Gram: A Graph Data Model and Query Language[1]

Bernd Amann
INRIA
F-78153 Le Chesnay Cedex, France

Michel Scholl
Cedric/CNAM
292 rue St Martin, F-75141 Paris Cedex 03, France

Abstract

We present a model for data organized as graphs. Regular expressions over the types of the node and edge labels are used to qualify connected subgraphs. An algebraic language based on these regular expressions and supporting a restricted form of recursion is introduced. A natural application of this model and its query language is hypertext querying.

1 Introduction

Recent database [13, 5] research work shows a growing interest in the definition of graph models and languages to allow a natural way of handling data appearing in applications such as hypertext or geographic database systems. Standard data models are often inefficient as they do not capture the inherent structure of data representing hypertext documents [4, 7, 18] or networks (highways, rivers, ...) [12].

In this paper, we present a graph data model. Its application to hypertext querying is illustrated by an example of a travel agency that organizes journeys. We think of a hypertext as a directed labeled graph where the nodes are typed documents and the edges correspond to typed span-to-span[2] links between documents [15].

A journey corresponds to a sequence of stops in several cities, where hotels, restaurants and monuments are visited. Figure 1 shows a schema, also structured as a graph.

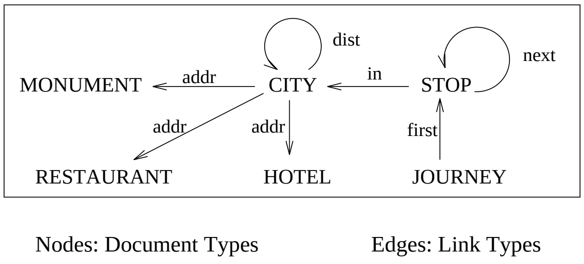


Figure 1: The Travel Agency Schema

A document can have one of the following types: STOP, JOURNEY, CITY, HOTEL, RESTAURANT and MONUMENT. A sample of the database graph is given in Figure 2. Note that links are typed and may contain information such as a date, an address or a distance in kilometers. Since restaurants like McDonalds can be described independently from the city where they are located, only their common characteristics are described in the RESTAURANT node. The information specific to a hotel or restaurant in a city, e.g. its address, is contained in the link connecting it with the city document. Consider a travel agency customer reading some touristic presentation of Paris:

> ...*A very nice hotel in the* 15^{th} *district is the* `Imperial` *hotel near the metro station Pasteur. From there you can visit* `Eiffel Tower`, `Trocadero`, `Orsay Museum`, ...

The two words `Eiffel Tower` are an anchor connecting this text to some other document describing Eiffel Tower. The customer navigates

[1] Work partially supported by the French *Programme de Recherche Coordonnée BD3* and the BRA Esprit Project *Amusing*. The second author is also affiliated to INRIA, France.

[2] Links connect not only documents but more precisely spans of characters called *anchors*, each anchor being located in a given document.

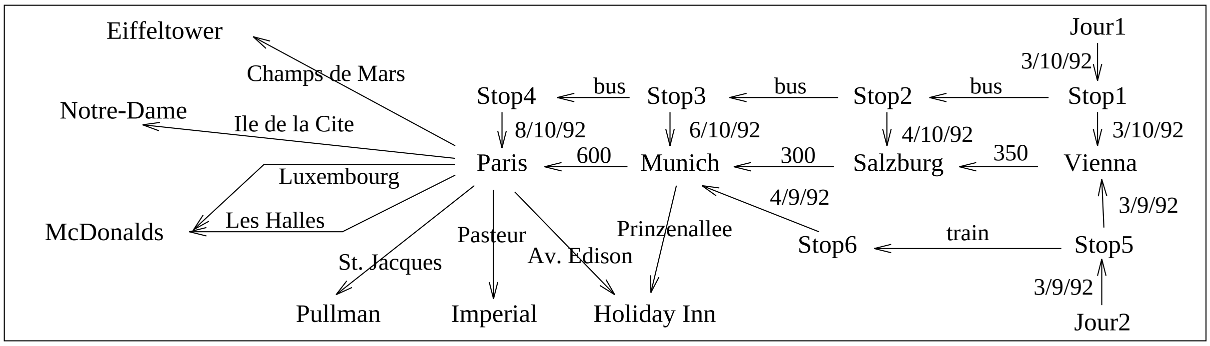


Figure 2: A Travel Agency Hypertext Graph

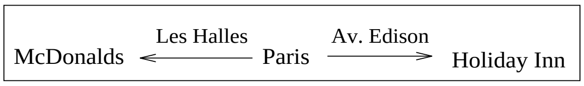


Figure 3: A Hyperwalk

through the database graph by clicking on the anchors. The corresponding target document, e.g. the description of Eiffel Tower, is displayed on the screen and contains itself anchors that can be selected. Such a navigation might be supported by a graphical *browser*, showing a map of the hypertext graph and the current position. One drawback of graph maps is that they easily become tangled, when the number of nodes and edges passes some limit (for example more than a few dozen documents [6]). Thus query languages become necessary to restrict the search space in navigational graphs [11, 14, 9].

We present a query algebra where regular expressions over data types are used to select walks in a graph. For example the regular expression JOURNEY first (STOP next)* STOP in CITY describes the walks going from a journey document (a node of type JOURNEY) to one of its stops in a city (node of type CITY). To illustrate the power of this language, we take three examples of queries on the travel agency hypertext. Figure 3 contains a subgraph of the database showing that there are a Holiday Inn and a McDonalds in Paris. Assume that for representing such information we have two relations HOTEL(NAME, ADDRESS, CITY) and RESTAURANT(NAME, ADDRESS, CITY).

The following SQL query extracts all cities with a Holiday Inn and a McDonalds restaurant:

```
select HOTEL.CITY
from HOTEL, RESTAURANT
where HOTEL.NAME='Holiday Inn'
  and RESTAURANT.NAME='McDonalds'
  and HOTEL.CITY=RESTAURANT.CITY
```

A set of tuples, an element of which is depicted in Figure 3, is selected and projected on the CITY nodes. In our model, such a tuple is called a **hyperwalk**. We present an algebra for hyperwalks, which are specified by regular expressions, called hyperwalk expressions. For example the schema of the hyperwalk in Figure 3 can be represented by hyperwalk expression CITY addr RESTAURANT + CITY addr HOTEL.

