Classification and traversal algorithmic techniques for general hyperpath optimization problems

Giorgio Ausiello Giuseppe F. Italiano Umberto Nanni Fabiano Sarracco

February 7, 2009

1 Introduction

Directed hypergraphs are a generalization of the concept of directed graphs. 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 *many-to-one* relations arises; a directed hypergraph is useful exactly in these scenarios. It is defined by a set of nodes and a set of hyperarcs, each connecting 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 [Ull82], Horn clauses in propositional calculus [AI91], AND-OR graphs [Nil82, MSS04], Petri nets [AFN92], Datalog [GR90], Operations research [GLNP93], Association Rules in Data Mining [CDP04], management of authorizations in privacy protection [MMZ05].
In several applications of directed hypergraphs, the notions of path and of traversal are required. A *hyperpath*, similarly to the notion of path in directed graphs, consists of a connection among nodes using hyperarcs. However, while paths and shortest paths in directed graphs are standard concepts, and efficient algorithms for their computations are well known, for hypergraphs the corresponding notions of hyperpaths and optimal hyperpaths are very subtle. Moreover hyperpaths are susceptible of different definitions of measure, corresponding to different concepts arising in its various applications. Not surprisingly, some of these measures make the problem of finding optimal hyperpaths NP-hard and even not approximable within a $\log(n)$ threshold. However, if the measure function on hyperpaths matches certain conditions (which we define as *value-based* measure functions), the problem turns out to be, at least in some cases, solvable in polynomial time.
Algorithms and data structures to find optimal hyperpaths have been developed by Knuth in the context of a *grammar problem* [Knu77]. Dynamic maintenance of optimal hyperpaths has been also considered, for example in [RR96], and complexities have been expressed in terms of both amortized and output complexity. In almost all of these works a Dijkstra-like approach is used to find optimal hyperpaths.

In this paper we investigate the threshold between NP-hard optimization problems and problems that are solvable in polynomial time; we provide a uniform framework for metrics in directed hypergraphs, by defining a general notion of measure function on hyperpaths, and specializing it to some of the most intuitive measures, some of which have already been considered in the literature. Furthermore we provide a deeper understanding of hypergraph and hyperpath ciclicity and we point out that some measure functions might induce cycles in optimal hyperpaths; in these cases it is shown that a pure Dijkstra-like approach does not achieve good complexities when updating optimal hyperpaths after hyperarc insertions and deletions.

At last we provide simple and efficient algorithms for the dynamic maintenance of optimal hyperpaths. These algorithms are proved to deal correctly with measure functions giving rise to a special case of cyclic optimal hyperpaths.

Organization of the paper. In Section 2 we present some basic notions. We start by providing the definitions of hypergraph, subhypergraph and hyperpath, and we show how hyperpaths can be expressed in two distinct but complementary ways. While attempting to provide a definition of cyclic hyperpaths (Section 3), it will be required to introduce some tools which will help us to efficiently characterize and manipulate the structure of a hyperpath. In Section 4 we introduce several metrics on hypergraphs and hyperpaths. In Section 5 we introduce a new characterization of measure functions on hyperpaths, and a classification, in Section 5, of such functions according to the behavior of their output. In Section 6 a new classification of optimization problems is provided. In Section 7 the state of the art of the algorithms presented in literature for finding and maintaining optimal hyperpath is reported; the analysis of their complexity provides some motivations for the algorithms we present in Section 8 for dynamic maintenance of optimal hyperpaths.

Part I
Hyperpath optimization problems on directed hypergraphs

2 Basic definitions

Definition 2.1 *A* directed hypergraph $\mathcal{H}$ *is a pair* $\langle N, H\rangle$ *where* N *is a set of nodes and* $H \subset 2^N \times N$ *is a set of hyperarcs. Each hyperarc is an ordered pair* $h = \langle S, t\rangle$*, where the* source set *(or* tail*)* $S \subseteq N$ *is an arbitrary nonempty set of nodes, and the* target node *(or* head*)* $t \in N$ *is a single node.*

Note that a directed graph is a special case of directed hypergraph in which all the source sets have cardinality one.

Remark 2.1 *In literature, different definitions of directed hypergraph are presented. For instance, Gallo et al. [GLNP93] lead to a more generalized definition of directed hypergraph by allowing both the source and the target of a hyperarc to be non-singleton subsets of* N*. Particular cases of hyperarcs are the so called backward hyperarcs (or B-arcs), having a single node as a target, and the forward hyperarcs (or F-arcs), having a single node as a source. A B-Hypergraph (F-Hypergraph) is a directed hypergraph whose hyperarcs are all B-arcs (F-arcs). According to such definitions, in this paper we are considering only B-Hypergraphs.*

Definition 2.2 *Given a node* n*, the* forward star *of* n*, or* $fstar(n)$*, is the set of all its outgoing hyperarcs (i.e. , hyperarcs having node* n *in the source set), while the* backward star *of* n*, or* $bstar(n)$*, is the set of all its incoming hyperarcs (i.e. , hyperarcs whose target node is* n*).*
The outdegree *of* n *is the cardinality of its forward star, while the* indegree *of* n *is the cardinality of its backward star; the* degree *of* n *is the sum of its indegree and outdegree:*

$$outdeg(n) = |fstar(n)|,\ indeg(n) = |bstar(n)|,\ deg(n) = outdeg(n) + indeg(n)$$

An example of the previous definitions is presented in Figure 1. Observe that $bstar(a) = \emptyset$, hence $indeg(a) = 0$, while $fstar(a) = \{\langle a\, b, c\rangle, \langle a, b\rangle\}$, hence $outdeg(a) = 2$; $bstar(c) = \{\langle a\, b, c\rangle\}$ ($indeg(c) = 1$), while $fstar(c) = \{\langle c, d\rangle\}$ ($outdeg(c) = 1$); $bstar(d) = \{\langle c, d\rangle, \langle b, d\rangle\}$ ($indeg(d) = 2$), while $fstar(d) = \emptyset$ ($outdeg(d) = 0$).

Since the source set of a hyperarc may have cardinality greater than one, the property that the sum of the indegrees of all nodes is equal to the sum of the outdegrees of all nodes cannot be extended from directed graphs to directed hypergraphs.

A *self-loop* is a hyperarc whose target node appears also in the source set.

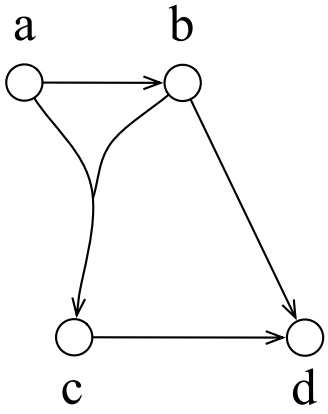

Figure 1: A directed hypergraph.

Definition 2.3 *A directed hypergraph $\mathcal{H}' = \langle N', H' \rangle$ is a* subhypergraph *of $\mathcal{H} = \langle N, H \rangle$ (denoted as $\mathcal{H}' \subseteq \mathcal{H}$) if:*

a) $N' \subseteq N$,

b) $H' \subseteq H$, *and, for each hyperarc* $\langle S, t \rangle \in H'$, $S \subseteq N'$ *and* $t \in N'$,

Furthermore, 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 $\mathcal{H}' = \langle N', H' \rangle$ is said to be the subhypergraph of $\mathcal{H}$ induced by H'.

Before defining the notion of hyperpath in directed hypergraphs, we relate some simple graphs definitions, like paths, walks and cycles, to directed hypergraphs, since they will be recalled afterwards.

Definition 2.4 *A* (directed) walk *of length k in a directed hypergraph from a node x to a node y, is a sequence of nodes and hyperarcs $[x \equiv n_1, h_1, n_2, h_2, \ldots, h_k, n_{k+1} \equiv y]$ such that, for each $j = 1, \ldots, k$, $h_j = \langle S_j, n_{j+1} \rangle \in H$, and $n_j \in S_j$.*
A directed cycle *is a directed walk of length $k > 1$ having $n_1 = n_{k+1}$.*

Now we define hyperpaths; unlike the definition of path, we define (the existence of) a hyperpath in a recursive way.

Definition 2.5 *Let $\mathcal{H} = \langle N, H \rangle$ 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 either*

a) $y \in X$ *(*extended reflexivity*);*

*b) there is a hyperarc $\langle Z, y \rangle \in H$ and hyperpaths from X to each node $z_i \in Z$ (*extended transitivity*).*

If there exists a hyperpath from X to y we say that y is reachable *from X and, in case b), that hyperarc $\langle Z, y \rangle$ is* traversable.

The above recursive definition of hyperpath can be naturally represented by a tree defined as follows:

Definition 2.6 *Let $\mathcal{H} = \langle N, H \rangle$ be a directed hypergraph, $X \subseteq N$ be a non-empty subset of nodes, and y be a node in N. A* hyperpath *(or* unfolded hyperpath *or* hyperpath tree*) from X to y (if it exists) is a tree $t_{X,y}$ recursively defined as follows:*

a) for each (sub)hyperpath obtained by extended reflexivity, the corresponding (sub)tree is empty;

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

A branch *of $t_{X,y}$ is a path from the root to a leaf node of $t_{X,y}$.*

Note that the root of the hyperpath tree $t_{X,y}$ is a hyperarc in $bstar(y)$. Furthermore if $\langle S, t\rangle$ is a leaf in the hyperpath tree, then it must be $S \subseteq X$. An example of hyperpath tree is presented in Figure 2.

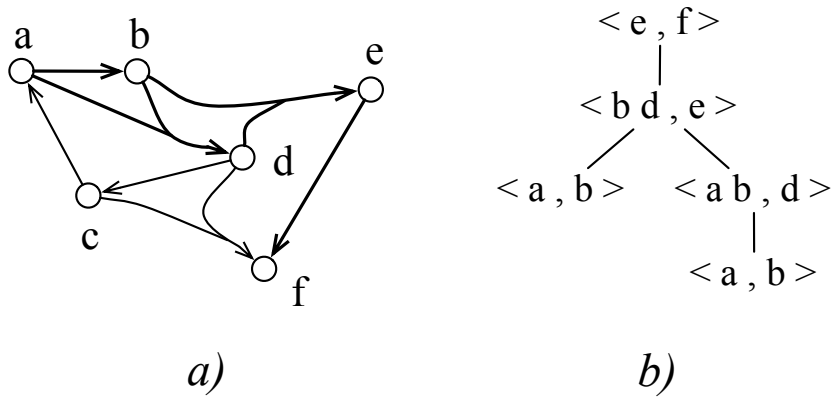

Figure 2: A hyperpath from a to f is represented with bolded hyperarcs in *a)*. In *b)*, the corresponding hyperpath tree ($t_{a,f}$) is showed.

This representation explicitly describes the sequence of hyperarcs as traversed while going from X to y. There is however an alternative and more concise way of describing hyperpaths, defined as follows:

Definition 2.7 *Let $\mathcal{H} = \langle N, H\rangle$ be a directed hypergraph and let $t_{X,y}$ be a hyperpath from a set of nodes $X \subseteq N$ to a target node $y \in N$. The* folded hyperpath *$F(t_{X,y})$ corresponding to $t_{X,y}$ is the subhypergraph of $\mathcal{H}$ induced by the hyperarcs in $t_{X,y}$.*

It is interesting to observe that there is not a one-to-one relationship between unfolded and folded hyperpaths, since distinct (unfolded) hyperpaths may have the same folded representation.
Note also that these two representations can be used in the case of paths in a directed graph as well: given a path π from a node x to a node y, we can describe π either by providing the sequence of all the edges in π as traversed while going from x to y (unfolded description), or by providing the subgraph of G containing exactly the edges of π (folded description) (see, for instance, Figure 3). While the former description may contain the same edge more than once and may not even be finite, since it may contain a cycle which is traversed an unbounded number of times, the latter description is more compact but it may hide the real length of the path.

While turning from graphs to hypergraphs, we can notice moreover that there is an even deeper difference between folded and unfolded hyperpaths: unlike simple paths, in fact, there are even acyclic hyperpaths whose unfolded tree representation is exponentially larger than the corresponding folded representation. An example of this is shown in Figure 4. This behavior partially motivates the analysis of cycles in hypergraphs and hyperpaths we introduce in the following section.

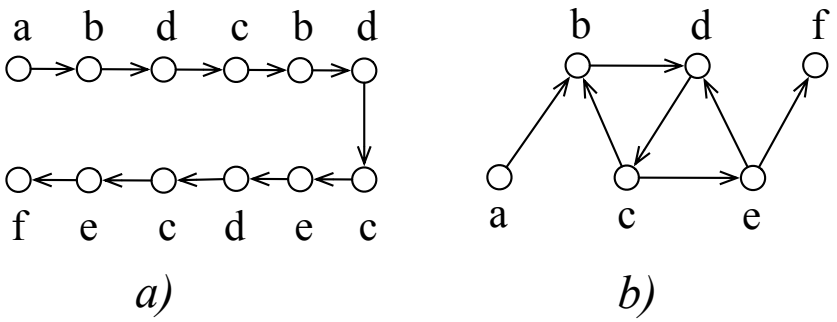

Figure 3: In *a)* an unfolded description of a path from a to f; in b the folded representation of the same path.

