Application of a Graph Model to Hypertext Querying*

BERND AMANN[‡], *MICHEL SCHOLL* [♯‡]

[‡] INRIA 78153 Le Chesnay Cedex France
[♯] Cedric/CNAM 292 rue St Martin 75141 Paris Cedex 03 France
Email: Bernd.Amann@inria.fr and Michel.Scholl@inria.fr

Abstract

We present a model for data organized as graphs with typed nodes and edges. Regular expressions over the types of the node and edge labels are used to select walks in the hypertext graph. An outline of the application of this model and its query language to the implementation of hypertext documents is given by using an extended example of a travel agency application.

1 Introduction

Recent database research work [7, 8, 11, 4, 16] 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 [6, 3] or geographic database [10] systems. Standard data models are often inefficient as they do not capture the inherent structure of data representing hypertext documents or networks (highways, rivers, ...).

In this paper, we address the problem of hypertext querying. 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[1] links between documents [13].

In the following, we take as an example a travel agency that organizes journeys. A journey corresponds to a document connected 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, and a sample of the hypertext graph is given in Figure 2. Note that links are typed and may contain information such as a date (link between stop descriptions and cities) or an address (links between cities and hotels, restaurants and monuments). Since hotels and restaurants like McDonalds can be described independently from the city where they are located, there exists only one description of their common characteristics. The information specific to a hotel or restaurant in a city, e.g. its address, is contained in a link.

Consider a travel agency customer reading some tourist 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 through the hypertext 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,

*Work partially supported by the French *Programme de Recherche Coordonnée BD3.*

[1] Links are not only between documents but between spans of characters, called *anchors*.

