Online Maintenance of Minimal Directed Hypergraphs

Giuseppe F. Italiano
Umberto Nanni

CUCS-435-89

ONLINE MAINTENANCE OF MINIMAL DIRECTED HYPERGRAPHS

Giuseppe F. Italiano [†‡](1), Umberto Nanni [†](2)

[†] Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza", Roma, Italy

[‡] Department of Computer Science
Columbia University, New York, NY 10027

ABSTRACT

In this paper we deal with directed hypergraphs, a generalization of directed graphs previously introduced in the literature. In particular, we investigate the problem of maintaining efficiently information about minimal hyperpaths while new hyperarcs are inserted. We consider several definitions of minimal hyperpath and we prove that accordingly to some of such definitions the problem of finding the minimal hyperpath is NP-complete, while hyperpaths with minimal GAP and minimal RANK can be found in polynomial time. We deal with this problem in an *online* fashion, by allowing insertions of hyperarcs in the hypergraphs. We present data structures and algorithms which allow to return a hyperpath with minimal GAP or RANK between an arbitrarily given pair of nodes in a time which is linear in its size. The total time required to maintain the data structure during the insertion of new hyperarcs is $\Theta(m\ n^2)$ for min-GAP and $\Theta(m\ n^2 \log n)$ for min-RANK (where m is the total size of the description of the hyperarcs and n is the number of nodes). These results are useful in applications where directed hypergraphs are known to be a suitable model (e.g. transition networks, rewriting systems, database schemes, logic programming and problem solving).

1 - INTRODUCTION

Various kinds of hypergraphs have been extensively used in computer science as a mathematical model to represent concepts and structures from different areas: transition networks, rewriting systems, databases, logic programming, problem solving. In all cases hypergraphs generalize the concept of graph in the sense that they consist of a set of nodes and a set of hyperedges (or hyperarcs) defined over the nodes. A model that has been used in several applications consists of *directed hypergraphs* . In a directed hypergraph a hyperarc is defined by a pair (X,i) where X is an arbitrary nonempty set of nodes and i is a node. Directed hypergraphs have a wide range of applications in computer science [B 77, GMM 81, Y 82, ADS 83, AI 87].

A first application arises in database theory, where directed hypergraphs may provide a natural representation for functional dependency [ADS 83, ADS 85]. Another area in which hypergraphs provide a valuable representation is problem solving. Directed hypergraphs may be interpreted in problem solving as *and-or graphs*, to describe the relationship existing among a given problem P and the set of problems whose solution is required to solve P [GMM 81, N 82]. Another interesting application of directed hypergraphs arises in the representation and manipulation of *Horn formulae*. Among various classes of logical formulae, Horn formulae are particularly interesting in view of the fact that in Knowledge Based Systems [K 79] knowledge is often represented by means of *if ... then ...* clausal rules. Also in the case of propositional Horn formulae, the use of directed hypergraphs is quite natural. In particular, each Horn clause corresponds to a hyperarc and testing the implication between propositional variables corresponds to checking the existence of a hyperpath between two nodes. A particularly interesting case is when, in the process of building a Horn formula which represents our knowledge on a given domain by progressively adding new clauses, we want to check online the existence of a hyperpath from T to F (two special nodes

(1) Partially supported by NSF grants CCR-86-05353, CCR-88-14977 and by an IBM Graduate Fellowship.

(2) Partially supported by SELENIA S.p.A. and by M.P.I. National Project on "Algoritmi e Strutture di Calcolo".

corresponding to the truth values *true* and *false*), because such a hyperpath would imply the unsatisfiability of the whole formula [DG 84, AI 87]. Finally, an interesting area we are going to investigate for possible applications of our ideas are Petri nets [P 66] or, more in general, the transition networks or parallel computation. Here directed hypergraphs are supposed to be a suitable model for the static properties of the networks.

In this paper, we focus on the *online* maintenance of *minimal* hyperpaths. In the case of simple directed graphs there exists a unique natural definition of minimal path, and the *offline* version of this problem have been extensively studied in the literature. The dynamic problem is considered for example in [R 85]. We investigate the problem of maintaining efficiently information about minimal hyperpaths while new hyperarcs are inserted. In the case of directed hypergraphs, there are several definition of minimality for hyperpaths. We prove that accordingly to some of such definitions the problem of finding the minimal hyperpath between a pair of nodes is NP-complete, while others are tractable and we give efficient online algorithms. Our solution is efficient in terms of *amortized* time [T 85]. In other words a single insertion might be expensive, but the overall time for the whole input sequence is efficient. We consider a dynamic environment in which we are given a static set of nodes N and an initially empty set H of hyperarcs, and we are allowed to perform an arbitrary sequence of operations of the following kinds:

(a) inserting a new hyperarc $\langle X,y\rangle$ in H;
(b) asking for the value of minimal gap (rank) from a single node x to a node y;
(c) asking for a hyperpath from x to y with minimal gap (rank).

We will also consider the more general case where (b) and (c) may start from a generic source set.