Intuitively, the user wants to navigate, i.e. follow some paths (walks) in the hypertext graph. He will choose walks according to

- a given hyperwalk expression.
- the values of the nodes and edges traversed.
- the types and values of nodes and edges of other walks that are related to the nodes and edges traversed.

With the algebra to be defined below, the above query is formulated as follows (a syntax a la SQL is used for clarity):

```
select CITY
from CITY addr RESTAURANT +
     CITY addr HOTEL
where RESTAURANT.name="McDonalds" and
     HOTEL.name="Holiday Inn"
```

The `from` clause specifies the range of the query: it is a regular expression r over types of nodes and edges and specifies the starting set of hyperwalks. The `select` and `where` clauses, as usual, specify the selection criteria, as well as what is to be projected: CITY in the `select` clause specifies that for each target hyperwalk we keep only the CITY document. RESTAURANT.name="McDonalds" in the `where` clause specifies that we select only the hyperwalks whose restaurant document is such that its attribute name value is "McDonalds".

As another example, the following query gives all restaurants in the third district of Paris:

`select` RESTAURANT
`from` CITY addr RESTAURANT
`where` CITY.name="Paris" `and` addr.district=3

The last query gives all information about journeys including a visit to the Mont St. Michel monument in March:

`select` *
`from` JOURNEY first (STOP next)* STOP
in CITY addr MONUMENT
`where` in.month="March" `and`
MONUMENT.name="Mont St. Michel"

Section 2 defines graph schemas and databases. Hyperwalk expressions and the notion of satisfaction of a hyperwalk expression are introduced in Section 3. Section 4 presents the algebraic operations on hyperwalks which are selection, projection, renaming, join and concatenation as well as the usual set operations. The hyperwalk algebra is then applied on the examples mentioned before. Finally, in Section 5, the application of this model to hypertext querying is illustrated.

2 Graph Databases

2.1 Database Schema

Definition 2.1 *A* (graph database) schema *is a directed weakly connected*[3] *labeled multigraph*
$S = (N_S, E_S, \psi_S, \lambda_S, T)$ *where*

1. *N_S is a set of nodes and E_S is a set of edges.*
2. *ψ_S is an incidence function from E_S into $N_S \times N_S$.*
3. *$T = T_N \cup T_E$ is a set of labels and λ_S is a labeling function from $N_S \cup E_S$ into T.*

Additionally, S has to satisfy the following restrictions:

- *Different nodes must have different labels (distinct node label property):*
- *Note that S is a multigraph, i.e. a pair of nodes can be connected by more than one edge. If two edges connect the same pair of nodes in the same direction then they must have different labels (distinct edge label property):* $\psi(e) = \psi(e') \wedge e \neq e' \Rightarrow \lambda_S(e) \neq \lambda_S(e')$.

We assume that each symbol t in T, called a type, is associated with a *domain* of values, denoted $dom(t)$. A set of values V is defined

[3] A directed graph is weakly connected if the corresponding undirected graph is connected [16]. From now on, connected will mean weakly connected.

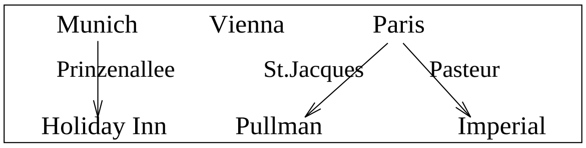


Figure 4: A Travel Agency Database

as the union of the domains of all types in T: $V = \bigcup_{t \in T} dom(t)$. We do not make any other assumptions on the type system T. Figure 1 shows a schema with 6 nodes and 6 edges labeled by the types T ={JOURNEY, STOP, CITY, HOTEL, RESTAURANT, MONUMENT, addr, in, dist, next, first}.

2.2 Graph Database

A (graph) database with schema S is a directed labeled graph with labels in V. A function associates with each node and edge in the database a node, respectively an edge in the schema and thus allows the specification of the databases satisfying a schema:

Definition 2.2 *A* (graph) database *with schema S is a directed labeled multigraph*
$G = (N, E, \psi, \lambda, V)$ *where*

1. *N is a set of nodes and E is a set of edges.*
2. *ψ is an incidence function from E into $N \times N$.*
3. *V is a set of labels and λ is a labeling function from $N \cup E$ into V,*

such that there exists a function τ from G to S associating with each node (each edge) in G a node (an edge) in S with:

- *τ returns for each node ν in G a node n in S and the label of ν is in the domain of the label of n: $\lambda(\nu) \in dom(\lambda_S(\tau(\nu)))$.*
- *τ returns for each edge ϵ in G, going from node ν to node ν', an edge e in S, which is going from node $\tau(\nu)$ to node $\tau(\nu')$ and the label of ϵ is in the domain of the label of e: $\psi_S(\tau(\epsilon)) = (\tau(\nu), \tau(\nu'))$ and $\lambda(\epsilon) \in dom(\lambda_S(\tau(\epsilon)))$.*

Conversely, an instance of a database schema S is a graph database. An instantiation function I associates with each node (edge) m in S a set of nodes (edges) in G as follows: $I(m) = \{\mu \mid \tau(\mu) = m\}$. Note that a database G is not necessarily connected (Figure 4) and there may be nodes without edge (Vienna) or more than one incoming or outgoing edge (Paris).

3 Hyperwalks and Hyperwalk Expressions

3.1 Walks and Hyperwalks

Walks (paths) are the basic objects of our model. A *walk* in a graph is an alternating sequence of nodes and edges $n_0 e_0 n_1 \ldots n_{i-1} e_{i-1} n_i$ beginning and ending with nodes, in which each edge is incident with the two nodes immediately preceding and following it [16]. For example, to get the cities in a journey, one chooses the walks in the database starting from a node of type JOURNEY[4] and ending with a CITY node.

As a matter of fact, a walk the user traverses, might be related to information in other walks. For example, he might choose all pairs of walks, starting in different journey nodes and ending in the same city. In the following, we will combine walks into sets of walks, called *hyperwalks*. Figure 4 shows a hyperwalk made of two walks connecting Paris with the Pullman and Imperial hotel.

