Alcom-FT Technical Report Series
ALCOMFT-TR-01-19

# Decremental Maintenance of Minimum Rank Hyperpaths and Minimum Models of Horn Formulae*

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

February 21, 2001

**Abstract**

We present a decremental algorithm for maintaining minimum rank hyperpaths in a directed hypergraph from a source node $s$ to all other nodes, under the assumption of unit hyperarc weights. Given a hypergraph $\mathcal{H}$ with $n$ nodes and $m$ hyperarcs, the total time needed to perform a sequence of hyperarc deletions is $O(n \cdot \mathrm{Size}(\mathcal{H}))$, where $\mathrm{Size}(\mathcal{H})$ is the sum of the sizes of the hyperarcs of $\mathcal{H}$; the total space needed is $O(\mathrm{Size}(\mathcal{H}))$. In the case of integer hyperarc weights in $[1, C]$ our solution requires $O(C \cdot n \cdot \mathrm{Size}(\mathcal{H}))$ total time and $O(C + \mathrm{Size}(\mathcal{H}))$ space to handle any sequence of hyperarc deletions and weight increase operations. This improves over the recomputation from scratch after each deletion, that requires $O(m \cdot \mathrm{Size}(\mathcal{H}))$ worst case time over the sequence.

The only algorithm proposed in the literature for the fully dynamic maintenance of minimum weight hyperpaths has been proposed in a more general framework. When applied to our restricted case, it may take $O(n^2 \cdot \mathrm{Size}(\mathcal{H}))$ total time to handle a sequence of hyperarc deletions. In this case, our algorithm gives an $n$ speedup factor for the amortized time per deletion.

Our algorithm can be used to maintain the satisfiability and the minimum model of a Horn formula $\mathcal{F}$ with $n$ propositional symbols in $O(n \cdot \mathrm{Length}(\mathcal{F}))$ total time over a sequence of clause deletions. Here, $\mathrm{Length}(\mathcal{F})$ is the sum of the number of literals in the clauses of $\mathcal{F}$.

**Keywords:** Hypergraphs, dynamic algorithms, minimum rank hyperpaths, amortized complexity, Horn formulae.

*Work partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

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

‡Università di Roma “La Sapienza”, Dipartimento di Statistica, Probabilità e Statistiche Applicate, P.le Aldo Moro 8, I-00185 Roma, Italy. On leave from Dipartimento di Informatica e Sistemistica. E-mail: paolo.franciosa@uniroma1.it

§Dipartimento di Ingegneria Elettrica, Università di L’Aquila, Monteluco di Roio I-67040 L’Aquila, Italy.

# 1 Introduction

Directed hypergraphs are a non trivial generalization of directed graphs, where arcs (called *hyperarcs*) go from a set of nodes, the *tail*, to a single node, the *head* (see e.g., [2, 14]). Directed hypergraphs play an important role in several application areas as, for instance, functional dependency manipulation [6], operations research [8], transportation networks [14], Boolean Horn formulae [3, 16, 17, 22], Horn formulae with uncertainty [7], Datalog formulae [15], Petri nets [1].

The concept of hyperpath on a directed hypergraph generalizes the notion of path on a directed graph: intuitively, there is a hyperpath from a source node $s$ to a target node $t$ if there exists a hyperarc $e$ entering $t$, and there are hyperpaths from $s$ to all the nodes in the tail of $e$. Finding hyperpaths is a basic problem occurring in all of the application areas mentioned above.

Several definitions of weighted hypergraphs have been provided, by defining functions that assign weights to hyperpaths. Unlike the case of graphs, where shortest paths can be computed in polynomial time, on directed hypergraphs the problem of finding minimum weight hyperpaths is NP-hard for some weight functions, while it is polynomial time computable for others. The reader can refer to [5] for a comprehensive description of weight functions and the complexity of various traversal problems.

Among the various weight functions, we take into consideration the *rank*, formerly defined in [18], and called *distance* in [14], that extends the usual concept of distance from directed graphs to hypergraphs. Minimum rank hyperpaths can be computed in polynomial time [5]. Noteworthy applications of the rank function can be found in the Generalized Grammar Problem [20], where the rank corresponds to the height of a derivation tree.

In several cases, it is required to maintain hyperpaths in a dynamic setting, that is, while modifying the hypergraph, along the same research line that has been developed for graphs (see [11] for a survey). 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 graph problems is among fully-dynamic ones, where insertions and deletions of arcs can be intermixed, and semi-dynamic ones (incremental or decremental), where only insertions or only deletions are allowed, respectively.

Previous results on the dynamic maintenance of different kinds of hyperpaths have been proposed in [3, 4, 7, 18, 23]. In [4] the notion of closure of a node (i.e., the set of nodes reachable from that node) is introduced, and incremental algorithms are given to efficiently maintain the closure. In [18], algorithms are given to maintain the minimum rank hyperpaths under a sequence of hyperarc insertions in overall $O(n \cdot \mathrm{Size}(\mathcal{H}))$ time, under the assumption of unit hyperarc weights. In [23] different techniques are given for the fully-dynamic setting.

