Alcom-FT Technical Report Series
ALCOMFT-TR-02-11

Partially Dynamic Maintenance of Minimum Weight Hyperpaths*

Giorgio Ausiello† Paolo G. Franciosa‡ Daniele Frigioni§†

March 19, 2002

Abstract

We address the problem of dynamically maintaining minimum weight hyperpaths in a directed hypergraph in a decremental setting. For such problem, we provide a new efficient algorithm that works for a wide class of hyperpath weight measures. This algorithm explicitly updates minimum weight hyperpaths in $O(\max(n, C) \cdot \text{size}(\mathcal{H}))$ worst case time under a sequence of hyperarc weight increments and hyperarc deletions, where C is the maximum weight of minimum hyperpaths in $\mathcal{H}$ and $\text{size}(\mathcal{H})$ is the size of the representation of the hypergraph. Hyperpath weight measures are only required to belong to the class of *strict weakly superior functions*.

Keywords: Directed hypergraph, minimum weight hyperpath, dynamic algorithm.

1 Introduction

Hypergraph is a common name for various combinatorial structures that generalize graphs. Beside the most known undirected hypergraphs, or simply *hypergraphs* (see [13]), a relevant role is played by *directed hypergraphs* (see [18]), a generalization of directed graphs, which find applications in several areas of computer science and mathematics for representing implicative structures. In a directed hypergraph we are given a set N of n nodes and a set A of m *hyperarcs*. Each hyperarc is a pair $\langle T, h\rangle$, where T is a subset of N, called the *tail* of the hyperarc, and h is a single node in N, called the *head* of the hyperarc. The most obvious interpretation of a hyperarc $\langle T, h\rangle$ is that the information associated to h functionally depends on the information associated to nodes in T.

Directed hypergraphs and other strictly related combinatorial structures are widely used in computer science. Notably, they are used in artificial intelligence for representing problem solving relationships (And-Or graphs [28] and recursive label-node hypergraphs [14]), in database theory for representing functional dependencies among attributes (FD-graphs [5] and connections in acyclic

*Partially supported by the Future and Emerging Technologies programme of the European Union under contract number IST-1999-14186 (ALCOM-FT).

†Dipartimento di Informatica e Sistemistica, Università di Roma “La Sapienza”, Via Salaria 113, I-00198 Roma, Italy. E-mail: ausiello@dis.uniroma1.it

‡Dipartimento di Statistica, Probabilità e Statistiche Applicate, Università di Roma “La Sapienza”, Piazzale Aldo Moro 5, I-00185 Roma, Italy. E-mail: paolo.franciosa@uniroma1.it

§Dipartimento di Ingegneria Elettrica, Università dell’Aquila, Monteluco di Roio I-67040 L’Aquila, Italy. E-mail: frigioni@ing.univaq.it

hypergraphs [25, 33]), in deductive databases [19], in fuzzy logic, for determining the reliability of facts [8], in propositional logic, for satisfiability check (namely in the case of Horn formulæ [9, 17, 29]), in formal languages (weighted context free grammars [22]), in the theory of concurrency (Petri net paths [34]), in model checking (dependency graphs [24] and boolean graphs [3]), in diagnostics [2]. More applications can be found in [1, 18, 30].

Any time a structure is represented by means of a hypergraph, it may be relevant to find hyperpaths that connect nodes or sets of nodes, or "minimum" hyperpaths, where the minimality is defined on the basis of a weight function that assigns weights to hyperpaths. For example, if the hypergraph is used to represent a Horn formula, hyperpaths correspond to proofs and the weight of a hyperpath can be related to different measures of the complexity of a proof (see [4, 9] for more details). Further applications of hyperpaths and minimum hyperpaths to different contexts can be found in [1, 8, 15, 20, 23, 26].

There are various ways in which a weight can be attached to a hyperpath. Depending on the measure used to assign a weight to a hyperpath the problem of finding minimum weight hyperpaths can be polynomially solvable or NP-hard. For example, in [11] it has been shown that the problem of finding minimum *cost* hyperpaths in a directed hypergraph is NP-hard, where the *cost* of a hyperpath is the sum of the weights of its hyperarcs. Conversely, it is possible to characterize a number of cases in which the problem can be solved in polynomial time, as in the case of the *rank* and the *gap* [21]. Observe that cost, rank and gap are all generalizations of the standard notion of distance in graphs, which can be computed in polynomial time.

Several attempts have been made in the literature to extend Dijkstra's algorithm to compute minimum weight hyperpaths. For example, in [22] Knuth defines the *grammar problem* as a generalization of the single source shortest path problem, and shows that Dijkstra's algorithm can be adapted to solve the grammar problem when the costs of derivations are expressed by *superior* functions. A function $g(x_1, x_2, \ldots, x_k)$ is superior if it is monotone non decreasing in each variable and $g(x_1, x_2, \ldots, x_k) \geq \max(x_1, x_2, \ldots, x_k)$; if the inequality is strict, then g is *strict superior*. Ramalingam and Reps in [31] identify a larger class of functions that are suitable for a Dijkstra-like visit strategy, that is the class of *weakly superior* functions. A function $g(x_1, x_2, \ldots, x_k)$ is weakly superior if it is monotone non decreasing in each variable and for each $i \in [1, k]$, if $g(x_1, x_2, \ldots, x_k) < x_i$, then $g(x_1, \ldots, x_i, \ldots, x_k) = g(x_1, \ldots, \infty, \ldots, x_k)$. If the inequality is not strict, then g is *strict weakly superior*.

In this paper, we consider the problem of dynamically maintaining minimum weight hyperpaths in a directed hypergraph subject to modifications as hyperarc weight changes and hyperarc insertions or deletions. In this setting we do not want to recompute hyperpaths from scratch after each change, but we want to take advantage of the part of the previous solution that is still valid. A common classification of dynamic problems in hypergraphs is among fully-dynamic ones, where weight changes, insertions and deletions of hyperarcs can be intermixed, and partially-dynamic ones (incremental or decremental), where only weight decrements and insertions or only weight increments and deletions are allowed, respectively. The problem of maintaining minimum weight hyperpaths arises in several application domains as, for example, in the minimum model maintenance in Horn formulæ [7, 8, 9, 31]. In particular, the decremental problem finds applications in the context of assumption-based truth maintenance systems [15, 23], where maintaining minimum hyperpaths under deletions of hyperarcs corresponds to maintaining a set of "small" explanations of all observations under the elimination of either a hypothesis or a clause in the background theory.

The first hypergraph problem stated in a dynamic framework has been the maintenance of the transitive closure of a directed hypergraph under hyperarc insertions [10]. Minimum weight traversal problems have been studied by various authors. For the incremental case, an algorithm is presented in [9], that maintains satisfiability and the minimum model of a Horn formula $\mathcal{F}$ in $O(n \cdot \text{Length}(\mathcal{F}))$ total time over a sequence of clause insertions, where $\text{Length}(\mathcal{F})$ is the sum of the number of literals in the clauses of $\mathcal{F}$. This algorithm is extended in [8] to the case of Horn formulæ with uncertainty. Algorithms are given in [21] for the incremental maintenance of minimum rank and minimum gap hyperpaths under a sequence of hyperarc insertions in overall $O(n \cdot \text{size}(\mathcal{H}))$ time, under the assumption of unit hyperarc weights. In [7] a decremental algorithm is proposed to maintain minimum rank hyperpaths under a sequence of hyperarc deletions and weight increments in overall $O(W \cdot n \cdot \text{size}(\mathcal{H}))$ worst case time in the case of integer hyperarc weights in $[1, W]$.

The fully dynamic version of the problem has been considered in [31]. The authors propose a Dijkstra-like procedure, applicable to strict weakly superior weight functions. The proposed algorithm takes $O(\|\delta\| \cdot (\log \|\delta\| + M))$ worst case time per update operation, where each operation may consist of several insertions/deletions of hyperarcs and hyperarc weight changes. Parameter $\|\delta\|$ represents the number of nodes interested by the operation plus the total number of hyperarcs incident to these nodes, while M is the time needed to recompute the value of a weight function. Note that in the worst case $\|\delta\| = m$. In the same paper, Ramalingam and Reps also provide an algorithm for the fully dynamic problem on strict superior weight functions, that requires $O(\|\delta\| \cdot \log \|\delta\|)$ worst case time per update operation.