We present data structures and algorithms which allow to return a hyperpath with minimal GAP or RANK between an arbitrarily given pair of nodes in a time which is linear in its size. The total time required to maintain the data structure during the insertion of new hyperarcs is $\Theta(m\, n^2)$ for min-GAP and $\Theta(m\, n^2 \log n)$ for min-RANK (where m is the total size of the description of the hyperarcs and n is the number of nodes). This means that the amortized cost is $\Theta(n^2)$ or $\Theta(n^2 \log n)$ per insertion respectively in the case of maintenance of minimal GAP or RANK. To correctly evaluate these performances, we remind that in the case of directed hypergraphs m can be exponential in n.

2 - BASIC DEFINITIONS AND REPRESENTATION OF DIRECTED HYPERGRAPHS

A directed hypergraph (from now on referred to as dhg) is a generalization of the concept of directed graph. Given a set N of nodes, we will denote as P(N) the power set of N.

Definition 2.1 A *directed hypergraph* $\mathcal{H}$ is a pair $\langle N,H\rangle$ where N is a set of *nodes* and H is a set of *hyperarcs*. Each hyperarc is an ordered pair $\langle S,i\rangle$ from an arbitrary nonempty set $S \in P(N)$ (*source set*) to a single node $i \in N$ (*target node*).

The basic parameters which will be taken into account in order to discuss the complexity of algorithms on directed hypergraphs are: the number of nodes (n_s), the number of hyperarcs (h), the number of source sets (n_c), the *source area*, that is the sum of cardinalities of all source sets (a), and the overall length of the description of the hypergraph ($m=|\mathcal{H}|$). If we represent a directed hypergraph by means of adjacency lists we have that $m = a + h$. According to the same representation the number of source sets is the same as the number of adjacency lists.

Definition 2.2 Given a hypergraph $\mathcal{H}=\langle N,H\rangle$, a nonempty subset of nodes $X \subseteq N$ and a node $y \in N$, a *hyperpath* $hp_{X,y}$ from X to y is defined to be a (minimal) set of hyperarcs such that one of the following conditions holds:

- $y \in X$ and $hp_{X,y}$ is empty (*extended reflexivity*);
- there is a hyperarc $\langle X',y\rangle \in hp_{X,y}$ and, for each node $x \in X'$, there is a hyperpath $hp_{X,x'}$ with $hp_{X,x'} \subseteq hp_{X,y}$ (*extended transitivity*).

In figure 1 a dhg is shown, where any arrow, corresponding to a hyperarc, can have a single or multiple source set. In figure 2 a hyperpath $hp_{A,G}$ is shown (we will often use this simpler notation instead of $hp_{\{A\},G}$).

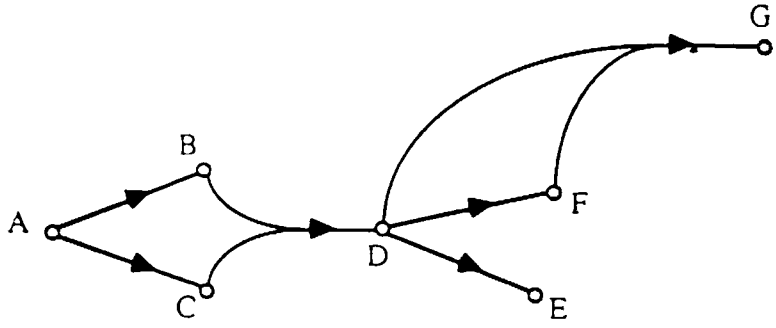

Figure 1: a directed hypergraph

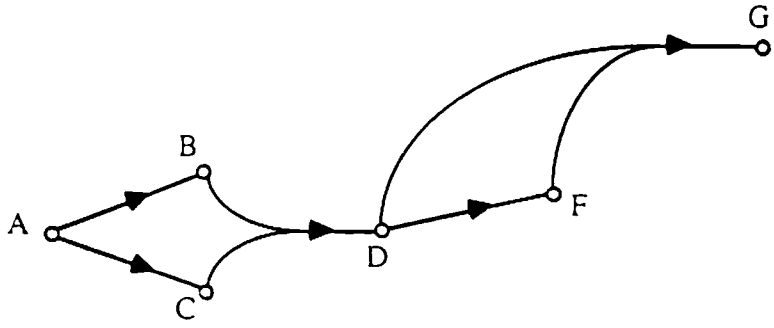

Figure 2: a hyperpath from {A} to G

In order to design efficient algorithms for the manipulation of directed hypergraphs, a data structure has been introduced for representing a dhg [ADS 83], which essentially is a simple graph with two kinds of nodes and two kinds of arcs.

Definition 2.3 Given a hypergraph $\mathcal{H}=\langle N,H\rangle$, let S_H be the set of non singleton source set, i.e. $S_H = \{X \mid \text{there exists a hyperarc } (X,i) \in H \text{ and } |X|>1\}$. The *FD-graph* of $\mathcal{H}$ is the labelled graph $G(\mathcal{H})=\langle N_H, A_f, A_d\rangle$, where:

N_c is a new set of nodes called *compound nodes* which is in bijective relationship with S_H (we will denote as X both the source set $X \subseteq N$ and the corresponding compound node $X \in N_c$). Each simple node j in the source set X will be called a *component* node of the compound node X.