**Results of the paper**. In this paper we present a decremental algorithm for the maintenance of minimum rank hyperpaths from a source node to all other nodes, under the assumption of unit hyperarc weights. Given a hypergraph $\mathcal{H}$ with $n$ nodes and $m$ hyperarcs, the total time needed to handle any sequence of hyperarc deletions is $O(n \cdot \mathrm{Size}(\mathcal{H}))$, where $\mathrm{Size}(\mathcal{H})$ is the sum of the sizes of the hyperarcs of $\mathcal{H}$; the total space needed is $O(\mathrm{Size}(\mathcal{H}))$. In the case of integer hyperarc weights in $[1, C]$ our solution requires $O(C \cdot n \cdot \mathrm{Size}(\mathcal{H}))$ total time and $O(C + \mathrm{Size}(\mathcal{H}))$ space over any sequence of hyperarc deletions and weight increase operations. This improves over recomputing minimum rank hyperpaths from scratch after each operation, that would require $O(m \cdot \mathrm{Size}(\mathcal{H}))$ ($O(C \cdot m \cdot \mathrm{Size}(\mathcal{H}))$) overall time.

The only solution known in the literature for the maintenance of minimum weight hyperpaths, handling hyperarc deletions, is the fully dynamic algorithm proposed in [23]. The authors propose a Dijkstra-like procedure that is applicable to the *grammar problem*. This problem is a generalization of the problem considered in this paper, by allowing different hyperarcs to have different weight functions associated with them. The algorithm in [23] 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. The parameter $||\delta||$ represents the number of nodes interested by the operation $\delta$ plus the total number of hyperarcs incident on all these nodes, while $M$ is the time needed to compute the weight function. The authors do not provide any amortized bound (see [24] for the definition of *amortized complexity*), but a simple analysis shows that, even neglecting term $M$, their algorithm may take up to $\Theta(n^2 \cdot \mathrm{Size}(\mathcal{H}))$ total time to handle a sequence of hyperarc deletions when all the hyperarcs have unit weight and the considered weight function is the *rank*. Thus, our algorithm gives an $n$ speedup factor for the amortized time per deletion with respect to the algorithm in [23], when applied to our simpler problem.

Our solution is based on a novel technique that, after the deletion of hyperarc $\langle X, y \rangle$, explores the nodes whose rank and/or port changes, starting from node $y$ and proceeding one level at a time. Here, we say that a node is at level $i$ if its rank 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 rank increases to $i$. Conversely, classical approaches for solving shortest path problems under arc deletions usually work in two phases: first the set of nodes affected by the deletion is selected, and then some Dijkstra-like algorithm is applied to the subgraph induced by the selected nodes.

**Applications**. Hypergraphs can be used for representing Horn formulae. In fact, a Horn formula can be mapped into a hypergraph, and satisfiability of the formula is strictly related to reachability properties on that hypergraph. A linear time off-line algorithm for Horn satisfiability is given in [10], while in [3] an incremental algorithm is presented, which maintains satisfiability and the minimum model in $O(n \cdot \mathrm{Length}(\mathcal{F}))$ total time over a sequence of clause insertions, where $\mathrm{Length}(\mathcal{F})$ is the sum of the number of literals in the clauses of $\mathcal{F}$. The incremental algorithm is extended in [7] to the case of Horn formulae with uncertainty.

Using the algorithm presented in this paper, we can also maintain satisfiability and the minimum model of a Horn formula $\mathcal{F}$ in $O(n \cdot \mathrm{Length}(\mathcal{F}))$ total time over any sequence of clause deletions. Actually, the algorithm does more, since for each propositional symbol in the formula it also maintains its minimum depth proof.

Finding hyperpaths also has applications in Assumption-based Truth Maintenance

Systems ([19, 21]), where a hyperpath can be used to represent what is called an *environment*, i.e., a set of facts that explains an observation. In this framework, maintaining minimum hyperpaths under deletions of hyperarcs corresponds to maintaining a set of "small" explanations of all observations under the elimination of either an hypothesis or a clause in the background theory.

**Structure of the paper**. After giving the basic definitions concerning directed hypergraphs in Section 2, we describe our algorithm and give its correctnes and complexity proofs in Section 3. In Section 4, we present the application to Horn formulae. Finally, in Section 5 some future research directions are addressed.

# 2 Directed Hypergraphs

The following definitions concerning directed hypergraphs are from [2], and are consistent with the more general definitions given in [14].

A *directed hypergraph* $\mathcal{H}$ is a pair $\langle N, A\rangle$, where $N$ is a non empty set of *nodes* and $A$ is a set of 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 the hyperarc $e$, and are denoted by $\mathrm{Tail}(e)$ and $\mathrm{Head}(e)$, respectively.

The *forward star* of a node $v$ is the set of hyperarcs $\mathrm{FStar}(v) = \{e \in A : v \in \mathrm{Tail}(e)\}$, and the *backward star* of a node $v$ is the set of hyperarcs $\mathrm{BStar}(v) = \{e \in A : v = \mathrm{Head}(e)\}$.