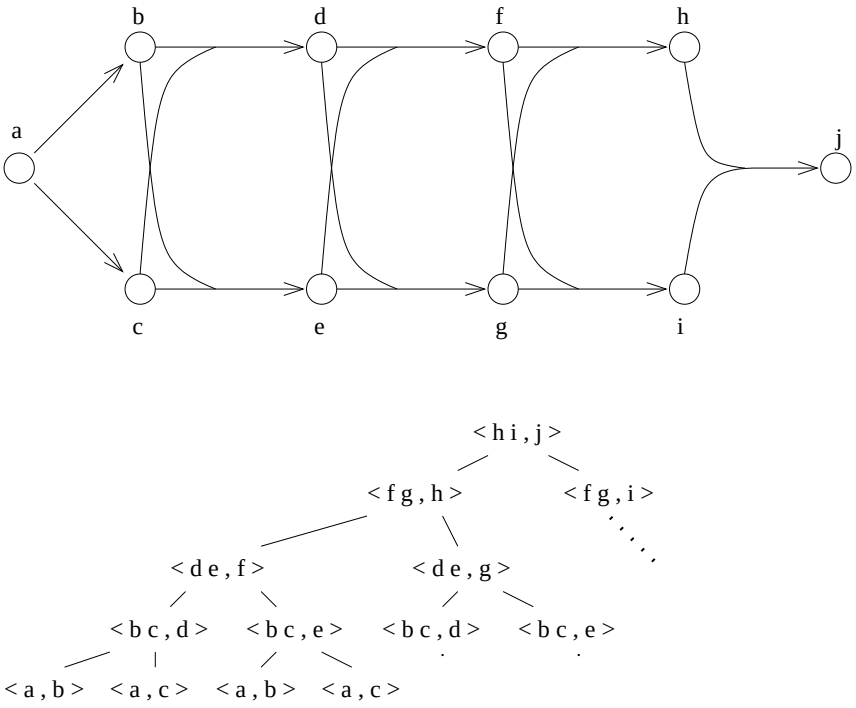

Figure 4: In b), an unfolded hyperpath $t_{a,j}$ of exponential size respect to its folded representation, in a).

3 Cycles in hypergraphs and hyperpaths

First we provide a definition of cyclic hypergraphs by using the notion of directed cycle previously introduced.

Definition 3.1 *A directed hypergraph $\mathcal{H} = \langle N, H \rangle$ is* cyclic *if it contains at least one directed cycle, otherwise it is* acyclic.

Also for directed hypergraphs we can relate the aciclicity of a hypergraph with the existence of a topological ordering of its nodes:

Lemma 3.1 *[GLNP93] A directed hypergraph $\mathcal{H} = \langle N, H \rangle$ is* acyclic *if and only if there exists a topological ordering of the nodes N:*

$$N = [n_{i_1}, n_{i_2}, \ldots, n_{i_n}]$$

such that, for each hyperarc $h = \langle S, t \rangle \in H$, each node in S precedes t in the ordering.

Now we extend the notion of ciclicity to hyperpaths. Since folded hyperpaths are (sub)hypergraphs, we can provide the following definition:

Definition 3.2 *A hyperpath $t_{X,y}$ is* cyclic *if and only if the corresponding folded hyperpath $F(t_{X,y})$ is cyclic, otherwise it is* acyclic.

It would be useful to have a definition of cyclicity that can be directly applied (and checked) on hyperpaths (i.e. , on hyperpath trees). However, unlike simple graphs, this task is not trivial, and requires a deeper understanding of such structures. The rest of this section is therefore devoted to introduce some concepts which will help us to efficiently characterize and manipulate hyperpaths. We define the following two measures.

Definition 3.3 *Let $t_{X,y}$ be a hyperpath. We define:*

- arc-multiplicity *of $t_{X,y}$ ($\Delta(t_{X,y})$): the maximum number of distinct hyperarcs in $t_{X,y}$ having the same node as target:*

  $$\Delta(t_{X,y}) = \max_{i=1,\ldots,|N|} \Delta_i$$

  where $\Delta_i = |\{h \in t_{X,y} : n_i \text{ is target of } h\}|$;

- node-multiplicity *of $t_{X,y}$ ($\Gamma(t_{X,y})$): the maximum number of times a node appears as target in the same branch of $t_{X,y}$:*

  $$\Gamma(t_{X,y}) = \max_{i=1,\ldots,|N|} \Gamma_i$$

  where $\Gamma_i = \max_b |\{h \in \text{branch } b \text{ of } t_{X,y} : n_i \text{ is target of } h\}|$;

Let $X_S \subseteq X$ be the set of sources*, i.e., the nodes in X that actually are within the source set of at least one hyperarc in $t_{X,y}$.*

In both the above definitions we consider all the sources $n_i \in X_S$ as targets of a fictitious hyperarc $h_i = \langle X, n_i \rangle$, therefore their Γ_i and their Δ_i must be increased by one.

Lemma 3.2 *Any hyperpath $t_{X,y}$ having $\Gamma(t_{X,y}) \geq 2$ is cyclic.*

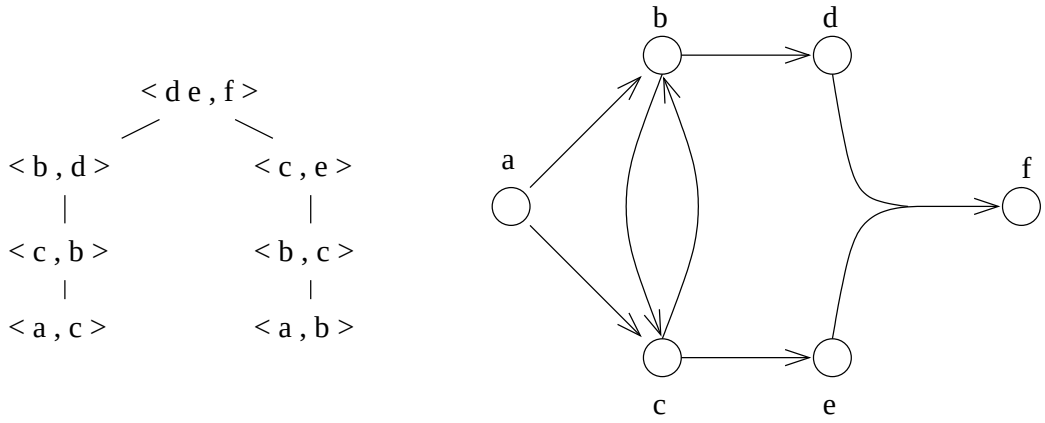

Figure 5: A cyclic hyperpath with $\Delta(t_{a,f})$ and $\Gamma(t_{a,f}) = 1$, and its folded representation.

Proof. Let h and h' be two hyperarcs having the same node n as target and belonging to the same branch of $t_{X,y}$ (we assume that h is the deeper hyperarc). Let $h_1, \ldots, h_k$ be the sequence of hyperarcs belonging to the subbranch delimited by h and h' (in down to up order). Let now n_i be the target node of hyperarc h_i, for each $i = 1, \ldots, k$. It is easy then to check that the sequence $[n_i, h_1, n_{t_1}, h_2, \ldots, h_k, n_{t_k}, h', n_i]$ is a directed cycle in $F(t_{X,y})$. According to Definition 3.2, the hyperpath $t_{X,y}$ is therefore cyclic. □

Note that the converse of Lemma 3.2 does not hold. There are cyclic hyperpaths (like that in Figure 5) such that the hypothesis of Lemma 3.2 is not satisfied (they have in fact $\Gamma = 1$).
Therefore, in order to provide a tighter definition of cyclic unfolded hyperpath, we need a further tool.

Definition 3.4 *Let $t_{X,y}$ be a hyperpath, and let $t^1_{X,z}$ and $t^2_{X,z}$ be two distinct (possibly empty if $z \in X$) subtrees of $t_{X,y}$, representing two (distinct) hyperpaths from X to z. We define as* subtree replacement *(or simply* ST-REP*) the operation of removing the subtree $t^2_{X,z}$ and replacing it with a copy of subtree $t^1_{X,z}$.*

A subtree replacement hence substitutes a hyperpath from X to a node z, with another hyperpath from X to z. Therefore, given a hyperpath $t_{X,y}$, the hyperpath $t'_{X,y}$ obtained by a finite sequence of ST-REPs on $t_{X,y}$ is still a valid hyperpath from X to y. We denote as $t_{X,y} \rightsquigarrow t'_{X,y}$ the fact that hyperpath $t_{X,y}$ is obtained by a (possibly empty) sequence of ST-REPs on $t_{X,y}$. Observe that if $t_{X,y} \rightsquigarrow t'_{X,y}$ then $F(t'_{X,y}) \subseteq F(t_{X,y})$.
Now we can state the following theorem:

Theorem 3.3 *A hyperpath $t_{X,y}$ is* cyclic *if and only if there exists a hyperpath $t'_{X,y}$, such that $t_{X,y} \rightsquigarrow t'_{X,y}$ and $\Gamma(t'_{X,y}) \geq 2$.*

Proof. ($\Rightarrow$) If $t_{X,y}$ is cyclic then $F(t_{X,y})$ contains at least a directed cycle. Let $P = [n_1, h_1, n_2, \ldots, h_k, n_1]$ be one of these cycles. Since each node n_i is the target of a hyperarc, if at least one n_i is a source in X_S then we would immediately have $\Gamma(t_{X,y}) \geq 2$. On the other side, if we assume that, for each $n_i \in P$, $n_i \notin X_{t_{X,y}}$, we can perform the following ST-REPs in $t_{X,y}$: let t_{h_k} be a subtree of $t_{X,y}$ having h_k as root, and let t_{h_1} be a subtree of $t_{X,y}$ having h_1 as root. One of the children of the root of t_{h_1} is a hyperarc h' having n_1 as target. We replace the subtree having h' as root with t_{h_k} still obtaining a valid hyperpath. Let now t_{h_2} be a subtree having h_2 as root. Again one of the root's children is a hyperarc h'' having n_2 as target. We replace the subtree having h'' as root with t_{h_1}. We can then iterate this for each of the k nodes of P, thus obtaining a new hyperpath $t'_{X,y}$ with the sequence of hyperarcs $h_k, h_1, \ldots, h_k$ in a same branch b. It is easy to check that $\Gamma(t'_{X,y}) \geq 2$ since n_1 appears twice in b.

($\Leftarrow$) Let us suppose now that, starting form a hyperpath $t_{X,y}$, we have obtained, by means of a finite sequence of ST-REPs, a new hyperpath $t'_{X,y}$ such that $\Gamma(t'_{X,y}) \geq 2$. By Lemma 3.2, $F(t'_{X,y})$ is cyclic, and since $F(t'_{X,y}) \subseteq F(t_{X,y})$, $F(t_{X,y})$ is cyclic as well. □

At this point it should be interesting to investigate the relationships between the structure of a hyperpath and its behavior respect to ST-REP operations.

Lemma 3.4 *Given a hyperpath $t_{X,y}$, the following statements on $t_{X,y}$ are equivalent:*

a) $t_{X,y}$ *is invariant with respect to ST-REP operations, i.e. if* $t_{X,y} \rightsquigarrow t'_{X,y}$ *then* $t'_{X,y} = t_{X,y}$*;*

b) $\Delta(t_{X,y}) = 1$*;*

c) if $t^1_{X,z}$ *and* $t^2_{X,z}$ *are two subtrees of* $t_{X,y}$ *having the same target node* z *on the root hyperarc, then* $t^1_{X,z} = t^2_{X,z}$*.*

Any hyperpath satisfying the previous statements is called st-replacement invariant *(or* ST-REP-INV*).*

Proof. *($c \Rightarrow a$)* Obvious. If all the subtrees of $t_{X,y}$ having the same root's target node z are identical, no ST-REP operation can modify the hyperpath $t_{X,y}$.
($a \Rightarrow b$) If, by contradiction, $\Delta(t_{X,y}) > 1$ then either one of the following holds:

1. there exist in $t_{X,y}$ two distinct hyperarcs having the same target node z: $h_1 = \langle S_1, z\rangle$ and $h_2 = \langle S_2, z\rangle$. Let us consider the subtrees $t^1_{X,z}$ and $t^2_{X,z}$ having respectively h_1 and h_2 as root; they have the same target node z in the root, we could replace (ST-REP) one of them with the other one, and obtain a distinct valid hyperpath $t'_{X,y}$, which yields a contradiction.

2. there exists in $t_{X,y}$ a hyperarc $h = \langle S, z\rangle$ and $z \in X_S$. We can replace (ST-REP) the subtree $t_{X,z}$ having h as root, with an empty hyperpath $t'_{X,z} = \emptyset$, obtaining a distinct valid hyperpath $t'_{X,y}$, which yields a contradiction.

($b \Rightarrow c$) Let us suppose, by contradiction, that there exist two distinct subtrees $t^1_{X,z}$ and $t^2_{X,z}$ in $t_{X,y}$. Being $\Delta(t_{X,y}) = 1$, it cannot be $z \in X$ since in this case we would have $t^1_{X,z} \equiv t^2_{X,z} \equiv \emptyset$ (leaf of the tree); therefore $z \notin X$ and there is in $t_{X,y}$ only one hyperarc $\langle S, z\rangle$ having z as target, thus $t^1_{X,z}$ and $t^2_{X,z}$ must have it as the same root hyperarc. Therefore there must exists two distinct subtrees:

$$\exists i : t^1_{X,s_i} \neq t^2_{X,s_i}$$

We can however reiterate the previous reasoning on these subtrees by showing that $s_i \notin X$ and there can be only one hyperarc $\langle S', s_i\rangle$ in $t_{X,y}$. In this way we can reach all the leaves of the tree and prove that $t^1_{X,z} \equiv t^2_{X,z}$. □

Remark 3.1 *A consequence of Lemma 3.4 is the fact that a ST-REP-INVhyperpath $t_{X,y}$ is acyclic. On the contrary, also in this case the reverse property does not hold, i.e., there are acyclic hyperpaths such that ST-REP-INV property is not satisfied (e.g. , see Figure 6).*

As a corollary of Theorem 3.3 and Lemma 3.4, if $\Delta(t_{X,y}) = 1$ then $\Gamma(t_{X,y}) = 1$, for each hyperpath $t_{X,y}$.