$N_H = N \cup N_c$ is the set of nodes where N is called the set of *simple nodes*.:

$A_f \subseteq N_H \times N = \{(X,i) \mid \text{there exists a hyperarc } (X,i) \text{ in } \mathcal{H}\}$ is the set of arcs referred to as *full arcs*;

$A_d \subseteq N_c \times N = \{(X,j) \mid X \in N_c \text{ and } j \in X\}$ is the set of arcs referred as *dotted arcs*.

In figure 3 it is shown the FD-graph corresponding to the hypergraph in figure 1.

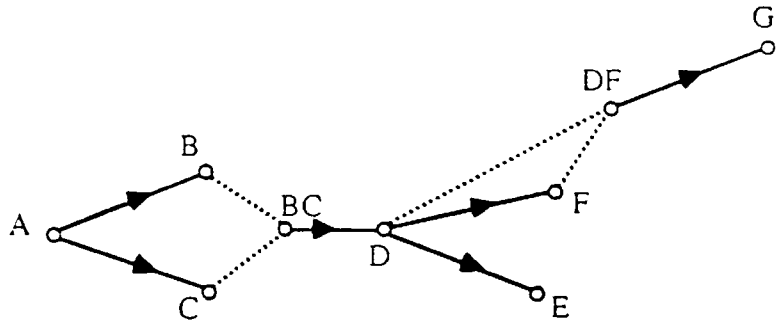

Figure 3: the FD-graph corresponding to the hypergraph in figure 1

Definition 2.4 Given an FD-graph $G(\mathcal{H})=<N_H,A_f,A_d>$, a nonempty subset of nodes $X\subseteq N_H$ and a node $i\in N_H$, an *FD-path* from X to i is a graph $G'(\mathcal{H})=<N'_H,A'_f,A'_d>$ which is a minimal subgraph of $G(\mathcal{H})$ and such that:

(i) $(X,i) \in A'_f \cup A'_d$,
or

(ii) there is a simple node j, a full arc $(j,i)\in A'_f$ and an FD-path from X to j in $G'(\mathcal{H})$,
or

(iii) there exists a compound node $Y\in N'_H$, a full arc $(Y,i)\in A'_f$ and, for each node $j\in Y$, we have $(Y,j)\in A'_d$ and there exists a FD-path from X to j in $G'(\mathcal{H})$.

3 - MINIMAL HYPERPATHS

We now introduce some parameters to characterize a hyperpath. While in the case of graphs the only way to characterize a path is to provide its length, in the case of hypergraphs, hyperpaths have a much more complex structure and different concepts can be provided to capture their size. We introduce the following terminology, some of which borrowed from [ADS 86].

Definition 3.1 The *gap* $g(hp_{X,y})$ of a hyperpath $hp_{X,y}$ is inductively defined as follows:

- if $hp_{X,y}$ is defined by extended reflexivity then
$$g(hp_{X,y})=0$$
- else, if $hp_{X,y}$ is defined by extended transitivity, i.e. there is a hyperarc $<X',y>\in hp_{X,y}$, then
$$g(hp_{X,y}) = 1 + \min_{x\in X'}\{g(hp_{X,x})\}$$

Definition 3.2 The *rank* $r(hp_{X,y})$ of a hyperpath $hp_{X,y}$ is inductively defined as follows:

- if $hp_{X,y}$ is defined by extended reflexivity then
$$r(hp_{X,y})=0$$
- else, if $hp_{X,y}$ is defined by extended transitivity, i.e. there is a hyperarc $<X',y>\in hp_{X,y}$, then
$$r(hp_{X,y}) = 1 + \max_{x\in X'}\{r(hp_{X,x})\}$$

Definition 3.3 The *number of hyperarcs* $n(hp_{X,y})$ of a hyperpath $hp_{X,y}$ is simply defined to be the cardinality of $hp_{X,y}$.

Definition 3.4 The *number of source sets* $ss(hp_{X,y})$ of a hyperpath $hp_{X,y}$ is defined as the cardinality of the set of source sets in $hp_{X,y}$ or, more formally:

$$ss(hp_{X,y}) = |\{S \mid \langle S,j\rangle \in hp_{X,y} \text{ for some } j\}|$$

Definition 3.5 The *source area* $a(hp_{X,y})$ of a hyperpath $hp_{X,y}$ is defined as the cardinality of the set of nodes which are in a source set of some hyperarc in $hp_{X,y}$ or, more formally:

$$a(hp_{X,y}) = |\{x \mid x \in S \text{ and } \langle S,j\rangle \in hp_{X,y} \text{ for some } j\}|$$

or

$$a(hp_{X,y}) = |\cup_{\langle S_i,j\rangle \in hp_{X,y}} S_i|$$

Definition 3.6 The *size* $s(hp_{X,y})$ of a hyperpath $hp_{X,y}$ is defined as:

$$s(hp_{X,y}) = a(hp_{X,y}) + n(hp_{X,y})$$

The size corresponds to the overall length of the description of $hp_{X,y}$.

Given a hypergraph $\mathcal{H}$, a set of nodes X, and a single node y, we are interested to investigate the problem of finding a minimal hyperpath from X to y according to any of the previous definitions. In parallel computation, for example, the minimal GAP of a hyperpath may be interpreted as the minimum number of subsequent transitions required for a *partial* result, while, perhaps more significantly, the minimal RANK may be interpreted as the maximum number of required transitions. An efficient solution for the dynamic problem allows to study some properties of the networks in real time, i.e. while modifications to the network are performed.

