Directed hypergraphs and applications*

Giorgio Gallo, Giustino Longo and Stefano Pallottino

Dipartimento di Informatica, Università di Pisa, Pisa, Italy

Sang Nguyen

Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, Qué., Canada

Abstract

Gallo, G., G. Longo, S. Palottino and S. Nguyen, Directed hypergraphs and applications, Discrete Applied Mathematics 42 (1993) 177–201.

We deal with directed hypergraphs as a tool to model and solve some classes of problems arising in operations research and in computer science. Concepts such as connectivity, paths and cuts are defined. An extension of the main duality results to a special class of hypergraphs is presented. Algorithms to perform visits of hypergraphs and to find optimal paths are studied in detail. Some applications arising in propositional logic, And-Or graphs, relational databases and transportation analysis are presented.

1. Introduction

Hypergraphs, a generalization of graphs, have been widely and deeply studied in [5–7], and quite often have proved to be a successful tool to represent and model concepts and structures in various areas of computer science and discrete mathematics.

Here we deal with directed hypergraphs. Sometimes with different names such as "labelled graphs" and "And-Or graphs", directed hypergraphs have been intro-

duced in the literature as a way to deal with particular problems arising in computer science and in combinatorial optimization (see, for example, [8, 11, 16, 22, 24, 25, 29-31, 34]).

Directed hypergraphs have also been explicitly introduced in [12, 23, 33]. In addition, particular instances of directed hypergraphs can be found in [2-4, 10, 14, 18, 28].

The remaining of the paper is organized as follows. After a general presentation of directed hypergraphs, Section 3 introduces the concept of connection in hypergraphs and defines paths and hyperpaths. Section 4 introduces cuts and cutsets in relation to connectivity. Sections 5 and 6 develop algorithms to visit hypergraphs and to solve some classes of minimum path problems defined on hypergraphs. Several applications are studied in Section 7. In particular, it is shown that hypergraph concepts and algorithms are elegant and powerful tools to model and to solve problems which arise in areas such as propositional logic [10, 14], And-Or graphs [16, 22, 25, 31], databases [1-3, 24, 34], and urban transportation [27-29].

2. Directed hypergraphs

A *hypergraph* is a pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of *vertices* (or *nodes*) and $\mathcal{E} = \{E_1, E_2, \ldots, E_m\}$, with $E_i \subseteq \mathcal{V}$ for $i = 1, \ldots, m$, is the set of *hyperedges*. Clearly, when $|E_i| = 2$, $i = 1, \ldots, m$, the hypergraph is a standard graph.

While the size of a standard graph is uniquely defined by n and m, the size of a hypergraph depends also on the cardinality of its hyperedges; we define the *size* of $\mathcal{H}$ as the sum of the cardinalities of its hyperedges:

$$size(\mathcal{H}) = \sum_{E_i \in \mathcal{E}} |E_i|.$$

It is worth noting that there is a one-to-one correspondence between hypergraphs and Boolean matrices. Indeed, any $n \times m$ matrix $A = [a_{ij}]$ such that $a_{ij} \in \{0, 1\}$ may be considered as the *incidence matrix* of a hypergraph $\mathcal{H}$ where each row i is associated with a vertex v_i and each column j with a hyperedge E_j.

A directed hyperedge or *hyperarc* is an ordered pair, $E = (X, Y)$, of (possibly empty) disjoint subsets of vertices; X is the *tail* of E while Y is its *head*. In the following, the tail and the head of hyperarc E will be denoted by $T(E)$ and $H(E)$, respectively.

A *directed hypergraph* is a hypergraph with directed hyperedges. In the following, directed hypergraphs will simply be called hypergraphs. An example of hypergraph is illustrated in Fig. 1. Note that hyperarc E_5 has an empty head.

As for directed graphs, the *incidence matrix* of a hypergraph $\mathcal{H}$ is an $n \times m$ matrix $[a_{ij}]$ defined as follows (see Fig. 1):

$$a_{ij} = \begin{cases} -1, & \text{if } v_i \in T(E_j), \\ 1, & \text{if } v_i \in H(E_j), \\ 0, & \text{otherwise.} \end{cases}$$

