Optimal Traversal of Directed Hypergraphs*

Giorgio Ausiello$^{1,4}$  Giuseppe F. Italiano$^2$
Umberto Nanni$^{3,4}$

TR-92-073
September 1992

Abstract

A *directed hypergraph* is defined by a set of nodes and a set of hyperarcs, each of which connects a set of *source* nodes to a single *target* node. Directed hypergraphs are used in several contexts to model different combinatorial structures, such as functional dependencies [20], Horn clauses in propositional calculus [6], AND-OR graphs [17], Petri nets [18]. A *hyperpath*, similarly to the analogous notion of path in directed graphs, consists of a connection among nodes using hyperarcs. Unlike paths in graphs, hyperpaths are suitable of different definitions of measure, corresponding to different concepts arising in various applications.

In this paper we consider the problem of finding optimal hyperpaths according to several optimization criteria. We show that some of these problems are NP-hard but, if the measure function on hyperpaths matches certain conditions (namely if it is *value-based*), the problem turns out to be tractable. We describe efficient algorithms and data structures to find optimal hyperpaths which can be used with any value-based measure function, since it appears in parametric form. The achieved time bound is $O(|\mathcal{H}| + n \log n)$ for a hypergraph $\mathcal{H}$ with $n$ nodes and an overall description of size $|\mathcal{H}|$. Dynamic maintenance of optimal hyperpaths is also considered, and the proposed solution supports insertions of hyperarcs.

---

*$^*$Work partially supported by the Italian MURST National Project "Algoritmi e Strutture di Calcolo" and by the ESPRIT II Basic Research Actions Program Project no. 3075 ("ALCOM").

$^1$Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113 - 00198 - Roma, Italy.

$^2$IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598. On leave from Università di Roma "La Sapienza", Italy.

$^3$Dipartimento di Matematica Pura e Applicata, Università di L'Aquila, Via Vetoio, Coppito - 67010 - L'Aquila, Italy.

$^4$This work was partly developed while this author was visiting the "International Computer Science Institute", Berkeley, California.
1 Introduction

Directed hypergraphs are a generalization of directed graphs, originally proposed in [3] to represent the implication relation inherent in the structure of several problems. While directed graphs are normally used for representing one-to-one functional relations over finite sets, in several areas of computer science the need for more general types of functional relations arises, such as many-to-one relations or even functional relations with variable arity.

A directed hypergraph (dhg) consists of a finite set of nodes $N$ and a set of hyperarcs, any hyperarc being a pair whose first element is a set of nodes $S \subseteq N$ and whose second element is a node $j \in N$. The natural interpretation is that an hyperarc from $i_1, \ldots, i_k$ to $j$ exists if a function relates $i_1, \ldots, i_k$ to $j$. Clearly a graph is a special case of hypergraph.

Classical examples of combinatorial structures that can be easily represented by hypergraphs are functional dependencies in relational databases [3] (where an hyperarc from $A_1, \ldots, A_k$ to $B$ represents a functional dependency between attributes $A_1, \ldots, A_k$ and attribute $B$), Horn formulae in propositional calculus [6] (in which a hyperarc represents a Horn clause $p_1, \ldots, p_k \rightarrow q$, where $p_1, \ldots, p_k$ and $q$ are positive or negative literals), implications in problem solving (where hypergraphs can be used as an alternative to and-or graphs), Datalog [13], Operations Research [12], Petri Nets [2].

In several applications of directed hypergraph the notion of path and of traversal are required. The concept of path in a directed graph is standard and shortest paths between nodes can be efficiently determined [19] and maintained [8]. In the case of a hypergraph the corresponding concept of hyperpath has to be carefully defined. Intuitively a hyperpath from a set of nodes $S$ to a single node $t$ exists if there is a hyperarc from a set $S'$ to $t$ and if there exist hyperpaths from $S$ to any of the nodes in $S'$. On the basis of such concept the notion of transitive closure of a hypergraph has been defined in [3] and efficient algorithms for transitive closure maintenance under dynamic insertion of hyperarcs have been introduced in [7].

More delicate is the problem of defining cost measures of hyperpaths. In fact, whereas the cost of a path in a graph may be uniquely measured on the basis of edge costs, hyperpaths can be measured and compared according to various measure notions and optimality criteria. Moreover, such optimality criteria lead to problems having dramatically different complexity properties. Example of such measures are: the overall number of hyperarcs involved in a hyperpath, the overall length of the hyperpath representation, the sum of the costs of all arcs belonging to the hyperpath. While in the case of graphs paths which are optimal with respect to all of these criteria may be efficiently determined, we show that in the case of hypergraphs the problem of determining optimal hyperpaths under the same optimality criteria are all NP-hard.

The situation is much simpler if we consider other hyperpath measures. The gap and the rank of a hyperpath are examples of this second type of measures. For them it has been shown in [15] that efficient data structures and algorithms can be defined which allow the retrieval and the dynamic maintenance of optimal hyperpaths. We call such measures value-based measures since, as will be better explained in the paper, their recursive definition on the structure of a hyperpath does not depend on combinatorial constraint on the set of affected hyperarcs, but only on the measures of the component sub-hyperpaths. Beside the rank and the gap of a hyperpath, other examples of value-based measures are the traversal cost, which corresponds to the cost of traversing the hypergraph and in which, when a hyperarc is traversed more than once the cost is repeatedly taken into account, and the bottleneck, which corresponds to the notion of critical (less reliable) hyperarc in a weighted hyperpath.

The main contribution of the paper consists of introducing the value-based measures, a general concept which capture several metrics on hyperpaths, and in showing that hyperpaths with optimal value-based measures can be determined in polynomial time and also efficiently maintained under dynamic insertion of new hyperarcs in the hypergraph.

The paper is organized as follows. In the next section the basic definitions of hypergraph and hyperpath are provided. Section 3 is devoted to the introduction of various measures and optimality
criteria for hyperpath and to a discussion of the complexity of optimal traversal problems. In particular, after showing the NP-hardness of various problems, the concept of value-based measure is provided and the main result of the paper concerning the fact that hyperpath with optimal value-based measures can be efficiently determined is stated, whose proof is given in Section 4. In Section 6 we analyze the on-line case in which a hypergraph is allowed to dynamically change by insertion of new hyperarcs and we provide efficient on-line algorithms for maintaining hyperpaths with optimal value-based measures. Applications for both the static and the dynamic case are sketched in Section 6.

2 Metric Concepts over Directed Hypergraphs

A directed hypergraph (in short dhg) is a generalization of the concept of directed graph. It was first introduced in [3] to represent functional dependencies in relational data base schemata.

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

A weighted hypergraph is a directed hypergraph in which a real cost is associated with each hyperarc.

Note that a directed graph is a special case of directed hypergraph, where all the source sets have cardinality one. There are several parameters which can be taken into account for directed hypergraphs:

- the number of nodes: \( n = |N| \)
- the number of hyperarcs: \( h = |H| \)
- the source area \( a \), that is the sum of cardinalities of all the source sets:
  \[
  a = \sum_{S \in S} |S|,
  \]
  where \( S \) denotes the set of source sets: \( S = \{S|(S, t) \in H \text{ for some } t\} \)
- the nonsingleton source area \( a' \), that is the sum of cardinalities of all the nonsingleton source sets:
  \[
  a' = \sum_{S \in S_M} |S|,
  \]
  where \( S_M \) denotes the set of nonsingleton source sets: \( S_M = \{S|(S, t) \in H \text{ for some } t, |S| > 1\} \)
- the size \( s \), that is the overall length of the description of the hypergraph (also denoted as \( |\mathcal{H}| \)).

If we represent a directed hypergraph by means of adjacency lists we have that \( |\mathcal{H}| \equiv s = n + a' + h \).