As far as finding hyperpaths with minimum number of hyperarcs, number of source sets, source area, and size, the concerned problems are shown to be NP-complete.

Theorem 3.1 Let $\mathcal{H}=\langle N,H\rangle$ be a directed hypergraph, x and y be two nodes in N, and n,ss,a,s be positive integers. The following four problems are NP-complete.

- finding whether there exists a hyperpath $hp_{x,y}$ with n hyperarcs or less;
- finding whether there exists a hyperpath $hp_{x,y}$ with ss source sets or less;
- finding whether there exists a hyperpath $hp_{x,y}$ with source area a or less;
- finding whether there exists a hyperpath $hp_{x,y}$ with size s or less.

Sketch of the proof: (by reduction to Minimum Cover). It is easy to check the membership in NP for all the above problems. Let $A=\{a_1,a_2,\ldots,a_m\}$ be a set of elements and $S=\{S_1,S_2,\ldots,S_M\}$ be a family of subsets of A such that $\cup_{i=1,2,\ldots,M}(S_i)=A$. The problem MC(A,S,k) is of deciding whether there exists a subfamily S' of S such that $|S'| \le k$, and $\cup_{S_i \in S'} S_i = A$. This problem is known to be NP-complete [GJ 79]. We can associate to the problem MC the hypergraph $H=\langle N,H\rangle$ such that each element $a_j \in A$ and each subset $S_i \in S$ have a corresponding node (which will be denoted in the same way) belonging to N: $N = A \cup S \cup \{p,q\}$. See figure 4 below.

Besides, the set H of hyperarcs $H = H_1 \cup H_2 \cup H_3$ is such that:

$H_1 = \{\langle p,S_i\rangle \mid S_i \in S\}$, $H_2 = \{\langle S_i,a_j\rangle \mid S_i \in S \text{ and } a_j \in S_i\}$, and $H_3 = \{\langle A,q\rangle\}$.

We remark that $\mathcal{H}$ does contain a hyperpath from p to q. Let us consider any hyperpath $hp'_{p,q}$ in $\mathcal{H}$ from p to q: it must consist of a set $hp'_{p,q} \subseteq H$ of hyperarcs which can be partitioned into three proper subsets H'_1, H'_2, and H'_3, where:

$$H'_1 \subseteq H_1, \quad H'_2 \subseteq H_2, \quad \text{and} \quad H'_3 = H_3.$$

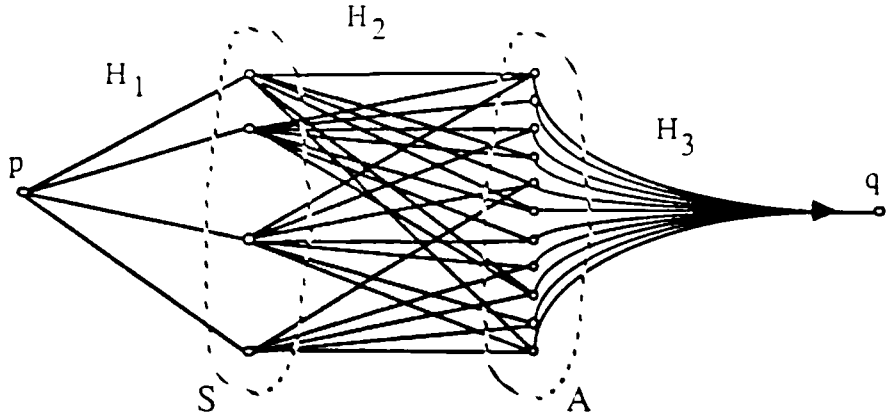

Figure 4: the hypergraph associated to an instance of the set covering problem (any arc is oriented toward the right)

For each $a_j \in A$, there exists $S_i \in S'$ such that $\langle S_i, a_j \rangle \in H'_2$, and $\langle p, S_i \rangle \in H'_1$. Furthermore we remark that, for each possible hyperpath from p to q, exactly n arcs must belong to H'_2, accordingly to the requirement of minimality in the definition of hyperpath. The source area of such a hyperpath can be computed as:

$$a(hp'_{p,q}) = 1 + |S'| + |A| = 1+k+m$$

The set of nodes S' individuated by the hyperpath $hp'_{p,q}$ provides a feasible solution to the problem MC(A,S,k) and, vice versa, given a feasible solution with size $\leq k$ for the problem MC, that is a subfamily $S'=\{S'_1, S'_2, \ldots, S'_k\} \subseteq S$ such that $\cup_{i=1,2,\ldots,k}(S'_i)=A$, we can easily find a hyperpath $hp'_{p,q}$ with source area $\leq a = 1+k+m$. Therefore, the problem of finding a solution for the problem MC(A,S,k) can be solved if and only if we are able to find a hyperpath with source area $\leq a = 1+k+m$ in the hypergraph $\mathcal{H}$. The same example provides a proof of the equivalence between MC(A,S,k) and any of the other considered problems, i.e. finding a hyperpath in $\mathcal{H}$ with:

number of hyperarcs $\leq n = k+m+1$
number of source sets $\leq ss = 2+k$
size $\leq s = 2+2k+2m$.