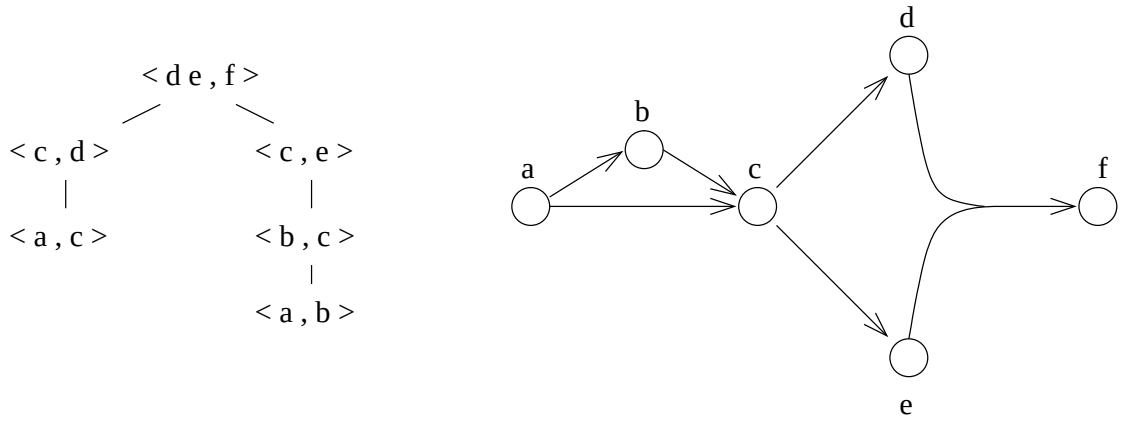

Figure 6: An acyclic hyperpath having $\Delta(t_{a,f}) = 2$ with its folded representation.

4 Metrics over unfolded hyperpaths

In the literature, several measure functions on directed hyperpaths have been considered. Interestingly, some of these measures are related to hypergraphs, and hence based on folded hyperpaths: the total number of source sets, the total number of hyperarcs, or the sum of their costs, in the case of weighted hypergraphs. Unfortunately, in [IN89] the authors prove, by means of a reduction from the Set Cover problem, that the problems of finding hyperpaths with minimum cost, size, number of hyperarcs are all **NP**-hard problems. Moreover, a deeper analysis of such reduction shows that such problems are not approximable by a polynomial time algorithm within a constant factor.

The scenario completely changes if we consider instead measures over hyperpaths. In this sections we define a general notion of measure function that can be applied to unfolded hyperpaths, and next specialize it to some of the most intuitive measures presented in literature. We will then show that measures defined in this way give rise to polynomially solvable optimization problems.

First, we extend the notion of hypergraph in order to take into account also possible weights associated to hyperarcs.

Definition 4.1 *A* weighted directed hypergraph $\mathcal{H}^{\mathcal{W}}$ *is a triple* $\langle N, H; w\rangle$*, where* $\langle N, H\rangle$ *is a directed hypergraph and each hyperarc* $\langle S, t\rangle \in H$ *is associated to a real value* $w_{\langle S,\, t\rangle} \in \mathbb{R}$ *called* weight of the hyperarc.

Definition 4.2 *Given a weighted directed hypergraph* $\mathcal{H}^{\mathcal{W}} = \langle N, H; W\rangle$*,* $\mu = (\mu_0, \mu_\infty, \psi, f)$ *is a* value-based functional measure on hyperpaths *(or* value-based measure function*) if for each set of nodes* $X \subseteq N$ *and* $y \in N$*:*

- *if* X *and* y *are not connected then* $\mu(t_{X,y}) = \mu_\infty$*;*
- *for each hyperpath* $t_{X,y} = \emptyset$ *(*$y \in X$*) then* $\mu(t_{X,y}) = \mu_0$*;*
- *for each hyperpath* $t_{X,y} = \{\langle Z, y\rangle\} \cup t_{X,z_1} \cup t_{X,z_2} \cup \ldots \cup t_{X,z_k}$*, then:*

$$\mu(t_{X,y}) = f_{\langle Z,\, y\rangle}(w_{\langle Z,\, y\rangle}, \psi_{\langle Z,\, y\rangle}(\mu(t_{X,z_1}), \mu(t_{X,z_2}), \ldots, \mu(t_{X,z_k}))).$$

where:

- ψ *associates to each hyperarc* $\langle S, t\rangle$ *a monotone non decreasing commutative and associative function from* $|S|$*-tuples of reals to reals:* $\psi_{\langle S,\, t\rangle} : \mathbb{R}^{|S|} \to \mathbb{R}$*;*

- *f associates to each hyperarc $\langle S, t\rangle$ a monotone non decreasing function from a pair of reals to reals:* $f_{\langle S,t\rangle} : \mathbb{R}^2 \to \mathbb{R}$.

Notice that the recursive definition of value-based measure function on the structure of a hyperpath does not depend on combinatorial constraints on the set of affected hyperarcs, but only on the measures of the component sub-hyperpaths. If we consider the cost of a (folded) hyperpath, defined as the sum of the costs of its hyperarcs, we can similarly define the traversal cost of an unfolded hyperpath as the cost of the root of the hyperpath, 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.

A number of metrics for directed hyperpaths have been introduced in the literature. As an example, the *traversal cost* $c(t_{X,y})$ of a hyperpath $t_{X,y}$ is inductively defined as follows:
a) if $t_{X,y} = \emptyset$ $(y \in X)$ then: $c(t_{X,y}) = 0$;
b) if $t_{X,y}$ has root $\langle Z, y\rangle$ with subtrees $t_{X,z_1}, t_{X,z_2}, \ldots, t_{X,z_k}$, then:

$$c(t_{X,y}) = w_{\langle Z,\, y\rangle} + \sum_{z_i \in Z} c(t_{X,z_i}).$$

Several other measures, which are of interest in some applications, can be defined in a recursive way on unfolded hyperpaths. In the table below we summarize the most common definitions.

measure function	notation	reflexivity $t_{X,y} = \emptyset$	general case $t_{X,y} = t_{X,z_1}, t_{X,z_2}, \ldots, t_{X,z_k}$	no hyperpath
traversal cost	$c(t_{X,y})$	0	$c(t_{X,y}) = w_{\langle Z,\, y\rangle} + \sum_{z_i \in Z} c(t_{X,z_i})$	$+\infty$
rank	$r(t_{X,y})$	0	$r(t_{X,y}) = w_{\langle Z,\, y\rangle} + \max_{z_i \in Z} r(t_{X,z_i})$	$+\infty$
gap	$g(t_{X,y})$	0	$g(t_{X,y}) = w_{\langle Z,\, y\rangle} + \min_{z_i \in Z} g(t_{X,z_i})$	$+\infty$
bottleneck	$b(t_{X,y})$	$+\infty$	$b(t_{X,y}) = \min\left(w_{\langle Z,\, y\rangle}, \min_{z_i \in Z}\left(b(t_{X,z_i})\right)\right)$	0
threshold	$t(t_{X,y})$	0	$t(t_{X,y}) = \max\left(w_{\langle Z,\, y\rangle}, \max_{z_i \in Z}\left(t(t_{X,z_i})\right)\right)$	$+\infty$

It is easy to see that all these metrics are value-based measure functions, and they can be restated as shown in the following table.

meas.funct. μ	$f(w, \psi)$	$\psi(\mu_1, \ldots, \mu_k)$	μ_0	μ_∞
traversal cost	$+$	$\sum$	0	$+\infty$
rank	$+$	$\max$	0	$+\infty$
gap	$+$	$\min$	0	$+\infty$
bottleneck	$\min$	$\min$	$+\infty$	0
threshold	$\max$	$\max$	0	$+\infty$

Note that in the special case where a directed hypergraph is a directed graph, the traversal cost, the rank and the gap collapse to the usual definition of cost of a (weighted) path. Furthermore, the bottleneck coincides with the bottleneck capacity of the path (i.e., the edge with minimum capacity of the path), and the threshold is the maximum cost in the path.

5 Properties of measure functions

In order to characterize optimization problems that can be solved in polynomial time, we now present a classification of measure functions. In particular, our interest is focused on the behavior of the output of a function related to the values of its arguments.

In his definition of grammar problem [Knu77], Knuth generalizes the single-source shortest path problem to context free grammars, and provides a classification on the functions associated to the productions of the grammar:

Definition 5.1 *[Knu77] Let $g(x_1, \ldots, x_k)$ be a function from D^k into D, monotone nondecreasing in each variable.*

- *g is a* superior function *on D^k ($g \in SUP$) if, for each $x_1, \ldots, x_k \in D^k$:*

$$g(x_1, \ldots, x_k) \geq \max(x_1, \ldots, x_k);$$

- *g is an* inferior function *on D^k ($g \in INF$) if, for each $x_1, \ldots, x_k \in D^k$:*

$$g(x_1, \ldots, x_k) \leq \min(x_1, \ldots, x_k).$$

Examples of *SUP* functions are $\max_{1 \leq i \leq k}\{x_i\}$ in $\mathbb{R}^k$, and $\Sigma_{i=1}^{k} x_i$ in $[0, +\infty]^k$. Examples of *INF* functions are: $\min_{1 \leq i \leq k}\{x_i\}$ in $\mathbb{R}^k$, and the product $\Pi_i\{x_i\}$ in $[0, 1]^k$,

Ramalingam and Reps in [RR96] introduce a generalization of these classes of functions:

Definition 5.2 *[RR96] Let $g(x_1, \ldots, x_k)$ be a function from D^k into D, monotone nondecreasing in each variable.*

- *g is a* weakly superior function *in D^k ($g \in WSUP$) if, for each $x_1, \ldots, x_k \in D^k$, and for each $i = 1, \ldots, k$:*

$$g(x_1, \ldots, x_k) < x_i \;\Rightarrow\; g(x_1, \ldots, x_i, \ldots, x_k) = g(x_1, \ldots, \infty, \ldots, x_k)$$

- *g is a* weakly inferior function *in D^k ($g \in WINF$) if, for each $x_1, \ldots, x_k \in D^k$, and for each $i = 1, \ldots, k$:*

$$g(x_1, \ldots, x_k) > x_i \;\Rightarrow\; g(x_1, \ldots, x_i, \ldots, x_k) = g(x_1, \ldots, -\infty, \ldots, x_k)$$

It is obvious that if a function is *SUP*, then it is *WSUP* ($SUP \subset WSUP$), and if it is *INF*, then it is *WINF* ($INF \subset WINF$). Examples of *WSUP* functions that are not *SUP* are $\min_{1 \leq i \leq k}\{x_i\}$ and any constant function. Examples of *WINF* functions that are not *INF* are $\max_i\{x_i\}$ and any constant function.

The following lemma summarizes the properties holding in the case of a generic composition of these functions:

Lemma 5.1 *If we are given the functions $f, g_1, \ldots, g_h$ then their composition $f(g_1(\ldots), \ldots, g_h(\ldots))$ has the following properties:*
1. if $f, g_1, \ldots, g_h \in SUP$, then $f(g_1, \ldots, g_h) \in SUP$;
2. if $f, g_1, \ldots, g_h \in INF$, then $f(g_1, \ldots, g_h) \in INF$;
3. if $f, g_1, \ldots, g_h \in WSUP$, then $f(g_1, \ldots, g_h) \in WSUP$;
4. if $f, g_1, \ldots, g_h \in WINF$, then $f(g_1, \ldots, g_h) \in WINF$.

Both [Knu77] and [RR96] consider also the class of *strict (weakly)* superior and inferior functions, characterized by a strict inequality between the value of the function at hand and each of its arguments: this leads to the classes *SSUP*, *SWSUP*, *SINF*, and *SWINF*.
For example a function $g(x_1, \ldots, x_k)$ from D^k into D is a *strict superior function* (*SSUP*) in D if it is monotone nondecreasing in each variable and if:

$$g(x_1, \ldots, x_k) > \max(x_1, \ldots, x_k), \text{for each } x_1, \ldots, x_k \in D$$

Also for strict functions we can state a lemma summarizing the properties holding in the case of a generic composition:

Lemma 5.2 *If we compose a function f with functions $g_1, \ldots, g_h$, the following properties hold:*

1. *if* $f \in SSUP$, *and* $g_i \in SUP$ *for all* i *(or vice versa) then* $f(g_1, \ldots, g_h) \in SSUP$;
2. *if* $f \in SINF$, *and* $g_i \in INF$ *for all* i *(or vice versa) then* $f(g_1, \ldots, g_h) \in SINF$;
3. *if* $f \in SWSUP$, *and* $g_i \in WSUP$ *for all* i *(or vice versa) then* $f(g_1, \ldots, g_h) \in SWSUP$;
4. *if* $f \in SWINF$, *and* $g_i \in WINF$ *for all* i *(or vice versa) then* $f(g_1, \ldots, g_h) \in SWINF$.

Let us state another important property of weak functions. For example, let us assume $g : D^k \to D$ to be a *SWSUP* function, and let $g(x_1, \ldots, x_k) = l$. We can then partition the set $\{x_1, \ldots, x_k\}$ into two subsets:

- the set $I = \{x_i \,|\, x_i \geq l\}$, of *irrelevant variables*;
- the set $R = \{x_i \,|\, x_i < l\}$ of *relevant variables*;

In [RR96] it is proved that for each value assigned to irrelevant variables in the interval $[l, \infty]$, the equality $g(x_1, \ldots, x_k) = l$ holds. An equivalent property can be stated for *WSUP*, *WINF* and *SWINF* functions as well.

We now relate all these properties to our definition of hypergraphs. Let us recall that the measure μ of any hyperpath $t_{X,y}$ having $\langle Z, y\rangle$ as the root hyperarc is given by

$$\mu(t_{X,y}) = f_{\langle Z,\, y\rangle}(w_{\langle Z,\, y\rangle}, \psi_{\langle Z,\, y\rangle}(\mu(t_{X,z_1}), \mu(t_{X,z_2}), \ldots, \mu(t_{X,z_k}))).$$

According to Lemma 5.1, if all the functions $f_{\langle S,\, t\rangle}$, $\psi_{\langle S,\, t\rangle}$ (for all $\langle S, t\rangle \in H$) composing the measure function μ, belong to the same class (say, *SUP*), then μ belongs to that class (*SUP*) as well. As a special case, we will consider in many cases measure functions where all $\psi_{\langle S,\, t\rangle}$, with $\langle S, t\rangle \in H$, are uniform, as well as all $f_{\langle S,\, t\rangle}$. For the measure functions defined in Section 4 we are exactly in this situation. For example, if we consider a weighted hypergraph having all positive weights, the computation of the traversal costs will result in a *SSUP* function, while the computation of the gap will result in a *SWSUP* function.