Note that if the directed hypergraph is more simply a directed graph, the number of vertices is equal to the number of nodes \( n \), and the number of edges is \( m = h \). Furthermore, \( a = n \), \( a' = 0 \), and \( s = n + m \).

Definition 2.2 Let \( \mathcal{H} = (N, H) \) be a directed hypergraph. A hypergraph \( \mathcal{H}' = (N', H') \) such that:

\[
N' \subseteq N
\]
\[
H' \subseteq H \text{ and, for each } (S, t) \in H', S \subseteq N'.
\]
is called a subhypergraph of $\mathcal{H}$. We denote this by $\mathcal{H}' \subseteq \mathcal{H}$.

**Definition 2.3** Let $\mathcal{H} = (N, H)$ be a directed hypergraph, and let $H' \subseteq H$ be a set of hyperarcs in $\mathcal{H}$. Let $N' \subseteq N$ be the union of source sets and target nodes of hyperarcs in $H'$. The hypergraph $\mathcal{H}' = (N', H')$ is said to be the subhypergraph of $\mathcal{H}$ induced by $H'$.

In Figure 1 an example of hypergraph and subhypergraph are shown.

Figure 1: a directed hypergraph (a) and a subhypergraph (b).

We now define the concept of hyperpath in directed hypergraphs. Before, we recall some terminology on directed graphs. A path $\pi$ in a directed graph is a sequence of edges $e_1, e_2, \ldots, e_k$ and vertices $v_0, v_1, \ldots, v_k$ such that $e_i = (v_{i-1}, v_i)$, $1 \leq i \leq k$. A path $\pi$ is simple if no vertex repeats twice. It is a cycle if $v_0 = v_k$. If $v_i = v_j$ for some $i \neq j$, then the path $\pi$ contains a cycle as a subpath.

Given a path $\pi$ from $x$ to $y$, we can describe $\pi$ in different ways. We can give the sequence of all the edges in $\pi$, as traversed while going from $x$ to $y$. Notice that this description may contain the same edge more than once and may not even be bounded, since $\pi$ may contain a cycle which is traversed an unbounded number of times. We refer to this description of a path as unfolded. An alternative description may be the subgraph of $G$ containing exactly the edges of $\pi$ (note that each edge is considered only once). If the graph $G$ is finite, this description is always bounded, and is referred to as folded.

We now turn to directed hypergraphs.

**Definition 2.4** Let $\mathcal{H} = (N, H)$ be a directed hypergraph, $X \subseteq N$ be a non-empty subset of nodes, and $y$ be a node in $N$. There is a hyperpath from $X$ to $y$ in $\mathcal{H}$ if

a) either $y \in X$ (extended reflexivity);
b) or there is an hyperarc $(Z, y) \in H$ and hyperpaths from $X$ to each node $z_i \in Z$ (extended transitivity).

The above recursive definition of hyperpath is naturally described by a tree labeled in the nodes, referred to as the hyperpath tree and defined as follows.

**Definition 2.5** Let $\mathcal{H} = (N, H)$ be a directed hypergraph, $X \subseteq N$ be a non-empty subset of nodes, and $y$ be a node in $N$ such that there is a hyperpath from $X$ to $y$. A hyperpath tree from $X$ to $y$ is a tree $t_{X,y}$ defined as follows:

a) if $y \in X$ (extended reflexivity) $t_{X,y}$ is empty;
b) if there is a hyperarc $(Z, y) \in H$ and hyperpaths from $X$ to each node $z_i \in Z$ (extended transitivity), then $t_{X,y}$ consists of a root labeled with hyperarc $(Z, y) \in H$ having as subtrees the hyperpath trees $t_{X,z_i}$ for each node $z_i \in Z$.

Note that the hyperpath tree $t_{X,y}$ is such that its root has the target node $y$ in its label. Furthermore, if $(S, t)$ is the label of a leaf in the hyperpath tree, the source set $S$ is contained in $X$. We refer to the hyperpath tree as the unfolded representation of a hyperpath. As in the case of a path in a directed graph, this representation describes explicitly the sequence of hyperarcs, as traversed while going from $X$ to $y$. Once again, the same hyperarc may appear more than once in the hyperpath tree. In what follows, we will use interchangeably the terms unfolded hyperpath and hyperpath tree. Also for hyperpaths there is an alternative and more concise description defined as follows.
Theorem 3.1 Let $\mathcal{H} = (N, E)$ be a directed hypergraph, $x$ and $y$ be two nodes in $N$, and $k$ be an integer. Consider the following problems.

(P1) Find a hyperpath $h_{x,y}$ with $k$ hyperarcs or less.

(P2) Find a hyperpath $h_{x,y}$ of cost $k$ or less.

(P3) Find a hyperpath $h_{x,y}$ of size $k$ or less.

Then (P1), (P2) and (P3) are NP-complete.

Proof: We consider the three problems separately.

Consider first problem (P1). We use a reduction to Minimum Cover (in short MC) [14]. Let 
$A = \{a_1, a_2, \ldots, a_n\}$ be a set of elements and $S = \{S_1, S_2, \ldots, S_m\}$ be a family of subsets of $A$ such that $\bigcup_{i=1}^{m} S_i = A$. We recall that, given $A, S$ and an integer $k$, MC consists of finding a cover of cardinality $k$ or less, that is, a subfamily $S'$ of $S$ such that $|S'| \leq k$, and $\bigcup_{S_i \in S'} S_i = A$.

Let $\mathcal{I} = (A, S, k)$ be an instance of MC. We now define a directed hypergraph $\mathcal{H}_{\mathcal{I}} = (N_{\mathcal{I}}, E_{\mathcal{I}})$ as follows. The set of nodes is $N_{\mathcal{I}} = A \cup S \cup \{p, q\}$ and the set of hyperarcs is $E_{\mathcal{I}} = E_1 \cup E_2 \cup E_3$, where:

$H_1 = \{(p, S_i) | S_i \in S\}$,

$H_2 = \{(S_i, a_j) | S_i \in S \text{ and } a_j \in S_i\}$, and

$H_3$ consists of the single hyperarc $(A, q)$ (see Figure 4).

Figure 4: the hypergraph associated to an instance of MC.

We now show that there exists a hyperpath from $p$ to $q$ in $\mathcal{H}_{\mathcal{I}}$ having no more than $k + n + 1$ hyperarcs if and only if there exists a cover $S'$ of $A$ whose cardinality is less than or equal to $k$.

Indeed assume there is a feasible solution for MC, that is $\bigcup_{S_i \in S'} S_i = A$, then $\mathcal{H}_{\mathcal{I}}$ always contains a hyperpath from $p$ to $q$ (see Definition 2.6):

$h_{p,q} = \langle A, q \rangle \bigcup_{a_j \in A} \{h_{p,a_j}\}$, with $h_{p,a_j} = \{(p, S_i), (S_i, a_j)\}$.

This is possible if and only if for each $a_j \in A$ there exists some $S_i$ which contains $a_j$, that is if $S$ includes a cover for $A$. Note that if the cover has cardinality $k$, then there is a hyperpath with $k + n + 1$ hyperarcs in $\mathcal{H}_{\mathcal{I}}$.

Vice versa let $h_{p,q}$ be a hyperpath with $k_n$ hyperarcs. As shown above, $h_{p,q}$ must contain the hyperarc $(A, q)$, plus $n$ hyperarcs from the set $H_2$, plus, say, $k$ hyperarcs from the set $H_1$, with:

$k = k_n - n - 1$.

Let $S'$ be the target nodes $S_1, S_2, \ldots, S_k$ of the $k$ hyperarcs in the set $h_{p,q} \cap H_1$. Then $S'$ is a cover for the set $A$. In fact for any node $a_j \in A$ there exists a hyperarc $(S_i, a_j)$ in $h_{p,q} \cap H_2$, and by construction, this means that any element of the set $A$ is contained in some $S_i \in S'$.