3.2 Walk and Hyperwalk Expressions

Let $T = T_N \cup T_E$ where T_N (T_E) denotes the sets of node (edge) types in T. A *walk expression* (*we*) is a regular expression (r.e.) over T without alternation (+), whose language contains only alternating sequences of node and edge types, starting and ending with a node type. For example (STOP next)* STOP in CITY addr HOTEL is a walk expression.

Definition 3.1 *A regular expression over T is a* hyperwalk expression (hwe) *iff 1) it can be rewritten as a sum of walk expressions, $r = \sum_{1 \leq i \leq n} r_i$ such that 2) the following undirected labeled graph $\mathcal{G}(r)$ is* connected*:*

$\mathcal{G}(r) = (\mathcal{N}, \mathcal{E}, \lambda_{\mathcal{G}})$, where with each r_i we associate a node n_i in $\mathcal{N}$ with label r_i ($\lambda_{\mathcal{G}}(n_i) = r_i$), and there exists an edge with label t between nodes n_j and n_k iff t is a node type in r_j and r_k.

Condition 2) is a necessary condition to enforce that information in walks of a hyperwalk are related to each other.

Example 3.1 *The r.e.* JOURNEY first STOP in CITY + STOP next STOP in CITY *is a hyperwalk expression. The walk expressions* JOURNEY first STOP in CITY *and* STOP next STOP in CITY *share the two node types* STOP *and* CITY *(Figure 5).*

[4] We say that a node is of type t, if its label is in $dom(t)$.

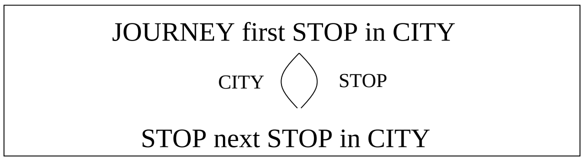


Figure 5: $\mathcal{G}(JOURNEY\ first\ STOP\ in\ CITY + STOP\ next\ STOP\ in\ CITY)$

JOURNEY first STOP + CITY addr HOTEL *is not a* hwe *since the walk expressions* JOURNEY first STOP *and* CITY addr HOTEL *do not share any node type.*

Hyperwalk satisfying a Hyperwalk Expression: The *label* of a walk is obtained by substituting each node and edge in the walk with its label: $\lambda(n_1 e_1 \ldots e_{k-1} n_k) = \lambda(n_1)\lambda(e_1) \ldots \lambda(e_{k-1})\lambda(n_k)$.

The *label* of a hyperwalk h is obtained by replacing each walk in h by its label:
$\lambda(\{w_1, w_2, \ldots w_n\}) = \{\lambda(w_1), \lambda(w_2) \ldots \lambda(w_n)\}$.

We define the following language over T:

$$\begin{aligned}
\mathcal{L}(t) &= dom(t) \text{ for all } t \in T. \\
\mathcal{L}(A.B) &= \mathcal{L}(A) \times \mathcal{L}(B) \text{ for all r.e. } A \text{ and } B. \\
\mathcal{L}(A+B) &= \mathcal{L}(A) \cup \mathcal{L}(B) \text{ for all r.e. } A \text{ and } B. \\
\mathcal{L}(A^*) &= \cup_{i \geq 0} \mathcal{L}(A^i) \text{ where } \mathcal{L}(A^0) = \{\epsilon\} \\
& \quad \text{for all regular expressions } A.
\end{aligned}$$

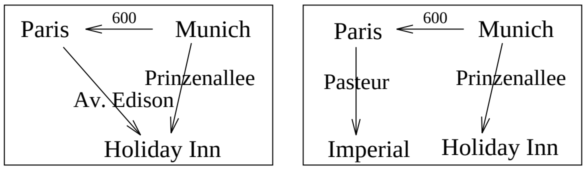
From now on, $\mathcal{L}(r)$ will be called the language of r. Let r be a *we* and w be a walk whose label is in $\mathcal{L}(r)$: $\lambda(w) \in \mathcal{L}(r)$. With each occurrence of a type t in r we can associate some nodes or edges in w, called instances of t in w, i.e. whose labels are in $dom(t)$. For example Vienna is an instance of CITY in the walk Stop1.3/10/92.Vienna whose label is in $\mathcal{L}$(STOP in CITY).

Definition 3.2 *A hyperwalk $h = \{w_1, w_2, \ldots w_n\}$* satisfies *a* hwe *$r$, denoted $h \models r$, iff*

1. *its label is a subset of $\mathcal{L}(r)$: $\lambda(h) \subseteq \mathcal{L}(r)$.*
2. *there exists a decomposition $\sum_{1 \leq i \leq n} r_i$ of r, such that each walk w_i in h has a label $\lambda(w_i)$ in $\mathcal{L}(r_i)$.*
3. *for* all *node types t shared by r_i and r_j, w_i and w_j share at least a node of type t.*

Condition 3) enforces that *the walks in h form a connected graph*: to each edge e in $\mathcal{G}(r)$, with label $t \in T$ and between two nodes with label r_i and r_j, correspond two walks w_i and w_j in h such that

- $\lambda(w_i) \in \mathcal{L}(r_i)$, $\lambda(w_j) \in \mathcal{L}(r_j)$ and
- w_i and w_j share a node n whose label is in the domain of t: $\lambda(n) \in dom(t)$.



a) b)

Figure 6: Hyperwalk Connectivity

Example 3.2 h={Jour1.3/10/92.Stop1, Stop1.bus-.Stop2} *is a hyperwalk satisfying* hwe r=JOURNEY-first STOP + STOP next STOP. *The two walks in h share a node whose label* Stop1 *is of type* STOP. {Jour1.3/10/92.Stop1, Stop2.bus.Stop3} *does not satisfy r. The two walks do not share any node.*

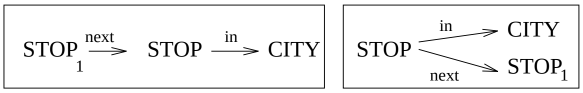
(STOP next)* STOP + STOP in CITY *is satisfied by* {Stop1.bus.Stop2.bus.Stop3, Stop2.4/10/92.Salzburg}. *The two walks share node* Stop2.