The diagram in Figure 7 summarizes the relationships between function classes presented in this section. The classes in the diagram are also populated by the value based measure functions presented in Section 4. For instance, the rank measure function is *SUP* if all the hyperarcs weights are non-negative, since rank is given by the composition $f(\psi(\cdot))$, where f (sum) is a *SUP* function, and ψ (max) is *SUP* as well; hence Lemma 5.1 holds. Notice that if hyperarc weights are all positive, then rank is *SSUP*, since f (sum) is *SSUP* in this case, and hence Lemma 5.2 holds.
Moreover, observe that any constant function is in $SWSUP \cap SWINF$, while average function, where $\psi(\mu_1, \ldots, \mu_k) = \sum_i^k \mu_i/k$ and $f = (w + \psi(\cdot))/2$, does not belong to any of the function classes presented in this section.

In the next section we will see how these properties on measure functions influence the structure of optimal hyperpaths, and hence the polynomial solvability of the problem.

6 Hyperpath Optimization Problems on Directed Hypergraphs

In this section we provide a characterization of the optimization problems. In particular, we classify them according to the structure of the optimal hyperpaths they produce.
An optimization problem $\mathcal{P} = (\Phi, \mu)$ on directed hyperpaths is characterized by an *optimization criterion* $\Phi \in \{\min, \max\}$, and a measure function μ on hyperpaths.

In the following we use the notation $a \prec b$ (or $a \preceq b$) to indicate that the value a is better (or not worse) than the value b, according to a specific optimization criterion Φ. Similarly, we denote as $a \succ b$ (or $a \succeq b$) the fact that the value a is worse (or not better) than the value b. For instance, if $\Phi = \max$, then the symbol $\prec$ is equivalent to $>$, $\preceq$ is equivalent to $\geq$, $\succ$ is equivalent to $<$ and $\succeq$ is equivalent to $\leq$.

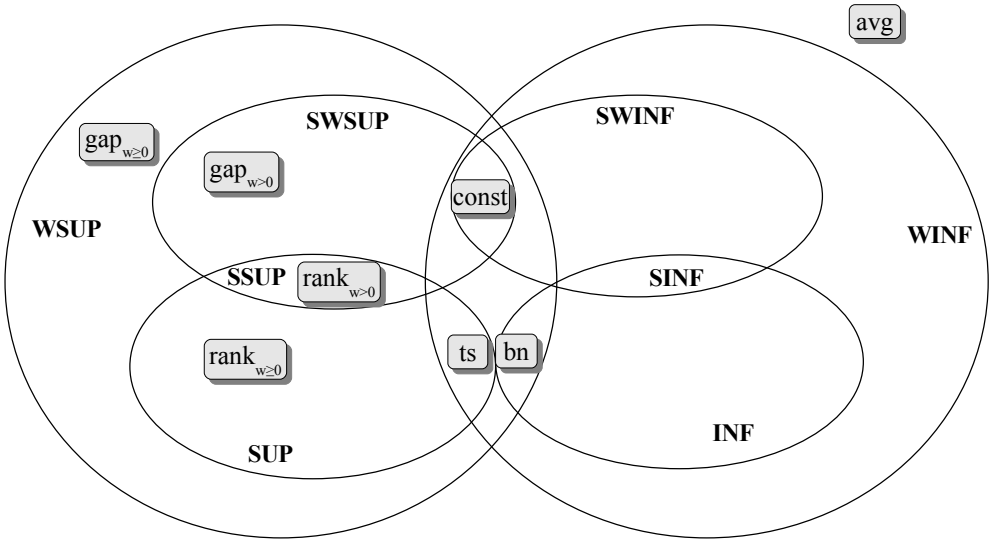

Figure 7: The relationships between measure functions and some examples (ts = threshold, bn = bottleneck).

Definition 6.1 *Given a directed hypergraph $\mathcal{H}$ and an optimization problem $\mathcal{P} = (\Phi, \mu)$, an* optimal hyperpath *from a set of nodes X to a node y, is a hyperpath $t^*_{X,y}$ whose measure $\mu(t^*_{X,y})$ is the best (minimum or maximum) among the measures of all the hyperpaths from X to y in $\mathcal{H}$:*

$$t^*_{X,y} \text{ is optimal} \Leftrightarrow \text{ for each hyperpath } t_{X,y} : \mu(t_{X,y}) \succeq \mu(t^*_{X,y})$$

The following lemma expresses the equivalence of hyperpaths obtained through subtree-replacements:

Lemma 6.1 *Let $t_{X,y}$ be a hyperpath for a problem $\mathcal{P} = (\Phi, \mu)$. Let $t^1_{X,z}$ and $t^2_{X,z}$ be two distinct subhyperpaths of $t_{X,y}$. If $\mu(t^1_{X,z}) \preceq \mu(t^2_{X,z})$, we can replace (ST-REP) $t^2_{X,z}$ with $t^1_{X,z}$ obtaining a hyperpath $\tilde{t}_{X,y}$ such that $\mu(\tilde{t}_{X,y}) \preceq \mu(t_{X,y})$.*

Proof. It is a straightforward consequence of the monotonicity of the ψ functions composing μ. □

We now give the following characterization on optimization problems:

Definition 6.2 *An optimization problem $\mathcal{P} = (\Phi, \mu)$ is* k-cycle-convergent *for some $k \geq 0$ if, for any hypergraph $\mathcal{H}^{\mathcal{W}}$ and for any optimal hyperpath $t^*_{X,y}$ between a set of nodes X and a target node y in $\mathcal{H}^{\mathcal{W}}$, there exists an optimal hyperpath $\tilde{t}^*_{X,y}$ such that $t^*_{X,y} \rightsquigarrow \tilde{t}^*_{X,y}$ and $\Gamma(\tilde{t}^*_{X,y}) \leq k + 1$.*
An optimization problem that is 0-CYCLE-CONV is said to be cycle-invariant.

Definition 6.3 *An optimization problem $\mathcal{P} = (\Phi, \mu)$ is* cycle-unbounded *(or* Γ-unbounded*) if it is not k-CYCLE-CONV, for any $k \in \mathbb{N}$.*

Note that if we are given a k-*CYCLE-CONV* optimization problem on hyperpaths, there can exist an optimal hyperpath $t^*_{X,y}$ of unbounded size, but there always exists another optimal equivalent (bounded) hyperpath $\tilde{t}^*_{X,y}$ where each node is target at most of $k + 1$ hyperarcs in the same branch of $\tilde{t}^*_{X,y}$. For a *CYCLE-UNB* optimization problem there exist optimal hyperpaths having no optimal equivalent bounded hyperpaths.

In the previous definitions we focused our attention on the value of Γ of a particular hyperpath, i.e. the maximum number of occurrences of the same node in the same branch of a hyperpath. Nothing has been stated about the value of Δ of such hyperpaths, i.e. the maximum indegree of the nodes in the folded representation. The following lemma provides a bound on this value.

Lemma 6.2 *Given a k-CYCLE-CONV optimization problem, and an optimal hyperpath $t^*_{X,y}$ there always exists an optimal hyperpath $\tilde{t}^*_{X,y}$ such that $t^*_{X,y} \rightsquigarrow \tilde{t}^*_{X,y}$ and*

$$\Delta(\tilde{t}^*_{X,y}) \leq \Gamma(\tilde{t}^*_{X,y}) \leq k+1.$$

Proof. For a k-*CYCLE-CONV* optimization problem, by Definition 6.2 we can always get an optimal hyperpath $\tilde{t}^*_{X,y}$ having $\Gamma(\tilde{t}^*_{X,y}) \leq k$. Starting from $\tilde{t}^*_{X,y}$, we consider each node $z \in N$ for which $\Delta_z(\tilde{t}^*_{X,y}) > k+1$. Let H^1_z be the set of the first hyperarcs having z as target, encountered while traversing $\tilde{t}^*_{X,y}$ from the root to any of its leaf nodes. Let T^1_z be the corresponding set of subtrees of $\tilde{t}^*_{X,y}$ having as root the hyperarcs in H^1_z. It is easy to notice that these subtrees are disjoint and their union includes all the hyperarcs in $\tilde{t}^*_{X,y}$ having z as target. Let $t^1_{X,z}$ be the element of T^1_z having the best measure μ. Since $\Gamma_z(t^1_{X,z}) \leq k+1$, we can perform a series of ST-REP in $\tilde{t}^*_{X,y}$, such that each subtree in T^1_z is replaced with a copy of $t^1_{X,z}$, still obtaining (by Lemma 6.1) an optimal hyperpath with $\Gamma_z(t^1_{X,z}) \leq k+1$. If $\Delta_z(t^1_{X,z}) \leq k+1$, then $\Delta_z(\tilde{t}^*_{X,y}) \leq k+1$ as well, and the task is completed (at least for z). In the other case, we have to iterate the previous procedure in each copy of $t^1_{X,z}$, in order to make $\Delta_z(t^1_{X,z}) \leq k+1$. Let then H^2_z be the set of the first hyperarcs having z as target (obviously excluding the root), encountered while traversing $t^1_{X,z}$ from the root to any of its leaves, let T^2_z be the corresponding set of subtrees having as root the hyperarcs in H^2_z, and let $t^2_{X,z}$ be the element of T^2_z having the best measure. We ST-REP each subtree in T^2_z (in each occurrence of $t^1_{X,z}$ in $\tilde{t}^*_{X,y}$) with $t^2_{X,z}$. Notice that in this case $\Gamma_z(t^2_{X,z}) = \Gamma_z(t^1_{X,z}) - 1 \leq k$, and we need that $\Delta_z(t^2_{X,z}) \leq k$, since in the rest of the tree we have only one distinct hyperarc having z as target (the root of $t^1_{X,z}$). We have possibly to iterate this procedure for $\Gamma_z(t^1_{X,z}) \leq k+1$ times, but at the end, at most $\Gamma_z(t^1_{X,z}) \leq k+1$ distinct hyperarcs having z as target will be left in $\tilde{t}^*_{X,y}$, hence $\Delta_z \leq k+1$ in the produced hyperpath.
By iterating this procedure for each node, at the end we obtain a new hyperpath $\tilde{t}^{*'}_{X,y}$ such that $\Delta(\tilde{t}^{*'}_{X,y}) \leq \Gamma(\tilde{t}^{*'}_{X,y}) \leq k$, since ST-REP operations do not introduce new hyperarcs in the hyperpath.
□

Observe that another way of stating Lemma 6.2 is to say that if we are given a k-*CYCLE-CONV* optimization problem, we can always represent an optimal hyperpath by providing at most $k+1$ predecessors for each node. Recall that we already proved that if $\Delta(t_{X,y}) = 1$ then $\Gamma(t_{X,y}) = 1$, hence, in presence of cycle-invariant optimization problems we can represent optimal hyperpaths by providing one predecessor for each node.

In Section 5 we presented some common value-based measures on hyperpaths and classified them as *SUP/INF* functions proposed by Knuth [Knu77] and Ramalingam and Reps [RR96]. In this section we provide a classification of optimization problems according to the structure of the optimal hyperpaths. The following theorems lay a bridge between these two classifications.

Theorem 6.3 *Let $\mathcal{P} = (\Phi, \mu)$ be an optimization problem on hypergraphs. Then the following properties hold:*
a) if $\Phi = \min$ and μ is a SUP function then $\mathcal{P} = (\Phi, \mu)$ is cycle-invariant;
b) if $\Phi = \max$ and μ is a INF function then $\mathcal{P} = (\Phi, \mu)$ is cycle-invariant;
c) if $\Phi = \min$ and μ is a WSUP function then $\mathcal{P} = (\Phi, \mu)$ is 1-CYCLE-CONV;
d) if $\Phi = \max$ and μ is a WINF function then $\mathcal{P} = (\Phi, \mu)$ is 1-CYCLE-CONV.