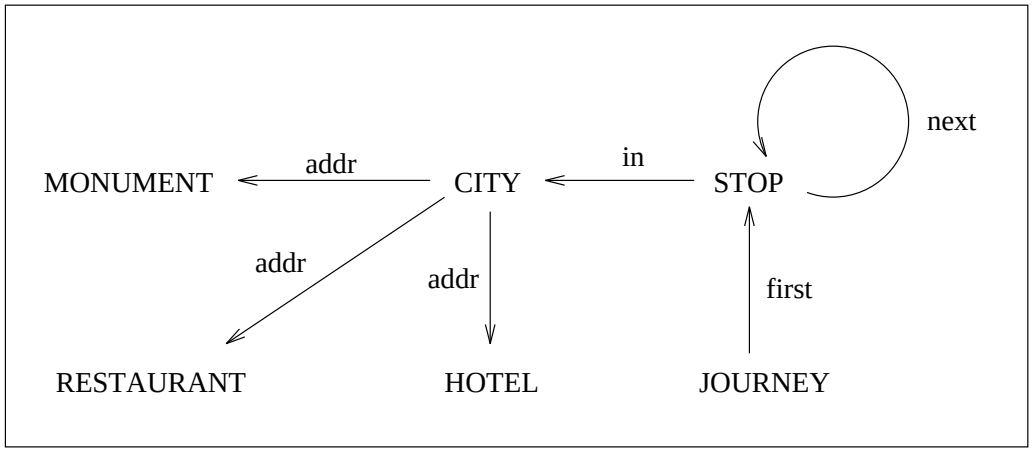


Figure 1: The Travel Agency Schema

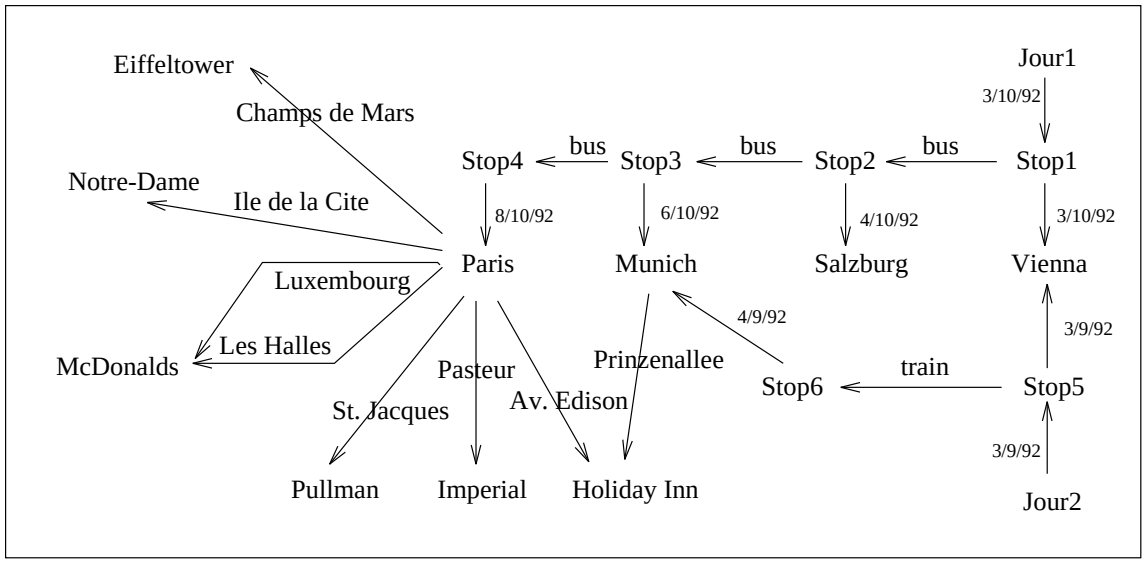


Figure 2: A Travel Agency Hypertext Graph

when the number of nodes and edges passes some limit (for example more than a few dozen documents [5]). Thus query languages become necessary to restrict the search space in navigational graphs [12].

We present a query algebra where regular expressions [15] 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 node (a node of type $JOURNEY$) to one of its stops in a city (node of type $CITY$). Our language is a variant of GraphLog [7, 8], in which regular expressions are also used for specifying paths in a graph. One main difference between GraphLog and our approach is that the former presents a graphical query language while we offer an algebraic language on regular expressions over types.

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 hypertext showing that there are a Holiday Inn and a McDonalds in Paris. Assume that for representing such informations we have two relations $HOT_CIT(HOTEL, ADDRESS, CITY)$ and $REST_CIT(RESTAURANT, ADDRESS, CITY)$.

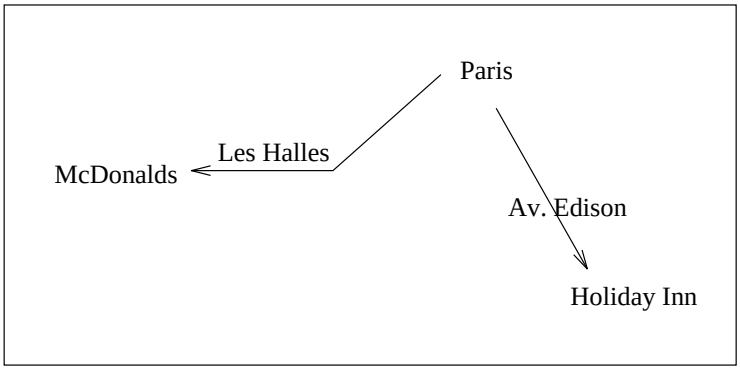


Figure 3: A Hyperwalk

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

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

A set of objects, an example of which is depicted in Figure 3, is selected and projected on the $CITY$ nodes. In our model, such an object 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 the hyperwalk expression $CITY\,address\,RESTAURANT + CITY\,address\,HOTEL$.

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 address RESTAURANT + CITY address 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 set of hyperwalks to be selected in the hyperdocument. The `select` and `where` clauses, as usual, specify the selection criteria, as well as what is to be projected.

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

```
select RESTAURANT
from CITY address RESTAURANT
where CITY.name="Paris" and address.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 address MONUMENT
where in.month="March" and MONUMENT.name="Mont St. Michel"
```

Section 2 defines hypertext schemas and graphs. 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. Finally in Section 5, an outline of the application of this model to hypertext querying is given.