Therefore the problem of finding a cover with cardinality $k$ is reduced to the problem of finding a hyperpath $h_{p,q}$ with a number of hyperarcs:

$n(h_{p,q}) = k_n = k + n + 1$.

The NP-completeness of (P2) follows immediately from the observation that finding a hyperpath with $k$ hyperarcs or less is a special case of (P2), when all the costs are 1.
To prove the NP-completeness of \((P_3)\), we observe that using the same reduction given for \((P_1)\), there is a cover with cardinality \(k\) or less if and only if there is a hyperpath \(h_{x,y}\) of size \(k = 2k + 2n + 2\) or less. □

Since the above decision problems are NP-complete, the NP-hardness of the corresponding optimization problems follows immediately.

**Corollary 3.2** Let \(\mathcal{H} = (\mathcal{N}, \mathcal{H})\) be a directed hypergraph, \(x\) and \(y\) be two nodes in \(\mathcal{N}\). The following problems are NP-hard:

i) finding the hyperpath \(h_{x,y}\) with minimum number of hyperarcs;

ii) finding the hyperpath \(h_{x,y}\) with minimum cost;

iii) finding the hyperpath \(h_{x,y}\) with minimum size\(^1\).

Let us now discuss under what conditions the problem of finding optimal hyperpaths may be solved in polynomial time. We introduce the concept of value-based measure over hyperpaths, which consists of some restrictions on the possible definitions of the measure function \(\mu\). If a given measure function is value-based, NP-hard problems do not arise because critical combinatorial constraints are avoided. On the contrary, there exist algorithms and data structures to solve efficiently these problems, as shown in the following sections. Such a definition applies to the folded hyperpaths.

**Definition 3.1** \(\mu\) is a value-based measure function on hyperpaths if it can be described by a triple \((f, \psi, \mu_0)\) such that:

a) if \(h_{x,y} = \emptyset\) then

\[ \mu(h_{x,y}) = \mu_0; \]

b) if \(h_{x,y} = \{(z,y)\} \cup h_{x,z_1} \cup h_{x,z_2} \cup \ldots \cup h_{x,z_k}\), then

\[ \mu(h_{x,y}) = f(w(z,y), \psi(\mu(h_{x,z_1}), \mu(h_{x,z_2}), \ldots, \mu(h_{x,z_k}))). \]

where:

- \(h_{x,y}\) is a hyperpath from the set \(X\) to the node \(y\);
- \(\mu_0\) is a real value;
- \(f: \mathbb{R}^2 \to \mathbb{R}^+\) is a monotone function;
- \(\psi: \mathcal{P}(\mathbb{R}) \to \mathbb{R}^+\) is a commutative and associative function from sets of reals to reals, monotone with respect to each of its arguments.

Examples of \(\psi\) are the min or max operators. As a consequence of this definition, the rank function is value-based, since it can be described by choosing \(\mu_0 = 0\), \(f\) to be the addition function, and \(\psi\) to be the max operator.

In this framework we can consider several measures of hyperpath, characterized by a given choice for the functions \(f, \psi\), and by the constant \(\mu_0\). Besides the definition of rank introduced in the previous section, examples of other value-based measure functions are the following.

**Definition 3.2** The bottleneck \(b(h_{x,y})\) of a hyperpath \(h_{x,y}\) is defined as follows:

\(^1\)Finding hyperpaths with minimum source area or minimum number of source sets are NP-hard problems as well.
a) if \( h_{X,y} = \emptyset \) (i.e. \( y \in X \)) then \( b(h_{X,y}) = \infty \);
b) if the hyperpath is defined by transitivity:
\[
b(h_{X,y}) = \min_{(S,t) \in h_{X,y}} \{ w(S,t) \}.
\]

**Definition 3.3** The gap \( g(h_{X,y}) \) of a hyperpath \( h_{X,y} \) is inductively defined as follows:

a) if \( h_{X,y} = \emptyset \) then: \( g(h_{X,y}) = 0 \);
b) if \( h_{X,y} = \bigcup_{z \in Z} h_{X,z} \cup \{(Z,y)\} \), then:
\[
g(h_{X,y}) = w(Z,y) + \min_{z \in Z} \{ g(h_{X,z}) \}.
\]

We remark that, as in the case of rank, the gap can be better defined with respect to the unfolded hyperpath \( t_{X,y} \), providing a uniquely defined value also in the case of cyclic hyperpaths. In fact the gap of the unfolded hyperpath, \( g_t(t_{X,y}) \), can be recursively defined as the sum of the cost of the root plus the minimum gap among its children. Again for acyclic hyperpaths we have:
\[
g(h_{X,y}) = g_t(t_{X,y}).
\]

In Definition 2.8 we defined the cost of a (folded) hyperpath as the sum of the costs of its hyperarcs. In the case of unfolded hyperpaths we define the **traversal cost** as the cost of the root plus the cost of all its subtrees. In other words if a hyperarc is traversed more than once, its cost is repeatedly taken into account.

**Definition 3.4** The traversal cost \( c_t(t_{X,y}) \) of an unfolded hyperpath \( t_{X,y} \) is inductively defined as follows:

a) if \( t_{X,y} \) is empty (i.e. \( y \in X \)) then: \( c_t(t_{X,y}) = 0 \);
b) if the unfolded hyperpath \( t_{X,y} \) has root \( (Z,y) \) with subtrees \( t_{X,z_1}, t_{X,z_2}, \ldots, t_{X,z_k} \), then:
\[
c_t(t_{X,y}) = w(Z,y) + \sum_{z \in Z} c_t(t_{X,z}).
\]

The rank (as seen above), gap, bottleneck, and traversal cost are all value-based measure functions. The following table summarizes these definitions.

<table>
<thead>
<tr>
<th>Function</th>
<th>( f )</th>
<th>( \psi )</th>
<th>( \mu_\varphi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>gap</td>
<td>+</td>
<td>min</td>
<td>0</td>
</tr>
<tr>
<td>rank</td>
<td>+</td>
<td>max</td>
<td>0</td>
</tr>
<tr>
<td>bottleneck</td>
<td>min</td>
<td>min</td>
<td>( \infty )</td>
</tr>
<tr>
<td>traversal cost</td>
<td>+</td>
<td>+</td>
<td>0</td>
</tr>
</tbody>
</table>

If we are given a value-based measure function and a choosing criterion \( \phi \) (from now on simply the min or max function), the complexity of finding minimal hyperpaths is polynomial in the description of \( \mathcal{H} \), as shown in the following section.
4 Algorithms for Value-based Measures over Hyperpaths

In this section we present efficient algorithms to find \( \mu \)-optimal hyperpaths, where \( \mu \) is a value-based measure function over hyperpaths.

4.1 Representing Directed Hypergraphs

In this section we deal with the problem of finding optimal hyperpaths from a given node \( T \) to any other node. The same technique works with the same time and space bounds for the problem of computing optimal hyperpaths from a source set to a node. Furthermore, it can be easily generalized to find the optimal hyperpath from any collection \( S \) of source sets to any other node (see [7] for further details). In this case, both the space and time complexity increase by an \( O(|S|) \) multiplicative factor.

In order to design efficient algorithms for directed hypergraphs, a data structure called \textit{FD-graph} has been introduced in [3] (the name \textit{FD-graph} derives from functional dependency). An FD-graph is essentially a labeled graph with two kinds of nodes and two kinds of edges, as the following definition shows.