Q.E.D.

As far as hyperpaths with minimal gap or rank are concerned, the problem can be shown to be polynomially solvable and an offline algorithm to get a hyperpath from a generic source set X to a node y requires $\Theta(m|X|) \leq \Theta(mn_S)$ time where m is the size of the description of $\mathcal{H}$.

4 - ONLINE MAINTENANCE OF MINIMAL HYPERPATHS

In this section we show how to perform an arbitrary sequence of operations of the following kinds:

(a) insert a new hyperarc $\langle X, y \rangle$;
(b) ask for the value of minimal gap (rank) from a single node x to a node y;
(c) ask for a hyperpath from x to y with minimal gap (rank).

In this case we might apply one of the following naive strategies:

(1) - In order to speed up answers to requests (b) and (c) we might use the offline algorithm to recompute the hyperpaths with minimal gap (rank) any time an operation of type (a) is performed. This could require $\Theta(mn)$ time per hyperarc insertion.

(2) - Apply the $\Theta(m)$ algorithm for each request of type (b) and (c).

An arbitrary sequence of $\Theta(m)$ operations could produce, in the worst case, $\Theta(m^2n)$ time with the strategy (1) and $\Theta(m^2)$ with the strategy (2). In the following we propose dynamic data structures and online algorithms for this problem which have a

worst case total time complexity of $O(mn^2)$ for minimal gap, and of $O(mn^2 \log n)$ for minimal rank, corresponding to an amortized time complexity of $O(n^2)$ and $O(n^2 \log n)$ respectively per insertion. As far as the operations of type (b) we get answers in constant time, and for type (c) we get answers in linear time with the size of the output (i.e., optimal).

Note that in order to maintain efficiently dhg's in an online fashion, we need a representation of FD-graphs such that new compound nodes can be inserted. In fact, while in the case of the static problem we know all the simple and compound nodes of the hypergraph, when dynamic dhg's are considered also new compound nodes might be inserted in the data structure for each hyperarc insertion. We suppose that the set of simple nodes is given, but the number of new compound nodes is not known *a priori* and the order of insertions of hyperarcs in the structure is arbitrary. In such a situation an efficient dynamic data structure for compound nodes has to be maintained. The idea underlying the data structure is to maintain the FD-graph corresponding to the original directed hypergraph. In such a representation, a simple node can be accessed by means of full arcs only, while compound nodes are accessed by means of dotted arcs. In what follows, we shall deal only with FD-graphs and FD-paths, but every result may be directly formalized in terms of directed hypergraphs and directed hyperpaths. When no danger of confusion arises, the two formalisms will be used interchangeably.

We now present algorithms and data structures to maintain hyperpaths with minimal GAP from any simple node to any other (simple or compound) node. This is done for the sake of clarity but any result can be properly extended, with a slight modification, to the case in which we maintain minimal hyperpaths *from* any compound node and/or any arbitrary set of simple nodes specified at any time during the insertions. The overall time and space complexity of such a problem can be expressed in terms of the number of source sets (being compound nodes in the FD-graph or not) we want to maintain.

The Data Structures

In order to deal with the first simpler problem (hyperpaths with minimal GAP from any simple node), we maintain the following data structures. FD-graphs are implemented by maintaining adjacency lists for each (simple or compound) node. In more detail, all the full [dotted] arcs leaving a node x are organized in the lists $L_f(x)$ [$L_d(x)$]. Obviously $L_d(x)$ will be empty for any compound node x. FD-paths with minimal gap are retrieved using an $n_S \times n_S$ array LAST, containing the collection of *backward pointers*, defined as follows. LAST[i,j] points to the last (simple or compound) node (except j) in the minimal FD-path from the simple node i to the simple node j. If no such FD-path exists, then LAST[i,j] has a special value *nil*. With this additional information, the existence of an FD-path from a simple node i to a simple node j can be checked in constant time by examining the [i,j]-th entry of the array LAST. If LAST[i,j] is null, then there is no FD-path from node i to node j, otherwise a minimal FD-path can be traced out by starting from j and proceeding in a depth first backward fashion as shown for instance in [ALN 89]. Notice that there is no need to store the node which has to be followed starting from a compound node during this backward search since in this case, in order to trace an FD-path, we are forced to go back to all the simple nodes contained in the corresponding source set.

To properly update the information contained in the array LAST, we introduce n_S arrays GAP of size n_S (one for each simple node) which have the following property:

$GAP_y[x] = k$, if there is an FD-path of minimal gap k from the simple node x to the simple node y.

$GAP_y[x] = +\infty$, if there is no FD-path from the simple node x to the simple node y.

Analogously, for each compound node w, we introduce an array of size n_S called GAP_w for which the following invariant is maintained:

$GAP_w[x] = \min \{ +\infty, \min \{GAP_y[x] \mid y \in w\} \}$, for each simple node x.

With this invariant, $GAP_w[x]$ is equal to $+\infty$ if the compound node w is not reachable from x. Otherwise, it gives the minimal gap of an FD-path from x to w.