2 Hypertext Graph

Hypertext Schema

Definition 2.1: A Hypertext Schema *is a directed weakly connected*[2] *labeled multigraph* $S = (N_S, E_S, \psi_S, \lambda_S, T)$ *where*

1. N_S *is a set of nodes.*
2. E_S *is a set of edges.*
3. ψ_S *is an incidence function from* E_S *into* $N_S \times N_S$.
4. T *is a set of labels.*
5. 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):* $n \neq n' \Rightarrow \lambda_S(n) \neq \lambda_S(n')$.
- *Note that* S *is a multigraph, i.e. a pair of nodes can be connected by more than one link. 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 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 labeled by $JOURNEY$, $STOP$, $CITY$, $RESTAURANT$, $HOTEL$ and $MONUMENT$ and 6 edges labeled by $first$, $next$, in and $address$.

Hypertext Graph

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

Definition 2.2: A Hypertext Graph *is a directed labeled multigraph* $G = (N, E, \psi, \lambda, V)$ *where*

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

[2] A directed graph is weakly connected if the corresponding undirected graph is connected [14]. From now on, connected will mean weakly connected.

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 hypertext schema S is a hypertext graph. An instantiation function I associates with each node and edge m in S a set of nodes respectively edges in G as follows: $I(m) = \{\mu \mid \tau(\mu) = m\}$. Note that a hypertext graph G is not necessarily connected (Figure 4-a). For an edge e in S

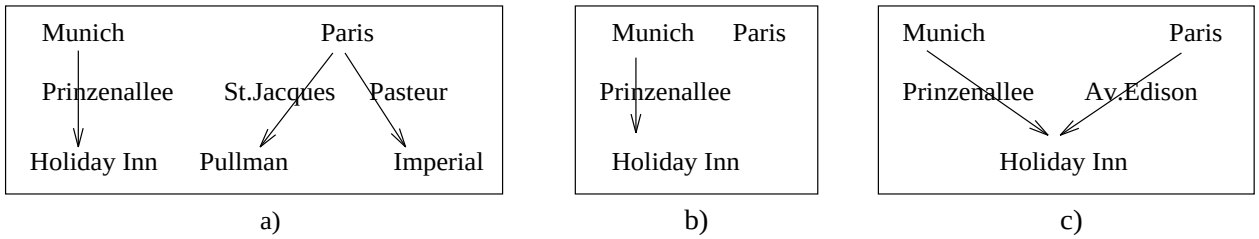


Figure 4: Travel Agency Hypertexts

from n_1 to n_2 there may be nodes in $I(n_1)$ respectively $I(n_2)$ where no edge (Figure 4-b) or more than one edge (Figure 4-a and 4-c) in $I(e)$ is leaving from (respectively coming in).

3 Hyperwalks and Hyperwalk Expressions

Walks and Hyperwalks

We consider walks as 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 [14]. For example, if somebody asks for the cities in a given journey, he chooses the walks in the graph starting from a node of type $JOURNEY$[3] and ending with a node of type $CITY$. 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-c shows a hyperwalk made of two walks connecting two city documents to the same hotel document.

In a forthcoming paper [1], we define an algebra on sets of hyperwalks whose structure is expressed by regular expressions over the set of types T, called *hyperwalk expressions*. In this paper we will only give an informal presentation of this model.

Walk and Hyperwalk Expressions

A *walk expression* is a regular expression over T without alternation (+), whose language[4] contains only alternating sequences of node and edge types, starting and ending with a node type. A walk expression r specifies all walks in a hypertext graph G whose structure is expressed by r. For example

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

[4] The language of a regular expression over T is defined as [15]:
$L(a) = \{a\}$ for all $a \in T$
$L(A\,B) = L(A) \times L(B) = \{a.b \mid a \in L(A) \text{ and } b \in L(B)\}$ for all expressions A and B over T
$L(A + B) = L(A) \cup L(B) = \{c \mid c \in L(A) \text{ or } c \in L(B)\}$ for all expressions over T
$L(A^*) = \bigcup_{i \geq 0} L(A^i)$ where $A^0 = \epsilon$ and $A^i = A\,A^{i-1}$ for all expressions A over T

$CITY\,address\,HOTEL$ is a walk expression *satisfied* by all walks of length 1 connecting nodes of type $CITY$ to nodes of type $HOTEL$ by an *address* link.