In this paper, we first show a simple extension of known strategies for dealing with the incremental case, and then propose and analyze in detail a new technique for tackling the decremental case. In both cases, the proposed algorithms apply to all strict weakly superior weight functions, as for example the rank and the gap.

For the incremental case, we describe an extension of Dijkstra's algorithm for managing an arbitrary sequence of hyperarc insertions and weight decrements in overall $O(C \cdot \text{size}(\mathcal{H}))$ worst case time, if minimum weight hyperpaths assume weights in an interval of integers $[1, C]$. When applied to rank or gap, we obtain the same time bounds of the *sort-by-structure* strategy given in [12].

Concerning the decremental problem, we propose a new algorithm for maintaining minimum weight hyperpaths under a sequence of hyperarc weight increments and hyperarc deletions, requiring $O(\max(n, C) \cdot \text{size}(\mathcal{H}))$ worst case time for the whole sequence. The algorithm assumes that each hyperarc is associated a strict weakly superior weight function, and minimum weight hyperpaths assume integer weight in $[1, C]$.

Our decremental solution is based on a novel technique that, after the update of hyperarc $\langle X, y\rangle$, explores the nodes whose weight and/or port changes (here, the port of a node is the hyperarc incoming that node in the minimum weight hyperpath from the source), starting from node y and proceeding one level at a time. Here, we say that a node is at level i if its weight is equal to i. For each i, the algorithm builds the *new* level i by selecting the nodes belonging to the *old* level i whose port changes, and nodes coming from *old* levels $j < i$ whose weight increases to i. Conversely, classical approaches for solving shortest path problems usually work in two phases: first the whole set of nodes affected by the update is selected, and then some Dijkstra-like algorithm is applied to the subgraph induced by the selected nodes.

The performance of our algorithm must be compared with the performance of the algorithms proposed by Ramalingam and Reps [31]. Adapting their algorithm for strict weakly superior weight functions to deal with integer weight values it could be possible to cut out the $\log \|\delta\|$ factor, but the time required to handle a single update can be $\Omega(n \cdot \text{size}(\mathcal{H}))$. In fact, term M depends on the size of hyperarc tails, and can be $\Omega(n)$, while $\|\delta\|$ can be $\text{size}(\mathcal{H})$. On the other hand, the algorithm given in [31] for strict superior weight functions could require $\Omega(\text{size}(\mathcal{H}))$ worst case time per update, while our algorithm is able to maintain minimum weight hyperpaths under a sequence of hyperarc weight increments and hyperarc deletions, in the more general case of strict weakly superior functions, within the same overall time bound.

The paper is organized as follows. In section 2 the basic definitions concerning hypergraphs and hyperpaths are given and various hyperpath weight measures are presented. Section 3 is devoted to a discussion of a variety of hyperpath weight measures. In Section 4 we tackle the dynamic problem. Here, we first propose an incremental solution for the maintenance of minimum weight hyperpaths by discussing the differences with respect to the previous solutions known in the literature (Subsection 4.1), and then we present a new efficient algorithm for maintaining minimum weight hyperpaths under hyperarc weight increments (Subsection 4.2) and hyperarc deletions (Subsection 4.3). In the last section we draw some conclusions and suggest future research directions.

2 Hypergraphs and hyperpaths

The following definitions concerning directed hypergraphs are from [6] and [12], and are consistent with the more general definitions given in [18].

A *directed hypergraph* $\mathcal{H}$ (see Fig. 1 for an example) is a pair $\langle N, A\rangle$, where N is a non empty set of n *nodes* and A is a set of m hyperarcs; a *hyperarc* e is an ordered pair $\langle T, h\rangle$, with $T \subseteq N$, $T \neq \emptyset$, and $h \in N \setminus T$; T and h are called the *tail* and the *head* of e, and are denoted by $\mathrm{tail}(e)$ and $\mathrm{head}(e)$, respectively. The *forward star* of $v \in N$ is the set $\mathrm{fstar}(v) = \{e \in A : v \in \mathrm{tail}(e)\}$, while the *backward star* of v is the set $\mathrm{bstar}(v) = \{e \in A : v = \mathrm{head}(e)\}$. The *size* of a hypergraph $\mathcal{H}$ is defined as $\mathrm{size}(\mathcal{H}) = \sum_{e\in A} |\mathrm{tail}(e)|$.

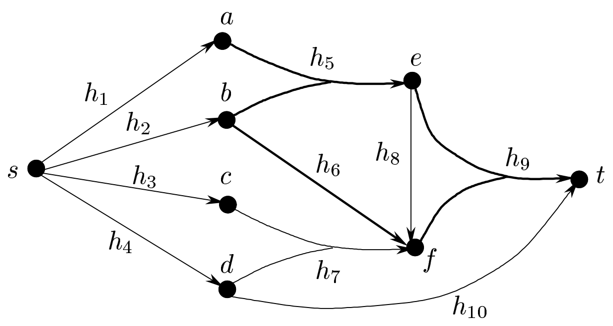

Figure 1: A directed hypergraph

Given a hypergraph $\mathcal{H}$, a *subhypergraph* of $\mathcal{H}$ is a hypergraph $\mathcal{H}' = \langle N', A'\rangle$ with $N' \subseteq N$ and $A' \subseteq A$. A subhypergraph is *proper* if at least one of the inclusions is strict. A *hyperpath* in $\mathcal{H}$ from a set of nodes $S \subset N$, with $S \neq \emptyset$, called *source*, to a target node $t \in N$ is a subhypergraph $\Pi_{S,t} = \langle N_{\Pi_{S,t}}, A_{\Pi_{S,t}}\rangle$ of $\mathcal{H}$ having the following property: if $t \in S$, then $A_{\Pi_{S,t}} = \emptyset$, otherwise its $k \geq 1$ hyperarcs can be ordered in a sequence $\langle e_1, \ldots, e_k\rangle$ such that:

1. $\forall e_i \in A_{\Pi_{S,t}}, \mathrm{tail}(e_i) \subseteq S \cup \{\mathrm{head}(e_1), \ldots, \mathrm{head}(e_{i-1})\}$
2. $t = \mathrm{head}(e_k)$
3. No proper subhypergraph of $\Pi_{S,t}$ is a hyperpath from S to t in $\mathcal{H}$.

The above definition of hyperpath is consistent with the notion of *folded hyperpath* given in [12], and generalizes the notion of simple path in a directed graph. A node t is said to be *reachable* in

$\mathcal{H}$ from source S if there exists a hyperpath $\Pi_{S,t}$ in $\mathcal{H}$. The *port* of a node v on $\Pi_{S,t}$, denoted by port(v), is the hyperarc in $A_{\Pi_{S,t}}$ having v as head; it is unique by condition 3 above. In Fig. 1, thick hyperarcs represent the hyperpath from source $S = \{a, b\}$ to node t consisting of hyperarcs $\langle h_5, h_6, h_9 \rangle$.

The most intuitive and classical measure of the weight of a hyperpath is its *cost*, defined as the sum of the weights of its hyperarcs (see, for example, [11]). A different approach leads to defining the weight of a hyperpath in an inductive manner. First attempts in this direction can be found in [27] and in [11]. This approach has been formalized in [12] in the following terms.

Definition 2.1 *A* functional hypergraph $\mathcal{H}_F = \langle N, A; F \rangle$ *is a directed hypergraph* $\mathcal{H} = \langle N, A \rangle$ *in which each hyperarc* $e = \langle X, y \rangle \in A$ *is associated to a triple* $F_e = (w_e, \psi_e, f_e)$*, where:*

- w_e *is a real value;*
- ψ_e *is a function from* $|X|$*-tuples of reals to reals;*
- f_e *is a function from pairs of reals to reals.*

Let $\Pi_{S,t}$ be a hyperpath from S to t, and let $\langle Z, t \rangle$ be the last hyperarc in $\Pi_{S,t}$ (i.e., the *port* of t), where $Z = \{z_1, z_2, \ldots, z_k\}$: then[1] $\Pi_{S,t} = \Pi_{S,z_1} \cup \Pi_{S,z_2} \cup \ldots \cup \Pi_{S,z_k} \cup \{\langle Z, t \rangle\}$, where Π_{S,z_i} is the subhyperpath of $\Pi_{S,t}$ going from S to z_i, $1 \leq i \leq k$. The weight of $\Pi_{S,t}$ depends on $w_{\langle Z,t \rangle}$, that gives the *weight* of hyperarc $\langle Z, t \rangle$, and on $\psi_{\langle Z,t \rangle}$, that takes into account the weights of all hyperpaths Π_{S,z_i}. Function $f_{\langle Z,t \rangle}$ combines these two weights, giving the weight of hyperpath $\Pi_{S,t}$. Since f_e and ψ_e together determine the role of a hyperarc e in the weight of a hyperpath containing e, we will often combine them in a single function $g_e(w_e, x_1, x_2, \ldots, x_k) = f_e(w_e, \psi_e(x_1, x_2, \ldots, x_k))$.