The *size* of a hypergraph $\mathcal{H}$ is defined as $\mathrm{Size}(\mathcal{H}) = \sum_{e\in A} |\mathrm{Tail}(e)|$. The following different notion of size is also used in the literature: a "compact" hypergraph $\mathcal{H}^c$ is derived from $\mathcal{H}$ where a set of $p$ hyperarcs having the same tail $T$ is represented by a single hyperarc from $T$ to a dummy node $c$, plus $p$ hyperarcs from $c$ to the original heads; the size of this hypergraph is denoted by $\mathrm{Minsize}(\mathcal{H}) = \mathrm{Size}(\mathcal{H}^c)$.

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* from node $s$ to node $t$ on $\mathcal{H}$ is a subhypergraph $\Pi_{s,t} = \left\langle N_{\Pi_{s,t}}, A_{\Pi_{s,t}}\right\rangle$ of $\mathcal{H}$ having the property that its $k$ hyperarcs can be totally ordered in a sequence $\{e_1, \ldots, e_k\}$ satisfying the following conditions:

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

This concept generalizes the notion of simple path on a directed graph. A node $t$ is said to be *reachable* from node $s$ on $\mathcal{H}$ if there exists a hyperpath $\Pi_{s,t}$ on $\mathcal{H}$. The *port* of a node $v$ on the hyperpath $\Pi_{s,t}$, denoted by $\mathrm{port}(v)$, is the hyperarc in $A_{\Pi_{s,t}}$ having $v$ as head; it is unique by condition 3. A hypergraph is shown in Figure 1; thick hyperarcs form a hyperpath from node $s$ to node $t$; hyperarc labels give an ordering satisfying conditions 1–3. For instance, $\mathrm{port}(t)$ is the hyperarc labelled 5.

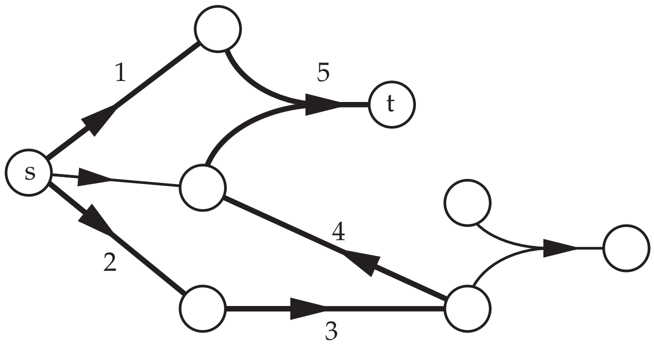


Figure 1: A directed hypergraph and (in bold) a hyperpath from node $s$ to node $t$.

A *weighted directed hypergraph* $\mathcal{H}$ is a triple $\langle N, A, W\rangle$, where $\langle N, A\rangle$ is a hypergraph and $W$ is a *weight function* which, given a hyperpath $\Pi_{s,t}$ on $\mathcal{H}$, assigns a weight $W_{\Pi_{s,t}}(v)$ to each node $v$ of $\Pi_{s,t}$; $W_{\Pi_{s,t}}(t)$ is the *weight of the hyperpath* $\Pi_{s,t}$.

There could be several hyperpaths on $\mathcal{H}$ from node $s$ to node $t$, possibly having different weights. The *minimum weight* of all hyperpaths from $s$ to $t$ is denoted by $W_{\min}(s,t)$; if no hyperpath $\Pi_{s,t}$ exists, then we assume $W_{\min}(s,t) = \infty$.

Several weight functions have been defined in the literature. In this paper we take into consideration the *rank* weight function, defined in [18] as follows:

$$\begin{aligned} W_{\Pi_{s,t}}(s) &= 0 \\ W_{\Pi_{s,t}}(v) &= \max\left\{W_{\Pi_{s,t}}(i) \mid i \in \mathrm{Tail}(\mathrm{port}(v))\right\} + w(\mathrm{port}(v)), \text{ if } v \neq s, \end{aligned}$$

where $w$ is a function that assigns a real value to each hyperarc. We denote by $\mathrm{rank}(s,v)$ the minimum weight $W_{\min}(s,v)$ of a hyperpath from the source node $s$ to node $v$; we simply call it rank of $v$ ($\mathrm{rank}(v)$) when there is no ambiguity on the source node $s$.

The rank weight function is one of the natural extensions of the concept of "distance" from directed graphs to hypergraphs; in fact, if all the tails are singletons the rank reduces to the usual notion of distance in digraphs. If a directed hypergraph represents a Petri net, the rank of node $v$ measures the length of the critical path from the source to $v$. In the case of Horn clauses, as described for example in [3], the rank of a node under unit hyperarc weights gives the depth of the minimum depth proof for the propositional symbol associated to that node.

In the following, we will use the rank function with the additional constraint that function $w$ assigns a unit value to each hyperarc.

# 3 Decremental Maintenance of the Rank

In this section we present an algorithm that maintains a set of minimum rank hyperpaths, from a given source node to all the other nodes of a directed hypergraph, during a sequence of hyperarc deletions. These hyperpaths are represented by explicitly storing the port and the rank of each node.