Note that condition 3 requires more than h connectivity. In h={Munich.600.Paris.AvEdison-.HolidayInn, Munich.Prinzenallee.HolidayInn} *(Figure 6a) satisfying* r=CITY dist CITY addr HOTEL + CITY addr HOTEL, *the two walks share* Munich *of type* CITY **and** HolidayInn *of type* HOTEL. *On the contrary,* h'={Munich.600.Paris-.Pasteur.Imperial, Munich.Prinzenallee.HolidayInn} *(Figure 6b) does not satisfy r. Indeed the two walks share* Munich *of type* CITY *but* **do not** *share any node of type* HOTEL *(they should !).*

Instance of a Hyperwalk-Expression: The *instance* of a *hwe* r in a database G is defined as the set of hyperwalks in G satisfying r: $I(r) = \{h \mid h \in G \wedge h \models r\}$.

4 Hyperwalk Algebra

The query language defined in this section is based on a hyperwalk algebra. A *query* is an expression of the form $\phi(S)$ or $\phi'(S, S')$ where S and S' are sets of hyperwalks and ϕ is an *algebraic operation* which is closed under the sets of hyperwalks. Unary operations (*projection, selection, renaming*) take as a source a set S of hyperwalks in the graph database G satisfying a given *hwe* r and return a target set of hyperwalks S' satisfying a *hwe* r' possibly different from r. Binary operations (*join, concatenation, set operations*) take two sets of source hyperwalks and return a target set of hyperwalks.



a) b)

Figure 7: Two Hyperwalk Expressions

4.1 Renaming

Hyperwalk expressions may have several occurrences of the same type. Renaming allows one to distinguish between the different occurrences of a given type in a given *hwe*. For example h_1 ={Vienna.350.Salzburg} and h_2 ={Munich.600.Paris} are two hyperwalks satisfying r=CITY dist CITY. To distinguish between the two occurrences of CITY in r, we rename the second occurrence into CITY'. In CITY dist CITY', CITY specifies the first city, which is Vienna for h_1 and Munich for h_2 and CITY' specifies the second city (Salzburg for h_1 and Paris for h_2). In relational algebra, renaming of attributes in a relation satisfying a given relational schema keeps the relation unchanged. Here, renaming of an instance $I(r)$ in G may select a subset of the hyperwalks in $I(r)$ as illustrated by Figure 7:

$$
\begin{aligned}
& I(\mathsf{STOP\ next\ STOP} + \mathsf{STOP\ in\ CITY}) = \\
& \quad I(\mathsf{STOP}_1\ \mathsf{next\ STOP} + \mathsf{STOP\ in\ CITY}) \\
& \quad \cup I(\mathsf{STOP\ next\ STOP}_1 + \mathsf{STOP\ in\ CITY})
\end{aligned}
$$

By renaming the first (second) instance of STOP in r, we get the subset of hyperwalks in $I(r)$ satisfying the schema in Figure 7-a (7-b).

Definition 4.1 *Let r and r' be two hyperwalk expressions such that r' is obtained from r by renaming some types in r without changing the type definition: if t in r has been renamed into t' then $dom(t) = dom(t')$. Then r and r' have the same language: $\mathcal{L}(r) = \mathcal{L}(r')$. Let S be a set of hyperwalks satisfying* hwe r. *Renaming $\rho_{r'}(S)$ keeps those hyperwalks in S that satisfy r' ($\rho_{r'}(S) \subseteq S$):*

$$\rho_{r'}(S) = \{h \mid h \in S, h \models r'\}$$

Example 4.1 h={Stop1.3/10/92.Vienna, Vienna-.350.Salzburg} *satisfies* hwe r=STOP in CITY + CITY dist CITY. *After renaming r into* r'=STOP in-CITY_1 + CITY_1 dist CITY, Vienna *is an instance of* CITY_1, *and* Salzburg *is an instance of* CITY. *Hyperwalk h satisfies r' but does not satisfy the renaming of r into* STOP in CITY_1 + CITY dist CITY_1.

4.2 Selection

Selection allows one to evaluate Boolean functions (selection conditions) on hyperwalk labels. Applied to a set S of hyperwalks it returns the subset of hyperwalks in S whose labels satisfy the given conditions.

We may distinguish in a regular expression r, "simple" subexpressions (without $*$) from "complex" subexpressions, i.e. those of the form $(s)^*$. In *hwe* r=(CITY$_1$ dist)* CITY$_2$ addr RESTAURANT, (CITY$_1$ dist)* is a complex subexpression and CITY$_2$ addr RESTAURANT is a simple subexpression. If h is a hyperwalk in the instance of r then with a simple subexpression v in r is associated a unique component of h, denoted $h : v$, which is not necessarily a hyperwalk. To each complex subexpression $(s)^*$ of r corresponds a *set* of components in h, each of them satisfying s. For hyperwalk h={Salzburg.300.Munich.600.Paris} in the instance of (CITY$_1$ dist)* CITY$_2$, the component h:CITY$_2$ denotes Paris and the component h:(CITY$_1$ dist)* denotes the set {{Salzburg.300}, {Munich.600}}.

Definition 4.2 Selection conditions *on* regular expressions *are defined as follows:*

1. *Let t be a type in a simple subexpression of r. A Boolean function $f(t)$ from $dom(t)$ into $\{true, false\}$ is a simple condition on r.*
2. *Let t and t' be two types in a simple subexpression of r. Then $t = t'$ is a simple condition on r.*
3. *If C is a condition on s, and $(s)^*$ is a complex subexpression of r, then $\exists C$ is a condition on r.*
4. *If C and C' are conditions on r, then so are $C \wedge C'$, $C \vee C'$ and* `not` *C.*

Definition 4.3 *Let S be a set of hyperwalks satisfying r and C be a condition on r. The selection on S with condition C, denoted $\sigma_C(S)$, returns the hyperwalks in S satisfying C:*

$$\sigma_C(S) = \{h \mid h \in S \wedge h \text{ satisfies } C\}.$$

A hyperwalk h *satisfies* a selection condition C (C is true for h) if one of the following holds:

1. If C is simple, then it is evaluated in the straightforward way on the label of the corresponding component in h (as for tuples in the relational model).
2. If $C \equiv \exists D$ and D is a selection condition on s, then C is true for h iff the selection on the $(s)^*$-component of h, denoted $h : (s)^*$, is not empty: $h \models \exists D$ iff $\sigma_D(h : (s)^*) \neq \emptyset$.
3. If $C \equiv D \wedge F$ then $h \models C$ iff $h \models D$ and $h \models F$.
4. If $C \equiv D \vee F$ then $h \models C$ iff $h \models D$ or $h \models F$.
5. If $C \equiv not\ D$ then $h \models C$ iff $h \not\models D$.