We now describe how to maintain the information in the array LAST during the insertion of new hyperarcs in the original directed hypergraph. In order to accomplish this task, we associate an array[1..n_S] referred to as $REACH_y$ to each simple node y in N_S. In the course of the update algorithms, the following invariant is maintained:

$REACH_y[x] = 0$, if there is an FD-path in the FD-graph from the simple node x to the simple node y.

$REACH_y[x] = 1$, otherwise.

The array REACH of size n_S is defined also for compound nodes and the following invariant is maintained for each compound node x (with components $x_1, x_2, \ldots, x_q$) and for each simple node i:

$$REACH_x[i] = \Sigma_{(k=1,\ldots,q)}\{REACH_{x_k}[i]\}.$$

That is, the entry $REACH_x[i]$ is equal to the number of simple nodes in X which are not reachable from the simple node i. Any array $REACH_x$ is initialized when the compound node x and the array itself are created: this happens the first time we insert a hyperarc with source set X. Note that the data structures for simple nodes are redundant, since $REACH_j[i]=1$ if and only if LAST[i,j]=nil. This is done for the sake of clarity and for a more uniform presentation of the algorithms. On the other side this increases the space by no more than a constant factor and could be easily avoided in the implementation.

While dealing with dynamic hypergraphs we are allowed to introduce hyperarcs with arbitrary source sets. As a consequence we might have several hyperarcs with the same source set X. The compound nodes will be maintained in a AVL tree [AVL 62] referred to as T_C, while N_C denotes the set of compound nodes. This will allow to efficiently check whether the source set of the hyperarc to be introduced corresponds to a compound node already existent in the FD-graph. With this technique, only the necessary compound nodes will be introduced, thus making our representation non redundant.

In the following procedures we will use a function *Compound* which, given an arbitrary source set X, searches in the balanced tree T_C and returns the corresponding compound node x if it already exists, otherwise it is created and inserted in T_C performing any necessary initialization.

The Algorithms

We now show how hyperpaths with minimal GAP starting from simple nodes can be maintained during the insertion of new hyperarcs in the hypergraph. The procedure insert provides the required updates to the data structures while inserting a hyperarc from a source set X to a target node y:

```
procedure insert (X  set of simple_node, y: simple_node);
var x: node; i: simple_node;
begin
  if |X|=1
    then x := the element of X
    else x := compound(X);
  insert y into L_1(x),
```

```
    for each {simple_node} i in Ns do
      if REACH_X[i] = 0
        then  begin
                closure_reach (i,y);
                closure_last (i,x,y)
              end
  end;
```

The aim of recursive procedures *closure_reach* and *closure_last* is respectively to update the arrays REACH and LAST after the insertion of the hyperarc <X,y>. This will be accomplished separately for each source node i from which y is reachable. In particular, each call to closure_reach(i,j) propagates to j the reachability *from* the node i, possibly updating the value of $REACH_j[i]$, while each call to closure_last(i,k,j) tries to find out whether the inserted hyperarc has introduced a new minimal hyperpath from the simple node i to the node j finishing with the arc <k,j>. If this is the case, the value of LAST[i,j] will be set to k and it will tried to propagate the new minimal hyperpath. More details are given in the following pseudo-code.

```
procedure closure_reach(i : simple_node; j : node);
var  w : node;
begin
if REACH_j[i] <> 0                { there is no hyperpath from i to j}
  then begin
    REACH_j[i] := REACH_j[i] - 1;
    if REACH_j[i] = 0 { a hyperpath from i to j has been just introduced}
       then for each w in L_f(j) union L_d(j)  do  closure_reach(i,w)
  end
end;
```

The second procedure, closure_last, performs a second scanning in the FD-graph in order to update the arrays LAST and GAP:

```
procedure closure_last(i: simple_node; k: node; j: simple_node);
var s: simple_node; C: compound_node;
begin
 if GAP_j[i] > GAP_k[i] + 1  { a new minimal hyperpath hp_i,j has been found}
   then  begin
           GAP_j[i] := GAP_k[i] + 1;
           LAST[i,j] := k;
           for each {simple node} s in L_f(j) do closure_last(i,j,s);
           for each {compound node} C in L_d(j) do
             if  REACH_C[i] = 0  and  GAP_C[i] > GAP_j[i]
               then  begin
                       GAP_C[i] := GAP_j[i];
                       for each simple node s in L_f(C) do
                         closure_last(i,C,s)
                     end
         end
end;
```

Note that LAST[i,y] is computed only when y is a simple node as should be done for the definition of LAST. The correctness of this approach hinges on the following invariants.

Lemma 4.1 After the insertion of any hyperarc, a node j is reachable from a simple node i if and only if $REACH_j[i]=0$.

Lemma 4.2 After the insertion of any hyperarc, for any simple node i and any compound node C, $REACH_C[i]$ equals the number of simple nodes in the source set of C which are not reachable from i.

Sketch of the proofs The arrays $REACH_j$ has to be updated by the procedure closure_reach. We remark that a call to closure_reach(i,j) (propagating to j the closure of the node i) is performed visiting an arc <x,j> as soon as the node x has become reachable from i, due to the insertion of some hyperarc in the data structure.The lemmata 4.1 and 4.2 can be proved proceeding by induction on the number of (full or dotted) arcs examined during the execution of the procedure closure_reach during the insertion of new (hyper)arcs.
QED

Lemma 4.3 After the insertion of any hyperarc, for any pair of simple nodes i and j, LAST[i,j] points to the last node of a minimal-gap FD-path from i to j, if there exists such a hyperpath.

