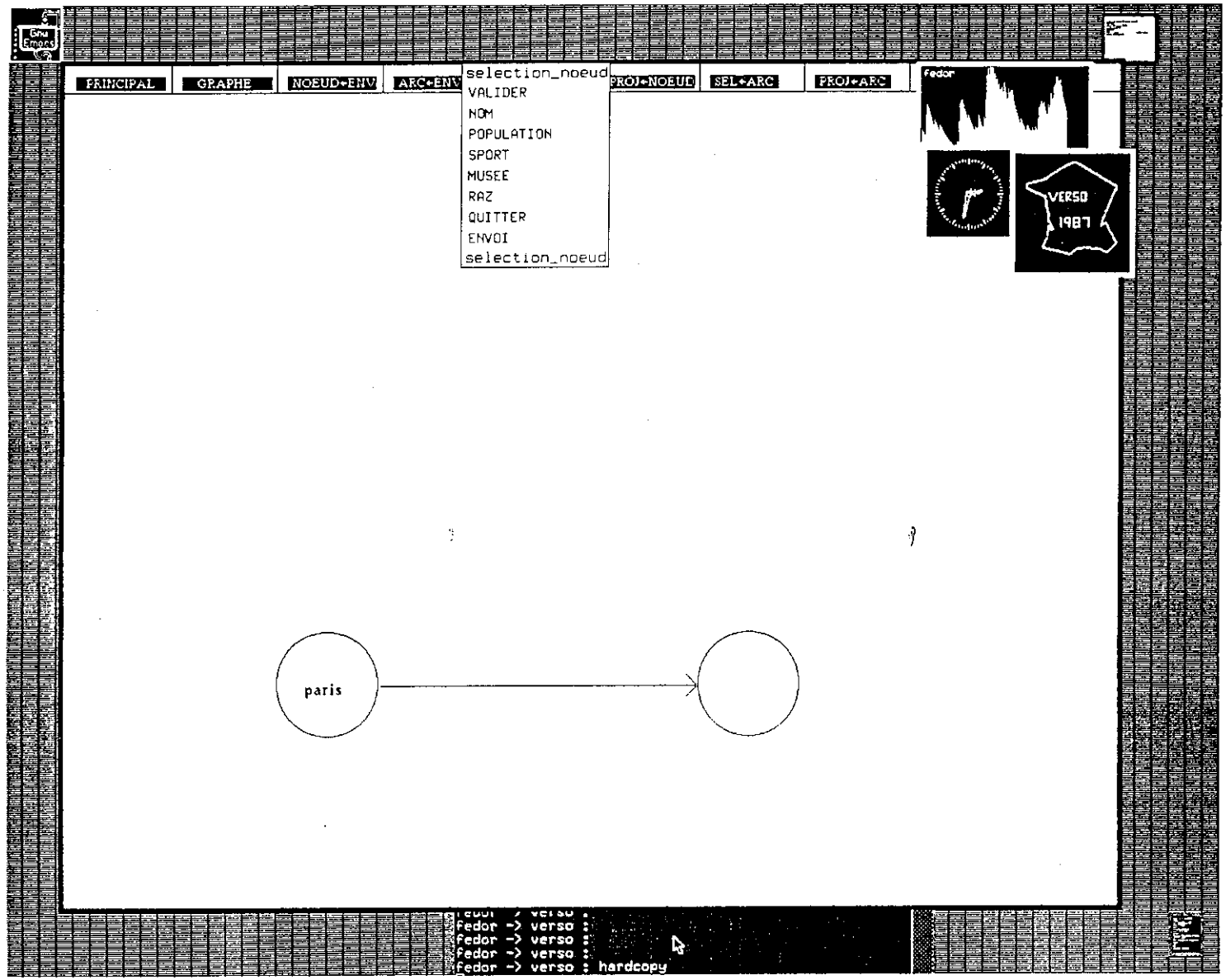
CMW 87 — IF Cruz AO Mendelzon PT Wood
A graphical Query Language supporting recursion
Proceedings of the SIGMOD conference
San Fransisco May 1987