Example 4.2 *To get all hotels in Vienna, we apply $\sigma_{vienna(\text{CITY})}$ on the hyperwalks satisfying* CITY addr HOTEL *where vienna(*CITY*) is true for the* CITY *node corresponding to the city of Vienna.*

Let S be a set of hyperwalks satisfying (CITY-dist)* CITY*. To get all hyperwalks where at least two neighbor cities are closer than 100 kilometers from each other, we apply $\sigma_{\exists less100(\text{dist})}$ on S, where $less100 : dom(\text{dist}) \rightarrow \{\text{true}, \text{false}\}$ is true for all* dist*-edges with a value less than 100 km.*

4.3 Projection

Informally speaking, the projection of a hyperwalk h on a *hwe* r' consists in keeping either a subset of the walks in h or subwalks of walks in h. For example {Stop1.bus.Stop2} is a projection of h={Stop1.bus.Stop2, Stop1.3/10/92.Vienna}. So is the set {Vienna}, containing only the city of Vienna.

We define a partial order on walk expressions as follows.

Definition 4.4 *Let r and r' be two walk expressions. Then r' is a* subexpression *of r, denoted $r' \leq r$, if r is of the form $u.r'.v$ where u and v are possibly equal to ϵ.*

For example STOP in CITY is a subexpression of JOURNEY first STOP in CITY. It is not a subexpression of STOP (next STOP)* in CITY.

Now we define a partial order on hyperwalk expressions as follows:

Definition 4.5 *A* hwe $r' = \sum_{1 \leq j \leq m} r'_j$ *is a* subexpression *of a* hwe $r = \sum_{1 \leq i \leq n} r_i$*, denoted $r' \leq r$, iff all of the following hold:*

1. *r' is a* hwe *with graph $\mathcal{G}(r')$*[5] *(see Definition 3.1).*
2. *$m \leq n$ and for all walk expressions r'_j, $j \in [1, m]$, there exists a* we *r_i in r, such that $r'_j \leq r_i$. r_i is a called superexpression of r'_j.*
3. *To each edge in $\mathcal{G}(r')$ with label t connecting n'_i with label r'_i to n'_j with label r'_j, corresponds an edge in $\mathcal{G}(r)$ with label t connecting n_i with label r_i (superexpression of r'_i) to n_j with label r_j (superexpression of r'_j). Then all occurrences of t in r_i (r_j) have been kept in r'_i (r'_j).*

[5] $\mathcal{G}(r')$ must be connected, otherwise r' is not a *hwe*.

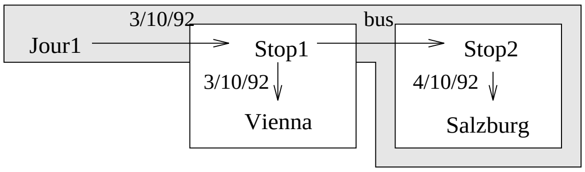


Figure 8: Hyperwalk Projection

In other words, not all projections on walks are legal: (i) $\mathcal{G}(r')$ must be connected (Condition 1) and (ii) when a type with multiple occurrences is shared, then **all** occurrences must be projected (Condition 3). To understand why Condition 3 is necessary, look at Figure 8. It shows one hyperwalk satisfying r=JOURNEY first STOP in CITY$_1$ + STOP next STOP in CITY$_2$. r'=JOURNEY first STOP + STOP next STOP in CITY$_2$ (grey) is a subexpression of r: $r' \leq r$. But this is not true any more for the white part satisfying r"=STOP in CITY$_1$ + STOP in CITY$_2$. r'' violates Condition 3 since its second component STOP in CITY$_2$ is a subexpression of STOP next STOP in CITY$_2$, in which we have not kept all occurrences of STOP. We should keep all occurrences of STOP in r, since we do not know for each hyperwalk h satisfying r, whether the stops shared by its walks is an instance of the first or the second occurrence of STOP. By removing one occurrence in the subexpression, the corresponding component of h (Figure 8) satisfying the subexpression is not anymore a hyperwalk (it is not connected): {Stop1.3/10/92.Vienna, Stop1.bus.Stop2.4/10/92.Salzburg} satisfies r, but the projection on r'', {Stop1.3/10/92.Vienna, Stop2.4/10/92.Salzburg} is not a hyperwalk.

Definition 4.6 *Let* hwe r' *be a subexpression of* hwe r. *Then the projection of a set of hyperwalks* $S \subseteq I(r)$ *on* r', *denoted* $\pi_{r'}(S)$, *is the set of the* r'*-components, denoted* $h : r'$, *of the hyperwalks* h *in* S*:*

$$\pi_{r'}(S) = \{h' \mid h \in S,\ h' = h : r')\}$$

$\pi_{r'}(S)$ is a subset of the instance of r', $I(r')$.

Example 4.3 *Figure 9 shows two hyperwalks satisfying* hwe JOURNEY first (STOP next)* STOP in CITY addr HOTEL. *The projection on* CITY addr HOTEL *returns* {Paris.Pasteur.Imperial} *and* {Munich.Prinzenallee.HolidayInn}. *Note that projecting on* (STOP next)* STOP *returns* {Stop1.bus.Stop2.bus.Stop3.bus.Stop4} *and* {Stop5.train.Stop6}.