\textbf{Definition 4.1} Given a hypergraph \( \mathcal{H} = (N, H) \), let \( S_M \) be the set of non-singleton source set, i.e.,
\[ S_M = \{ Z \mid \text{there exists a hyperarc } (Z, i) \in H \text{ and } |Z| > 1 \} \]

The FD-graph of \( \mathcal{H} \) is the labeled graph \( G(\mathcal{H}) = (N_s \cup N_c, A_f \cup A_d) \), where:

- \( N_s \equiv N \) is the set of simple nodes;
- \( N_c \) is the set of compound nodes which is in bijective relationship with \( S_M \). If \( Z \in S_M \) is a source set then \( z \) will denote the corresponding compound node, and any simple node \( z_i \) in the source set \( Z \) will be called a component node of the compound node \( z \);
- \( A_f \subseteq (N_c \times N_s) \cup (N_s \times N_s) = \{(z, x) \mid (Z, x) \in H\} \) is the set of edges referred to as full edges, in bijective relationship with \( H \);
- \( A_d \subseteq N_s \times N_c = \{(z_i, z) \mid z \in N_c \text{ and } z_i \in Z\} \) is the set of edges referred to as dotted edges, connecting any compound node to its components.

An example of FD-graph is shown in Figure 5.

![Figure 5: the FD-graph corresponding to the hypergraph in Figure 1.](image)

Note that there is a one-to-one correspondence between the hyperarcs of a given hypergraph \( \mathcal{H} \) and the full arcs of the corresponding FD-graph \( G(\mathcal{H}) \). If the hyperarcs of \( \mathcal{H} \) have weights, the same weight is given to the corresponding full arc.

The concept of \textit{FD-path} from a generic source \( X \subseteq N \) to a target node \( y \in N \) is related to that of (folded) hyperpath \( h_{X,y} \): it is the FD-graph of such a hypergraph. While dealing with FD-graphs, we extend the notion of FD-path (and its measure) considering also a compound node as a possible target node.

\textbf{Definition 4.2} Let \( \mathcal{H} = (N, H) \) be a directed hypergraph, and \( G(\mathcal{H}) = (N_s \cup N_c, A_f \cup A_d) \) be the corresponding FD-graph. An FD-path from a source \( X \subseteq N_s \) to any (simple or compound) target node \( y \in N_s \cup N_c \) is an FD-graph \( h^{	ext{FD}}_{X,y} \) contained in \( G(\mathcal{H}) \) and such that:

- \( h^{	ext{FD}}_{X,y} = G(h_{X,y}) \) if \( y \in N_s \);
- \( h^{	ext{FD}}_{X,y} = \bigcup_{y_i \in y} G(h_{X,y_i}) \cup \{(y_i, y) \mid y_i \text{ is a component node of } y\} \) if \( y \in N_c \).
The notion of measure is extended to FD-paths so as to be consistent with the measure of the corresponding hyperpath. In particular we give the definition for a value-based measure function.

**Definition 4.3** Let \( \mathcal{H} = (N, H) \) be a directed hypergraph, \( G(\mathcal{H}) = (N_s \cup N_c, A_f \cup A_d) \) be the corresponding FD-graph, and \( \mu = (f, \psi, \mu_0) \) be a value-based measure function over hyperpaths. The corresponding measure function \( \mu_{FD} \) over FD-paths is defined as follows:

- a) if \( y \) is a simple node and \( (v, y) \) is the full edge in \( h_{X,y}^{FD} \) entering \( y \) then:
  \[
  \mu_{FD}(h_{X,y}^{FD}) = f(w_{|v,y|}, \psi(\mu_{FD}(h_{X,v}^{FD}))); \]

- b) if \( y = \{y_1, y_2, \ldots, y_q\} \) is a compound node then:
  \[
  \mu_{FD}(h_{X,y}^{FD}) = \psi(\mu_{FD}(h_{X,y_1}^{FD}), \mu_{FD}(h_{X,y_2}^{FD}), \ldots, \mu_{FD}(h_{X,y_q}^{FD})); \]

- c) if \( y \in X \) and \( h_{X,y}^{FD} \) is empty, then \( \mu_{FD}(h_{X,y}^{FD}) = \mu_{FD}(\emptyset) = \mu_0 \).

It is straightforward to prove that, for any source \( X \) and any simple node \( y \):

\[
\mu_{FD}(h_{X,y}^{FD}) = \mu(h_{X,y}).
\]

From now on we only consider the problem of finding optimal FD-paths, using the symbol \( \mu \) to denote also the measure function over FD-paths.

Analogously the notion of distance from a source \( X \), given an optimization criterion \( \phi \), is extended over FD-paths:
- if \( y \) is a simple node and \( h_{X,y}^{FD} \) ranges over any possible FD-path between \( X \) and \( y \), then:
  \[
  \delta(X, y) = \phi_{h_{X,y}^{FD}}(\mu(h_{X,y}^{FD})) \]

- if \( y = \{y_1, y_2, \ldots, y_q\} \) is a compound node then:
  \[
  \delta(X, y) = \psi(\delta(X, y_1), \delta(X, y_2), \ldots, \delta(X, y_q)) \]

- if there exists no FD-path from \( X \) to \( y \) then \( \delta(X, y) = \mu_{\infty} \).

Now let us turn to the data structure which will be used in the algorithms. FD-graphs are implemented by maintaining adjacency lists for each (simple or compound) node. Namely, all the full [dotted] edges leaving a node \( y \) are organized in the lists \( L_f(y) \) \( L_d(y) \). \( L_d(y) \) is empty if \( y \) is a compound node.

In the following we will describe how to find an optimal hyperpath from a given source \( T \). In order to retrieve optimal hyperpaths the basic idea is to store, for each simple node \( z \), the incoming hyperarc belonging to the optimal hyperpath (from \( T \). This "backward pointer" technique is implemented on FD-graphs by using an array \( LAST \) with size \( n_s \) containing the collection of backward pointers and defined as follows:
- \( LAST[z] \) points to the last (simple or compound) node (except \( z \)) in the optimal FD-path from the simple node \( T \) to the simple node \( z \);
- \( LAST[z] \) has a special value \textit{null} if no such FD-path exists.

Notice that there is no need to store an analogous information for compound nodes, 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.
The measure of the optimal hyperpaths, that is the distance, from $T$ to any other simple or compound node $y$, is stored in the variable $DIST[y]$:

$DIST[y] = d$, if there is an FD-path of optimal distance $d$ from $T$ to $y$;

$DIST[y] = \mu_{\infty}$, if there is no FD-path from $T$ to the node $y$.

In addition, for any simple node $x$ the variable $REACH[x]$ is defined in the following way:

$REACH[x] = 0$, if there is an FD-path in the FD-graph from $T$ to $x$.

$REACH[x] = 1$, otherwise.

The variable $REACH$ is defined also for each compound node $z$ (with components $z_1, z_2, \ldots, z_k$):

$REACH[z] = \sum_{k=1}^{z_k} REACH[z_k]$.

In other words, the entry $REACH[z]$ is equal to the number of simple nodes in the set $Z$ which are not reachable from the simple node $T$. Note that the data structures for simple nodes are redundant, since $REACH[z] = 1$ if and only if $LAST[z] = \text{null}$. 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 can be easily avoided in the implementation.

The existence of an FD-path from $T$ to any (simple or compound) node $y$ can be checked in constant time by examining the variable $REACH[y]$. If $REACH[y] > 0$, then there is no FD-path from $T$ to $y$, otherwise an optimal FD-path can be traced out by starting from $y$ and proceeding in a backward fashion, following all the incoming dotted edges for any compound node, and the pointer $LAST$ for any simple node (see [7] for further details).

4.2 Polynomial Algorithms for Value-Based Measures

If we are given a value-based measure function $\mu$ and an optimization criterion, the function $\phi$ (as said before simply the min or max function), the problem of finding $\mu$-optimal hyperpaths is polynomially solvable and in this section we show that this problem can be solved very efficiently, providing a general algorithm for this problem. We distinguish several cases, depending on the nature of the functions $f, \mu$ and of the optimization criterion $\phi$.