Proof. Let us start proving *(a)*. Let $t^*_{X,y}$ be an optimal hyperpath between a source set X and a target node y, for an optimization problem $\mathcal{P} = (\Phi, \mu)$, where μ is a *SUP* function. If $\Gamma(t^*_{X,y}) > 1$, then there is a branch in such tree containing two hyperarcs with the same node as a target: $\langle Y_1, z\rangle$, $\langle Y_2, z\rangle$. Without loss of generality, suppose that the subtree $t^1_{X,z}$ rooted at $\langle Y_1, z\rangle$, contains the subtree

$t^2_{X,z}$ rooted at $\langle Y_2, z\rangle$. Let $\mu(t^1_{X,z})$ and $\mu(t^2_{X,z})$ be the measures of $t^1_{X,z}$ and $t^2_{X,z}$ respectively. Due to the *SUP* nature of μ, we have that $\mu(t^1_{X,z}) \geq \mu(t^2_{X,z})$. By Lemma 6.1, we can replace (ST-REP) subtree $t^1_{X,z}$ with subtree $t^2_{X,z}$, still obtaining an optimal hyperpath $t^{*'}_{X,y}$. We can repeat this operation until $\Gamma(t^*_{X,y}) = 1$.
Similar considerations can be used to prove *(b)*.
To prove case *(c)*, suppose that we have an optimal hyperpath $t_{X,y} = \{\bigcup_{i=1,\ldots,k} t_{X,z_i}\} \cup \{\langle Z, y\rangle\}$, where $Z = \{z_1, \ldots, z_k\}$, such that $\Gamma(t_{X,y}) > 2$. Since the measure function is *WSUP*, we can split the set of nodes Z into two subsets Z_R and Z_I such that:
1. $z_i \in Z_R \iff \mu(t_{X,y}) \geq \mu(t_{X,z_i})$ (relevant nodes)
2. $z_i \in Z_I \iff \mu(t_{X,y}) < \mu(t_{X,z_i})$ (irrelevant nodes)

For each irrelevant node z_i, the value of $\mu(t_{X,y})$ does not depend on the value of $\mu(t_{X,z_i})$, in fact $\mu(t_{X,y}) = \mu(t_{X,y})\Big|_{\mu(t_{X,z_i})=\infty}$. The role of the irrelevant subhyperpath t_{X,z_i} is to propagate the reachability from the root to z_i, regardless of its measure. Hence we can replace this generic hyperpath with one having $\Gamma(t^a_{X,z_i}) = 1$, by using the same technique described in case *(a)*, and without increasing the value of the measure $\mu(t_{X,y})$. Note that this hyperpath may have to contain node y as a possible target in some intermediate node. We can recursively proceed on the subtrees t_{X,z_i} until all relevant edges have been found, and the irrelevant subtrees have been replaced by subhyperpaths having $\Gamma = 1$. Notice that the subtree $t^R_{X,y}$ induced by the relevant edges has the same property that we have considered in case *(a)*: if a target node z appears twice in a branch of $t^R_{X,y}$, say $\langle Y_1, z\rangle$, and $\langle Y_2, z\rangle$ with the former node above the latter, we can replace (ST-REP) the whole subtree having $\langle Y_1, z\rangle$ as root, with the subtree having $\langle Y_2, z\rangle$ as root, and still obtain a hyperpath from X to y, whose measure can not be larger than $\mu(t_{X,y})$.
Case *(d)* can be proved similarly. □

The proof of Theorem 6.3 has an intrinsic relevance, because it allows us to point out an interesting property of 1-*CYCLE-CONV* problems which will be exploited in our algorithms. In the case that we are minimizing a *WSUP* function or we are maximizing a *WINF* function there always exists an optimal hyperpath, having $\Gamma = \Delta = 2$ in which this property is satisfied: when a node in the hyperpath has $\Gamma = 2$ then the measure of the first traversal of that node is irrelevant for the measure of the whole hyperpath, but provides the reachability of the node itself. As we will see, this will allow us to use a two phase algorithm in which the first (irrelevant) predecessor of a node in the hyperpath is computed with a simple propagation technique while the relevant one is computed by using a priority queue of nodes. A relevant consequence to point out is that, when dealing with 1-*CYCLE-CONV* problems, in order to provide optimal hyperpaths as output, we need data structures able to maintain two predecessors for each node.

In Theorem 6.3 we found some problems that have always bounded solutions. There are however common problems that are *CYCLE-UNB*.

Theorem 6.4 *Let $\mathcal{P} = (\Phi, \mu)$ be an optimization problem on hypergraphs. The following properties hold:*
a) if $\Phi = \max$ and μ is a SSUP function then $\mathcal{P} = (\Phi, \mu)$ is CYCLE-UNB;
b) if $\Phi = \min$ and μ is a SINF function then $\mathcal{P} = (\Phi, \mu)$ is CYCLE-UNB.

By using the previous results, we can characterize in a general and unified framework several optimization problems on hypergraphs, as summarized in the table of Figure 8. In this table we recall some measure functions defined in the previous section in terms of the functions $f(w, \psi)$ and $\psi(\mu_1, \ldots, \mu_k)$. The resulting properties of the measure function and the properties of the related optimization problems are directly derived from the arguments given before.

We also provide a characterization of the *closure* problem as an optimization problem: in this case

measure function μ	$f(w,\psi)$	$\psi(\mu_1,\ldots,\mu_k)$	resulting properties	MIN problem	MAX problem
traversal cost $w>0$	$+$ *SSUP*	$\sum$ *SSUP*	*SSUP*	*CY-INV*	*CYCLE-UNB*
rank $w>0$	$+$ *SSUP*	$\max$ *SUP,WINF*	*SSUP*	*CY-INV*	*CYCLE-UNB*
gap $w>0$	$+$ *SSUP*	$\min$ *WSUP,INF*	*SWSUP*	1-*CYCLE-CONV*	*CYCLE-UNB*
bottleneck $w>0$	$\min$ *WSUP,INF*	$\min$ *WSUP,INF*	*WSUP,INF*	1-*CYCLE-CONV*	*CY-INV*
threshold $w>0$	$\max$ *SUP,WINF*	$\max$ *SUP,WINF*	*SUP,WINF*	*CY-INV*	1-*CYCLE-CONV*
closure $w=1$	$\min$ *WSUP,INF*	$\min$ *WSUP,INF*	*WSUP,INF*	1-*CYCLE-CONV*	*CY-INV*

Figure 8: Characterization of measure functions on hypergraphs.

we can use the $\min$ function with uniform hyperarcs weights equal to 1. Of course, the *maximum closure* corresponds to the traditional transitive closure over hypergraphs, investigated in [AI91, AIN90].

By observing the previous results, one might ask if the classes of k-*CYCLE-CONV* optimization problems do not collapse for $k \geq 2$. The following class of problems provides a negative answer to this question.

Let $||$ be the bitwise OR operation on natural numbers represented in binary format, e.g. $5\,(101)\,||\,2\,(10) = 7$, $7\,(111)\,||\,3\,(11) = 7$, and let χ be a fixed positive integer value. Let $\bar{\mu}_\chi = (\mu_0, \mu_\infty, \psi, f)$ be the following measure function:

- $\mu_0 = 0$, $\mu_\infty = -\infty$
- $\psi_{\langle S,t\rangle} : \mathbb{N}^{|S|} \rightarrow \mathbb{N}$ is the following function:

$$\psi_{\langle S,t\rangle} = \bar{\mu}_\chi(t_{X,s_1})\,||\,\bar{\mu}_\chi(t_{X,s_2})\,||\,\ldots\,||\,\bar{\mu}_\chi(t_{X,s_{|S|}})$$

- $f_{\langle S,t\rangle} : \mathbb{N}^2 \rightarrow \mathbb{N}$ is given by

$$f_{\langle S,t\rangle} = \psi_{\langle S,t\rangle}\,||\,2^{\min(|S|,\chi)-1}$$

It is easy to check that the measure function $\bar{\mu}_\chi$ keeps track of the cardinality of the source sets in each hyperarc of $t_{X,t}$. In particular, the i-th less relevant digit (with $i \leq \chi$) of the binary representation of $\bar{\mu}_\chi(t_{X,t})$ is equal to 1 if and only if there is at least one source set of cardinality i in the hyperarcs of $t_{X,t}$.

Given a hypergraph $\mathcal{H}$, a source set X and a target node y, we are considering the family of optimization problems $\mathcal{P}_\chi = (\max, \bar{\mu}_\chi)$. The following theorem holds:

Theorem 6.5 *The problem $\mathcal{P}_\chi = (\max, \bar{\mu}_\chi)$ is $(\chi-1)$-CYCLE-CONV and it is not $(\chi-2)$-CYCLE-CONV, for any $\chi \geq 2$.*

Proof. Let us fix χ and let $t_{X,y}$ be an optimal solution for the $\mathcal{P}_\chi = (\max, \bar{\mu}_\chi)$ problem such that $\Gamma(t_{X,y}) > \chi$. This means that there exists a branch in $t_{X,y}$ containing at least $\chi+1$ hyperarcs with the same node (say z) as a target. Let us consider the sequence of subtrees $t^1_{X,z} \supset t^2_{X,z} \supset \ldots \supset t^{\chi+1}_{X,z}$

having such elements as roots. Due to the nature of $\bar{\mu}_\chi$, if the binary representation of $\bar{\mu}_\chi(t^i_{X,z})$ has the j-th digit set to 1, then the j-th digit of the binary representation of $\bar{\mu}_\chi(t^{i-1}_{X,z})$ is equal to 1 ($\bar{\mu}_\chi$ is a *SUP* function) since both $t^{i-1}_{X,z} \supset t^i_{X,z}$. Therefore only χ distinct values are allowed for the sequence of $\mu(t^i_{X,z})$, hence there must exist two subtrees $t^i_{X,z}, t^{i-1}_{X,z}$ such that $\mu(t^i_{X,z}) = \mu(t^{i-1}_{X,z})$. We can then ST-REP the subtree $t^{i-1}_{X,z}$ with $t^i_{X,z}$ thus reducing the Γ of this branch, without altering the measure of the hyperpath. We can repeat this operation until the Γ of all the branches of $t_{X,y}$ is at most χ. Therefore $\mathcal{P}_\chi = (\max, \bar{\mu}_\chi)$ is $\chi - 1$-*CYCLE-CONV*.
To prove that $\mathcal{P}_\chi$ is not $\chi - 2$-*CYCLE-CONV*, consider the following hypergraph $\mathcal{H}_\chi = \langle N, H \rangle$. The set N is composed by nodes $\{x, n_1, \ldots, n_\chi, y\}$. There is a hyperarc $\langle x, n_i \rangle$ in H for each $i = 1, \ldots, \chi$; moreover H contains hyperarc $\langle n_1, y \rangle$ and a set of self-loops $l_j = \langle y n_2 \ldots n_j, y \rangle$ (for $j = 2, \ldots, \chi$). An example of $\mathcal{H}_\chi$ for $\chi = 4$ is provided in Figure 9$-a$). We are interested in finding the hyperpath from x to y which maximizes $\bar{\mu}_\chi$. We can observe that such a hyperpath $t^*_{x,y}$ contains in the same branch hyperarc $\langle n_1, y \rangle$ and all the self-loops l_j (see Figure 9$-b$) for $\chi = 4$). Therefore $\Gamma(t^*_{x,y}) = \chi$ and $\bar{\mu}_\chi(t^*_{x,y}) = 2^\chi - 1$. It is easy to check that no ST-REP on $t^*_{x,y}$ can produce a hyperpath with the same measure and a smaller value of Γ. □

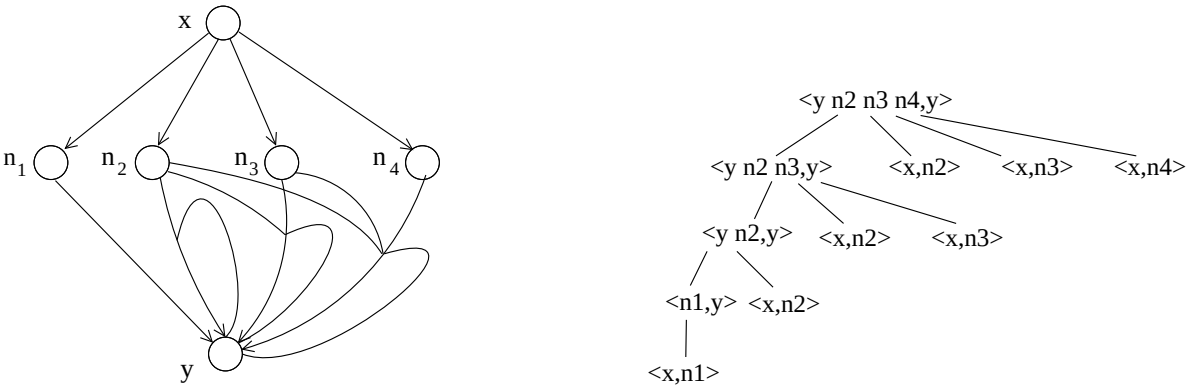

Figure 9: In a) hypergraph $\mathcal{H}_\chi$ for $\chi = 4$; in b) an optimal hyperpath from x to y.

The previous theorem proves that the classes of k-*CYCLE-CONV* maximization problems do not collapse for $k \geq 2$. A similar result can be straightforwardly proved for minimization problems by considering a measure function symmetric to $\bar{\mu}_\chi$.

We conclude this section by representing in Figure 10 the main relationships between the classes of measure function and the properties of the corresponding minimization problems. A similar (but symmetrical) result can be obtained in case of maximization problems.

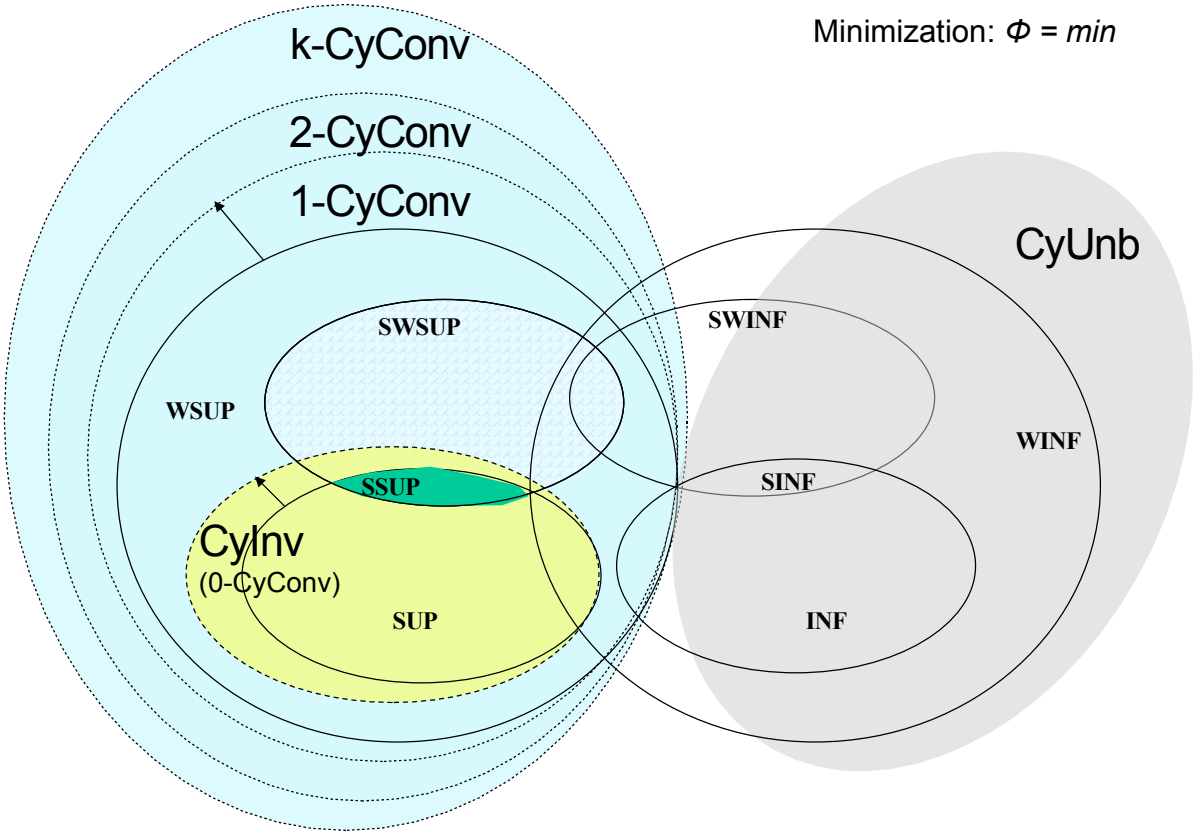

Figure 10: Relationships between the classes of measure function and the properties of the corresponding minimization problems.