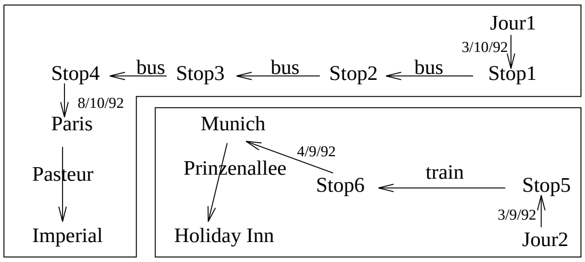


Figure 9: Two Hyperwalks

4.4 Join

The join operation takes pairs of hyperwalks from two different hyperwalk sets $S \subseteq I(r)$ and $S' \subseteq I(r')$ and returns their union that satisfies *hwe* $r + r'$:

Definition 4.7 *Let* S *and* S' *be two subsets of* $I(r)$ *respectively* $I(r')$. *If* $r + r'$ *is a* hwe *then the* join *of* S *and* S', *denoted* $S \bowtie S'$, *returns a subset of* $I(r + r')$ *each hyperwalk of which is the union of a hyperwalk in* S *and a hyperwalk in* S'*:*

$$S \bowtie S' = \{h \cup h' \mid h \in S, h' \in S', h \cup h' \models r + r'\}.$$

Note that the union of two hyperwalks $h \in S$ and $h' \in S'$ is not necessarily a hyperwalk satisfying $r + r'$. For example the union of {Paris.Luxembourg.McDonalds} and {Munich.Prinzenallee.HolidayInn} is not a hyperwalk. In fact, $h \cup h'$ is a hyperwalk in $S \bowtie S'$ if for each pair of walk expressions r_i in r and r'_j in r' sharing a node type t, the corresponding walks in h and h' share a node of type t.

Example 4.4 *Suppose we have a set of journeys* $S \subseteq I($JOURNEY first(STOP next)*.STOP$)$. *To keep only those journeys that make a stop in Munich, we join* S *with* $\sigma_{munich(\text{CITY})} I($STOP in CITY$)$.

All journeys stopping in Munich before Paris can be obtained by the following query (we want to keep information not only about the journey itself but also about all stops in the journey):

$$\sigma_{paris(\text{CITY})}\ I(\text{JOURNEY first (STOP next)* STOP in CITY}) \bowtie \sigma_{munich(\text{CITY})}\ I(\text{STOP in CITY})$$

4.5 Concatenation

The concatenation of two hyperwalk sets S and S' concatenates each hyperwalk h in S with a hyperwalk h' in S' whenever it is possible. The idea is to concatenate walks in h with walks in h'. If the ending node of a walk $w \in h$ is equal to the starting node of a walk $w' \in h'$, then the concatenation of w and w', denoted $w \circ w'$, is obtained by replacing the ending node in w by w'. For example the concatenation of Jour1.3/10/92.Stop1 and Stop1.bus.Stop2 is Jour1.3/10/92.Stop1.bus.Stop2. If the ending node of w is not equal to the starting node of w', then $w \circ w' = \epsilon$.

Concatenation of walk expressions is defined as follows. Let r and r' be two walk expressions. If r is of the form $u.t$ and r' is of the form $t.v$, where $t \in T_N$ and u, v are regular expressions over T, then the concatenation of r and r', denoted $r \odot r'$, is $u.t.v$. It is ϵ otherwise. For example (STOP next)* STOP $\odot$ STOP in CITY=(STOP next)*STOP in CITY.

For two hyperwalk expressions $r = \sum_{1 \leq i \leq n} r_i$ and $r' = \sum_{1 \leq j \leq m} r'_j$, concatenation, denoted $r \odot r'$, is defined as the sum of the concatenations of the walk expressions in r with the walk expressions in r':

$$r \odot r' = \sum_{1 \leq i \leq n, 1 \leq j \leq m} r_i \odot r'_j.$$

For example STOP in CITY $\odot$ (CITY addr HOTEL + CITY addr RESTAURANT) is STOP in CITY addr HOTEL + STOP in CITY addr RESTAURANT. The concatenation of STOP next STOP with CITY addr HOTEL is empty: STOP next STOP $\odot$ CITY addr HOTEL=ϵ.

Definition 4.8 *Let S and S' be two subsets of $I(r)$ respectively $I(r')$. If $r \odot r' \neq \epsilon$ then the* concatenation *of S and S', denoted $S \odot S'$, returns a set of hyperwalks in $I(r \odot r')$ each element of which is the concatenation of a hyperwalk in S with a hyperwalk in S', denoted $h \circ h'$ and defined below.*

$$S \odot S' = \{h \circ h' \mid h \in S, h' \in S', h \circ h' \models r \odot r'\}$$

Let h and h' be two hyperwalks satisfying $r = \sum_{1 \leq i \leq n} r_i$ respectively $r' = \sum_{1 \leq j \leq m} r'_j$. The concatenation of h and h' contains for each pair r_i, r'_j where $r_i \odot r'_j \neq \epsilon$ the walk $w_i \circ w'_j$, where w_i (w'_j) in h (h') satisfies r (r'). Note that if for some $r_i \odot r_j \neq \epsilon$, $w_i \circ w'_j = \epsilon$, then $h \circ h'$ does not satisfy $r \odot r'$:

$$h \circ h' = \{w_i \circ w'_j \quad | \quad w_i \in h, w'_j \in h' \wedge w_i \models r_i, w'_j \models r'_j \wedge r_i \odot r'_j \neq \epsilon\}.$$

Example 4.5 *Assume that we have two sets of hyperwalks $S \subseteq I($STOP in CITY$)$ and $S' \subseteq I($CITY addr HOTEL + CITY addr RESTAURANT$)$. S contains stops with their cities and S' represents cities together with their hotels and restaurants. We would like to concatenate these information to obtain the hotels and restaurants which can be visited during a stop. For each pair of hyperwalks $h \in S$ and $h' \in S'$, we concatenate the walks in h with the walks in h' to get the set of hyperwalks $S \odot S'$ satisfying* hwe STOP in CITY addr HOTEL + STOP in CITY addr RESTAURANT.

Set **union** ($\cup$), **intersection** ($\cap$) and **difference** ($\setminus$) are defined as usual on two sets of hyperwalks satisfying the same *hwe*.