Definition 2.2 *Given a functional directed hypergraph* $\mathcal{H}_F = \langle N, A; F \rangle$*, a* weight measure μ *associates a real weight to a hyperpath* $\Pi_{S,t}$ *as follows:*

- *if* $\Pi_{S,t}$ *has no hyperarcs (i.e.,* $t \in S$*), then* $\mu(\Pi_{S,t}) = \mu_0$*, where* μ_0 *is a proper constant;*
- *if* $\Pi_{S,t} = \Pi_{S,z_1} \cup \Pi_{S,z_2} \cup \ldots \cup \Pi_{S,z_k} \cup \{\langle Z, t \rangle\}$*, where* $Z = \{z_1, z_2, \ldots, z_k\}$*, then* $\mu(\Pi_{S,t}) = g_{\langle Z,t \rangle}(w_{\langle Z,t \rangle}, \mu(\Pi_{S,z_1}), \mu(\Pi_{S,z_2}), \ldots, \mu(\Pi_{S,z_k}))$.

Several weight measures have been introduced in the literature to define the weight of a hyperpath in a functional directed hypergraph, by considering, given a hyperarc e such that tail(e) $= \{x_1, x_2, \ldots, x_k\}$, different choices for functions ψ_e and f_e:

rank: $f_e(x, y) = x + y$, $\psi_e(x_1, x_2, \ldots, x_k) = \max_{1 \leq i \leq k}\{x_i\}$, and $\mu_0 = 0$;

gap: $f_e(x, y) = x + y$, $\psi_e(x_1, x_2, \ldots, x_k) = \min_{1 \leq i \leq k}\{x_i\}$, and $\mu_0 = 0$;

threshold: $f_e(x, y) = \max\{x, y\}$, $\psi_e(x_1, x_2, \ldots, x_k) = \max_{1 \leq i \leq k}\{x_i\}$, and $\mu_0 = 0$;

bottleneck: $f_e(x, y) = \min\{x, y\}$, $\psi_e(x_1, x_2, \ldots, x_k) = \min_{1 \leq i \leq k}\{x_i\}$, and $\mu_0 = +\infty$;

traversal cost: $f_e(x, y) = x + y$, $\psi_e(x_1, x_2, \ldots, x_k) = \sum_{1 \leq i \leq k}\{x_i\}$, when $w_e > 0$ for each $e \in A$, and $\mu_0 = 0$;

[1]By a little abuse of notation, given two hypergraphs $\mathcal{H}_1 = \langle N_1, A_1 \rangle$ and $\mathcal{H}_2 = \langle N_2, A_2 \rangle$ we denote the hypergraph $\langle N_1 \cup N_2, A_1 \cup A_2 \rangle$ by $\mathcal{H}_1 \cup \mathcal{H}_2$.

For example, if we assume that all hyperarcs of the hypergraph in Fig. 1 but h_5 have unit weight, while hyperarc h_5 has weight 2, then the thick hyperpath has rank 3, gap 2 and traversal cost 4.

Some of the weight measures listed above have been used by other authors with different names. For example, in [17, 30], the *rank* is called *distance*, and the *traversal cost* is called *value*.

Note that, the notion of *cost* differs from that of *traversal cost*. In fact, in the computation of traversal cost, the weight of a hyperarc may be considered many times. Actually, this is the reason why the *cost* cannot be defined inductively as in Definition 2.2.

3 Minimum weight traversal problems

As already mentioned, there are various ways in which a weight can be attached to a hyperpath. Depending on the weight measure used to assign a weight to a hyperpath the problem can be polynomially solvable or NP-hard. For example, in [11] it has been shown that the problem of finding minimum *cost* hyperpaths in a directed hypergraph is NP-hard. Conversely, when we take into consideration inductively defined measures, it is possible to characterize a number of cases in which the problem can be solved in polynomial time. Examples of inductively defined measures that can be computed in polynomial time are the *rank* and the *gap* [21]. Observe that cost, rank and gap are all generalizations of the standard notion of distance in graphs, which can be computed in polynomial time.

The more general approach for finding minimum weight hyperpaths in a functional directed hypergraph is finding a *fixed point* of the following set of equations, known as *generalized Bellman-Ford equations* (see, e.g., [17, 31]):

$$L(v) = \begin{cases} 0 & \text{if } v \in S \\ \min\limits_{e \in \mathrm{bstar}(v)} \{f_e(w_e, \psi_e(x_1, x_2, \ldots, x_k))\} & \text{if } v \in N \setminus S \end{cases}$$

where $\mathrm{tail}(e) = \{x_1, x_2, \ldots, x_k\}$, and f_e and ψ_e depend on the particular weight measure used. The complexity of this problem strongly depends on the characteristics of functions f_e and ψ_e, since, as we will see in the remainder of this section, the solution of the general problem can be achieved in several cases by a Dijkstra-like computation. Classes of weight functions that can be managed by Dijkstra's algorithm have been first introduced in [22] in the framework of context-free grammars as follows:

Definition 3.1 *A function $g(x_1, \ldots, x_k)$ from $(\mathbb{R}^+)^k$ to $\mathbb{R}^+$ is:*

- superior *(SUP) if it is monotone nondecreasing in each variable and $g(x_1, \ldots, x_k) \geq \max(x_1, \ldots, x_k)$. If the inequality is strict, then g is* strict superior *(SSUP).*
- inferior *(INF) if it is monotone nondecreasing in each variable and $g(x_1, \ldots, x_k) \leq \min(x_1, \ldots, x_k)$. If the inequality is strict, then g is* strict inferior *(SINF).*

Examples of superior functions are $\max_{1 \leq i \leq k}\{x_i\}$, and $\Sigma_{i=1}^k x_i$. Examples of inferior functions are $\min_{1 \leq i \leq k}\{x_i\}$, and $\Pi_{i=1}^k\{x_i\}$ when $0 \leq x_i \leq 1$, $i = 1, \ldots, k$.

Successively, Ramalingam and Reps introduced in [31] the following generalization of superior (inferior) functions:

Definition 3.2 *A function $g(x_1, \ldots, x_k)$ from $(\mathbb{R}^+)^k$ to $\mathbb{R}^+$ is:*

- weakly superior *(WSUP) if it is monotone nondecreasing in each variable and, for* $1 \leq i \leq k$, $g(x_1, \ldots, x_k) < x_i \;\Rightarrow\; g(x_1, \ldots, x_i, \ldots, x_k) = g(x_1, \ldots, \infty, \ldots, x_k)$
- weakly inferior *(WINF) if it is monotone nondecreasing in each variable and, for* $1 \leq i \leq k$, $g(x_1, \ldots, x_k) > x_i \;\Rightarrow\; g(x_1, \ldots, x_i, \ldots, x_k) = g(x_1, \ldots, \infty, \ldots, x_k)$

Examples of weakly superior functions that are not superior are $\min_{1 \leq i \leq k}\{x_i\}$, $\min_{1 \leq i \leq k}\{x_i\} * 2$, and any constant function. Examples of weakly inferior functions that are not inferior functions are $\max_{1 \leq i \leq k}\{x_i\}$, $\max_{1 \leq i \leq k}\{x_i\}/2$, and any constant function.

Ramalingam and Reps [31] considered also the classes of *strict weakly superior*, and *strict weakly inferior* functions, as defined below.