We first deal with the following two problems.

1. find the $\mu$-minimal hyperpaths from $T$ to any other node in $H$, provided that $\mu$ is monotone nondecreasing, that is if $h_{T,x} \subseteq h_{T,y}$ then $\mu(h_{T,x}) \leq \mu(h_{T,y})$;

2. find the $\mu$-maximal hyperpaths from $T$ to any other node in $H$, provided that $\mu$ is monotone nonincreasing.

Examples of the former case are finding the minimum rank or the minimum gap hyperpaths. Finding the maximum bottleneck FD-path is an example of the latter case.

A description of the algorithms (given in Figures 6,7,8) follows.

From now on we say that a (simple or compound) node $y$ has been visited by the algorithm if the value of $REACH[y]$ is equal to 0; a full edge is scanned if it has been passed as an argument to the procedure Scan.

All the optimal hyperpaths from $T$ are found in nonincreasing (nondecreasing) order, depending on the optimality criterion. Algorithm Distance generalizes to directed hypergraphs Dijkstra's shortest path algorithm [9].

Algorithm Distance operates in conjunction with procedure Scan. The algorithm uses a priority queue $PQ$ where a node $x$ is enqueued by procedure Scan (line 7) as soon as it is visited (that is when a first edge $(t',x)$ is scanned for some $t'$), together with the node $t$ itself, and using as priority the measure of the first FD-path from $T$ to $x$ found by the algorithm. When a full edge $(t',x)$ is
Algorithm Distance:
input:
\( \mathcal{H} \): weighted hypergraph; \{given in terms of adjacency lists of the FD-graph \( G(\mathcal{H}) \)\}
\( T \): node;
\( \mu_0 \): distance; \{the distance corresponding to the empty FD-path\}
\( f, \psi \): functions; \{defining the measure function \( \mu \)\}
criterion: \((\min, \max)\); \{the optimization criterion\}
output:
\( \text{DIST}[y] \): distance; \{for any simple and compound node \( y \)\}
\( \text{REACH}[y] \): integer; \{for any simple and compound node \( y \)\}
\( \text{LAST}[x] \): node; \{for any simple node \( x \)\}
\( \text{aux} \):
\( \text{PQ} \): priority queue; \{based on the given criterion (max or min)\}
\( \mu_\infty \): distance; \{the extremal distance representing no FD-path\}
\( x, t \): simple nodes;
\( z \): compound node;
\( s \): node;
1. begin
2. InitializeData;
3. make-PQ-empty;
4. PQ-insert(\( \mu_0, (T, T) \));
5. REACH[T] = 0;
6. while PQ-nonempty do
7. begin
8. PQ-extract(D, (s, t));
9. \( \text{DIST}[t] := D_t \);
10. \( \text{LAST}[t] := s \);
11. for each \{full edge\} \( (t, x) \in L_f(t) \) do Scan(t, x);
12. for each \{dotted edge\} \( (t, z) \in L_d(t) \) do
13. begin
14. decrement(REACH[z]);
15. if REACH[z] = 0 then \{all the component nodes \( z \) have been visited\}
16. begin
17. \( \text{DIST}[z] := \psi(\text{DIST}[z_{i_1}], \text{DIST}[z_{i_2}], \ldots, \text{DIST}[z_{i_n}]) \);
18. for each \{full edge\} \( (z, x) \in L_f(z) \) do Scan(z, x);
19. end
20. end
21. end
22. end;

Figure 6: algorithm Distance
Procedure InitializeData;
1. begin
2. if criterion=min
3. then $\mu_\infty = +\infty$
4. else $\mu_\infty = 0$;
5. for each {simple node} $x \in N_s$ do
6. begin
7. $REACH[x] := 1$;
8. $DIST[x] := \mu_\infty$;
9. $LAST[x] := \text{nil}$;
10. end
11. for each {compound node} $z \in N_c$ do
12. begin
13. $REACH[z] := |z|$;
14. $DIST[z] := \mu_\infty$;
15. end;
16. end;

Figure 7: procedure InitializeData

Procedure Scan($t$ : node; $z$ : simple-node);
1. begin
2. $D_{t,x} := f(w_{t,z}, DIST[t])$;
3. if $REACH[t] = 1$
4. then
5. begin
6. decrement($REACH[t]$); {the value is set to zero}
7. $PQ$-insert($D_{t,x}, (t, z)$);
8. end
9. else if $D_{t,x} < D_z$ {for minimization criterion}
10. then $PQ$-decrease($D_{t,x}, (t, z)$)
11. end;

Figure 8: procedure Scan
scanned but the node $z$ has been previously visited, the priority of $z$ is updated if and only if the
e edge $(t', z)$ yields an FD-path which improves the old priority (procedure Scan, lines 9-10).

Algorithm Distance initializes all the data structures (line 2), then insert as a first element in
the priority queue the item $(\mu_0, (T, T))$. Repeatedly, the algorithm extracts from the queue $PQ$ the
node $t$ with minimum priority $D_t$ which is assumed to be the measure of the optimal hyperpath
from $T$ to $t$. At this point all the outgoing full edges $(t, z)$ are scanned, all the dotted edges $(t, z)$
are considered (line 12) and, for any compound node $z$ such that $t \in Z$, $REACH[z]$ is decreased
(line 14), and two possible cases arise:

$REACH[z] > 0$ then nothing else is done;
$REACH[z] = 0$ then $t$ is the last component of the compound node $z$ to be visited. In this case
the node $z$ is visited as well, the measure of the optimal hyperpath from $T$ to $z$ is computed (line
17), and all the outgoing full edges $(z, z)$ are scanned (line 18).

Using the above argument, it is easy to prove the following invariants.

a) Each node is visited at most once;
b) Each full edge is scanned at most once;
c) After any number of execution of the "while" loop (lines 7-21) the set of scanned full edges
coincides with the set of full edges exiting from any visited node;
d) A compound node $z$ is visited if and only if all the simple nodes $z; z \in Z$ are visited.

Lemma 4.1 At the end of the execution of algorithm Distance, any (simple or compound) node $y$
has been visited if and only if there exists an FD-path from $T$ to $y$ in $G(\mathcal{H})$.

Proof:
(if case) Note that the node $T$ is visited before the first iteration of the loop (line 5). Suppose by
contradiction that there exists an FD-path $h_{T,y}$ in $G(\mathcal{H})$ but $y$ is not visited by the algorithm. Due
to the recursive definition of hyperpath, there exists a hyperarc $(s,t) \in h_{T,y}$ such that all the nodes
in the source set $S$ are visited by the algorithm, while the target node $t$ is not. This means that at
least one of the following conditions arises:
- there exists a full edge $(s, t)$ in $h_{T,y}$ such that $s$ is visited, while $t$ is not;
- there exists a compound node $z = \{z_1, z_2, \ldots, z_k\}$ such that each $z_i \in z$ is visited, but $z$ is not.

Due to invariants $c$ and $d$, neither case arises.

(only if case) Due to invariant $d$, we consider only the case of simple nodes. Let us proceed by
induction on the number of executions of the while cycle (lines 7-21), that is on the number of
visited nodes.

Before the first execution of the loop, the queue contains only the item $(\mu_0, (T, T))$ and the first
visited node is $T$, which is reachable from $T$ by an empty hyperpath.