4.6 Examples

In the Introduction Section we gave some examples of queries on a graph database with a syntax a la SQL. We assumed in these examples that the type system is a relational one, i.e. any node or edge has a tuple structure where each attribute is of atomic type (string, integer, ...). For example a CITY node is structured as a tuple: [name: string, country: string, population: integer]. An *in* edge might contain the following information: [year: integer, month: integer, day: integer, duration: integer].

We shall now inspect some of the queries given in the Introduction Section and see how each of them can be solved by an expression of the above algebra. For each example, we give the corresponding (hyperwalk) algebraic expression.

1) All restaurants in the third district of Paris are obtained by a selection applied on all hyperwalks satisfying r=CITY addr RESTAURANT. Each target hyperwalk is projected on the RESTAURANT node:

$\pi_{RESTAURANT}(\sigma_{paris(\mathrm{CITY}) \wedge 3rdDistr(\mathrm{addr})}(r))$.

Observe that this is a regular relational selection/projection.

2) Assume somebody is addicted to McDonalds restaurants and Holiday Inn hotels. Those cities that meet his wishes can be found as follows. Let r=CITY addr RESTAURANT and r'=CITY addr HOTEL. As a matter of fact, we look for all cities with a McDonalds restaurant and join them with the cities with a Holiday Inn hotel:

$$\pi_{\mathrm{CITY}}(\sigma_{mcDonalds(\mathrm{RESTAURANT})} I(r) \bowtie \sigma_{holInn(\mathrm{HOTEL})} I(r'))$$

or

$$\pi_{\mathrm{CITY}}(\sigma_{mcDonalds(\mathrm{RESTAURANT}) \wedge holInn(\mathrm{HOTEL})} \mathrm{I}(\mathrm{r} + \mathrm{r}'))$$

where, e.g., the application of the Boolean function $mcDonalds$(RESTAURANT) corresponds to testing whether the attribute value of the restaurant name is "mcDonalds".

3) The last query could not have been solved with a relational query language. It illustrates the fact that our language supports recursion. We want to get all journeys which traverse Munich. It might be a journey that either starts in, or ends in, or goes through Munich:

$$\pi_{\text{JOURNEY}}(\sigma_{munich(\text{CITY})} I(\mathsf{JOURNEY\ first(STOP\ next)^*\ STOP\ in\ CITY})).$$

Cycles in the schema, e.g. the *next* edge, allow to construct hyperwalk expressions with Kleene closure $(*)$.

5 Application to Hypertexts

In the following, we sketch several mechanisms for integrating the query language into hypertext applications.

When navigating through the hypertext, a user can be positioned at arbitrarily many nodes at one time (nodes corresponding to documents displayed on the screen), called the current *user state* U [18]. U is changing according to the user's actions: clicking on an anchor results in adding the target node to U; closing a document removes it from U, etc.

5.1 Multiple Steps Navigation

Instead of navigating through the hypertext by selecting anchors step by step, in state U the user might be helped in reaching directly node y, following a path satisfying a query Q. This means that all nodes y reachable from a node x in U by a walk in Q are added to U, while document x is closed (x is deleted from U).

Example 5.1 *Figure 10-a shows a hypertext graph with some cities and the* Holiday Inn. U={Vienna, Salzburg, Munich, Paris}. *Query* $\sigma_{holiday\_inn}(\text{HOTEL})$ $I(\mathsf{CITY\ addr\ HOTEL})$ *adds* Holiday Inn *to* U *and removes all cities with a Holiday Inn from* U: U'={Vienna, Salzburg, Holiday Inn}

5.2 Map Supported Navigation

Browsers are visual representations of hypertext graphs that allow the user to change the user state interactively by selecting some displayed nodes. A graphical *schema browser* displays the schema subgraph corresponding to a *hwe* r. By selecting a node n in the schema browser, all nodes whose labels are of type n can be displayed.

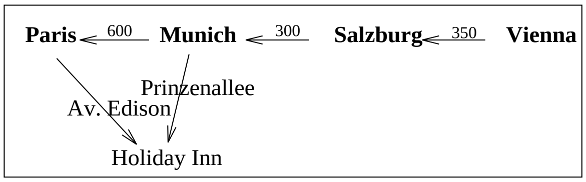


Figure 10: Query Navigation

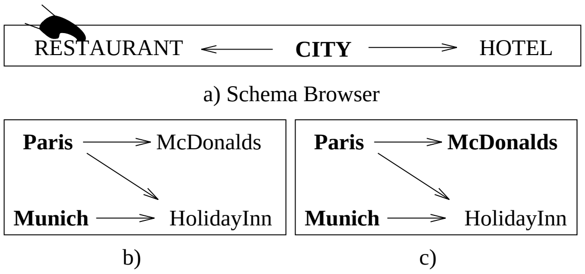


Figure 11: Map supported Navigation

Example 5.2 *By clicking on the node with label* RESTAURANT *(Figure 11-a),* McDonalds *is added to the user state* U *(Figure 11-b):* U'={McDonalds, Paris, Munich} *(Figure 11-c).*

5.3 Navigation Space Restriction

Several mechanisms for navigation space restriction have been proposed in the literature [11, 18]. The idea is to allow the user to select some subgraph for further navigation.

A query Q might then define a *script* or *guided tour* [1, 20]. The user navigates as usual, but being in a given node n, he may only follow those links which are contained in some hyperwalk in Q.

Example 5.3 *Assume a customer is reading some description about Paris and wants to get more information about the hotels in the* 15^{th} *district. The query* $\sigma_{15^{th}\_distr}(\text{addr})$ $I(\mathsf{CITY\ addr\text{-}HOTEL})$ *changes the displayed document such that only the anchors connecting the text to a description of a hotel in the* 15^{th} *district remain* `active`.

> ...A very nice hotel in the 15^{th} district is the `Imperial` hotel near the metro station Pasteur. From there you can visit *Eiffel Tower*, *Trocadero*, *Orsay Museum*, ...