Let $\mathcal{H} = \langle N, A \rangle$ be a directed hypergraph with $n$ nodes, $m$ hyperarcs, and unit hyperarc weights. We denote by $\mathcal{H}'$ the hypergraph obtained from $\mathcal{H}$ after deleting hyperarc $e = \langle X, y \rangle$, and by $\mathrm{r}(v)$ ($\mathrm{r}'(v)$) the rank in $\mathcal{H}$ ($\mathcal{H}'$) of node $v$. Analogously, $\mathrm{p}(v)$ ($\mathrm{p}'(v)$) denotes the port of $v$ in $\mathcal{H}$ ($\mathcal{H}'$), while $\mathrm{L}_i$ ($\mathrm{L}'_i$) identifies level $i$ of $\mathcal{H}$ ($\mathcal{H}'$).

First of all, we observe that after deleting $e$ from $\mathcal{H}$, each hyperpath in $\mathcal{H}$ that does not contain $e$ is contained in $\mathcal{H}'$. Hence, $\mathrm{r}'(v) = \mathrm{r}(v)$ for each node $v$ whose minimum rank hyperpath in $\mathcal{H}$ does not contain $e$.

As a consequence of the above observation, we concentrate on the set of nodes in $\mathcal{H}$ whose current minimum rank hyperpath from $s$ contains $e$. Given one of these nodes $v$, either $\mathrm{r}'(v) = \mathrm{r}(v)$, by exploiting a different hyperpath, or $\mathrm{r}'(v) > \mathrm{r}(v)$ (possibly $\mathrm{r}'(v) = +\infty$, if $v$ becomes unreachable).

The hypergraph $\mathcal{H}$ is represented by associating to each node $v$ two simple lists containing all hyperarcs in $\mathrm{BStar}(v)$ and $\mathrm{FStar}(v)$. The minimum rank hyperpaths are represented as follows. For each node $v$, we store:

- $\mathrm{rank}(v)$, that coincides with $\mathrm{r}(v)$ ($\mathrm{r}'(v)$) before (after) processing the deletion;
- $\mathrm{port}(v)$, that coincides with $\mathrm{p}(v)$ ($\mathrm{p}'(v)$) before (after) processing the deletion.

In addition, for each hyperarc $e$, we store the value $\mathrm{Trank}(e) = \max\{\mathrm{rank}(v) \mid v \in \mathrm{Tail}(e)\} + 1$. In order to keep the space occupancy within $O(\mathrm{Size}(\mathcal{H}))$, we explicitly represent each hyperarc in $\mathcal{H}$ only once; all the occurrences of hyperarcs in the above data structures are implemented as references.

The algorithm we propose for handling hyperarc deletions is given in Figure 2 and is called `HDelete`. It explores the set of nodes whose rank and/or port changes under the deletion of hyperarc $e = \langle X, y \rangle$, and builds the new minimum rank hyperpaths one level at a time, as follows. The $i$-th iteration of the for loop at Line 4 identifies the set of nodes in $\mathrm{L}'_i$, whose rank and/or port changes due to the deletion, by selecting all nodes $v \in \mathrm{L}_i$ for which $\mathrm{r}'(v) = \mathrm{r}(v)$ and $\mathrm{p}'(v) \neq \mathrm{p}(v)$, and nodes $v \in \mathrm{L}_j$, $j < i$, such that $\mathrm{r}'(v) > \mathrm{r}(v)$ and $\mathrm{r}'(v) = i$.

The inspected nodes are temporarily stored in an array of sets of nodes, denoted as Newlevel. A node $v$ is put in $\mathrm{Newlevel}(i)$ if and only if $\mathrm{r}'(v)$ is known to be at least $i$. Item $\mathrm{Newlevel}(n+1)$ temporarily contains the nodes that are found to be unreachable from $s$ after the deletion.

The visit starts from node $y$ and is pruned any time a node is found whose rank does not increase after the deletion. When a node $v \in \mathrm{Newlevel}(i)$ is examined, $\mathrm{BStar}(v)$ is scanned looking for the hyperarc having minimum Trank (Line 7). If there is a hyperarc $h \in \mathrm{BStar}(v)$ having $\mathrm{Trank}(h) = i$, then we set $\mathrm{port}(v)$ to $h$. Otherwise, if all hyperarcs $h \in \mathrm{BStar}(v)$ have $\mathrm{Trank}(h) > i$, then $\mathrm{r}'(v) > i$. In this case, we check whether this causes any increase in the Trank of any hyperarc $f \in \mathrm{FStar}(v)$, and eventually update this value. In case $f$ is contained in some minimum weight hyperpath (i.e., $f = \mathrm{port}(\mathrm{Head}(f))$), we update $\mathrm{Trank}(f)$ and put $\mathrm{Head}(f)$ in $\mathrm{Newlevel}(\mathrm{rank}(\mathrm{Head}(f)))$, since $\mathrm{r}'(\mathrm{Head}(f))$ will be at least $\mathrm{rank}(\mathrm{Head}(f))$.

```
Procedure HDelete(e = ⟨X, y⟩: hyperarc)
begin
    remove e from the hypergraph
    if port(y) = e then
        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
                    port(v) ← h
                else
                    port(v) ← NIL
                    if BStar(v) = ∅ then
                        rank(v) ← n + 1
                    else
                        MinTrank ← min {Trank(h) | h ∈ BStar(v)}
                        rank(v) ← min {n + 1, MinTrank}
                    endif
                    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 = port(z) and z ∉ Newlevel(rank(z)) then
                                insert z into Newlevel(rank(z))
                            endif
                        endif
                    endfor
                endif
            endfor
        endfor
        foreach v ∈ Newlevel(n + 1) do
            delete v from Newlevel(n + 1)
            rank(v) ← +∞
            port(v) ← NIL
            foreach h ∈ FStar(v) do
                Trank(h) ← +∞
            endfor
        endfor
    endif
end
```