Definition 3.3 *A function* $g(x_1, \ldots, x_k)$ *from* $(\mathbb{R}^+)^k$ *to* $\mathbb{R}^+$ *is:*

- strict weakly superior *(SWSUP) if it is monotone nondecreasing in each variable and, for* $1 \leq i \leq k$, $g(x_1, \ldots, x_k) \leq x_i \;\Rightarrow\; g(x_1, \ldots, x_i, \ldots, x_k) = g(x_1, \ldots, \infty, \ldots, x_k)$
- strict weakly inferior *(SWINF) if it is monotone nondecreasing in each variable and, for* $1 \leq i \leq k$, $g(x_1, \ldots, x_k) \geq x_i \;\Rightarrow\; g(x_1, \ldots, x_i, \ldots, x_k) = g(x_1, \ldots, \infty, \ldots, x_k)$

Let us now see how the above defined classes of functions relate to hypergraphs. Remember that in a functional directed hypergraph, each hyperarc $e \in \mathcal{H}$ is associated to a triple (w_e, ψ_e, f_e), thus defining a function g_e as a combination of f_e and ψ_e. Given $e = \langle X, t \rangle$, with $X = \{x_1, x_2, \ldots, x_k\}$, the weight μ of any hyperpath $\Pi_{S,t}$ having e as the last hyperarc is given by $\mu(\Pi_{S,t}) = g_e(w_e, \mu(\Pi_{S,x_1}), \mu(\Pi_{S,x_2}), \ldots, \mu(\Pi_{S,x_k}))$. If functions g_e, for all $e \in \mathcal{H}$, are, say, superior functions, then the overall weight measure μ is superior as well (with respect to arguments μ_0 and w_e, for each hyperarc e in the considered hyperpath). The table in Fig. 2 summarizes the properties of the weight measures known in the literature with respect to the classes of function defined above. Concerning the relationship among the different classes of weight measures, as well as their compositions, we refer to [12].

	$f_e(w_e, \psi_e)$	$\psi_e(\mu_1, \ldots, \mu_k)$	$g_e(w_e, \mu_1, \ldots, \mu_k)$	μ
rank $w_e > 0$	$+$ *SSUP*	max *SUP*, *WINF*	*SSUP*	*SSUP*
gap $w_e > 0$	$+$ *SSUP*	min *WSUP*, *INF*	*SWSUP*	*SWSUP*
bottleneck $w_e > 0$	min *WSUP*, *INF*	min *WSUP*, *INF*	*WSUP*, *INF*	*WSUP*, *INF*
threshold $w_e > 0$	max *SUP*, *WINF*	max *SUP*, *WINF*	*SUP*, *WINF*	*SUP*, *WINF*
traversal cost $w_e > 0$	$+$ *SSUP*	$\sum$ *SSUP*	*SSUP*	*SSUP*

Figure 2: Properties of hyperpath weight measures.

The generalization of Dijkstra's algorithm to directed hypergraphs (or similar structures) was first tackled by Martelli and Montanari in [26] and subsequently extended by Knuth in [22]. For weight measures based on superior (inferior) functions, it is possible to determine minimum (maximum) cost derivation trees in a weighted context free grammar (or, equivalently, a minimum (maximum) weight hyperpath in a functional directed hypergraph) in $O(\text{size}(\mathcal{H}) + m \log n)$ worst case time (or $O(\text{size}(\mathcal{H}) + n \log n)$ by using Fibonacci heaps [16]). This result was subsequently extended in [12, 27, 31] to weakly superior (inferior) functions, thus including all weight measures in Fig. 2. In [12], it is also shown how the suitability of Dijkstra-based algorithms is related to the properties of weight measures and to the existence of particular types of cycles in hypergraphs.

4 Maintaining minimum weight hyperpaths

Let $\mathcal{H} = \langle N, A; F \rangle$ be a functional directed hypergraph with n nodes and m hyperarcs. We assume that the weight of any minimum weight hyperpath in $\mathcal{H}$ is an integer in the interval $[1, C]$; for example, if all hyperarc weights are integers in $[1, w_{MAX}]$, then in the case of the rank and gap weight measure we have $C \leq n \cdot w_{MAX}$, while in the case of the bottleneck weight measure we have $C \leq w_{MAX}$.

Here, we consider the problem of maintaining minimum weight hyperpaths of $\mathcal{H}$ in the following partially dynamic settings:

incremental: hyperarc weight decrements and hyperarc insertions are allowed;

decremental: hyperarc weight increments and hyperarc deletions are allowed.

We denote by $\mathcal{H}'$ the hypergraph obtained from $\mathcal{H}$ after applying a weight change, an insertion or a deletion to hyperarc $e = \langle X, y \rangle$, and by $\mathrm{w}(v)$ ($\mathrm{w}'(v)$) the weight of the minimum weight hyperpath from a fixed source set S to v in $\mathcal{H}$ ($\mathcal{H}'$). Analogously, $\mathrm{p}(v)$ ($\mathrm{p}'(v)$) denotes the port of v in $\mathcal{H}$ ($\mathcal{H}'$).

Hypergraph $\mathcal{H}$ is represented by associating to each node v two simple lists containing all hyperarcs in $\text{bstar}(v)$ and $\text{fstar}(v)$. Minimum weight hyperpaths are represented as follows. For each node v, we store $\text{weight}(v)$, that coincides with $\mathrm{w}(v)$ ($\mathrm{w}'(v)$) before (after) the update, and $\text{port}(v)$, that coincides with $\mathrm{p}(v)$ ($\mathrm{p}'(v)$) before (after) the update. For each hyperarc e in $\mathcal{H}$, we explicitly store the value $G(e) = g_e(w_e, \text{weight}(x_1), \text{weight}(x_2), \ldots, \text{weight}(x_k))$, where $\{x_1, x_2, \ldots, x_k\} = \text{tail}(e)$. In order to keep the space occupancy within $O(\text{size}(\mathcal{H}))$, we explicitly represent each hyperarc in $\mathcal{H}$ only once; all the occurrences of hyperarcs in the above data structures are implemented as references.

In a minimization problem, as a consequence of a hyperarc weight increment (decrement) or a hyperarc deletion (insertion), the weight of some hyperpath may increase (decrease) or some hyperpaths may disappear (appear), and the weight of minimum weight hyperpaths can only increase (decrease). Analogous considerations hold for a maximization problem.

In what follows, we tackle the minimization problem and assume that the functions g_e are *SWSUP* (*SWINF*), for each hyperarc e in $\mathcal{H}$. For example, it is sufficient that f_e and ψ_e are both monotone non decreasing (non increasing), and either f_e or ψ_e is *SWSUP* (*SWINF*). Since all *SSUP* functions are *SWSUP*, our algoritms are able to maintain minimum weight hyperpaths under the rank, gap, and traversal cost weight measures (see Fig. 2).

A fundamental step in our procedures is updating the value of $G(e)$. In the complexity analysis we assume that for any hyperarc e in $\mathcal{H}$ this can be done within the following time bounds:

T1. $g_e(w_e, x_1, x_2, \ldots, x_k)$ can be maintained under increments of w_e within $O(1)$ worst case time;

T2. $g_e(w_e, x_1, x_2, \ldots, x_k)$ can be maintained under increments of any x_i within $O(1)$ amortized time over an arbitrary sequence of weight increment operations.

It is simple to see that bounds *T1* and *T2* can be accomplished for the rank, gap and traversal cost weight measures.

In the following, we introduce an incremental and a decremental algorithm. The formal proof of correctness and the complexity analysis are given in detail only for the decremental case. The incremental case, which is a straightforward extension of Dijkstra's algorithm, is provided here for sake of completeness. For the complexity analysis of the incremental case we assume that bounds *T1* and *T2* apply to the maintenance of g_e under weight decrements.

4.1 The incremental problem

In this section, we describe Procedure `Weight_Decrease` that maintains the minimum weight hyperpaths of $\mathcal{H}$ from a fixed source $S \subset N$ to all other nodes of $\mathcal{H}$ under hyperarc weight decrements. We then discuss how to modify Procedure `Weight_Decrease` in order to handle hyperarc insertions. The algorithm is detailed in Fig. 3.

First of all, we observe that the weight decrement of hyperarc $e = \langle X, y \rangle$ in $\mathcal{H}$ can only affect nodes reachable from y in $\mathcal{H}$, and the minimum weight of these nodes can only decrease. We thus concentrate on the nodes in $\mathcal{H}$ reachable from y.