A *hyperwalk* in a hypertext graph G is a set of walks sharing nodes. More exactly, the walks of the set form together a connected subgraph of G. Figure 4-c is a hyperwalk opposed to Figures 4-a and 4-b, which are not hyperwalks, since their graphs are not connected. Let h be a hyperwalk containing walks $w_1, w_2, \ldots w_n$ where each walk w_i satisfies some walk expression r_i. The sum $r = \sum_{1 \le i \le n} r_i$ is a *hyperwalk expression*, *hwe* for short, *satisfied* by h. The formal definition of hyperwalk expressions and their satisfaction is given in [1].

Example 3.1:

1) The set $h = \{Jour1.3/10/92.Stop1, Stop1.bus.Stop2, \}$ is a hyperwalk satisfying the hyperwalk expression $r = JOURNEY\,first\,STOP + STOP.next.STOP$. The two walks in h share a node whose label $Stop1$ is of type $STOP$. Note that $\{Jour1.3/10/92.Stop1,\ Stop2.bus.Stop3, \}$ does not satisfy r. The two walks do not share any node. □

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

4 Hyperwalk Algebra

For querying a hypertext graph we define an algebra on sets of hyperwalks. We present the different operations of the algebra by a sample of queries on the travel agency hypertext (Figure 2).

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: the result of a query is itself a set of hyperwalks. Unary operations (*projection, selection, renaming*) take as a source a set S of hyperwalks in the hypertext graph 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.

The *instance* of a hyperwalk expression r in hypertext G is defined as the set of hyperwalks in G satisfying r: $I(r) = \{h \mid h \in G \wedge h \text{ satisfies } r\}$. For example $I(CITY\,address\,HOTEL)$ contains all walks connecting cities to their hotels.

Each walk has a value, called its *label*. The *label of a walk* w is obtained by substituting each node and edge in w with its label:

$$\lambda(n_1 e_1 n_2 \ldots e_{k-1} n_k) = \lambda(n_1)\lambda(e_1)\lambda(n_2)\ldots\lambda(e_{k-1})\lambda(n_k).$$

The *label of a hyperwalk* h is obtained by replacing each walk in h by its label:

$$\lambda(\{w_1, w_2, \ldots w_n\}) = \{\lambda(w_1), \lambda(w_2)\ldots\lambda(w_n)\}.$$

The first operation we present selects hyperwalks according to their labels.

The **Selection** operation σ evaluates Boolean functions (selection conditions) on hyperwalks. Applied to a set S of hyperwalks, it returns those hyperwalks in S whose labels satisfy the given condition. Let r be a hyperwalk expression and h be a hyperwalk satisfying r. A *selection condition* on r is constructed from Boolean functions that can be applied on the labels of the nodes and edges in h.

Let the Boolean function $vienna(CITY) : dom(CITY) \rightarrow \{true, false\}$ be true for all hyperwalks containing a $CITY$ node with name $Vienna$. To get all hotels in Vienna, we apply the selection operation σ with selection condition $vienna(CITY)$ on the hyperwalks satisfying $CITY\,address\,HOTEL$:

$$Q = \sigma_{vienna(CITY)} I(CITY\,address\,HOTEL)$$

As links are first class objects, we can also define selection conditions on link types. For example to get all journeys making a stop in Paris in April, we can apply the following selection condition on the instance of $JOURNEY\ first\ (STOP\ next)^*\ STOP\ in\ CITY$:

$$Q = \sigma_{paris(CITY) \wedge april(in)} I(JOURNEY\ first\ (STOP\ next)^*\ STOP\ in\ CITY)$$

where $paris(CITY)$ and $april(in)$ are two Boolean functions which are true for hyperwalks containing the city of Paris and an *in* link with a date in April.