6 Related Work

GraphLog [7, 8] is also a query language based on a graph representation of data and whose expressive power is equivalent to stratified linear Datalog [19]. We may underline a few significant differences between Gram and GraphLog: 1) the graph representation is not the same; in Gram, we do not make any assumption on the types of edges and nodes. Even if we assume relational types, it is not clear yet, whether the two graph representations are equivalent. 2) GraphLog [7, 8] is a query language based on graphs: a query is a graph pattern in which nodes are matched against nodes in the database and edges (specified by regular expressions on the values) are matched against paths in the database. In Gram, queries are not expressed by graphs. Instead, regular expressions on the (node and edge) types specify subgraphs (hyperwalks) in the database. 3) Finally it remains to compare the expressive power of Gram's algebra to that of GraphLog. We intend first to compare Gram's expressive power (with types restricted to be relational) to that of Datalog.

The graph model *GOOD* [13] provides a graphical language that allows to modify graph databases. Nodes and edges can be added and removed in subgraphs satisfying a given graph pattern. Opposed to GraphLog, the underlying data model is object oriented and functional edges are distinguished from non-functional (multivalued) edges.

Among other related approaches, [4] describes a query language for hypertexts which is based on modal logics and [5] is an extension of SQL for querying graph relations.

7 Future Work

7.1 Queries and Scripts

Assume an agency employee prepares a new trip using information already existing in the hypertext database. He would like to describe by a text document the overall organization of the journey, linking each stop description to an already existing document in the hypertext database.

He might create a new text document D (node of type JOURNEY), describing the beginning of the journey, then link it to the first city (link of type stop), then create a link back to description D. This step is repeated for each stop: each time a new anchor and links of type stop and back are added. The resulting journey satisfies hwe (JOURNEY stop CITY back)* JOURNEY. The disadvantage of such an approach is that the complexity of the database and the schema might increase unnecessarily: there exists a link back between each city and the description.

An alternate solution is to accept queries as virtual links instead of concrete edges. By activating an anchor, the user enters a guided tour defined by a query: together with anchor "Paris" in journey description D, we store the following query:

$$\sigma_{paris(\mathrm{CITY})} I(\ \mathsf{CITY\ addr\ MONUMENT} + \mathsf{CITY\ addr\ RESTAURANT}).$$

Then by activating anchor "Paris", the user can get information about monuments and restaurants in Paris.

Such a mechanism would allow to implement rather complex scripts, navigation strategies, strategies for returning to previous states etc. Compared to "hardwired" incremental updating of the hypertext, such a "programmable" approach of hypertext provides easy updating and customization, ease of changes in navigation strategies and reuse of existing hypertexts.

As an example, one can construct multiple level hypertexts from (low level) existing hypertext databases, in order to allow hierarchical navigation and zooming [10, 12].

7.2 Prototype

In order to validate the applicability of this model to hypertext querying, we are currently integrating [2] the hypermedia system MultiCard [17] with the object oriented database management system O_2 [3].

Acknowledgments

We are grateful to Ralf Güting. Deep exchanges with him on his graph data model [12] gave birth to this simpler, although less powerful, model. We would also like to thank Serge Abiteboul and Claude Delobel for helpful comments.

References

[1] F. Afrati and C. Koutras, A hypertext model supporting query mechanisms, *ECHT '90*, 52–66, Versailles, Nov. 1990.

[2] B. Amann and V. Christophides, Providing persistence to the hypermedia system MultiCard by using the OODBMS O_2, internal note, Verso project, INRIA, Sept. 1992.

[3] F. Bancilhon, C. Delobel, and P. Kannelakis, *The O_2Book*, Morgan Kaufmann, 1992.

[4] C. Beeri and Y. Kornatzky, A logical query language for hypertext systems, *ECHT '90*, 67–80, Versailles, Nov. 1990.

[5] J. Biskup, U. Räsch, and H. Stiefeling, An extension of SQL for querying graph relations, *Computer Languages*, 15(2), 65–82, 1990.

[6] B. Conklin, Hypertext: an introduction and survey, *IEEE Computer*, 20(19), 17–41, Sept. 1987.

[7] M. Consens and A. Mendelzon, Expresssing structural hypertext queries in GraphLog, *Proceedings of Hypertext '89*, Nov. 1989, 269–292.

[8] M. Consens and A. Mendelzon, GraphLog: a visual fromalism for real life recursion, *Proceedings of the ACM Symposium on Principles of Database Systems*, Nashville, Tennessee, 1990, 404–416.

[9] N. Delisle and M. Schwartz, Contexts: A partitioning concept for hypertexts, *ACM Transactions on Information Systems*, 5(2), 168–186, Apr. 1987.

[10] S. Feiner, Seeing the forest for the trees: Hierarchical display of hypertext structure, *Proceedings of the Conference on Office Information Systems*, Palo Alto, Calif., Mar. 1988, 205–212.

[11] R. Furuta and P. Stotts, Separating hypertext content from structure in trellis, *Hypertext: State of the Art*, 205–213, 1989.

[12] R. Güting, Extending a spatial database system by graphs and object class hierarchies, *Proceedings of the Int. Workshop on Database Management Systems for Geographical Applications*, Capri, May 1991, G. Gambosi, H. Six, and M. Scholl (eds.), Capri, May 1991.

[13] M. Gyssens, J. Paredaens, and D. V. Gucht, A graph-oriented object database model, *Proceedings of the ACM Symposium on Principles of Database Systems*, Nashville, Tennessee, 1990, 417–424.

[14] F. G. Halasz, Reflections on notecards: Seven issues for the next generation of hypermedia systems, *Communications of the ACM*, 31(7), 836–852, July 1988.

[15] F. G. Halasz and M. Schwartz, The Dexter hypertext reference model, *Proceedings of the NIST Hypertext Standardization Workshop*, Jan. 1990, 95–133.

[16] F. Harary, *Graph Theory*, Addison Wesley Series in Mathematics, 1971.

[17] A. Rizk and L. Sauter, Multicard: An open hypermedia system, this volume, 1992.

[18] F. W. Tompa, A data model for flexible hypertext database systems, *ACM Transactions on Information Systems*, 7(1), 85–100, July 1989.

[19] J. Ullman, *Principles of Database and Knowledgebase Systems*, volume 2, Computer Science Press, 1989.

[20] P. T. Zellweger, Active paths through multimedia documents, *Proceedings of the 1^{st} int. Conf. on Electronic Publishing: Document Manipulation and Typography*, Nice (France), Apr. 1988, 19–34.