Part II
Algorithmic techniques for hyperpaths optimization problems

7 Known techniques

Several authors have proposed algorithms that can be used to find optimal hyperpaths. The first solution was proposed, with a different formalism, by [Knu77]. That paper is aimed to solve the so called *context free grammar problem*. The algorithm presented can be easily adapted to find the optimal hyperpaths from a single *source node* (the *axiom* of the grammar), to all the other nodes in a weighted hypergraph $\mathcal{H} = \langle N, H; w\rangle$. The algorithm is based on Dijkstra's shortest path algorithm, and uses a priority queue. A generic node x is enqueued when the distance of a neighbor from the source has been computed. The priority of x in the queue may decrease if further neighbors provide better connection from the source. When a node is dequeued, its distance from the source is computed. Knuth's algorithm requires $O(|H| \log |N| + |\mathcal{H}|)$ worst case time, that can be reduced to $O(|N| \log |N| + |\mathcal{H}|)$ by using Fibonacci heaps [FT87] for implementing priority queues. We must notice however that there are some differences between the superior functions of Knuth and the value based measures we presented here, which makes them somehow incomparable. First, Knuth's paper deals with generic functions (not necessarily commutative), while the domain of a value-based function μ is more precisely a set of values, and hence the function ψ has to be commutative and associative. Furthermore, in the grammar problem each production may be associated to a different function; on our side, whereas in the definition we provided μ can associate a distinct function to each hyperarc, we will assume in our algorithms that the functions ψ and f are always the same, for all the hyperarcs. On the other hand, our assumptions allow to represent hypergraphs in a compact way by means of FD graphs, presented for example in [AI91], hence obtaining better complexities and, above all, simpler representations and implementations.

The dynamic maintenance of hyperpaths has been studied by several authors [IN89, AIN90, AI91, RR96, AG97]. Some of them considered only the *incremental single source problem*. This problem consists of maintaining the minimal hyperpaths from a single source node to every other node, while performing insertions of hyperarcs (*insert* operations), or weight decreases (*decrease* operations).

In [AFF04], it is proposed a simple extension of known strategies for dealing with the incremental case, and a new technique for tackling the decremental case. The proposed algorithms apply to *SWSUP* functions, provided that the function value associated to a hyperarc is simple enough to be efficiently updated and minimum weight hyperpaths assume integers measures in [1,C]. Their algorithms manage an arbitrary sequence of L hyperarc insertions and weight decrements in overall $O(L \cdot C + C \cdot |\mathcal{H}|)$ worst case time, and maintains minimum weight hyperpaths under a sequence of L hyperarc weight increments and hyperarc deletions, requiring $O(L \cdot C + \max\{|N|, C\} \cdot |\mathcal{H}|)$ worst case time for the whole sequence.

Ramalingam and Reps provide in [RR96] a fully dynamic solution of Knuth's grammar problem. Their solution is more general and applies also to a whole set of hyperarc operations of various kinds, and not only to single operations. The main idea behind the algorithm is to use a global priority queue in which are inserted only the nodes whose distance from the source has to be updated. Moreover, for each node $n \in N$, it is maintained a heap, containing all the hyperarcs leading to n with a "good" measure. If, after a sequence of updates, the measure of the best hyperpath to n changes, then the new correct measure is taken by picking the best hyperarc on the heap of n. The time complexity of this algorithm is given in terms of *output complexity*; the cost of the computation is defined as a function of the following parameters:

- $|\delta|$: the size of the set δ of the affected nodes, i.e. the nodes changing their distance after the

update;

- $||\delta||$, i.e. the number of affected nodes plus the number of hyperarcs having at least one endpoint[1] in δ. In the worst case $||\delta||$ can be $O(|H| + |\delta|)$.
- $||\delta||_B$, i.e. the number of affected nodes plus the sum of their outdegrees:

$$||\delta||_B = |\delta| + \sum_{n \in \delta} |fstar(n)|$$

This value corresponds to the value of $||\delta||$ on the bipartite directed graph $B = (N, P, E)$, where N is in bijection with the set of nodes of the hypergraph, P is in bijection with the productions (the hyperarcs) of the hypergraph and there is a directed edge $\langle n_i, p_j \rangle$ in E if n_i is in the source set of p_j, and a directed edge $\langle p_j, n_i \rangle$ if n_i is the target node of p_j. Note that in the worst case $||\delta||_B$ can be $O(|H| \cdot |\delta|)$.

Namely, the algorithm presented in [RR96] requires $O(||\delta||_B \cdot (\log ||\delta||_B + M))$ worst-case time to recompute the new measures after a sequence of updates. M is an upper bound on the time required to compute the production function associated with any hyperarc. In our discussion we will assume that the recomputations require constant amortized time. In this case, the stated complexity is reduced to $O(||\delta||_B \log ||\delta||_B)$.

Sort by Priority

We will refer to this kind of algorithms as *Sort-by-Priority*, in the sense that they use a priority queue to select the next node to be examined.

In the next section we will point out the issues that may arise in managing hyperarcs insertions (and deletions) with a pure *Sort-by-Priority* approach, in case of *WSUP* and *WINF* function. We will show a case in which a node is enqueued more than once, and the stated complexity is no longer satisfied.

Sort by Structure

A different approach to maintain optimal hyperpaths, which we will refer to as *Sort-by-Structure*, is described in [IN89]. The algorithm to update optimal hyperpaths from X after the insertion of a hyperarc $\langle S, t \rangle$, can be described as follows:

`Insert`$(S, t; w)$

1. Compute the nodes that become reachable from the source X with the insertion of the new hyperarc. Collect those nodes (together with node t) in a simple queue Q;
2. Extract a node z from Q and compute the optimal hyperpath from X to z. Next, scan all the hyperarcs $\langle Z, y \rangle$ whose source set Z contains z, possibly inserting node y in queue Q if a better path from the source to node y has been found;
3. Repeat the previous step until Q becomes empty.

Step 1 consists of computing the set of nodes reachable from X: a dynamic solution of this problem was proposed in [AIN90]. In the case of a weight decrease operation this step is skipped, and queue Q is initialized with node t only. *Sort-by-Structure* can be used with small changes also in case of the "static" problem as an alternative to Knuth's algorithm. Furthermore *Sort-by-Structure* can be easily adapted to handle acyclic hypergraphs in $O(|\mathcal{H}|)$ total time in any incremental sequence of updates. This solution does not make use of priorities, but the nodes whose optimal hyperpath from the source is to be improved are enqueued in a simple queue Q. Again, each node may be enqueued several times. If we consider an incremental sequence of operations, consisting in both insertion of new hyperarcs and weight decreases, for each node enqueued, $O(W)$ time is needed to process the whole sequence of updates, where W denotes the codomain of function μ, i.e., the number of possible values that the measure of a hyperpath may assume. The function $f_{\langle S,t \rangle}(w_{\langle S,t \rangle}, \psi_{\langle S,t \rangle}(\ldots))$ associated to each hyperarc $\langle S, t \rangle$ is (re)computed at most once for each improvement of nodes in the set S. This leads to an overall bound of $O(|\mathcal{H}| \cdot W)$ total time in any incremental sequence of updates, in the hypothesis that the (re)computation of each function associated to hyperarcs requires constant time.

[1] recall that the endpoints of a hyperarc are the nodes of its source set and its target node

In the algorithms we present in the following section, *Sort-by-Structure* and *Sort-by-Priority* are combined; we will refer to our algorithms as *Improved Sort-by-Priority*. As we will see *Improved Sort-by-Priority* algorithms provide the best bound for the general case, since each node is enqueued in the priority queue at most once in each call to the update procedure, also in case of the *WSUP* and *WINF* measure functions.

8 Incremental algorithms for maintaining optimal hyperpaths

In this section we present our algorithms for dynamic incremental maintenance of optimal hyperpaths. The algorithms update the optimal hyperpaths in a hypergraph $\mathcal{H} = \langle N, H \rangle$ from a fixed source set $X \subseteq N$ to each node $y \in N$, in the case of 0-*CYCLE-CONV* and 1-*CYCLE-CONV* optimization problems. Therefore minimization of *SUP* and *WSUP* functions and maximization of *INF* and *WINF* functions are covered. In Section 8.1 we present the algorithm which updates the optimal hyperpaths in the case that the weight of a hyperarc has been improved (i.e., decreased in the case of minimization problems or increased in the case of maximization problems). In Section 8.2 we present the algorithm for maintaining optimal hyperpaths under hyperarc insertions. Furthermore, we show an example where a pure *Sort-by-Priority* approach does not achieve a "good" computational complexity.

8.1 Weight updates

In this section we describe Algorithm `Improve_Weight`, which updates optimal hyperpaths after an improvement on the weight of a hyperarc $h = \langle S, t \rangle \in H$. Depending on the optimization function, this procedure thus manages weight decreases in case of hyperpath minimization, and weight increases in case of hyperpath maximization. The pseudocode of the algorithm is presented in Figure 1.
The entities computed and maintained by the algorithm are, for each node $y \in N$:

- `m[y]` : the measure of the optimal hyperpath from X to y, if any exists (also called the *distance* of y);
- `p[y]` : the last hyperarc (having y as target) traversed in the optimal hyperpath from X to y (the *port* of y); in case of weak measure functions, this value refers to the hyperarc belonging to the relevant subtree of $t_{X,y}$;

Algorithm 1 Algorithm `Improve_Weight`

```
Algorithm Improve_Weight (h = ⟨S, t; w⟩: a weighted hyperarc)
Assign weight w to hyperarc ⟨S, t⟩ ∈ H
if h ∈ H^T then
    Scan_Hyperarc(h)
end if
while PQ ≠ ∅ do
    z ⟵ ExtractMin from PQ
    for all h' ∈ (fstar(z) ∩ H^T) do
        Scan_Hyperarc(h')
    end for
end while
```

The modified hyperarc is passed to Procedure `Scan_Hyperarc`: if the measure computed for it improves the measure of the target node, then the node is enqueued into a priority queue PQ, with a priority equal to its new measure.
In the main cycle of the algorithm, the enqueued node with the best measure is extracted from PQ and

Algorithm 2 Procedure `Scan_Hyperarc`

Procedure Scan_Hyperarc ($h = \langle S, t; w\rangle$: a weighted hyperarc)
$m \longleftarrow$ recompute $\mu(h)$
if $m \prec \mathtt{m}[t]$ **then**
ChangePort (t, h)
EnqueueUpdate (t, m) in PQ
end if

Algorithm 3 Algorithm `Batch_Improve_Weight`

Algorithm Batch_Improve_Weight ($h_1 = \langle S_1, t_1; w_1\rangle,\ h_2 = \langle S_2, t_2; w_2\rangle,\ \ldots$)
for all hyperarc h_i **do**
Assign weight w_i to hyperarc $\langle S_i, t_i\rangle \in H$
if $h_i \in H^T$ **then**
Scan_Hyperarc(h_i)
end if
end for
while $PQ \neq \emptyset$ **do**
$z \longleftarrow$ *ExtractMin* from PQ
for all $h' \in (fstar(z) \cap H^T)$ **do**
Scan_Hyperarc(h')
end for
end while

its forward star is examined. More precisely, we consider only the set H^T of traversable hyperarcs; an hyperarc is traversable if all the nodes of its source set are reachable from X. Each hyperarc in $fstar() \cap H^T$ is passed to procedure `Scan_Hyperarc`.

The pseudocode of Procedure `Scan_Hyperarc` is presented in Figure 2. When invoked on hyperarc h, first the measure $\mu(h)$ is recomputed; then (line 2) it is checked if such measure improves the distance of the target node of h. If the distance is not improved, then the procedure stops here. Otherwise the distance and the port of the target node are updated (line 3) and Procedure *EnqueueUpdate* is invoked. Such procedure enqueues node t in PQ (if not already enqueued) and sets its priority to m.

Now we prove the correctness of Algorithm `Improve_Weight`. We first state some preliminary properties, and introduce the following notation. For each node $y \in N$,

- m_y and m'_y are the actual distances of y respectively before and after the weight update;
- $\mathtt{m}[y]$ is the stored distance of y before the algorithm execution;
- $\mathtt{m}'[y]$ is the computed distance of y after the algorithm execution;
- $\hat{\mathtt{m}}[y]$ is the currently stored distance of node y in a specific instant of the algorithm execution; sometimes we denote with $\hat{\mathtt{m}}'[y]$ a new value of distance of y computed during the execution of the algorithm.

Precondition for the algorithm is that, for each node $y \in N$, $\mathtt{m}[y] = m_y$, i.e. the correct distance of each node before the update is correctly stored. To prove the correctness of the algorithm we have to prove that, for each node $y \in N$, $\mathtt{m}'[y] = m'_y$.
We partition the set of nodes N into three subsets: unreachable nodes (N^U) (remaining unreachable also after the weight improvement), reachable nodes whose distance improves (N^A) after the weight improvement (the affected nodes), and reachable nodes whose distance from X does not change with the update (N^F).