Procedure `Weight_Decrease` explores the set of nodes whose weight changes after decreasing the weight of hyperarc $e = \langle X, y \rangle$, starting from node y, and examining the affected nodes by increasing minimum weight. Nodes affected by the weight decrement are temporarily stored in an array of sets of nodes, named NewWeightSet. A node v is put in NewWeightSet(i) if and only if its weight decreases to i.

```
Procedure Weight_Decrease(e = ⟨X, y⟩; δ)
w_e ← w_e − δ
update G(e)
if G(e) ≥ weight(y) then EXIT
NewWeightSet(G(e)) ← {y}
port(y) ← e
weight(y) ← G(e)
for i ← G(e) to C do
    foreach v ∈ NewWeightSet(i) do
        delete v from NewWeightSet(i)
        foreach e' ∈ fstar(v) do
            update G(e')
            let z be head(e')
            if G(e') < weight(z) then
                if z ∈ NewWeightSet(·) then
                    delete z from NewWeightSet(·)
                insert z into NewWeightSet(G(e'))
                port(z) ← e'
                weight(z) ← G(e')
        endfor
    endfor
endfor
```

Figure 3: Procedure `Weight_Decrease`.

The algorithm works for *SWSUP* weight functions. In fact, the weight of a hyperpath depends on the weights of nodes and hyperarcs in the hyperpath. On the other hand, if one of these values is greater or equal than the function value, then it is not relevant, in the sense that, as can be seen by Definition 3.3, this value can grow to $+\infty$ without affecting the function value. Thus, all relevant nodes in a minimum hyperpath from S to v have weight less than $\mathrm{w}'(v)$.

Let us observe what happens in Line 13: since nodes are examined by non-decreasing weight, if there was a hyperpath from S to z having weight less than i, all its relevant nodes should have been already examined. In this case the port and weight of each node in this hyperpath would have been already assigned, and the condition in Line 13 would be false. This implies that values assigned in Lines 5, 6, 17 and 18 are correct.

The complexity of Procedure `Weight_Decrease` strictly depends on which nodes are inserted in and extracted from NewWeightSet. By conditions in Lines 3 and 13, we see that only nodes whose weight changes are considered. Any time a node is considered, the dominating cost is due to scanning its forward star and updating hyperarc weights (see Lines 10 and 11). Since each node weight can decrease at most C times, and exploiting bound *T2*, the overall time required for managing any sequence of L hyperarc weight decrements is $O(L + C \cdot \mathrm{size}(\mathcal{H}))$.

Hyperarc insertions can be managed by Procedure `Weight_Decrease` by simply changing the very first lines as follows: in Line 1 the new hyperarc e must be properly inserted in the data structures, and in Line 2 the value of $G(e)$ must be computed from skratch. In this case the bound in *T1* does not apply, but the same overall complexity is maintained if $G(e)$ can be computed from scratch in $O(|\mathrm{tail}(e)|)$ time.

4.2 A new decremental algorithm

In this section, we describe a new algorithm for maintaining the minimum weight hyperpaths of $\mathcal{H}$ from a fixed source $S \subset N$ to all other nodes of $\mathcal{H}$ under hyperarc weight increments and hyperarc deletions. The algorithm is called `Weight_Increase` and is given in Fig. 4. We consider the case that the weight of a hyperarc e is increased by a positive quantity δ, and then discuss how to modify Procedure `Weight_Increase` in order to handle hyperarc deletions.

First of all, we observe that after updating w_e in $\mathcal{H}$, each hyperpath in $\mathcal{H}$ that does not contain e preserves its weight. Hence, $\mathrm{w}'(v) = \mathrm{w}(v)$ for each node v whose minimum weight hyperpath in $\mathcal{H}$ does not contain e. We thus concentrate on the set of nodes in $\mathcal{H}$ whose current minimum weight hyperpath from S contains e. Given one of these nodes v, either $\mathrm{w}'(v) = \mathrm{w}(v)$, or $\mathrm{w}'(v) > \mathrm{w}(v)$.

Procedure `Weight_Increase` explores the set of nodes whose weight and/or port changes under the update of hyperarc $e = \langle X, y \rangle$, and builds the new minimum weight hyperpaths as follows. The hypergraph is visited starting from node y and examining nodes by increasing minimum weight. The visit is pruned any time a hyperarc is found whose weight does not change or does not belong to any minimum weight hyperpath.

Each iteration of the **for** loop at Line 5 identifies the set of nodes having weight i (after the update), whose weight and/or port changes due to the hyperarc update, by selecting all nodes v such that $\mathrm{w}(v) = i$, $\mathrm{w}'(v) = i$ and $\mathrm{p}'(v) \neq \mathrm{p}(v)$, and all nodes v such that $\mathrm{w}(v) < i$ and $\mathrm{w}'(v) = i$.

Inspected nodes are temporarily stored in an array of sets of nodes, named NewWeightSet. A node v is put in NewWeightSet(i) if and only if $\mathrm{w}'(v)$ is known to be at least i. Nodes are extracted from set NewWeightSet(i) by increasing i, and for each node v we check whether there is a hyperarc h such that $f_h(w_h, \psi_h(\mathrm{weight}(x_1), \mathrm{weight}(x_2), \ldots, \mathrm{weight}(x_k))) = i$, where $\{x_1, x_2, \ldots, x_k\} = \mathrm{tail}(h)$. If this is the case, the weight of node v is set to i, otherwise v is inserted into some set NewWeightSet(j), where $j > i$, for future inspection. In order to improve

```
Procedure Weight_Increase(e = ⟨X, y⟩; δ)
1.   w_e ← w_e + δ
2.   update G(e)
3.   if port(y) ≠ e or G(e) has not changed then EXIT
4.   insert y into NewWeightSet(weight(y))
5.   for i ← weight(y) to C do
6.       foreach v ∈ NewWeightSet(i) do
7.           delete v from NewWeightSet(i)
8.           search the first h in bstar(v) such that G(h) = i
9.           if h exists then
10.              port(v) ← h
11.          else
12.              port(v) ← NIL
13.              weight(v) ← min {G(h) | h ∈ bstar(v)}
14.              insert v in NewWeightSet(weight(v))
15.              foreach e′ ∈ fstar(v) do
16.                  update G(e′)
17.                  if G(e′) has changed then
18.                      let z be head(e′)
19.                      if e′ = port(z) and z ∉ NewWeightSet(weight(z)) then
20.                          insert z into NewWeightSet(weight(z))
21.              endfor
22.      endfor
23.  endfor
```

Figure 4: Procedure `Weight_Increase`.

the algorithm performance, it is convenient to set j as $\min\{G(h) \mid h \in \text{bstar}(v)\}$.

Theorem 4.1 *Procedure `Weight_Increase` correctly updates weight(v) and port(v), for each node v, after increasing the weight of a hyperarc.*

Proof. We prove the theorem by showing that, if the weight of hyperarc $\langle X, y\rangle$ is increased by a quantity δ, then upon termination of `Weight_Increase`$(\langle X, y\rangle;\ \delta)$, for each node v, we have $\text{weight}(v) = \text{w}'(v)$ and $\text{port}(v) = \text{p}'(v)$. We assume that $\text{port}(y) = \text{p}(y) = \langle X, y\rangle$ and $G(\langle X, y\rangle)$ changes as a consequence of the weight increment on $\langle X, y\rangle$, otherwise minimum weight hyperpaths do not change. We also assume that $\text{weight}(v) = \text{w}(v)$ and $\text{port}(v) = \text{p}(v)$, for each node v, before executing procedure `Weight_Increase`.

The following properties hold during the execution of `Weight_Increase`:

P1. *For each hyperarc e, $G(e) = g_e(w_e, weight(x_1), weight(x_2), \ldots, weight(x_k))$, where $\{x_1, x_2, \ldots, x_k\} = tail(e)$.*
It is sufficient to observe that, any time weight(v) is modified for some node v (Line 13), $G(e')$ is modified accordingly for any $e' \in \text{fstar}(v)$ (Line 16). Moreover, if w_e increases $G(e)$ is updated in Line 2.