Sketch of the proof By induction on the number of hyperarcs inserted. The base of the induction is easily proved since at the beginning, when no hyperarc has been yet introduced, for each simple node i $GAP_i[i] = 0$, while the other entries are initialized to $+\infty$. Hence, the theorem trivially holds. Suppose now that the theorem hold before inserting a hyperarc from a source set X to a simple node i. During the first scan performed by the recursive calls of procedure closure_reach, all the entries of the arrays REACH are correctly computed as proved in lemmata 4.1 and 4.2. The second scanning performed by procedure closure_last is in charge of updating the arrays GAP and the backword pointers in the array LAST. Let us call gap(i,j) the the minimal gap of a hyperpath (if any) from node i to node j. We will prove that after the insertion of the hyperarc from X to y, $GAP_j[i]$=gap(i,j) for each pair of simple nodes for which there is a hyperpath from i to j, and $GAP_j[i]=+\infty$ otherwise. Let us call gap' and GAP' the values of gap and GAP after the insertion of the hyperarc <X,y>. It is easy to see that procedure closure_reach(i,*,*) can access a node j only if there exist a (simple or compound) node k for which $REACH_k[i]=0$ (and thus by either Lemma 4.1 or Lemma 4.2 reachable from i), and a full arc from k to j. As a consequence, a node j will be subsequently examined during the same call of closure_reach(i,*,*) only if there is a hyperpath from i to j. Since procedure closure_last is the only one which can update the arrays GAP after their initialization, this argument assures that if there is no hyperpath from i to j, then $GAP_j[i]$ remains unchanged to $+\infty$ also after the insertion of the hyperarc (X,y). Assume now that there is a hyperpath from i to j. Due to how the entries of the arrays GAP are updated, $GAP'_j[i]$= $\min\{GAP_j[i], GAP'_k[i] + 1\}$= gap'(x,y), as can be proved by induction on the hyperarcs visited by closure_last(i,*,*).
QED

Lemma 4.1 and Lemma 4.2 assure that the closures of the simple nodes are correctly updated during the insertions of new hyperarcs, while Lemma 4.3 guarantees that the entries of the array LAST allow to trace correctly a minimal-gap FD-path from any simple node i to any other (simple or compound) node. The following theorem summarizes the behavior of the data structure under the insertion of new hyperarcs and provides an analysis of the involved costs.

Theorem 4.1 Given an FD-graph $\mathcal{H}$, with n (simple or compound) nodes and m (full or dotted) arcs in which hyperarcs are to be inserted in an online fashion, there exists a data structure which allows:

(i) to check whether there is an FD-path between any pair of simple nodes in constant time;
(ii) to return a FD-path with minimal GAP (if one exists) between any pair of simple nodes in time which is linear in the length of the description of such an FD-path.

The total time involved in maintaining the data structure while new hyperarcs are inserted is $\Theta(mn^2)$. The space required is $\Theta(m + n^2)$.

Sketch of the proof The points (i) and (ii) and the space bound are a consequence of the definition of the data structures. As far the time bound is concerned, we remark that for each simple node i a full arc <k,j> is scanned by a call to closure_last (i,k,j) as soon $GAP_k[i]$ decreases. This can happen at most n times (because $GAP_k[i] \leq n-1$). Hence the theorem follows.
QED

We can extend algorithms and data structures to deal with the more general problem of maintaining hyperpaths with minimal GAP from any arbitrary set of simple nodes: this does not require a significant modification. In order to maintain the hyperpaths with minimal RANK, we have to modify the data structure. For each compound node Y and for each simple node x we maintain a heap of the components of Y reachable from x to allow the following operations:

- select the component y_i of Y with maximum $RANK(x,y_i)$: when REACH(x,Y)=0 this is equal to RANK(x,Y);
- modify $RANK(x,y_i)$ in the heap: this is required if we have added some arc reducing such a rank.

Since both these operations can be performed in $\Theta(\log n)$ we can easily modify the previous algorithms to perform online the updating of hyperpaths with minimal RANK with an overall complexity which is stated by the following theorem.

Theorem 4.2 Given an FD-graph $\mathcal{H}$, with n (simple or compound) nodes and m (full or dotted) arcs in which hyperarcs are to be inserted in an online fashion, there exists a data structure which allows:

(i) to check whether there is an FD-path between any pair of simple nodes in constant time;

(ii) to return a FD-path with minimal RANK (if one exists) between any pair of simple nodes in time which is linear in the length of the description of such an FD-path.

The total time involved in maintaining the data structure while new hyperarcs are inserted is $\Theta(m\, n^2 \log n)$. The space required is $\Theta(m + n^2)$.

5 - CONCLUSIONS

In this paper we have considered the problem of maintaining online minimal hyperpaths in a directed hypergraph. We have taken into account several definitions of minimality for hyperpaths and we have shown tthat some of them lead to NP-complete problems. Furthermore, we have presented algorithms and data structures to deal with the maintenance of hyperpaths with minimal GAP and minimal RANK while new hyperarcs are inserted. We have shown how these data structures allow us to answer queries in linear time with the size of the output (i.e., in optimal time).