Figure 2: Procedure HDelete.

**Theorem 3.1** *Procedure `HDelete` correctly updates values of rank($v$) and port($v$), for each node $v$, under the deletion of a hyperarc.*

**Proof.** We prove the theorem by showing that, if hyperarc $\langle X, y\rangle$ is deleted from a hypergraph $\mathcal{H} = \langle N, A\rangle$, then upon termination of `HDelete`($\langle X, y\rangle$), $\mathrm{rank}(v) = \mathrm{r}'(v)$ and $\mathrm{port}(v) = \mathrm{p}'(v)$, for each $v \in N$. We assume $\mathrm{port}(y) = \mathrm{p}(y) = \langle X, y\rangle$, otherwise minimum rank hyperpaths do not change.

The following properties hold during the execution of Procedure `HDelete`:

***P1.*** *For each hyperarc $h$, $Trank(h) = \max\{rank(v) \mid v \in Tail(h)\} + 1$.*
It is sufficient to observe that, any time $\mathrm{rank}(v)$ is modified for some node $v$ (Lines 13, 16 and 33), $\mathrm{Trank}(f)$ is modified accordingly for any $f \in \mathrm{FStar}(v)$ (Lines 21 and 36).

***P2.*** *If $v \in Newlevel(j)$ for some $j$, then $rank(v) = j$.*
This is proved by inspecting Lines 18 and 24. Note that, when the value of $\mathrm{rank}(v)$ is modified at Line 13 or 16, node $v$ has been already extracted from some set Newlevel at Line 6 and inserted into $\mathrm{Newlevel}(\mathrm{rank}(v))$ at line 18.

***P3.*** *For each node $v$, $rank(v)$ can only increase during the execution of Procedure `HDelete`.*
Only nodes in Newlevel can change the rank. Let $v$ be a node in $\mathrm{Newlevel}(i)$, for some $\mathrm{r}(y) \leq i \leq n$, considered in an iteration of the loop at Line 5. Due to Property *P2*, $\mathrm{rank}(v) = i$. If $v$ gets rank $n + 1$ (Line 13), then Property *P3* trivially holds. Otherwise, node $v$ can get a new rank value at Line 16. This can only happen if the test at Line 8 fails, thus implying that for each hyperarc $f \in \mathrm{BStar}(v)$, $\mathrm{Trank}(f) > i$, hence $\mathrm{rank}(v) > i$.

***P4.*** *For each node $v$, $r(v) \leq rank(v) \leq r'(v)$.*
Since after processing the previous hyperarc deletion we had $\mathrm{rank}(v) = \mathrm{r}(v)$, then Property *P3* implies that $\mathrm{r}(v) \leq \mathrm{rank}(v)$.
To prove that $\mathrm{rank}(v) \leq \mathrm{r}'(v)$, let us consider by absurd the node $x$ with minimum $\mathrm{r}'(x)$ such that $\mathrm{rank}(x) > \mathrm{r}'(x)$. It follows that, for each $z \in \mathrm{Tail}(\mathrm{p}'(x))$, $\mathrm{rank}(z) \leq \mathrm{r}'(z) < \mathrm{r}'(x)$.
The current value of $\mathrm{rank}(x)$ has been set either at Line 13 or at Line 16. In the former case, $\mathrm{BStar}(x) = \emptyset$, hence $\mathrm{r}'(x) = +\infty$, thus contradicting that $\mathrm{rank}(x) > \mathrm{r}'(x)$. In the latter case, due to Property *P1*, for each hyperarc $h \in \mathrm{BStar}(x)$, $\mathrm{Trank}(h) > \mathrm{r}'(x)$ and there is at least one node $z \in \mathrm{Tail}(h)$ having $\mathrm{rank}(z) \geq \mathrm{r}'(x)$, which contradicts that for each $z \in \mathrm{Tail}(\mathrm{p}'(x))$, $\mathrm{rank}(z) < \mathrm{r}'(x)$.