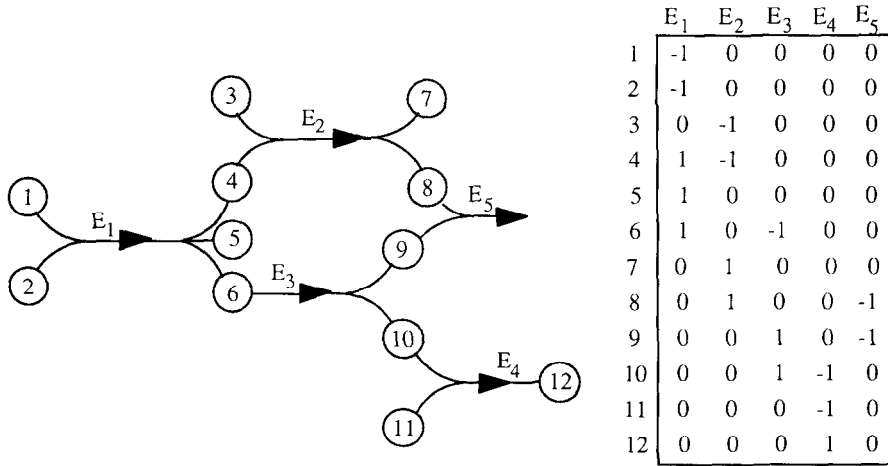|	E_1	E_2	E_3	E_4	E_5
1	-1	0	0	0	0
2	-1	0	0	0	0
3	0	-1	0	0	0
4	1	-1	0	0	0
5	1	0	0	0	0
6	1	0	-1	0	0
7	0	1	0	0	0
8	0	1	0	0	-1
9	0	0	1	0	-1
10	0	0	1	-1	0
11	0	0	0	-1	0
12	0	0	0	1	0

Fig. 1. A hypergraph and its incidence matrix.

Clearly, there is a one-to-one correspondence between hypergraphs and $(-1, 0, 1)$ matrices.

A *backward* hyperarc, or simply *B-arc*, is a hyperarc $E = (T(E), H(E))$ with $|H(E)| = 1$ (Fig. 2(a)). A *forward* hyperarc, or simply *F-arc*, is a hyperarc $E = (T(E), H(E))$ with $|T(E)| = 1$ (Fig. 2(b)).
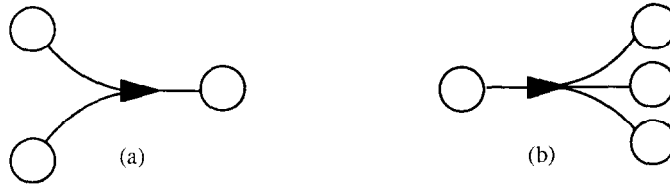
Fig. 2. (a) A B-arc. (b) An F-arc.

A *B-graph* (or B-hypergraph) is a hypergraph whose hyperarcs are B-arcs. An *F-graph* (or F-hypergraph) is a hypergraph whose hyperarcs are F-arcs. A *BF-graph* (or BF-hypergraph) is a hypergraph whose hyperarcs are either B-arcs or F-arcs.

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, we define its *symmetric image* the hypergraph $\tilde{\mathcal{H}} = (\mathcal{V}, \tilde{\mathcal{E}})$ where $\tilde{\mathcal{E}} = \{(X, Y): (Y, X) \in \mathcal{E}\}$. Note that the symmetric image of a B-graph is an F-graph, and viceversa.

Note that it is always possible to transform a general hypergraph into a BF-graph, by adding a dummy node to each hyperarc which is neither a B-arc nor an F-arc,
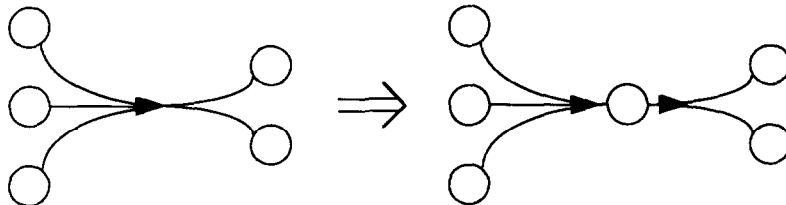
Fig. 3. Transformation of a hyperarc into a B-arc and an F-arc.

and thus replacing the hyperarc by one backward and one forward hyperarc (see Fig. 3).

Let $FS(v) = \{E \in \mathscr{E}: v \in T(E)\}$ and $BS(v) = \{E \in \mathscr{E}: v \in H(E)\}$ denote the *forward star* and the *backward star* of node v, respectively.

B-graphs and F-graphs are of particular relevance in applications. Indeed, they have been introduced many times in the literature with various names. The *labelled graphs*, used by Dowling and Gallier [10] and Gallo and Urbani [14] to represent Horn formulae, are B-graphs; B-graphs have been introduced as tools to analyze *deductive databases* [2–4, 18], and to study *Leontiev substitution matrices* and *Leontiev flow problems* [20]; F-graphs have been studied in the context of urban transit problems [28, 29] and applications of F-graphs to the analysis of And-Or graphs are reported in [12].

Torres and Aráoz [33] introduced hypergraphs and B-graphs, called directed hypergraphs and rule hypergraphs respectively, to represent deduction properties in databases as paths in hypergraphs.