When the item $(D_t, (s, t))$ is dequeued, $s$ has been previously visited and then, by inductive
hypothesis, $s$ is reachable from $T$ (there exists an FD-path $h_{T,s}$ in $G(\mathcal{H})$). But, by definition 4.2,
we have that $h_{T,s} \cup (s, t)$ is an FD-path from $T$ to $t$. During the same iteration, a node $z$ is inserted
in $PQ$ only if a full arc $(t, z)$ is scanned. \( \square \)

Theorem 4.1 Let $\mu$ be a value-based measure function over FD-paths. Algorithm Distance com-
putes correctly the $\mu$-optimal FD-paths from $T$ to any other node in $G(\mathcal{H})$ under the following
conditions:

1. if $\mu$ is a nondecreasing function, algorithm Distance computes the $\mu$-minimal FD-paths from
   $T$ to any other node in $G(\mathcal{H})$ using $PQ$ as a minimum-based priority queue;
2. if $\mu$ is a nonincreasing function, algorithm Distance computes the $\mu$-maximal FD-paths from $T$ to any other node in $G(\mathcal{H})$ using $PQ$ as a maximum-based priority queue.

Proof: (case 1) Note that the priority $D_t$ of any item $(D_t, (s, t))$ in the queue $PQ$ is computed by procedure Scan (line 2) as the measure of an actual FD-path whose last full arc is $(s, t)$. As far as compound nodes are concerned, we have that the distance from $T$ to any compound node $z$ is computed by Algorithm Distance (line 17) as soon as the distance from $T$ to any component node $z_i \in Z$ has been computed. In the following we refer only to simple nodes.

Let $X_t \subseteq N$ be the set of nodes already visited by the algorithm when the item $(D_t, (s, t))$ is extracted from the queue $PQ$. We will prove by induction on the number of items extracted from the queue that the following holds:

(i) $D_t$ is the measure of the optimal hyperpath using only nodes in $X_t$;

(ii) for any node $y$ such that $y \notin X_t$, $\delta(T, y) \geq D_t$.

This will prove the theorem since Lemma 4.1 assures that any node reachable from $T$ will be eventually visited by the algorithm.

In the base case the first dequeued item is $(\mu_e, (T, T))$ and the above claim is satisfied, since (i) $\delta(T, T) = \mu_e$ is the measure of the empty hyperpath, and (ii) no hyperpath can have a measure smaller than $\mu_e$.

Suppose now that the item $(D_t, (s, t))$ is extracted from the priority queue and algorithm Distance has visited the nodes in the set $X_t \subseteq N$. By inductive hypothesis, for any node $x \in X_t$ the algorithm has correctly computed the optimal hyperpath from $T$ to $x$ using only nodes in the set $X_t$. In this case we have that: (i) any full edge $(x, t)$ exiting from node $x \in X_t$ has been scanned and $D_t$ is the minimum priority computed for $t$; (ii) when the item $(D_t, (s, t))$ is dequeued, $D_t$ is the minimum priority among the elements in the queue and, since function $\mu$ is nondecreasing, any full arc scanned later will produce a priority value greater or equal $D_t$.

The same argument applies in case 2. □

We remark that, for each simple node $x$, the last node in an optimal FD-path from $T$ is stored in the variable $\text{LAST}[x]$ (line 10) as soon as the distance from $T$ is stored in $\text{DIST}[x]$ (line 9).

Theorem 4.2 The time complexity of Algorithm Distance is $O(|\mathcal{H}| + n \log n)$, where $n$ is the number of (simple) nodes in $\mathcal{H}$.

Proof: By invariant $b$ a full edge can be scanned at most once and, on the other side, a constant number of steps are performed by algorithm Distance for any full and dotted edge, as can be verified by inspecting the code.

During the execution of algorithm Distance, at most $n$ insertions and $n$ deletions of minimum are performed in the priority queue $PQ$, plus at most $h$ priority decreases.

The claimed bound derives from the implementation of $PQ$ as an $F$-heap [11], provided that for any compound node $z$, the function $\psi(\text{DIST}[z_1], \text{DIST}[z_2], \ldots, \text{DIST}[z_h])$ is computable in $O(q)$ time. □

We now turn to the case where the optimization criterion $\phi$ is the maximum (resp. minimum) and the measure function $\mu$ is monotone nondecreasing (resp. monotone nonincreasing). Although, the problem of finding $\mu$-optimal FD-paths is not harder than the previous case.

As an example, suppose that function $\mu$ is monotone increasing, and (for simplicity) that the hypergraph is unweighted. In this case the problem of finding maximal FD-paths from $T$ (with measure $\mu_e$) becomes depending on the possibility of interlacing cycles: these FD-paths leads to unfolded hyperpaths with unbounded depth.
This situation can be handled first finding all the nodes reachable from \( T \) through cyclic FD-paths, and then computing the maximal FD-paths from \( T \) to any other node in \( \mathcal{H} \).

1. compute the transitive closure of \( T \) in \( \mathcal{H} \) and let \( \mathcal{H}' \) be the subhypergraph induced by the hyperarcs whose source set is reachable from \( T \); 
2. find the set \( U_T \) of the nodes reachable from \( T \) by cyclic paths performing a *depth first search* of the FD-graph \( G(\mathcal{H}') \) starting from the node \( T \).

Note that both this steps can be performed in time bounded by the size of \( \mathcal{H} \).

It is straightforward to prove that \( U_T \) coincides with the set of nodes reachable from \( T \) through cyclic FD-paths. On the other side, using the same arguments as in the proof of theorem 4.1 it is possible to prove that algorithm *Distance* given above computes the right \( \mu \)-maximal distances from \( T \) for all the nodes *not* reachable through cyclic hyperpaths, by using \( PQ \) as a max-based priority queue.

These considerations lead to state the following theorem.

**Theorem 4.3** Let \( \mathcal{H} \) be an unweighted directed hypergraph, and \( \mu \) be an increasing (resp. decreasing) value-based measure function over FD-paths. It is possible to find \( \mu \)-maximal (resp. \( \mu \)-minimal) FD-paths from \( T \) to any other node in \( G(\mathcal{H}) \) in time \( O(|\mathcal{H}| + n \log n) \), where \( n \) is the total number of nodes.

**Proof:** In the general case in which \( \mu \) is monotone nondecreasing (or nonincreasing) and the hypergraph is weighted, the problem of finding \( \mu \)-maximal (\( \mu \)-minimal) hyperpaths from \( T \) reduces to the problem of passing through particular hyperarcs or particular cycles of the hypergraphs. \( \square \)

## 5 Dynamic Maintenance of Value-Based Measures

In many applications it is required to update the solution of a given problem while input data are modified. In this case, the data structures must be maintained while an intermixed sequence of query and update operations are performed in an on-line fashion.

The problem of maintaining information about the existence of hyperpaths in a directed hypergraph has been considered already in [7]. In this section we generalize those results and consider the problem of maintaining a quite general notion of distance in a directed hypergraph while new hyperarcs are inserted. More precisely, if we consider a directed hypergraph \( \mathcal{H} \), a distinguished node \( T \) in \( \mathcal{H} \), a value-based measure function \( \mu \), and an optimization criterion \( \phi \), we show how to perform an arbitrary sequence of operations of the following kinds:

a) Insert a new weighted hyperarc \( \langle X, y; w(x,y) \rangle \) in \( \mathcal{H} \);

b) Decrease the cost of an existing hyperarc;

c) Report the value of the \( \mu \)-distance from \( T \) to any node \( y \).

In order to deal efficiently with this problem we make some reasonable assumptions:

1. The value of the function \( \psi \) on \( q \) arguments can be recomputed in \( O(\log q) \) when one of its argument changes its value (recall that by definition 3.1 \( \psi \) is commutative and associative);

2. The measure function \( \mu \) can only have values in an ordered discrete set \( D = \{ d_0, d_1, \ldots, d_K \} \).
It is possible to show that without condition 2, there are sequences of operations where each update to the hypergraph requires time $O(|\mathcal{H}|)$, while we are aiming at more efficient bounds.