***P5.*** *If $(r(v) \neq r'(v)) \vee (p(v) \neq p'(v))$ and $rank(v) = k$, then $v \in Newlevel(k)$.*
If $(\mathrm{r}(v) \neq \mathrm{r}'(v)) \vee (\mathrm{p}(v) \neq \mathrm{p}'(v))$ then the rank of at least one node in $\mathrm{Tail}(\mathrm{port}(v))$ has been increased at least once during the previous iterations of the loop at Line 4. This implies that hyperarc $\mathrm{port}(v)$ has been visited at Line 19, and $v$ has been inserted into $\mathrm{Newlevel}(\mathrm{rank}(v))$ at Line 24. It is possible that $\mathrm{rank}(v)$ has been increased again, say from $x$ to $x'$, during the previous iterations, but in that case $v$ had been moved from $\mathrm{Newlevel}(x)$ (Line 6) to $\mathrm{Newlevel}(x')$ (Line 18). So, since $\mathrm{rank}(v)$ is now $k$, $v \in \mathrm{Newlevel}(k)$.

We show that at the end of Procedure `HDelete` $\mathrm{rank}(v) = \mathrm{r}'(v)$, for each node $v$. The proof proceeds by induction on the rank values, by studying the behaviour of a single iteration of the loop at Line 4. Let $k$ be the current value of variable $i$, $k \geq \mathrm{r}(y)$. We show that, during the $k$-th iteration, if $\mathrm{rank}(v) = \mathrm{r}'(v)$ for all nodes having $\mathrm{rank}(v) < k$ before Line 5, then $\mathrm{rank}(v) = \mathrm{r}'(v)$ for all nodes having $\mathrm{rank}(v) \leq k$ after Line 29.

**Basic step:** before starting the loop at Line 4, $\mathrm{r}'(v) = \mathrm{r}(v)$, for each node $v$ having $\mathrm{r}(v) < \mathrm{r}(y)$. Hence, by Property *P4*, $\mathrm{rank}(v) = \mathrm{r}'(v)$.

**Inductive step:** We must show that, after the $k$-th iteration, if $\mathrm{rank}(v) = k$ then $\mathrm{rank}(v) = \mathrm{r}'(v)$.

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

Nodes in $\mathrm{Newlevel}(k)$ are examined in the for loop at Line 5. Given a node $v \in \mathrm{Newlevel}(k)$, we distinguish two cases: either there is a hyperarc $h \in \mathrm{BStar}(v)$ such that $\mathrm{Trank}(h) = k$ or there is not such a hyperarc.

If $h$ exists (Line 9), then, thanks to Property *P1*, we know that $\max\{\mathrm{rank}(w) \mid w \in \mathrm{Tail}(h)\} = k - 1$. By the inductive hypothesis $\mathrm{rank}(w) = \mathrm{r}'(w)$ for all $w \in \mathrm{Tail}(h)$, hence $\mathrm{Trank}(h) = \max\{\mathrm{r}'(w) \mid w \in \mathrm{Tail}(h)\} + 1 = k$ and $\mathrm{r}'(v) = \mathrm{Trank}(h) = k$.

Otherwise, $\mathrm{rank}(v)$ is assigned a value greater than $k$ (Line 13 or 16), $v$ is inserted into $\mathrm{Newlevel}(\mathrm{rank}(k))$ (Line 18), and $v$ will be considered in a next iteration.

Note that, if a node gets a rank value greater than $n$, then it is unreachable from the source node. Unreachable nodes are put in the set $\mathrm{Newlevel}(n + 1)$ and considered at Line 31.

A node $v$ gets its correct port any time a hyperarc $h \in \mathrm{BStar}(v)$ is found having $\mathrm{Trank}(h) = \mathrm{r}'(v)$ (Line 9). □

Both the rank and the port of each node are explicitly updated during a sequence of hyperarc deletions, thus a query asking for the rank of a node can be answered in constant time. The minimum rank hyperpath from $s$ to a node $v$ can be retrieved in $O(\ell)$ time, where $\ell$ is the number of hyperarcs in the retrieved hyperpath, by a backward traversal of port pointers starting from $v$. The initial port and rank values for each node can be easily computed in $O(\mathrm{Size}(\mathcal{H}))$ worst case time. In the next theorem we show the time complexity of our algorithm.

**Theorem 3.2** *Let $\mathcal{H}$ be a directed hypergraph with $n$ nodes and $m$ unit weight hyperarcs. The total worst case time needed by Procedure* ***`HDelete`*** *for handling any sequence of hyperarc deletions is $O(n \cdot \mathit{Size}(\mathcal{H}))$.*

**Proof.** Algorithm `HDelete` examines nodes in $\mathrm{Newlevel}(i)$, for $i = \mathrm{r}(y), \ldots, n$ (loops at Lines 4 and 5). Sets $\mathrm{Newlevel}(\cdot)$ are implemented by means of an array of $n + 1$ lists, in which the $i$-th list contains all nodes in $\mathrm{Newlevel}(i)$. Insertion of a node can be done

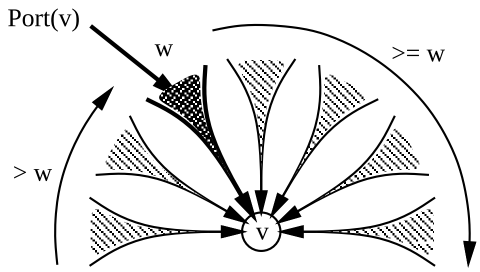


Figure 3: Scanning BStar($v$).

in constant time. Node deletions are also performed in constant time per node, since all nodes in Newlevel($i$) are sequentially deleted in the same loop.