P2. *If $v \in NewWeightSet(j)$ for some j, then $weight(v) = j$.*
When a node is inserted into NewWeightSet (see Lines 4, 14 and 20), the property trivially holds. Note that, any time the value of weight(v) is modified at Line 13, node v has already

been extracted from NewWeightSet at Line 7.

P3. *For each node v, weight(v) can only increase during the execution of Procedure* `Weight_Increase`.
Only the weight of nodes extracted from NewWeightSet can change. Let v be a node in NewWeightSet(k), for some $k \geq \mathrm{w}(y)$, considered during an iteration of the loop at Line 6. Due to Property *P2*, weight$(v) = k$. Node v can get a new weight value at Line 13 only if the test at Line 9 fails. In this case, $G(e') > k$ for each hyperarc $e' \in \mathrm{bstar}(v)$, hence weight$(v) > k$.

P4. *For each node v, $w(v) \leq$ weight$(v) \leq w'(v)$.*
By assumption, weight$(v) = \mathrm{w}(v)$, for each node v, before processing every weight increment. Thus, Property *P3* implies that weight$(v) \geq \mathrm{w}(v)$. It remains to show that weight$(v) \leq \mathrm{w}'(v)$. Let us assume by absurd that x is the first node that gets weight$(x) > \mathrm{w}'(x)$. The value of weight(x) has been set at Line 13. Since x is the first node that gets a "wrong" weight, we know that weight$(z) \leq \mathrm{w}'(z)$, for each node $z \neq x$. By monotonicity of g_h, for each hyperarc $h \in \mathrm{bstar}(x)$, $g_h(w_h, \mathrm{weight}(z_1), \mathrm{weight}(z_2), \ldots, \mathrm{weight}(z_k)) \leq g_h(w_h, \mathrm{w}'(z_1), \mathrm{w}'(z_2), \ldots, \mathrm{w}'(z_k))$, where $\{z_1, z_2, \ldots, z_k\} = \mathrm{tail}(h)$. Let us denote by $G'(h)$ the value $g_h(w_h, \mathrm{w}'(z_1), \mathrm{w}'(z_2), \ldots, \mathrm{w}'(z_k))$. Now, by Property *P1*, $G(h) = g_h(w_h, \mathrm{weight}(z_1), \mathrm{weight}(z_2), \ldots, \mathrm{weight}(z_k))$, and, by Line 13, weight$(x) = \min_{h \in \mathrm{bstar}(x)} G(h) \leq \min_{h \in \mathrm{bstar}(x)} G'(h) = \mathrm{w}'(x)$, contradicting the assumption that weight$(x) > \mathrm{w}'(x)$.

P5. *If $(w(v) \neq w'(v)) \vee (p(v) \neq p'(v))$ and weight$(v) = k$, then v is inserted into NewWeightSet(k).*
If $\mathrm{w}(v) \neq \mathrm{w}'(v)$ or $\mathrm{p}(v) \neq \mathrm{p}'(v)$, then two cases are possible:

1. $v \equiv y$ ($w_{\mathrm{p}(v)}$ has been increased). In this case, v has been inserted into NewWeightSet(weight(v)) at Line 4.
2. $\mathrm{w}(z) < \mathrm{w}'(z)$, for some $z \in \mathrm{tail}(\mathrm{p}(v))$. Since $g_{\mathrm{p}(v)}$ is *SWSUP*, then it can increase only due to changes in $\mathrm{w}(z)$ such that $\mathrm{w}(z) < g_{\mathrm{p}(v)}$. This implies that:
   - z has been increased at least once during the previous iterations of the loop at Line 5;
   - hyperarc port(v) has been visited at Line 15;
   - v has been inserted into NewWeightSet(weight(v)) at Line 20.

In both cases, v has been inserted into NewWeightSet(weight(v)). It is possible that weight(v) has been increased again, say from a to a', during the previous iterations, but in that case v had been moved from NewWeightSet(a) (Line 7) to NewWeightSet(a') (Line 14). Since weight(v) is now k, then $v \in$ NewWeightSet(k).

By exploiting the above properties, we show now that, at the end of Procedure `Weight_Increase`, weight$(v) = \mathrm{w}'(v)$ for each node v. The proof proceeds by induction on the weight values, by studying the behaviour of a single iteration of the loop at Line 5. Let k be the current value of variable i, $k \geq \mathrm{w}(y)$. We show that, during iteration k, if weight$(v) = \mathrm{w}'(v)$ for all nodes having weight$(v) < k$ before Line 6, then weight$(v) = \mathrm{w}'(v)$ for all nodes having weight$(v) \leq k$ after Line 22.

Basic step: $(k < \mathrm{w}(y))$ Let us consider a minimum weight hyperpath $\Pi_{S,v}$ in $\mathcal{H}$ such that $\mu(\Pi_{S,v}) < \mathrm{w}(y)$. Two cases may arise: either $\langle X, y\rangle \notin \Pi_{S,v}$, or $\langle X, y\rangle \in \Pi_{S,v}$. In the first case, $\mu(\Pi_{S,v})$ cannot be affected by the increment in $w_{\langle X,y\rangle}$, hence $\mathrm{w}'(v) = \mathrm{w}(v)$. In the second case, $\mu(\Pi_{S,v})$ can be obtained as the composition of some functions g_h, for some $h \in \Pi_{S,v}$, that use $\mathrm{weight}(y)$ and w_h as arguments. Since all g_h are *SWSUP*, also $\mu(\Pi_{S,v})$ is *SWSUP* with respect to arguments $\mathrm{weight}(y)$ and w_h's. Thus, if $\mu(\Pi_{S,v}) < \mathrm{w}(y)$, then $\mu(\Pi_{S,v})$ is not affected by the increment in $w_{\langle X,y\rangle}$, hence $\mathrm{w}'(v) = \mathrm{w}(v)$.

Inductive step: ($k \geq \mathrm{w}(y)$) We must show that, after iteration k, if $\mathrm{weight}(v) = k$ then $\mathrm{weight}(v) = \mathrm{w}'(v)$.

Let us consider a node v such that $\mathrm{weight}(v) = k$. If $\mathrm{w}(v) = \mathrm{w}'(v)$, then, thanks to Property *P4*, $\mathrm{weight}(v) = \mathrm{w}'(v)$. Otherwise, thanks to Property *P5*, v is in $\mathrm{NewWeightSet}(k)$.

Nodes in $\mathrm{NewWeightSet}(k)$ are examined in the for loop at Line 6. Given a node $v \in \mathrm{NewWeightSet}(k)$, we distinguish two cases: either there is a hyperarc $h \in \mathrm{bstar}(v)$ such that $G(h) = k$ or not.

If h exists (Line 10), then by Property *P1* it follows that $G(h) = g_h(w_h, \mathrm{weight}(v_1), \mathrm{weight}(v_2), \ldots, \mathrm{weight}(v_j))$, where $\{v_1, v_2, \ldots, v_j\} = \mathrm{tail}(h)$. By the inductive hypothesis, $\mathrm{weight}(v_i) = \mathrm{w}'(v_i)$, for all v_i such that $\mathrm{weight}(v_i) < G(h)$. Moreover, since function $g_h(w_h, \mathrm{weight}(v_1), \mathrm{weight}(v_2), \ldots, \mathrm{weight}(v_j))$ is *SWSUP*, then g_h is invariant with respect to all $\mathrm{weight}(v_i) \geq G(h)$.

Otherwise, $\mathrm{weight}(v)$ is assigned a value greater than k (Line 13), v is inserted into $\mathrm{NewWeightSet}(\mathrm{weight}(k))$ (Line 14), and v will be considered in a next iteration.

Note that, a node v gets its correct port any time a hyperarc $h \in \mathrm{bstar}(v)$ is found having $G(h) = \mathrm{w}'(v)$ (Line 10). □

In what follows we analyze the time complexity of Procedure `Weight_Increase`. The time required to manage a single weight increment operation strongly depends on the size of the affected portion of the hypergraph and on the amount of the weight change for the affected nodes. It is not easy to give a non trivial upper bound on the time spent in processing a single update, and hence we provide an amortized bound [32], measuring the performance of our algorithm on a whole sequence of hyperarc weight increments.