Let $h$ be the number of hyperarcs in $\mathcal{H}$ after all the operations have been processed. Because of condition 2, the total number of operations is at most $O(Kh)$, since one hyperarc can be inserted at most once and decreased at most $O(K)$ times. In order to support the operations, one might use the following two strategies:

1. In order to speed up answers to requests (c) one might use the $O(|\mathcal{H}| + n \log n)$ offline algorithm described in the previous section to recompute the hyperpaths with optimal measure any time an operation of type (a) or (b) is performed.

2. To make modifications (a) and (b) cheaper, one might apply the same offline algorithm when a request of type (c) occurs.

Let $Q$ the total number of queries of type (c). Strategy (1) leads to a total of $O(Q + Kh(|\mathcal{H}| + n \log n))$ time in the worst case, while strategy (2) yields a $O(Kh + Q(|\mathcal{H}| + n \log n))$ bound. In this section we describe algorithms and data structures to solve this problem in a total of $O(K(|\mathcal{H}| \log h + Q)$ time, which is better than the previous two bounds (we remark that $h$ may be exponentially greater than $n$).

As in the previous section, we maintain the FD-graph, corresponding to the original hypergraph, and the $\mu$-optimal FD-paths from $T$ to any other (simple or compound) node. Moreover, for a practical description of the algorithms, we only consider the cases where the optimization criterion $\phi$ is the min, and the function $\mu$ is monotone increasing.

Our algorithms can be properly extended, with a slight modification, to the case in which we maintain optimal hyperpaths from any collection $S$ of source sets specified at any time during the insertsions. In this case the overall time complexity is roughly increased by a $|S|$ factor, leading to an $O(Q + K|\mathcal{H}|(|S| + \log h))$ time bound for any sequence of operation.

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 compound nodes will be maintained in an AVL tree $T_c$ 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 $Z$, searches in the balanced tree $T_c$ and returns the corresponding compound node $z$ if it already exists, otherwise it is created and inserted in $T_c$, performing any necessary initialization, including the initialization of the variable $DIST[z] = \psi(DIST[z_1], DIST[z_2], \ldots, DIST[z_x])$.

We now show how optimal hyperpaths from $T$ to any simple node can be maintained during the insertion of new hyperarcs or edge cost decreases. The procedure Insert provides the required updates to the data structures while either inserting a hyperarc from a source set $X$ to a target node $y$ with cost $w(x,y)$, or decreasing the cost of an existing hyperarc to a new value $w'(x,y)$.

In the procedures we are going to describe, $\psi$ is simply the max function, and we are interested in maintaining the minimal hyperpath. In this case, the distance in an FD-graph from $T$ to any
other simple node $x$ is given by:

$$
\delta(T, x) = \min_{\{y, z\}} f(w_{[y, z]}, \delta(T, x))
$$

where $\{y, z\}$ ranges over any full edge entering $x$. The distance from $T$ to any compound node $z = \{z_1, z_2, \ldots, z_q\}$ can be computed as:

$$
\delta(T, z) = \max\{\delta(T, z_1), \delta(T, z_2), \ldots, \delta(T, z_q)\}.
$$

An example of this kind is the problem of maintaining the hyperpaths with minimum rank. Since, as stated before, the measure function $\mu$ has values in an ordered discrete set $D = \{d_0, d_1, \ldots, d_K\}$ and we are interested in maintaining minimal hyperpaths, we have that the measure of the empty hyperpath, that is $\delta(T, T)$, has value $\mu_0 = d_0$ and for each node $y$ such that there exists no hyperpath from $T$ we have $\delta(T, y) = \mu_\infty = d_K$.

In order to efficiently maintain the invariant $DIST(z) = \delta(T, z)$ for any compound node $z$, the value of the function $\psi$ must be updated when one of its argument changes. This could require an additional data structure. Therefore, for each compound node $z$, a maximum-based priority queue $\text{heap}(z)$ is used, where any component node $z_i$ is contained together with the current value of $DIST(z_i)$ and its position in the heap is updated when the value of $DIST(z_i)$ decreases.

Note that modifying the cost of a hyperarc is handled as an insertion of a new one, if this decreases the cost, otherwise it is rejected.

The aim of recursive procedures $\text{ClosureReach}$ and $\text{ClosureLast}$ is to update the values of $\text{REACH}$, $\text{DIST}$, and $\text{LAST}$ due to the insertion of a new hyperarc $(X, y; s)$. In particular, each call to the recursive procedure $\text{ClosureReach}(j)$ propagates to $j$ the reachability from the node $T$, possibly updating the value of $\text{REACH}[j]$, while each call to $\text{ClosureLast}(i, j; w)$ tries to find out whether the inserted hyperarc has introduced a new optimal hyperpath from $T$ to the node $j$. If this is the case, the value of $\text{LAST}[j]$ will be set to $i$ and it will try to propagate the new optimal value with recursive calls to $\text{ClosureLast}$. More details are given in the pseudo-code (see Figures 9,10,11).

The second procedure, $\text{ClosureLast}$, performs a second scanning in the FD-graph in order to update the variables $\text{LAST}$ and $\text{DIST}$.

The correctness of this approach hinges on the following invariants, which are similar to the ones considered in the previous section and will not be proved.

- After the insertion of any hyperarc, any simple node $x$ is reachable from $T$ if and only if $\text{REACH}[x] = 0$.
- After the insertion of any hyperarc, for any compound node $z$, $\text{REACH}[z]$ equals the number of simple nodes, component of $z$, which are not reachable from $T$.

**Lemma 5.1** After the insertion of any hyperarc, for any simple node $x$, $\text{LAST}[x]$ points to the last node of a minimal $\mu$-measure FD-path from $T$ to $x$, if there exists such a hyperpath. Moreover, for any simple or compound node $y$, $\text{DIST}[y] = \delta(T, y)$, where $\delta(T, y)$ denotes the $\mu$-distance from $T$ to $y$.

**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, $\text{DIST}[T] = \mu_\infty$, $\text{DIST}[y] = \mu_\infty$, and $\text{LAST}[x] = \text{nil}$ for any simple node different from $T$. Hence, the thesis trivially holds.

Suppose now that the lemma holds before inserting a hyperarc $(X, y; w_{[x,y]})$. During the first scan performed by the recursive calls of procedure $\text{ClosureReach}$, all the entries of the arrays $\text{REACH}$ are correctly computed as proved in lemma 4.1. For any new optimal FD-path from $T$ to any node $j$ the second scanning performed by procedure $\text{ClosureLast}$ is in charge of updating the variables $\text{DIST}[j]$ and, in the case of a single node, the backward pointer $\text{LAST}[j]$. Let $\ell(j)$ be the distance
procedure Insert
input:
   \(X\): set of simple nodes; \{source set\}
   \(y\): node; \{target node\}
   \(w\): cost; \{(new) cost of the hyperarc to be inserted\}
side effects:
   updated the structure of the FD-graph;
   updated the data structures \(DIST[y]\), \(REACH[y]\), \(LAST[x]\);
data structures:
   heap(\(z\)): max-heap; \{for each compound node \(z\)\}
aux:
   \(z\): node;
1. begin
2. if \(|X| = 1\)
3. then \(z := \) the single element of \(X\)
4. else \(z := \) Compound(\(X\))
5. if there exists a full edge \((x, y; w(x,y)) \in L_f(x)\)
6. then if \(w < w(x,y)\)
7. then \(w_{x,y} := w\)
8. else exit
9. else insert \((x, y; w(x,y))\) into \(L_f(x)\)
10. if \(REACH[x] = 0\) then
11. begin
12. ClosureReach(\(y\));
13. ClosureLast(\((x, y; w(x,y))\))
14. end
15. end;