Any time a node $v$ is extracted from Newlevel($i$), either there is a hyperarc $e \in$ BStar($v$) such that Trank($e$) $= i$ (Line 7), or rank($v$) gets a value strictly greater than $i$ (Line 13 or 16). In the latter case, $v$ is inserted into Newlevel(rank($v$)) at Line 18.

The rank of a node $v$ can only increase as a consequence of a hyperarc deletion (see Property *P3* in the proof of Theorem 3.1). Since the rank of $v$ can be at most $n$, then rank($v$) can increase at most $n$ times during a single execution of `HDelete`. Moreover, the value r$'$($v$) at the end of a deletion coincides with r($v$) at the beginning of the next deletion. This implies that rank($v$) can increase at most $n$ times during an arbitrary sequence of hyperarc deletions.

Any time rank($v$) increases, all the hyperarcs in FStar($v$) are scanned. By the above discussion, the loop at Line 19 requires an overall $O(n \cdot \sum_{v \in N} |\text{FStar}(v)|) = O(n \cdot \text{Size}(\mathcal{H}))$ worst case time during any sequence of hyperarc deletions.

We show now that the overall time spent at Lines 7 and 15 is $O(n \cdot m)$. We implement BStar($v$) as a simple list, in which hyperarcs are stored in any fixed order.

If port($v$) $\neq$ NIL, we assume that all hyperarcs that precede port($v$) in BStar($v$) have rank strictly greater than Trank(port($v$)), as depicted in Figure 3. Thus, when looking for a hyperarc at Line 7, we can sequentially scan BStar($v$) starting from port($v$) or, if port($v$) = NIL, we restart scanning from the beginning of BStar($v$). Since port($v$) becomes NIL only when r$'$($v$) $>$ r($v$) (Line 11), at most one sequential scan of BStar($v$) is performed for each different value of rank($v$). Hence, Line 7 requires $O(|\text{BStar}(v)|)$ worst case time for each possible value of rank($v$), giving overall $O(n \cdot \sum_{v \in N} |\text{BStar}(v)|) = O(n \cdot m)$ worst case time during any sequence of hyperarc deletions.

A similar argument applies to Line 15, that requires a whole scan any time rank($v$) increases, giving $O(n \cdot \sum_{v \in N} |\text{BStar}(v)|) = O(n \cdot m)$ worst case time during any sequence of hyperarc deletions.

Since $m \leq \text{Size}(\mathcal{H})$, it follows that the overall cost of Procedure `HDelete` is $O(n \cdot \text{Size}(\mathcal{H}))$. □

Concerning the space requirements, in addition to sets BStar($v$) and 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 Newlevel at a time. It follows that the total worst case space needed by Procedure `HDelete` is $O(\text{Size}(\mathcal{H}))$.

It is straightforward to characterize the complexity of our algorithm in terms of Minsize$(\mathcal{H})$. Since the algorithm is decremental, it suffices to assume that the hypergraph has been compacted into $\mathcal{H}^c$ at the beginning. In this way, the total time needed for any sequence of hyperarc deletions is $O(n \cdot \text{Minsize}(\mathcal{H}))$, using $O(\text{Minsize}(\mathcal{H}))$ space.

If the algorithm described in this paper is applied to a hypergraph whose hyperarcs have singleton tails (which can be seen as a directed graph) the overall worst case time needed to manage any sequence of deletions is $O(nm)$, giving the same performances as in [12, 13].

The algorithm described in this section can be extended to deal with integer hyperarc weights in the range $[1, C]$ by using standard techniques (see [12]). In this case, since ranks are now bounded by $C \cdot n$, the following theorem holds.

**Theorem 3.3** *Let $\mathcal{H}$ be a directed hypergraph with $n$ nodes and $m$ hyperarcs having integer weights in the range $[1, C]$. Any sequence of hyperarc deletions and weight increase operations can be handled in $O(C \cdot n \cdot \mathit{Size}(\mathcal{H}))$ worst case time and $O(C + \mathit{Size}(\mathcal{H}))$ space.*

# 4 An application to propositional Horn formulae

A *propositional Horn formula* (in the following, simply, Horn formula) is a conjunction of propositional Horn clauses, where a *Horn clause* is a disjunction of literals with at most one positive literal. Using the implication operator $\leftarrow$, a Horn clause has one of the following structures:

$$\begin{array}{ccl} Q & \leftarrow & P_1 \wedge \ldots \wedge P_i \\ Q & \leftarrow & \\ & \leftarrow & P_1 \wedge \ldots \wedge P_i \end{array}$$

where $Q, P_1, \ldots, P_i$ are propositional symbols, $P_1, \ldots, P_i$ are the *premises* and $Q$ is the *conclusion* of the clause.

Given a Horn formula and a propositional symbol $P$, a *proof* of $P$ is a set of clauses that imply $P$ ([3]). The *depth* of a proof of $P$ can be intuitively defined as the maximum length of a chain of implications. The depth is one of the possible measures of the "difficulty" of a proof.