Theorem 4.2 *The total worst case time needed by Procedure* `Weight_Increase` *for handling any sequence of L hyperarc weight increments is $O(L + C \cdot size(\mathcal{H}))$.*

Proof. Term L in the stated complexity bound comes from Condition *T1* above, since for each weight increment on hyperarc e, the cost of Lines 1 to 3 is $O(1)$.

Algorithm `Weight_Increase` examines nodes in $\mathrm{NewWeightSet}(i)$, for $i = \mathrm{w}(y), \ldots, C$ (loops at Line 5 and 6). Sets NewWeightSet are implemented by means of an array of C lists, in which the i-th list contains all nodes in $\mathrm{NewWeightSet}(i)$. Insertion of a node can be done in constant time. Node deletions are also performed in constant time per node, since all nodes in $\mathrm{NewWeightSet}(i)$ are sequentially deleted in the same loop (Line 7).

Let us observe what happens when a node v is extracted from $\mathrm{NewWeightSet}(i)$, for some $\mathrm{w}(y) \leq i \leq C$: either there is a hyperarc $e \in \mathrm{bstar}(v)$ such that $G(e) = i$ (Line 10), or $\mathrm{weight}(v)$ gets a value strictly greater than i (Line 12). In the latter case, v is inserted into $\mathrm{NewWeightSet}(\mathrm{weight}(v))$ at Line 14. The value $\mathrm{weight}(v)$ can only increase while processing a hyperarc weight increment (see Property *P3* in the proof of Theorem 4.1). Since $\mathrm{weight}(v)$ is an integer in $[1, C]$, then it can increase at most C times during a single execution of `Weight_Increase`. Moreover, the value $\mathrm{w}'(v)$ after processing a weight increment coincides with $\mathrm{w}(v)$ before processing the next weight increment. This implies that $\mathrm{weight}(v)$ can increase at most C times while processing an arbitrary sequence of hyperarc weight increments.

Any time a node v is extracted from $\mathrm{NewWeightSet}(\mathrm{weight}(v))$ — this happens at most once for each possible value of $\mathrm{weight}(v)$ — all hyperarcs in $\mathrm{fstar}(v)$ are scanned. By the above discussion, the loop at Line 15 requires an overall $O(C \cdot \sum_{v \in N} |\mathrm{fstar}(v)|) = O(C \cdot \mathrm{size}(\mathcal{H}))$ worst case time during any sequence of hyperarc weight increments.

Line 16 is performed any time the value of $\mathrm{weight}(z)$ changes, where $z \in \mathrm{tail}(e')$, for each hyperarc e'. This gives at most $C \cdot |\mathrm{tail}(e')|$ executions of Line 16 for each hyperarc e' during the whole sequence of hyperarc weight increments. Since, by Condition *T2*, Line 16 requires $O(1)$ amortized time, it follows that the overall time spent to perfom this line is $\sum_{e' \in A} C \cdot |\mathrm{tail}(e')| = O(C \cdot \mathrm{size}(\mathcal{H}))$.

We show now that the overall time spent at Lines 8 and 13 is $O(C \cdot m)$. We implement $\mathrm{bstar}(v)$ as a simple list, in which hyperarcs are stored in any fixed order. The value of $\mathrm{port}(v)$ is actually implemented as a pointer to a hyperarc in this list. We assume that, if $\mathrm{port}(v) \neq$ NIL, then each hyperarc h that precedes $\mathrm{port}(v)$ in $\mathrm{bstar}(v)$ has $G(h) > G(\mathrm{port}(v))$; since, while performing iteration k of the loop at Line 5, the search in Line 8 always stops on the first hyperarc having $G(h) = k$, this property is always maintained. Thus, when looking for a hyperarc at Line 8, we can sequentially scan $\mathrm{bstar}(v)$ starting from $\mathrm{port}(v)$ or, if $\mathrm{port}(v) =$ NIL, we restart scanning from the beginning of $\mathrm{bstar}(v)$. Since $\mathrm{port}(v)$ becomes NIL only when $\mathrm{w}'(v) > \mathrm{w}(v)$ (Line 12), at most one sequential scan of $\mathrm{bstar}(v)$ is performed for each different value of $\mathrm{weight}(v)$. Hence, Line 8 requires $O(|\mathrm{bstar}(v)|)$ worst case time for each possible value of $\mathrm{weight}(v)$, giving overall $O(C \cdot \sum_{v \in N} |\mathrm{bstar}(v)|) = O(C \cdot m)$ worst case time during any sequence of hyperarc weight increments.

The execution of Line 13 requires a whole scan of $\mathrm{bstar}(v)$ any time $\mathrm{weight}(v)$ increases, thus giving $O(C \cdot \sum_{v \in N} |\mathrm{bstar}(v)|) = O(C \cdot m)$ worst case time during any sequence of hyperarc weight increments.

Summing up the above bounds, it follows that the worst case time required by Procedure `Weight_Increase` to manage a sequence of L hyperarc weight increments is given by $O(L + C \cdot m + C \cdot \mathrm{size}(\mathcal{H}))$. Since $m \leq \mathrm{size}(\mathcal{H})$, the thesis follows. □

Concerning the space requirements, in addition to sets $\mathrm{bstar}(v)$ and $\mathrm{fstar}(v)$, we only need to represent constant size information with each node and hyperarc. Moreover, each node occurs in at most one of the sets in NewWeightSet at a time. It follows that the total worst case space needed by Procedure `Weight_Increase` is $O(\mathrm{size}(\mathcal{H}))$.

4.3 Dealing with hyperarc deletions

The main problem in managing hyperarc deletions on a functional directed hypergraph $\mathcal{H}$ is detecting unreachable nodes. The approach of procedure `Weight_Increase` cannot be directly extended to manage hyperarc deletions, since we cannot ensure a *monotonicity* property on hyperpaths. In fact, due to Definition 3.3, in the case of *SWSUP* weight functions, if a hyperpath $\Pi_{S,v}$ is a proper subhypergraph of hyperpath $\Pi_{S,w}$, then $\Pi_{S,v}$ and $\Pi_{S,w}$ may have the same weight. Instead, in the case of *SSUP* weight functions, the weight is strictly increasing along a hyperpath.

Actually, reachability is only related to the structure of hypergraph $\mathcal{H}$, not to the particular weight function. Thus, unreachable nodes can be detected on $\mathcal{H}$ by maintaining minimum weight hyperpaths in the simple case of the *rank* weight function, with unit hyperarc weights, under a sequence of hyperarc weight increments and hyperarc deletions. This is only done in order to detect unreachable nodes before updating minimum weight hyperpaths under the actual *SWSUP* weight function.

Procedure `Detect_Unreachable`, shown in Figure 5, is based on the same approach as Procedure `Weight_Increase`, but weight values are computed on the basis of the rank weight functions. Thus, each node v is associated to its current $\mathrm{rport}(v)$ (the last hyperarc in the minimum rank hyperpath from S to v), $\mathrm{rank}(v)$ (the analogous of $\mathrm{weight}(v)$, storing the current rank value), and for each hyperarc e, we store the value $\mathrm{Trank}(e) = \max\{\mathrm{rank}(v) \mid v \in \mathrm{tail}(e)\} + 1$ (the analogous of $G(e)$).

```
Procedure Detect_Unreachable(e = ⟨X, y⟩)
remove e from the hypergraph
if rport(y) ≠ e then EXIT
NewLevel(rank(y)) ← {y}
for i ← rank(y) to n do
    foreach v ∈ NewLevel(i) do
        delete v from NewLevel(i)
        search the first h in bstar(v) such that Trank(h) = i
        if h exists then
            rport(v) ← h
        else
            rport(v) ← NIL
            if bstar(v) = ∅ then
                rank(v) ← n + 1
            else
                MinTrank ← min {Trank(h) | h ∈ bstar(v)}
                rank(v) ← min {n + 1, MinTrank}
                insert v in NewLevel(rank(v))
                foreach f ∈ fstar(v) do
                    if rank(v) ≥ Trank(f) then
                        Trank(f) ← rank(v) + 1
                        let z be head(f)
                        if f = rport(z) and z ∉ NewLevel(rank(z)) then
                            insert z into NewLevel(rank(z))
                endfor
    endfor
endfor
foreach v ∈ NewLevel(n + 1) do
    mark v as unreachable
endfor
```

Figure 5: Procedure `Detect_Unreachable`.