Finally, we have proposed data structures and algorithms for this problem which allow us to achieve a worst case total time complexity of $\Theta(mn^2)$ for minimal gap, and of $\Theta(mn^2 \log n)$ for minimal rank, corresponding to an amortized time complexity of $\Theta(n^2)$ and $\Theta(n^2 \log n)$ per insertion respectively.

The following problems seem worthy of further investigation. Are there significant classes of hypergraphs such that min-size and other problems can be solved in polynomial time? Moreover, is there any efficient data structure for the fully dynamic maintenance of minimal hyperpaths (i.e., considering also deletion of hyperarcs)? Another interesting problem is studying the trade-off between the complexity of query and update operations, that is investigating whether removing the optimality for reporting hyperpaths can improve the bounds for updating a suitable data structure.

ACKNOWLEDGEMENTS We are indebted to Giorgio Ausiello for many suggestions and for discussing the main ideas of this paper. Without his help this work would not have started, progressed or ended. We are also grateful to the anonymous referees for their useful comments.

REFERENCES

[AVL 62] G.M. Adelson-Velskii, Y.M. Landis, *"An algorithm for the organization of information"*, Soviet Math. Dokl., 3, 1259-1262 (1962).

[AGU 72] A.V. Aho, M.R. Garey, J.D. Ullman, *"The transitive reduction of a directed graph"*, SIAM J. Comput., 1, 131-137 (1972).

[AHU 74] A.V. Aho, J.E. Hopcroft, J.D. Ullman, *"The design and analysis of computer algorithms"*, Addison-Wesley, Reading, MA (1974).

[ADS 83] G. Ausiello, A. D'Atri, D. Saccà, *"Graph algorithms for functional dependency manipulation"*, J. ACM , 30, 752-766 (1983).

[ADS 85] G. Ausiello, A. D'Atri, D. Saccà, *"Strongly equivalent directed hypergraphs"*, in Analysis and Design of Algorithms for Combinatorial Problems, Annals of Discrete Mathematics, 25, 1-25 (1985).

[ADS 86] G. Ausiello, A. D'Atri, D. Saccà, *"Minimal representation of directed hypergraphs"* , SIAM J. Comput., 15, 418-431 (1986).

[AI 87] G. Ausiello, G.F. Italiano, *"Online algorithms for polynomially solvable satisfiability problems"*, Journal of Logic Programming, to appear.

[AI 88] G. Ausiello, G.F. Italiano, U. Nanni, *"Dynamic Maintenance of Dyrected Hypergraphs"*, to appear in Theoretical Computer Science (1989).

[BM 72] R. Bayer, E. McCreight, *"Organization and maintenance of large ordered indices"*, Acta Informatica, 1, 173-179 (1972).

[B 73] C. Berge, *"Graphs and Hypergraphs"*, North Holland, Amsterdam, 1973.

[B 77] H. Boley, *"Directed recursive labelnode hypergraphs: A new representation language"*, Artificial Intelligence , 9, 49-85 (1977).

[DG 84] W.F. Dowling, J.H. Gallier, *"Linear-time algorithms for testing the satisfiability of propositional Horn formulae"*, J. Logic Programming, 3, 267-284 (1984).

[ES 81] S. Even, Y. Shiloach, *"An online edge deletion problem"*, J. ACM , 28, 1-4 (1981).

[F 83] R. Fagin, *"Acyclic database schemes of various degrees: A painless introduction"*, in Proceedings of CAAP 83, LNCS, 159, Springer Verlag, 65-89 (1983).

[F 85] G. N. Frederickson," Data structures for online updating of minimum spanning trees with applications", *SIAM J Comput.*, 14, 781-798 (1985).

[GJ79] M. R. Garey, D. S. Johnson, *"Computers and intractability: a guide to the theory of NP-completeness"*, W. H. Freeman Co., S. Francisco, (1979).

[GMM 81] S. Gnesi, U. Montanari, A. Martelli, *"Dynamic programming as graph searching: An algebraic approach"*, J. ACM , 28, 737-751 (1981).

[I 86] G.F. Italiano, *"Amortized efficiency of a path retrieval data structure"*, Theoretical Computer Science, 48, 273-281 (1986).

[K 79] R. Kowalski, *"Logic for problem solving"*, North Holland, New York (1979).

[N 82] N.J. Nilsson, *"Principles of Artificial Intelligence"*, Springer Verlag, (1982).

[P 66] C.A. Petri, "Communication with automata", Techn. Rep. RADC-TR-65377, 1, Griffith Air Force Base, New York (1966).

[R 85] H. Rohnert, "A dynamization of the all pairs least cost path problem", *Proc. 2nd Symp. on Theoretical Aspects of Computer Science*, 279-286 (1985).

[ST 83] D.D. Sleator, R.E. Tarjan, *"A data structure for dynamic trees"*, J. Comput. System Sci., 26, 362-381 (1983).

[T 85] R.E. Tarjan, *"Amortized computational complexity"*, SIAM J. Alg. Disc. Meth. , 6, 306-318 (1985).

[Y 82] M. Yannakakis, *"A theory of safe locking policies in Database Systems"*, J. ACM , 29, 718-740 (1982).