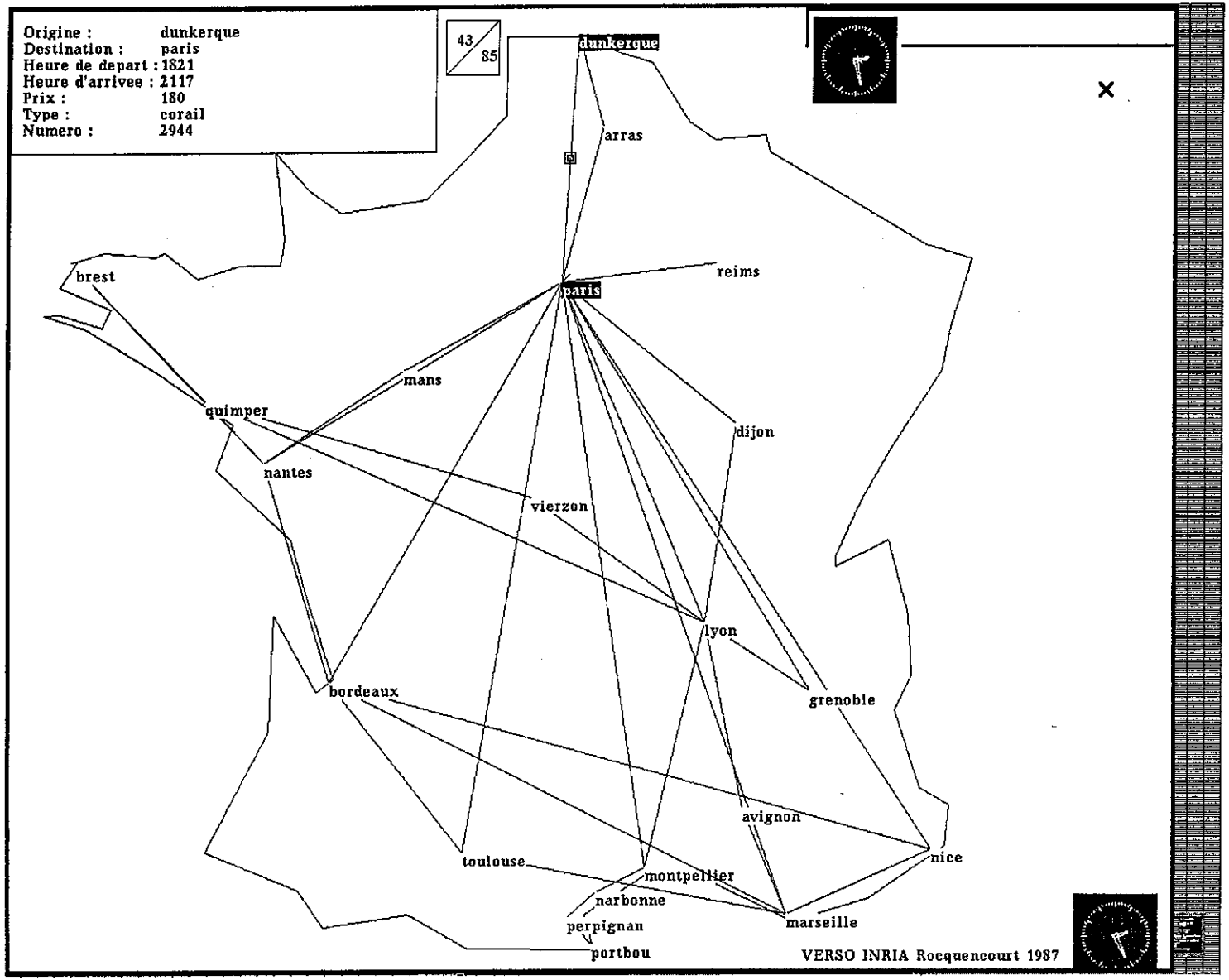
M89 — M. Mainguenaud
Un langage visuel de manipulation de graphes : Application aux Bases de Données Géographiques
Doctorate Thesis - Université Paris VI 6/28/89

P87 — P. Pfeffer
LGV : un langage graphique de manipulation de réseaux
DESS Report
Université de Paris-Sud Orsay July 1987

S88 — M. Scholl and all
VERSO : a DBMS based on Nested Relations
Nested Relations and Complex Objects Workshop - Darmstadt
Lecture Notes in Computer Science
Springer Verlag December 1988

SMSE 87 — TR Smith S. Menon JL Star JE Ester
Requirements and principles for implementation and construction of large scale GIS
International Journal of GIS Vol 1 n° 1 1987


PRINCIPAL
GRAPHE
NOEUD•ENV
ARC•ENV
selection_noeud
VALIDER
NOM
POPULATION
SPORT
MUSEE
RAZ
QUITTER
ENVOI
selection_noeud
PROJ•NOEUD
SEL•ARC
PROJ•ARC
fedor
VERSO
1987
paris
fedor -> verso :
fedor -> verso :
fedor -> verso :
fedor -> verso : hardcopy


Origine : dunkerque
Destination : paris
Heure de depart : 1821
Heure d'arrivee : 2117
Prix : 180
Type : corail
Numero : 2944
43
85
dunkerque
arras
reims
brest
paris
mans
quimper
dijon
nantes
vierzon
lyon
bordeaux
grenoble
avignon
nice
toulouse
montpellier
narbonne
marseille
perpignan
portbou
VERSO INRIA Rocquencourt 1987