Inspected nodes are temporarily stored in an array of sets of nodes, denoted as NewLevel (corresponding to sets NewWeightSet). A node v is put in $\mathrm{NewLevel}(i)$ if and only if its new rank value is known to be at least i. With respect to Procedure `Weight_Increase`, an additional set $\mathrm{NewLevel}(n+1)$ is maintained, that contains the nodes that are found to be unreachable from S after the deletion. These are the nodes whose minimum rank exceeds the maximum allowed value.

The correctness of Procedure `Detect_Unreachable` can be proven by following the same line of the proof of Theorem 4.1, and exploiting the monotonicity property on the rank weight function. Concerning the cost of Procedure `Detect_Unreachable`, in a way similar to the proof of Theorem 4.2

it is possible to show that the overall time needed to update minimum rank hyperpaths and to detect unreachable nodes during an arbitrary sequence of hyperarc deletions is $O(n \cdot \text{size}(\mathcal{H}))$.

Based on the above considerations, we can extend Procedure `Weight_Increase` to manage intermixed sequences of hyperarc weight increments and hyperarc deletions as follows.

In the case of a hyperarc weight increment, the topology of the hypergraph does not change, therefore also unit weight minimum rank hyperpaths are unchanged, and Procedure `Weight_Increase` is performed. In the case of a hyperarc deletion, we perform the following steps:

1. minimum rank hyperpaths are updated and unreachable nodes are detected by performing Procedure `Detect_Unreachable`;

2. unreachable nodes, and the corresponding backward and forward stars, are deleted from the hypergraph;

3. any node v whose port has been deleted is put into NewWeightSet(weight(v)), and its port is set to **nil**;

4. Procedure `Weight_Increase` is performed, starting from Line 5.

The worst case time required by the above algorithm for handling any sequence of L intermixed hyperarc weight increments and hyperarc deletions is $O(L+\max(n,C)\cdot\text{size}(\mathcal{H}))$. This easily follows from Theorem 4.2 and from the fact that minimum rank hyperpaths in the case of unit hyperarc weights can be decrementally maintained in overall $O(n \cdot \text{size}(\mathcal{H}))$ worst case time.

5 Concluding remarks

We have considered the problem of dynamically maintaining minimum weight hyperpaths in a directed hypergraph subject to modifications as hyperarc weight changes and hyperarc insertions or deletions. We have proposed a decremental solution for this problem that works for a wide class of weight functions and is efficient in terms of amortized complexity.

The algorithms proposed in this paper can easily be extended in order to update minimum weight hyperpaths after a batch of updates, instead of adjusting hyperpaths after any single hyperarc update. Although the worst case time required to deal with a sequence of batched updates asymptotically equals the bounds proved for the single updates, we expect the batched approach to be faster in practice.

A possible future research direction could be to apply some of the techniques presented in this paper to the fully-dynamic problem. It is easy to see that our algorithm could be very easily extended to a fully-dynamic one. Anyway, the complexity analysis is no longer valid, since if we intermix weight increments and decrements then weight values can arbitrarily increase and descrease. Nevertheless, due to the fact that these algorithms essentially visit only nodes whose weight changes, and their incoming and outgoing hyperarcs, we expect them to be fast in practice.

References

[1] P. Alimonti, E. Feuerstein, and U. Nanni. Linear time algorithms for liveness and boundedness in conflict–free Petri nets. In *Proceedings of Latin American symposium on Theoretical INformatics (LATIN'92)*, volume 583 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 1992.

[2] C. Alonso, B. Pulido, and G. G. Acosta. On line industrial diagnosis: an attempt to apply artificial intelligence techniques to process control. In *11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA-98-AIE)*, volume 1415 of *Lecture Notes in Artificial Intelligence*, pages 804–813. Springer-Verlag, 1998.

[3] H. R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3–30, 1994.

[4] J. Aráoz. Forward chaining is simple(x). *Operations Research Letters*, 26:23–26, 2000.

[5] G. Ausiello, A. D'Atri, and D. Saccà. Graph algorithms for functional dependency manipulation. *Journal of the ACM*, 30(4):752–766, 1983.

[6] G. Ausiello, A. D'Atri, and D. Saccà. Minimal representation of directed hypergraphs. *SIAM Journal on Computing*, 15(2):418–431, 1986.

[7] G. Ausiello, P. G. Franciosa, D. Frigioni, and R. Giaccio. Decremental maintenance of minimum rank hyperpaths and minimum models of Horn formulæ. Technical Report ALCOMFT-TR-01-19, IST Programme of the EU IST-1999-14186 (ALCOM-FT), 2001. http://www.brics.dk/cgi-alcomft/db?state=reports.

[8] G. Ausiello and R. Giaccio. On-line algorithms for satisfiability problems with uncertainty. *Theoretical Computer Science*, 171(1–2):3–24, 1997.

[9] G. Ausiello and G. F. Italiano. On-line algorithms for polynomially solvable satisfiability problems. *Journal of Logic Programming*, 10(1/2/3–4):69–90, 1991.

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

[11] G. Ausiello, G. F. Italiano, and U. Nanni. Optimal traversal of directed hypergraphs. Technical Report TR-92-073, International Computer Science Institute, Berkeley, CA, September 1992.

[12] G. Ausiello, G. F. Italiano, and U. Nanni. Hypergraph traversal revisited: Cost measures and dynamic algorithms. In *Symposium on Mathematical Foundations of Computer Science (MFCS'98)*, volume 1450 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 1998.

[13] C. Berge. *Graphs and Hypergraphs.* North-Holland, Amsterdam, The Netherlands, 1973.

[14] H. Boley. Directed recursive labelnode hypergraphs: A new representation language. *Artificial Intelligence*, 9(1):49–85, 1977.

[15] J. de Kleer. An assumption based truth maintenance system. *Artificial Intelligence*, 28:127–162, 1986.

[16] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[17] G. Gallo, C. Gentile, D. Pretolani, and G. Rago. Max Horn SAT and the minimum cut problem in directed hypergraphs. *Mathematical Programming*, 80:213–237, 1998.

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

[19] G. Gallo and G. Rago. A hypergraph approach to logical inference for datalog formulæ. Technical Report TR-28/90, Dipartimento di Informatica, Università di Pisa, Italy, 1990.

[20] S. Gnesi, U. Montanari, and A. Martelli. Dynamic programming as graph searching: An algebraic approach. *Journal of the ACM*, 28(4):737–751, 1981.

[21] G. F. Italiano and U. Nanni. On-line maintenance of minimal directed hypergraphs. In *Italian Conference on Theoretical Computer Science*, pages 335–349, 1989.

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

[23] K. Konolige. Abductive theories in artificial intelligence. In *Principles of Knowledge Representation*, pages 129–152. CSLI Publications, 1996.

[24] X. Liu and S. A. Smolka. Simple linear-time algorithms for minimal fixed points. In *International Colloquium on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 53–65. Springer-Verlag, 1998.

[25] D. Maier and J. D. Ullman. Connections in acyclic hypergraphs. In *Symposium on Principles of Database Systems (PODS'82)*, pages 34–39. ACM Press, 1982.

[26] A. Martelli and U. Montanari. Additive and/or graphs. In *3rd International Joint Conference on Artificial Intelligence (IJCAI'73)*, pages 1–11, 1973.

[27] S. Nguyen and S. Pallottino. Hypergraphs and shortest hyperpaths. In *Combinatorial Optimization*, volume 1403 of *Lecture Notes in Mathematics*, pages 258–271. Springer-Verlag, 1986.

[28] N. J. Nilsson. *Problem solving methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.

[29] D. Pretolani. Satisfiability and hypergraphs. Technical Report TD-12/93, Dipartimento di Informatica, Università di Pisa, Italy, 1993.

[30] D. Pretolani. A directed hypergraph model for random time dependent shortest paths. *European Journal of Operational Research*, 123:315–324, 2000.

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

[32] R. E. Tarjan. Amortized computational complexity. *SIAM Journal on Algebraic and Discrete Methods*, 6(2):306–318, 1985.

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

[34] H. C. Yen. A unified approach for deciding the existence of certain Petri net paths. *Information and Computation*, 96(1):119–137, 1992.