Note that the result of this query is a set of walks, connecting the $JOURNEY$ nodes to the node representing the city of Paris. To keep only the descriptions of the journeys themselves we have to apply the projection operator:

Informally speaking, the **Projection** π of a hyperwalk h on a hyperwalk expression 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.

To get all hotels in Vienna, we first select the hyperwalks containing Vienna and then apply the projection operation π with hwe $HOTEL$:

$$Q = \pi_{HOTEL}(\sigma_{vienna(CITY)} I(CITY\ address\ HOTEL))$$

Now we present two operations, *join* and *concatenation*, that allow to construct new hyperwalks from two sets of hyperwalks.

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

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

For example, let $S \subseteq I(CITY\ address\ HOTEL)$ be a set of cities and their hotels. To get also the restaurants in these cities we join S with the instance of $CITY\ address\ RESTAURANT$:

$$Q = S \bowtie I(CITY\ address\ RESTAURANT)$$

The result of Q then satisfies hwe $CITY\ address\ HOTEL + CITY\ address\ RESTAURANT$. A hyperwalk in Q is for example $\{Paris.StJacques.Pullman, Paris.LesHalles.McDonalds\}$. It comes from the union of $\{Paris.StJacques.Pullman\}$ and $\{Paris.LesHalles.McDonalds\}$.

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.

The **Concatenation** $\odot$ 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 the starting node of a walk $w' \in h'$, then w and w' can be concatenated by replacing the ending node in w by w'. For example the concatenation of $Jour1.3/10/92.Stop1$ and $Stop1.bus.Stop2$ is $Jour1.3/10/92.Stop1.bus.Stop2$.

Assume that we have two sets of hyperwalks $S \subseteq I(STOP\ in\ CITY)$ and $S' \subseteq I(CITY\ address\ HOTEL + CITY\ address\ RESTAURANT)$. S contains stops with their cities and S' represents cities together with their hotels and restaurants. We would like to concatenate these informations 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 $STOP\ in\ CITY\ address\ HOTEL + STOP\ in\ CITY\ address\ RESTAURANT$.

The remaining operations are *Renaming* and the set operations *Union, Intersection* and *Difference* which are defined on sets of hyperwalks satisfying the same hyperwalk expression.

Hyperwalk expressions may have several occurrences of the same type. **Renaming** ρ allows one to distinguish between the different occurrences of a given type in a given hwe. For example $h = \{Stop1.bus.Stop2\}$ is a hyperwalk satisfying $r = STOP.next.STOP$. To distinguish between the two occurrences of $STOP$ in r, we rename the second occurrence into $STOP'$: $\rho_{STOP\,next\,STOP'}(STOP\,next\,STOP)$. Then $STOP$ specifies the first stop in h ($Stop1$) and $STOP'$ specifies the second stop, which is $Stop2$.

Let S and S' be two sets of hyperwalks in $I(r)$.

1. The **Union** of S and S', denoted $S \cup S'$, returns the set of all paths which are in S *or* S'.
2. The **Intersection** of S and S', denoted $S \cap S'$, returns the set of all paths which are *both* in S *and* S'.
3. The **Difference** between S and S', denoted $S \setminus S'$, returns the set of all paths which are in S *and not in* S'.

Examples

In the Introduction we gave some examples of queries on a hypertext 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* link 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 and see how each of them can be solved by an expression of the above algebra.