Lemma 8.1 *During the execution of Algorithm* `Improve_Weight`*, for each node* $y \in N$*:*

1. *the computed distance of* y *never gets worse;*
2. *the computed distance of* y *is never better than the actual distance* m'_y *after the update.*

Proof. The first property is a trivial consequence of the precondition for the update (see Procedure `Scan_Hyperarc`, line 2).
For the second property, by hypothesis, before the update all the distances are correctly stored, i.e. $\forall y \in N,\ \mathtt{m}[y] = m_y$. By contradiction, let us consider the first node z that, during the update procedures, receives a value $\hat{\mathtt{m}}[z] \prec m'_z$. The former quantity is computed during the execution of Procedure `Scan_Hyperarc` on the hyperarc $h = \langle S, z\rangle$. Since z is the first node violating the lemma, the stored values are such that $\forall s_i \in S,\ \hat{\mathtt{m}}[s_i] \succeq m'_{s_i}$ (i.e. worse or equal to the actual distance). Nevertheless, by the monotonicity properties of function μ, the computation of $\mu(h)$ cannot return a value which is better than the actual value, i.e.:

$$\hat{\mathtt{m}}[z] = \mu\left(w_h, \hat{\mathtt{m}}[s_1], \hat{\mathtt{m}}[s_2], \ldots\right) \succeq \mu\left(w_h, m'_{s_1}, m'_{s_2}, \ldots\right) \succeq m'_z$$

□

We can now prove the correctness theorem:

Theorem 8.2 *Given a hypergraph* $\mathcal{H} = \langle N, H\rangle$ *and a set of nodes* $X \subseteq N$*, Algorithm* `Improve_Weight` *correctly updates the measure and the port of the optimal hyperpaths from* X *to each node* $y \in N$*, after an improvement on the weight of a hyperarc* $h \in H$*.*

Proof. The proof is trivial for nodes in N^U, since the algorithm does not affect reachability. Also for nodes in N^F, the theorem is a straightforward consequence of Lemma 8.1 since $\mathtt{m}[y] = m'_y$. We can then focus our attention only on the affected nodes (those in N^A). Let us suppose, by contradiction, that there exists a non-empty set of affected nodes $W \subseteq N^A$ for which, at the end of the execution of Algorithm `Improve_Weight`, the computed distance is wrong: $\mathtt{m}'[w] \neq m'_w$. Let us consider the node $z \in W$ having the best actual distance m'_z among all the nodes in W, and let $p_z = \langle S, z\rangle$ be a port of z ($\mu(p_z) = m'_z$). Note that, in the case that two or more nodes in W having the same distance, there surely will be at least one of these with none of the others in the source set of one of its ports[2]. We pick this one.
Now we prove that the correct measure of p_z must have been computed by Procedure `Scan_Hyperarc` and assigned to z. We can split S, i.e. the source set of p_z, in two subsets:

- The set S^R of nodes relevant for the computation of $\mu(p_z)$, i.e. $S^R = \{r \in S \mid m'_r \preceq m'_z\}$;
- The set $S^I = S - S^R$;

Furthermore, let S^A be the set of nodes in S^R whose distance is improved ($S^A = S^R \cap N^A$). The algorithm correctly computes the distance of all the nodes in S^R, since nodes in W either have a worse distance than z or are not in S. Therefore, all the nodes in S^A are enqueued in PQ. Now let us consider the instant when the last node of S^A is dequeued for the last time from PQ: all its forward star, thus p_z is passed to Procedure `Scan_Hyperarc`; and its measure is computed. All the measures of the other nodes in S^A have been fixed, then each node $r \in S^R$ has a computed distance equal to the actual final distance m'_r. Moreover the values of nodes in S^I are irrelevant for the computation of $\mu(p_z)$. Therefore the execution of Procedure `Scan_Hyperarc` on p_z correctly computes the measure m'_z for z and assigns it to z. By Lemma 8.1 the measure of z cannot change anymore, hence $\mathtt{m}'[z] = m'_z$.
□

Now we analyze the complexity of Algorithm `Improve_Weight`, by assuming that the recomputation of $\mu(h)$ requires (possibly amortized) constant time. We first need to state a further property of the algorithm.

[2] this is a consequence of the fact that the relevant part of an optimal hyperpath is acyclic

Lemma 8.3 *Let z be a node extracted from PQ and let $\hat{\mathtt{m}}[z]$ be its priority (its current computed distance). The invocation of Procedure* `Scan_Hyperarc` *on a hyperarc $h \in fstar(z)$ cannot insert or update a node in PQ, with a priority $\mu(h) \prec \hat{\mathtt{m}}[z]$.*

Proof. This is obvious in case of *SUP* (for minimization) or *INF* (for maximization) measure functions, since the value of function $\mu(h)$ cannot be better than any of its arguments (among them $\hat{\mathtt{m}}[z]$). In the case of *WSUP* and *WINF* functions, let us suppose, by contradiction, that the invocation of Procedure `Scan_Hyperarc` on $h \in fstar(z)$ has produced a measure $\mu(h) \prec \hat{\mathtt{m}}[z]$. Let $\hat{\mathtt{m}}[s_1], \hat{\mathtt{m}}[s_2], \ldots$ be the computed distances of nodes in the source set of h, used for the previous computation of $\mu(h)$. Since $\mu(h) \prec \hat{\mathtt{m}}[z]$, z is irrelevant for the computation of $\mu(h)$, and hence there must be at least one node s_r in the source set of h such that:

1. the computed distance of s_r has been improved by the algorithm after the previous computation of $\mu(h)$: $\hat{\mathtt{m}}'[s_r] \prec \hat{\mathtt{m}}[s_r]$;
2. the measure $\hat{\mathtt{m}}'[s_r]$ is relevant for the computation of $\mu(h)$, i.e. $\hat{\mathtt{m}}'[s_r] \preceq \mu(h)$;
3. the value of $\mu(h)$ has not been recomputed using, for node s_r, the new value $\hat{\mathtt{m}}'[s_r]$;

Condition 1 implies that node s_r must have been enqueued (or updated) in PQ with priority $\hat{\mathtt{m}}'[s_r]$, since the weight update and the enqueueing are executed in the same block of code. Since priority of s_r can not decrease and $\hat{\mathtt{m}}'[s_r] \preceq \mu(h) \prec \hat{\mathtt{m}}[z]$, then s_r must have been already dequeued before z. But since $h \in fstar(s_r)$, it must have been passed to Procedure `Scan_Hyperarc`, and $\mu(h)$ must have been recomputed using $\hat{\mathtt{m}}'[s_r]$ for node s_r, thus contradicting condition 3. □

Corollary 8.4 *Nodes in PQ are dequeued with monotonically not improving measures.*

Proof. This is a straightforward consequence of Lemma 8.3. □

Now we can state the complexity of the algorithm:

Theorem 8.5 *The complexity of Algorithm* `Improve_Weight` *is $O(||\delta||_B + |\delta| \log |\delta|)$.*

Proof. It is easy to check in the code that only the $|\delta|$ nodes that improve their measure are enqueued in PQ. Moreover, Corollary 8.4 implies that each affected node is enqueued in PQ exactly once. Then exactly $|\delta|$ *insert* (and $|\delta|$ *delete min*) operations are performed on PQ.
For each node t dequeued from PQ, Procedure `Scan_Hyperarc` is invoked at most $|fstar(t)|$ times; thus it is invoked at most $||\delta||_B$ total times. Since exactly $|\delta|$ times it performs an *insert* on `PQ`, at most $||\delta||_B - |\delta|$ times it performs a *decrease key* on `PQ`. Moreover no *increase key* are performed in the queue. We can have an efficient implementation of the priority queue by using Fibonacci heaps [ST83] (or Relaxed heaps [DGST88]); in this case the amortized complexity (or worst case complexity for Relaxed heaps) is $O(1)$ for *decrease key* operations and $O(\log n)$ for *insert* and *delete min* operations. The complexity of Algorithm `Improve_Weight` is thus $O(||\delta||_B + |\delta| \log |\delta|)$. □

In Figure 3 Algorithm `Batch_Improve_Weight` is presented, which updates optimal hyperpaths when a set of hyperarcs improves its weight. Algorithm `Batch_Improve_Weight` has complexity $O(||\delta||_B + |\delta| \log |\delta|)$ as well. In this case δ is the union of the set of nodes changing their distance after each hyperarc update.

Remark: in the case of minimization problems where function ψ is min (e.g., gap) and in case of maximization problems where function ψ is max (e.g., threshold), the complexity of the algorithm can be further improved to $O(||\delta|| + |\delta| \log |\delta|)$. The monotonic ordering in which we extract the nodes from the queue, in fact, allows us to invoke Procedure `Scan_Hyperarc` at most once for each hyperarc, i.e. when the best node of its source set is extracted from the queue. Therefore, in this case the factor $||\delta||_B$ is reduced to $||\delta||$.

8.2 Hyperarc insertions

The insertion of a hyperarc can provide in both cases (minimization and maximization) an improvement on the measures of hyperpaths. Nevertheless, a hyperarc insertion can modify (expand) the reachability of the hypergraph, since nodes previously unreachable may become reachable from the source X. Therefore, we can split the set of affected nodes N^A, into two subset: the set N^{AR} of nodes previously unreachable that become reachable, and the set N^{AI} of nodes already reachable which simply improve their distance. In this case, N^U represents the set of unreachable nodes that remain unreachable after the hyperarc insertion.

In order to efficiently deal with hyperarc insertions, the technique used in Algorithm `Improve_Weight` must be cautiously adapted. We must notice, in fact, that if we use the same *Sort-by-Priority* approach, then Lemma 8.3, does not hold anymore in case of minimization of *WSUP* functions or maximization of *WINF* functions, and the complexity results obtained for Algorithm `Improve_Weight` cannot be achieved.

The problem arises from that fact that, while in the case of weight improvement no new hyperarcs are discovered and the "optimality" is propagated (monotonically not improving) from the updated hyperarc, in the case of hyperarc insertions new traversable hyperarcs may be discovered and, for weak functions, "optimality" can be fetched from other sources. This can be effectively pointed out by the example represented in Figure 11. In this case we want to maintain hyperpaths from node s which are minimal with respect to the *gap* measure. If we use a pure *Sort-by-Priority* approach to manage the insertion of hyperarc $\langle a, b\rangle$ of weight $2i$ (with i large enough), then nodes b and c would be enqueued in PQ exactly $i+1$ times. This happens because the correct port of b, $\langle a\, x_i, b\rangle$, by which $m_b = i+1$, is discovered only when node x_i is dequeued from PQ, i.e. after all the nodes $x_1, x_2, \ldots, x_{i-1}$ have been dequeued and their outgoing hyperarcs have been assigned as tentative ports of b.

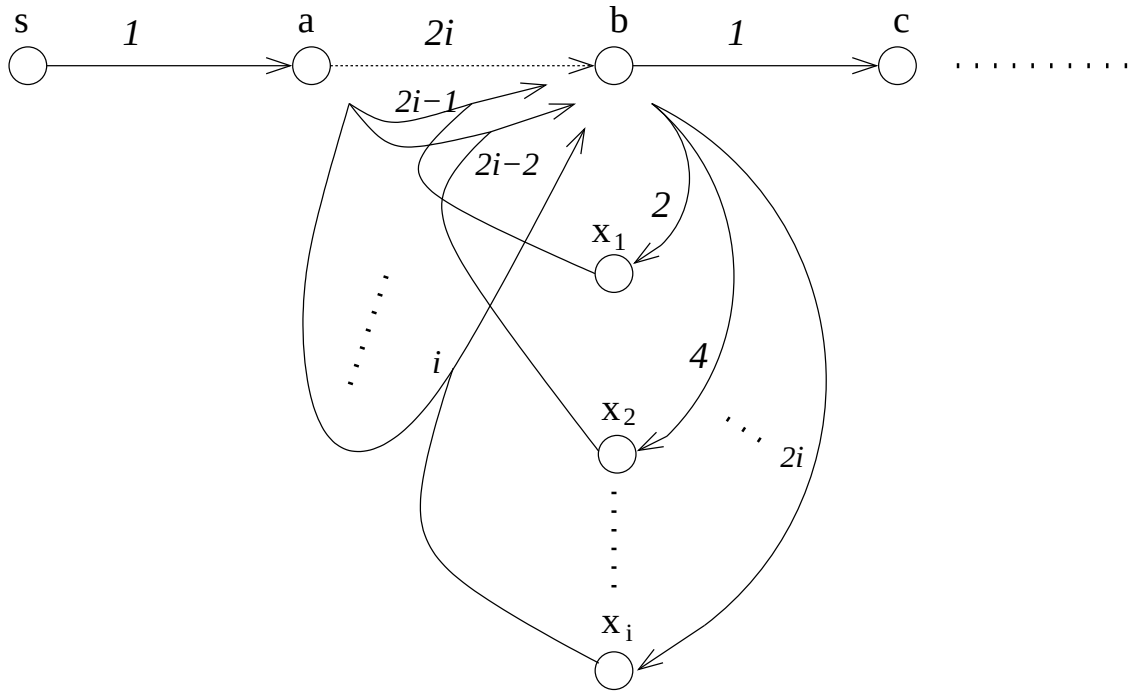

Figure 11: The insertion of hyperarc $\langle a, b\rangle$ in the hypergraph.

Furthermore, in these cases, the measure μ_∞ is not sufficient to propagate unreachability, since, for example, $\min(k, +\infty, \ldots, +\infty) = k$ ant not $+\infty$ as we expected if we give to $+\infty$ the meaning of "unreachable".