In [7] it is shown how a propositional Horn formula can be represented by a hypergraph, whose nodes correspond to propositional symbols (plus two nodes representing the constants True and False), and whose hyperarcs correspond to clauses.

If we represent a Horn formula by a hypergraph, we can see that the minimum depth of a proof of a propositional symbol is exactly the rank of the corresponding node(see [7]). Hence, our algorithm maintains a proof of minimum depth for each propositional symbol occurring in the formula.

**Theorem 4.1** *The total worst case time needed to maintain satisfiability and the minimum model of a Horn formula $\mathcal{F}$ over any sequence of clause deletions is $O(n \cdot \mathit{Length}(\mathcal{F}))$. The total space needed is $O(n + \mathit{Length}(\mathcal{F}))$.*

If a formula $\mathcal{F}$ is unsatisfiable, rank(FALSE) is the minimum depth of a proof showing the inconsistency of $\mathcal{F}$.

Note that, an alternative measure of the complexity of a proof is its "length", i.e. the number of clauses in the proof. This corresponds, on the hypergraph representing the formula, to the number of hyperarcs in a hyperpath. By applying a result in [5], it is easy to see that to determine the shortest proof of a propositional symbol in a Horn formula is an NP-hard problem. A weight function giving this measure is the "cost" with unit hyperarc weights. Finding a hyperpath with minimum cost is NP-hard.

## 5 Future work

At least two main issues deserve further investigation. First, it would be interesting to extend the results in this paper to different weight functions that can be computed in polynomial time, for instance the *gap* or the *bottleneck* [5]. Along this line, an important extension would be to maintain weight functions which are not strictly increasing along a hyperpath. Among other applications, this would allow us to deal with Horn formulae with uncertainty.

A second line of research would be to use some of the techniques presented here for a fully-dynamic algorithm. 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 insertions and deletions, we cannot exploit the monotonicity property of rank values.

Nevertheless, due to the fact that this algorithm essentially visits only nodes whose rank changes, and their incoming and outgoing hyperarcs, we expect it to be very fast in practice.

## Acknowledgments

We thank Francesco Donini for useful discussions on applications of hypergraphs to logic. We also wish to thank the anonymous referees for their careful revision of the paper and their comments, that allowed us to significantly improve the quality of the paper. We are specially grateful to one of them, who pointed out a mistake in a preliminary version of the paper.

## References

[1] Alimonti, P., Feuerstein, E., Nanni, U.: Linear time algorithms for liveness and boundedness in conflict-free Petri nets. Latin American Symposium on Theoretical Informatics (LATIN'92). *Lecture Notes in Computer Science* vol. 583, 1–14, 1992.

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

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

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

[5] Ausiello, G., Italiano, G. F., Nanni, U.: Hypergraphs traversal revisited: cost measures and dynamic algorithms. Mathematical Foundations of Computer Science (MFCS'98). *Lecture Notes in Computer Science* vol. 1450, 1–16, 1998.

[6] Ausiello, G., D'Atri, A., Saccà, D.: Graph algorithms for functional dependency manipulation. J. ACM, **30** (1983), 752–766.

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

[8] Cambini, R., Gallo, G., Scutellà, M. G.: Flows on hypergraphs. Mathematical Programming **78** (1997), 195–217.

[9] Cormen, T. H., Leiserson, C. E., Rivest, R. L.: *Introduction to Algorithms*. MIT Press, Cambridge, MA (1990).

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

[11] Eppstein, D., Italiano, G. F., and Galil, Z.: Dynamic graph algorithms. Handbook of Algorithms and Theory of Computation, CRC Press, 1999.

[12] Franciosa, P. G., Frigioni, D., Giaccio, R.: Semi-dynamic shortest paths and breadth first search in digraphs. Symposium on Theoretical Aspects of Computer Science (STACS'97). *Lecture Notes in Computer Science* vol. 1200, 33–46, 1997.

[13] Franciosa, P. G., Frigioni, D., Giaccio, R.: Semi-dynamic breadth-first search in digraphs. Theoretical Computer Science **250** (2001), 201–217.

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

[15] Gallo, G., Rago, G.: A hypergraph approach to logical inference for Datalog formulae. Technical Report TR-28/90, Università degli Studi di Pisa, Dipartimento di Informatica, Italy, 1990.

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

[17] Liu, X. L., Smolka, S. A.: Simple linear-time algorithms for minimal fixed points. International Colloquium on Automata Languages and Programming (ICALP'98). *Lecture Notes in Computer Science* vol. 1443, 53–65, 1998.

[18] Italiano, G. F., Nanni, U.: On line maintenance of minimal directed hypergraphs. Proceedings 3rd Italian Conference on Theoretical Computer Science, 335–349, World Scientific Co., 1989.

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

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

[21] Konolige, K.: Abductive theory in artificial intelligence. In G. Brewka, editor, *Principles of Knowledge Representation*, pages 129–152. CSLI Publications, Stanford (CA), USA, 1996.

[22] Pretolani, D.: Satisfiability and hypergraphs. TD 12/93, Dipartimento di Informatica, Università di Pisa (1993).

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

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