1) Assume somebody is addicted to McDonalds restaurants and Holiday Inn hotels. Those cities that meet his wishes can be found as follows:

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

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:

$$\pi_{CITY}(\ \sigma_{mc_donalds(RESTAURANT)} I(CITY\,address\,RESTAURANT) \bowtie \sigma_{hol_inn(HOTEL)} I(CITY\,address\,HOTEL))\ \square$$

2) All restaurants in the third district of Paris are selected by the following query:

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

All hyperwalks satisfying $CITY\,address\,RESTAURANT$ such that the city's name is Paris and the restaurant is in the third district are selected. For each hyperwalk we are interested only in the $RESTAURANT$ document:

$$\pi_{RESTAURANT}(\sigma_{paris(CITY) \wedge third_distr(address)} I(CITY\,address\,RESTAURANT))$$

Observe that this is a regular relational selection/projection. In a hypertext such a query would correspond to the following situation. A document (node contents) representing Paris is displayed on the screen and we would like to navigate through all restaurants in the third district. □

3) The last query could not have been solved with a relational query language. It illustrates the fact that our language, as GraphLog [8], 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:

```
select JOURNEY
from JOURNEY first (STOP next)* STOP in CITY
where CITY.name= "Munich
```

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

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

5 Future Work

When navigating through the hypertext, a user can be positioned at arbitrarily many nodes at one time. This set of nodes will be called the current *user state* U [17]. U is changing constantly according to the user's actions: clicking on an anchor results in adding the target document (node) to U; closing a document removes it from U, etc.

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

5.1 Multiple Stops Navigation

Instead of navigating through the hypertext by selecting anchors step by step, in state x 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 the user state.

Example 5.1: Figure 5-a hypertext graph with two cities (Paris and Munich), a hotel (Holiday Inn) and a restaurant (McDonalds). The nodes in the current user state U are in **boldface**: $U = \{Paris, Munich\}$. The simple query $I(CITY\,address\,RESTAURANT)$ marks all restaurants where there exists a city in the current user state (figure 5-b)). □

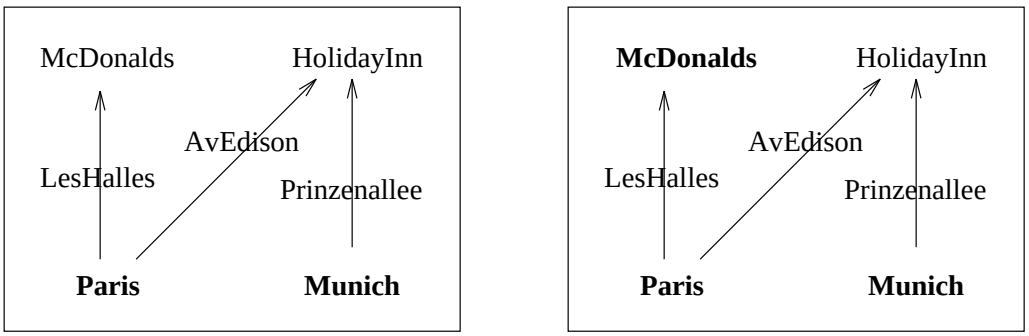


Figure 5: A User State

5.2 Navigation Space Restriction

Several mechanisms for navigation space restriction have been proposed in the literature [9, 17]. The idea is to allow the user to select some subgraph of the hypertext graph for further navigation and manipulation.

The hyperwalks of the result of a query Q define a *script* or *guided tour* [20] (with more than one starting node) useful for several applications. The user navigates as usual, but he may only follow paths in Q. In a given node n, he may only follow those links which are contained in some hyperwalk in Q; the other links are inactivated.

Example 5.2: 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 $\sigma_{15^{th}_distr(address)}I(CITY\,address\,HOTEL)$ 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`.