For these reasons, propagating at the same time the reachability and the measure of the function

we are interested in, is not a good choice (at least in case of weak functions). We have to decouple these two aspects. First, we propagate reachability by maintaining, for each node t, a `reach[t]` value which indicates the hyperarc that brings reachability to t (or **nil** if the node is unreachable). This is a key-aspect of this algorithm since, for 1-*CYCLE-CONV* functions, the value of `reach[t]` allows us to reconstruct the irrelevant acyclic hyperpath subtree from the source X to t. Then, we can update the values for measure μ by applying a *Sort-by-Priority* approach. This is achieved by invoking Algorithm `Batch_Improve_Weight` with a set L of "improving" hyperarcs. Such set is composed by the hyperarc inserted (if it is traversable) and by all the hyperarcs that become traversable after the update.
Algorithm `Insert_Hyperarc`, which creates the set L of the new traversable hyperarcs and invokes Algorithm `Batch_Improve_Weight` on it, is presented in Figure 4; Procedure `Scan_Reachable`, which scans the hyperarcs in order to propagate reachability, is presented in Figure 5.

Algorithm 4 Algorithm `Insert_Hyperarc`

Algorithm Insert_Hyperarc ($h = \langle S, t; w\rangle$: a weighted hyperarc)
/* L will be the set of new traversable hyperarcs */
$L \longleftarrow \emptyset$
$H \longleftarrow H \cup h$
Scan_Reachable(h)
while $Q \neq \emptyset$ **do**
  $z \longleftarrow$ *Dequeue* from Q
  for all $h' \in fstar(z)$ **do**
    Scan_Reachable(h')
  end for
end while
Batch_Improve_Weight(L)

Algorithm 5 Procedure `Scan_Reachable`

Procedure Scan_Reachable ($h = \langle S, t; w\rangle$: a weighted hyperarc)
if each node in S is reachable **then**
  $H^T \longleftarrow H^T \cup h$
  if (`reach[t]` = **nil**) **then**
    `reach[t]` $\longleftarrow h$
    Enqueue (t) in Q
  end if
  $L \longleftarrow L \cup h$
end if

We separately prove the two aspects related to the correctness of the algorithm: first we prove that reachability of all the nodes in N^{AR} is correctly updated, then we prove that the correct distance is computed for each node in N^A.
We say that the reachability is correctly computed for a node z if:

- `reach[z]` = **nil**, if z is not reachable from X, i.e., there exists no hyperpath from X to z;
- `reach[z]` $= \langle S, t\rangle$, if z and each node in S are reachable from X (and $\langle S, t\rangle \in H$).

Lemma 8.6 *At the end of the execution of Algorithm* `Insert_Hyperarc`*, the reachability is correctly computed for each node* $y \in N$.

Proof. First, observe that Procedure `Scan_Reachable` does not alter the value of `reach[]` for nodes already reachable (i.e. `reach[]` $\neq$ **nil**). Since such nodes remain reachable also after the update, their reachability is correctly computed. Possible errors might therefore occur only when nodes that become reachable (i.e. nodes in N^{AR}) are not updated by the algorithm, or when the algorithm marks as reachable nodes that are actually not reachable. However, we can observe that Procedure `Scan_Reachable` has the same structure as Procedure `Scan_Hyperarc`; while in the latter case we are propagating a measure function, in the former we are propagating the reachability. We can indeed prove by contradiction that these errors cannot happen, by using arguments similar to the ones we used to prove Lemma 8.1 (i.e. considering the first node that is erroneously marked as reachable, or the first node that is not correctly updated). □

Corollary 8.7 *For each invocation of Algorithm* `Insert_Hyperarc` *the following properties are satisfied:*

($\mathcal{P}_1$) *node t is enqueued in Q if and only if $t \in N^{AR}$*

($\mathcal{P}_2$) *each node $t \in N^{AR}$ is target of at least one hyperarc belonging to L, that is, $bstar(t) \cap L \neq \emptyset$.*

Proof. Observe that a node t can be enqueued in Q only in line 5 of Procedure `Scan_Reachable`. Such code is executed, on a hyperarc $h \in bstar(t)$, only if the value of `reach[t]` changes from **nil** to h. By Lemma 8.6 this happens if and only if $t \in N^{AR}$ ($\mathcal{P}_1$). Moreover, at the end of the procedure (line 7) hyperarc h is inserted in L ($\mathcal{P}_2$). □

We can now state the correctness of Algorithm `Insert_Hyperarc`.

Theorem 8.8 *Given a hypergraph $\mathcal{H} = \langle N, H \rangle$ and a set of nodes $X \subseteq N$, Algorithm* `Insert_Hyperarc` *correctly updates the optimal hyperpaths from X to each node $y \in N$, after the insertion of a hyperarc h in H.*

Proof. As in Theorem 8.2, we separately analyze nodes in N^U, N^F and N^A. By Lemma 8.6, Procedure `Insert_Hyperarc` works correctly for nodes in N^U (recall that reachability is not altered by Algorithm `Batch_Improve_Weight`). For nodes in N^F we can observe that neither Algorithm `Insert_Hyperarc` nor Procedure `Scan_Reachable` alter the measure of any node in N, and all the measure updates are performed by the invocation of Algorithm `Batch_Improve_Weight`. By Theorem 8.2, Algorithm `Batch_Improve_Weight` does not alter the measure of nodes in N^F, hence Algorithm `Insert_Hyperarc` works correctly for nodes in N^F as well. To prove correctness of the algorithm we have to prove that the distance of each node in N^A is computed by Algorithm `Batch_Improve_Weight`, that is, that each node in N^A is correctly enqueued in PQ. By Property ($\mathcal{P}_2$) of Corollary 8.7 this is straightforwardly true for nodes in N^{AR}, since Procedure `Scan_Hyperarc` is invoked for each hyperarc in L. Moreover, observe that the new port of any node in N^{AI} has at least a node in N^A in its source set. We can prove that that also all the nodes in N^{AI} are enqueued in PQ, by using a similar technique as the one used in Theorem 8.2, i.e., picking, by contradiction, the first node in N^{AI} that is erroneously not enqueued in PQ and showing that this leads to a contradiction. □

Let us now analyze the time complexity of Algorithm `Insert_Hyperarc`. By Lemma 8.7, in Q are inserted all and only the nodes in N^{AR}. Moreover, the check on `reach[t]` (line 3) guarantees that the insertion is performed only once for each node. Procedure `Scan_Reachable` is invoked on all the forward stars of the nodes in N^{AR}, and it has cost $O(1)$. Therefore the complexity of the first part of `Insert_Hyperarc` algorithm is $O(||N^{AR}||_B)$, or $O(||\delta||_B)$, since $N^{AR} \subseteq N^A \equiv \delta$.
The invocation of Algorithm `Batch_Improve_Weight` has as input the set

$$L = \left[\bigcup_{n \in N^{AR}} fstar(n) \cup h \right] \cap H^T$$

Even in this case, the procedure can perform at most $|\delta|$ *insert* (and *delete min*) operations and $O(||\delta||_B)$ *decrease key* operations; the execution of Algorithm `Batch_Improve_Weight` has then cost $O(||\delta||_B + |\delta|log|\delta|)$. Therefore, we can state the following theorem.

Theorem 8.9 *The complexity of Algorithm* `Insert_Hyperarc` *is* $O(||\delta||_B + |\delta|log|\delta|)$.

A relevant implementation detail to point out is that the reachability information about the source nodes of hyperarcs can be checked in constant time (line 1 of Procedure `Scan_Reachable`), and maintained in $O(|H|)$ space, by associating to each hyperarc a counter representing the number of unreachable nodes in its source set. This value must be properly set during the execution of Algorithm `Insert_Hyperarc` and decreased (by 1) for each invocation of Procedure `Scan_Reachable`.

9 Conclusions

In this paper we have considered the problem of dynamically maintaining optimal hyperpaths in a directed hypergraph with respect to a given measure. We classified measures, thus optimization problems, according to the structural characteristics of optimal hyperpaths, and showed how some of them cannot be efficiently solved with a pure Dijkstra-like approach. We then proposed two algorithms for maintaining optimal hyperpath in case of weight improvement and hyperarc insertion which are proved to work well also in such cases. Their worst case complexity, $O(||\delta||_B + |\delta|log|\delta|)$, improves the previously better known algorithm presented in [RR96], which has a claimed worst case complexity of $O(||\delta||_B log||\delta||_B)$.

Decremental algorithms for maintaining optimal hyperpaths after hyperarc deletions or weight changes are under analysis, but it seems that also in these cases the *Improved Sort-by-Priority* approach provides better results.

A possible future research in this direction could merge these algorithms into an algorithm able to solve the fully dynamic problem. Although it is not difficult to produce such algorithm, the complexity analysis provided seems to be no longer valid.

In Section 6 we provided an example of k-*CYCLE-CONV* optimization problems with $k \geq 2$. These example result however quite artificial, and it still remains an open question whether there exist more natural problems belonging to those classes.

Frigioni et al. [FMN03] proposed a novel approach to reduce to $O(|\delta|log|\delta|)$ the problem of fully dynamic maintenance of shortest paths for some types of directed graphs. It would be interesting to investigate how and in what cases such technique could be extended to directed hypergraphs.

Other interesting aspects that can be further investigated concerns the possibility to develop approximation algorithms to deal with the presented NP-hard optimization problems.

For what concerns hypergraphs applications, we are still investigating their possible use for fast mining of association rules; moreover it seems that some connectivity and communication problems arising from ad-hoc wireless networks can be effectively modeled by hypergraphs.

References

[AFF04] G. Ausiello, P. G. Franciosa, and D. Frigioni. Partially dynamic maintenance of minimum weight hyperpaths. *Journal of Discrete Algorithms*, Article in press, 2004.

[AFN92] P. Alimonti, E. Feuerstein, and U. Nanni. Linear time algorithms for liveness and boundedness in conflict-free petri nets. In *1st Latin American Theoretical Informatics*, volume 583, pages 1–14. Lecture Notes in Computer Science, Springer-Verlag, 1992.

[AG97] G. Ausiello and R. Giaccio. On-line algorithms for satisfiability formulae with uncertainty. *Theoretical Computer Science*, 171:3–24, 1997.

[AI91] G. Ausiello and G. F. Italiano. Online algorithms for polynomially solvable satisfiability problems. *Journal of Logic Programming*, 10:69–90, 1991.

[AIN90] G. Ausiello, G. F. Italiano, and U. Nanni. Dynamic maintenance of directed hypergraphs. *Theoretical Computer Science*, 72(2-3):97–117, 1990.

[CDP04] Sanjay Chawla, Joseph Davis, and Gaurav Pandey. On local pruning of association rules using directed hypergraphs. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 832. IEEE Computer Society, 2004.

[DGST88] James R. Driscoll, Harold N. Gabow, Ruth Shrairman, and Robert E. Tarjan. Relaxed heaps: an alternative to fibonacci heaps with applications to parallel computation. *Commun. ACM*, 31(11):1343–1354, 1988.

[FMN03] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic shortest paths in digraphs with arbitrary arc weights. *Journal of Algorithms (Academic Press)*, 49(1):86–113, 2003.

[FT87] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.

[GLNP93] G. Gallo, G. Longo, S. Nguyen, and S. Pallottino. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42:177–201, 1993.

[GR90] G. Gallo and G. Rago. A hypergraph approach to logical inference for datalog formulae. Technical Report 28/90, Dip. di Informatica, Univ. of Pisa, Italy, 1990.

[IN89] G. F. Italiano and U. Nanni. On line maintenance of minimal directed hypergraphs. In *3rd Italian Conf. on Theoretical Computer Science*, pages 335–349. World Scientific Co., 1989.

[Knu77] D. E. Knuth. A generalization of dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5, 1977.

[MMZ05] F. Massacci, J. Mylopoulos, and N. Zannone. Minimal disclosure in hierarchical hippocratic databases with delegation. In *10th European Symposium on Research in Computer Security (ESORICS 2005)*, pages 438–454. LNCS 3679, Springer-Verlag, 2005.

[MSS04] R. H. Möhring, M. Skutella, and F. Stork. Scheduling with and-or precedence constraints. *SIAM Journal of Computing*, 33(2):393–415, 2004.

[Nil82] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer Verlag, Berlin, 1982.

[RR96] G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, 21(2):267–305, 1996.

[ST83] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Science*, 24:362–381, 1983.

[Ull82] J. D. Ullman. *Principles of Database Systems*. Computer Science Press, Rockville, MD, 1982.

Algorithm 6 Static_Sort_By_Structure

```
1: /* Transitive closure from source set X */
2: for all nodes n ∈ N do
3:    back[n] ⟵ 0
4:    reach[n] ⟵ null
5:    m[n] ⟵ μ_∞
6: end for
7: for all nodes x ∈ X do
8:    Enqueue x in Q
9:    reach[x] ⟵ ∅
10:   m[n] ⟵ μ_0
11: end for
12: for all hyperarcs h = ⟨S, t; w⟩ ∈ H do
13:   unreachable[h] ⟵ |S|
14: end for
15: while Q ≠ ∅ do
16:   z ⟵ Dequeue from Q
17:   for all hyperarcs h = ⟨S, t; w⟩ ∈ fstar(z) do
18:      unreachable[h] ⟵ unreachable[h] -1
19:      if unreachable[h] = 0 then
20:         back[t] ⟵ back[t]+1
21:         if reach[t]=null then
22:            reach[t] ⟵ h
23:            Enqueue t in Q
24:         end if
25:      end if
26:   end for
27: end while
28: /* Measures computation */
29: for all nodes x ∈ X do
30:   Enqueue x in Q
31: end for
32: for all hyperarcs h = ⟨S, t; w⟩ ∈ H do
33:   unreachable[h] ⟵ |S|
34: end for
35: while Q ≠ ∅ do
36:   z ⟵ Dequeue from Q
37:   for all hyperarcs h = ⟨S, t; w⟩ ∈ fstar(z) do
38:      unreachable[h] ⟵ unreachable[h] -1
39:      if unreachable[h] = 0 then
40:         m ⟵ compute μ(h)
41:         if m ≺ m[t] then
42:            ChangePort (t, h)
43:         end if
44:         back[t] ⟵ back[t]-1
45:         if back[t]=0 then
46:            Enqueue t in Q
47:         end if
48:      end if
49:   end for
50: end while
```