Figure 9: procedure Insert

procedure ClosureReach(\(j\): node);
aux \(k\): node;
1. begin
2. if \(REACH(j) > 0\) then \{there is no previous hyperpath from \(T\) to \(j\)\}
3. begin
4. \(REACH(j) := REACH(j) - 1\);
5. if \(REACH(j) = 0\) then
6. for each \{full or dotted edge\} \((j, k; w_{(j,k)}) \in L_f(j) \cup L_d(j)\) do
7. ClosureReach(\(k\))
8. end
9. end;

Figure 10: procedure ClosureReach
procedure ClosureLast((i, j; w_{i,j}): full edge);
   var x, k: simple nodes; z: compound node;
1. begin
2. if f(w_{i,j}, DIST[i]) < DIST[j] then
3. begin {a new μ-minimal hyperpath has been just found}
4. DIST[j] := f(w_{i,j}, DIST[i]);
5. LAST[j] := i;
6. for each (full edge) (j, k; w_{j,k}) ∈ L_f(j) do ClosureLast((j, k; w_{j,k}));
7. for each (dotted edge) (j, z) ∈ L_d(j) do
8. begin
9. update(heap(z), (j, DIST[j]));
10. if REACH[z] = 0 and DIST[top(heap(z))] < DIST[z] then
11. begin {a new μ-minimal hyperpath has been just found}
12. DIST[z] := DIST[top(heap(z))];
13. for each (full edge) (z, k; w_{z,k}) ∈ L_f(z) do
14. ClosureLast((z, k; w_{z,k}));
15. end
16. end
17. end
18. end;

Figure 11: procedure ClosureLast

from T to j. We will prove that after the insertion of the hyperarc from X to y, DIST[j] = δ(j)
for any simple or compound node for which there is an FD-path from T to j, and DIST[j] = μ∞
otherwise.

Let us call δ'(j) and DIST'[j] the values of δ(j) and DIST[j] after the insertion of the hyperarc
(X, y). It is easy to see that procedure ClosureReach(j) can access a node j only if there exist a
(simple or compound) node k for which REACH[k] = 0 and (thus by lemma 4.1 it is reachable
from T), and a full edge from k to j. As a consequence, a node j will be examined during an
Insert(X, y; w_{x,y}) only if there is a hyperpath from T to j.

This argument assures that if there is no hyperpath from T to j, then DIST[j] remains unchanged
to μ∞ also after the insertion of the hyperarc (X, y). Assume now that there is a hyperpath
from T to j. Due to how the variable DIST[j] is updated, we have:

DIST'[j] = min(DIST[j], f(w_{k,j}, DIST'[k])) = δ'(j),

as can be proved by induction on the hyperarcs visited by ClosureLast using the same argument
as in the proof of lemma 4.1. □

Lemma 5.1 guarantees that the variables LAST allows one to trace correctly a μ-optimal FD-path
from T 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 5.1 Let μ = (f, max, μ_o) be an increasing value-based measure function with values in
the ordered discrete set D = {d_0, d_1, ..., d_K}. Given a hypergraph H with n nodes, in which weighted
hyperarcs may be inserted or their cost decreased in an online fashion, there exists a data structure
which allows:
(i) to check whether there is an FD-path from \( T \) to any other (simple or compound) node in constant time;

(ii) to return an FD-path from \( T \) with minimal \( \mu \)-measure (if one exists) 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 performing \( O(Kh) \) operations (insertion of new hyperarcs or cost decreases) is \( O(K|\mathcal{H}||\log h) \), where \( |\mathcal{H}| \) is the final size of the hypergraph, and \( h \) is the total number of hyperarcs. The space required is \( O(|\mathcal{H}| + n^2) \).

**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 separately consider (a) the time spent by the procedure \texttt{Insert} to handle each modification to the structure of the hypergraph, and (b) the total time required by the procedure \texttt{ClosureLast}, to update the distances from \( T \) (the procedure \texttt{ClosureReach} is called at most once for any (full or dotted) edge in the FD-graph \( G(\mathcal{H}) \)).

(a) Each of the \( O(Kh) \) operations (insertions or cost decreases) might require a search on \( T \), the tree of compound nodes, containing \( O(h) \) nodes. The other operations performed in the procedure \texttt{Insert} require constant time.

(b) For any insertion of hyperarc or any cost decrease, the distance from \( T \) to any node can only decrease. On the other side, any of the \( |\mathcal{H}| \) full or dotted edge \((j,k)\) is scanned by a call to \texttt{ClosureLast} as soon \( \text{DIST}[j] \) decreases. This can happen at most \( O(K) \) times, since \( \text{DIST}[j] \in \{d_0, d_5, \ldots, d_K\} \). The only operation operation in the procedure \texttt{ClosureLast} requiring more than constant time is the handling of the heap.

Hence the theorem follows. \( \square \)

Simple changes are required in procedure \texttt{ClosureLast} in order to deal with the problem of finding maximal hyperpaths when the function \( \psi \) is the \texttt{min} function: this is the case, for example, of the problem of maintaining the maximum bottleneck. In fact, besides the obvious changes in the initialization values for the variables, just the following modifications are required:

- in lines 2 and 10 the occurrences of the symbol "<" are to be changed in ">";
- the heap to be maintained for any compound node is based on \texttt{min}.

The case in which we want to maintain minimal (maximal) hyperpaths and the function \( \psi \) is the \texttt{min} (\texttt{max}) function (as in the problem of maintaining the minimum gap hyperpaths), the situation is even simpler, since we do not need any heap for compound nodes.

### 6 Conclusions

The problem of finding optimal hyperpaths in a dhg is in general NP-hard, also in some simple cases. In section 2 we have introduced the concept of value-based measure as a sufficient condition to guarantee that the problem is tractable. Furthermore we have provided general algorithms to find and maintain hyperpaths matching such optimality criteria.

Analogously to what happens with minimal hyperpaths it may be easily noticed that several combinatorial problems (cuts, matchings etc.) have higher complexity when extended to dhg's. Further research is necessary to investigate under what conditions such problems may remain tractable for dhg's, thereby contributing to a better understanding of this expressive structure.

We finally address some application areas in which the approach based on dhg has provided promising results.

In [3, 4] it is shown how dhg's can be successfully used to answer queries concerning the closure of a set of functional dependencies in a relational database schema. In this field, it seems interesting to explore the possibility of using optimal hyperpaths in a dhg describing the functional dependencies
holding in a database schema, in order to minimize chains of joins for materializing views and for answering queries.

In [6] it is shown that the satisfiability of a set of propositional Horn clauses depends on the existence of a hyperpath in a dgh representing the whole Horn formula. More in general, we might consider a knowledge-based system, based on Horn calculus, with fuzzy production rules. If each rule is given a confidence value representing its reliability, it is interesting to consider various problems, such as the degree of consistency of a set of rules, or the most reliable proof that a given fact is true, or finding the weakest step of such a proof. In addition, the design of a rule-based system may be supported by our algorithms and data structures for dynamic maintenance of dgh's and optimal hyperpaths.

Gallo and Rago [13] show how hypergraphs may be used in the context of deductive databases to check the satisfiability of a Horn formula in a Datalog program. Also in this more general model (first order predicative Horn calculus) minimization of hyperpaths may provide a powerful tool to optimize computations according to several possible criteria, related, for example, to the processing time, or to the number of accesses to secondary storage unit.

In [2] dgh's are used to model a special class of Petri Nets, namely the Conflict-Free Petri nets. This new approach leads to solve several problems (the sets of reachable places and potentially firable transitions, the liveness, and the boundedness of the net) devising algorithms with a time cost which is linear in the size of the net, improving on previous solutions for these problems. If we consider temporized Petri Nets, where a given amount of time is associated with each transition (corresponding to the time required to fire the transition), it would be interesting to investigate some time bounds on the behavior of the net by using the approach proposed in this paper.

References


Figure 4

Figure 5