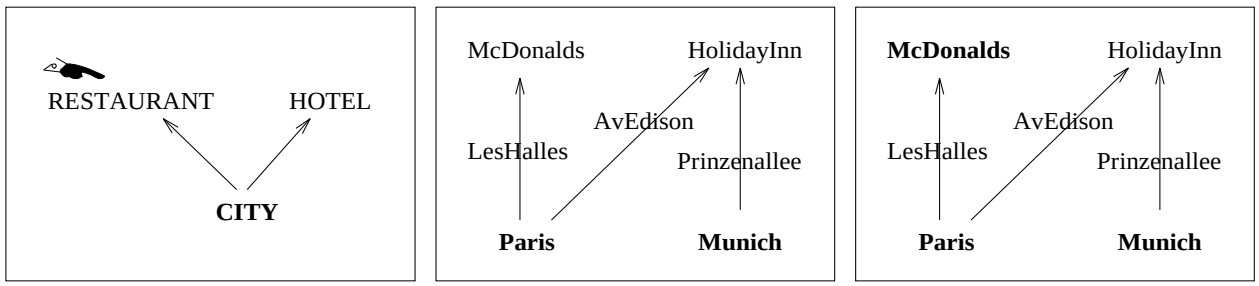
> ...A very nice hotel in the 15^{th} district is the `Imperial` hotel near the metro station Pasteur. From there you can visit *Eiffel Tower*, *Trocadero*, *Orsay Museum*, ...

□

Browsers are visual representations of hypertext graphs that allow the user to change the user state interactively by selecting some displayed nodes. Another possibility is not to display the instance of the query's answer but the corresponding schema subgraph.

A graphical *schema browser* displays the schema graph of Q on the screen. By selecting a node n in the schema browser, all nodes whose labels are of type n and which are the target nodes of some walk in Q are displayed.

Example 5.3: The schema in Figure 5-a can be considered as the schema graph of the query $I(CITY\,address\,RESTAURANT + CITY\,address\,HOTEL)$. By clicking on the node with label $RESTAURANT$ (Figure 5-a), the user state $U = \{Paris, Munich\}$ (Figure 6-b) is changed into $U' = \{McDonalds, Paris, Munich\}$ (Figure 6-c). □



a) Schema Browser b) c)

Figure 6: Schema Browser (a) and Hypertext Graphs (b and c)

5.3 Prototype and Expressive Power

In order to validate this model we are currently integrating the hypermedia system MultiCard developed by Bull S.A. with the object oriented database system O_2 [2] developed by O_2Technology.

The expressive power of the query language will be compared to Datalog [18, 19] with the assumption that types are those of the relational model.

Acknowledgements

We are grateful to Ralf Güting. Deep exchanges with him on his powerful graph data model [10] gave birth to this simpler, although less powerful, model.

References

[1] B. Amann and M. Scholl. Gram: A graph data model and query language. In *Proc. of the VIIIème Journées Bases de Données Avancées*, pages 86–101, Trégastel, France, September 1992.

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

[3] C. Beeri and Y. Kornatzky. A logical query language for hypertext systems. In *Proc. of the 1st European Conf. on Hypertext and Hypermedia (ECHT'90)*, pages 67–80, Versailles, November 1990.

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

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

[6] M.P. Consens and A.O. Mendelzon. Expressing structural hypertext queries in GraphLog. In *Proc. of Hypertext '89*, pages 269–292, November 1989.

[7] M.P. Consens and A.O. Mendelzon. GraphLog: a visual fromalism for real life recursion. In *Proc. of the ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pages 404–416, Nashville, Tennessee, 1990.

[8] I.F. Cruz, A.O. Mendelzon, and P.T. Wood. A graphical query language supporting recursion. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 323–330, San Francisco, 1987.

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

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

[11] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object database model. In *Proc. of the ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pages 417–424, Nashville, Tennessee, 1990.

[12] 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.

[13] F.G. Halasz and M. Schwartz. The dexter hypertext reference model. In *Proc. Hypertext Standardization Workshop, NIST*, page January, 1990.

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

[15] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison–Wesley Series in Computer Science, 1979.

[16] A. Rosenthal, S. Heiler, U. Dayal, and F. Manola. Traversal recursion: A practical approach to supporting recursive applications. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 166–176, 1986.

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

[18] J.D. Ullman. *Principles of Database and Knowledgebase Systems*, volume 1. Computer Science Press, 1988.

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

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