3. Paths, hyperpaths and connection

A *path* P_{st} of length q, in hypergraph $\mathscr{H} = (\mathscr{V}, \mathscr{E})$, is a sequence of nodes and hyperarcs $P_{st} = (v_1 = s, E_{i_1}, v_2, E_{i_2}, \ldots, E_{i_q}, v_{q+1} = t)$, where:

$$s \in T(E_{i_1}), \quad t \in H(E_{i_q}), \quad \text{and} \quad v_j \in H(E_{i_{j-1}}) \cap T(E_{i_j}), \quad j = 2, \ldots, q.$$

Nodes s and t are the *origin* and the *destination* of P_{st}, respectively, and we say that t is *connected* to s. If $t \in T(E_{i_1})$, then P_{st} is said to be a *cycle*; this is in particular true when $t = s$. In a *simple* path all hyperarcs are distinct, and a simple path is *elementary* if all nodes $v_1, v_2, \ldots, v_{q+1}$ are distinct. Similarly we may define *simple* and *elementary cycles*. A path is said to be *cycle-free* if it does not contain any subpath which is a cycle.

In Fig. 4, node 8 is connected to node 1, while node 9 is not. The elementary path connecting 8 to 1 is drawn in thick line.

Consider a hypergraph $\mathscr{H} = (\mathscr{V}, \mathscr{E})$. A *B-path* (or *B-hyperpath*) Π_{st} is a minimal hypergraph $\mathscr{H}_\Pi = (\mathscr{V}_\Pi, \mathscr{E}_\Pi)$, such that:
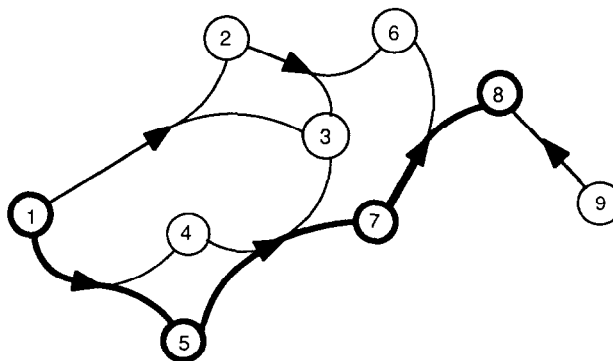

Fig. 4. A path $P_{1,8}$.

(i) $\mathcal{E}_\Pi \subseteq \mathcal{E}$;

(ii) $s, t \in \mathcal{V}_\Pi = \bigcup_{E_i \in \mathcal{E}_\Pi} E_i \subseteq \mathcal{V}$;

(iii) $x \in \mathcal{V}_\Pi \Rightarrow x$ is connected to s in $\mathcal{H}_\Pi$ by means of a cycle-free simple path.

We say that $\mathcal{H}_\Pi = (\mathcal{V}_\Pi, \mathcal{E}_\Pi)$ is an *F-path* (or *F-hyperpath*) from s to t if its symmetric image is a B-path from t to s.

A *BF-path* (or *BF-hyperpath*) from s to t is a hypergraph which is at the same time a B-path and an F-path from s to t.

Node y is *B-connected* (*F-connected*, *BF-connected*) to node x if a B-path (F-path, BF-path) Π_{xy} exists in $\mathcal{H}$.

The hypergraph in Fig. 5(a) is a B-path; note that the cycle $(4, E_4, 5, E_5, 4)$ is not contained in any simple path from node 1 to node 7. On the contrary, the hypergraph in Fig. 5(b) is not a B-path because the only path connecting node 3 to the origin contains the cycle $(2, E_3, 4, E_2, 3)$.

The following proposition trivially holds:

Proposition 3.1. *Given a B-path Π_{st} and a hyperarc $E \in \mathcal{E}_\Pi$, one has that each node $x \in \mathrm{T}(E)$ is B-connected to s.*

Given a hyperarc $E = (\mathrm{T}(E), \mathrm{H}(E))$, a *B-reduction* of E is a B-arc $E^* = (\mathrm{T}(E^*), \{v\})$ such that $\mathrm{T}(E) = \mathrm{T}(E^*)$ and $v \in \mathrm{H}(E)$.

A *B-reduction* of a hypergraph $\mathcal{H}$ is the B-graph $\mathcal{H}_B$ obtained from $\mathcal{H}$ by replacing each hyperarc by one of its B-reductions. Clearly, a hypergraph may have many B-reductions; we shall denote by $\mathcal{B}(\mathcal{H})$ the set of all the B-reductions of $\mathcal{H}$.

In an analogous way it is possible to define *F-reductions* and *BF-reductions* of hypergraphs. Note that a BF-reduction of a hypergraph is a standard digraph.

We say that node y is *super-connected* to node x in hypergraph $\mathcal{H}$ if y is B-connected to x in any B-reduction $\mathcal{H}_B$ of $\mathcal{H}$. Then to say that y is not super-connected to x we need at least one B-reduction in which y is not B-connected to x.

Note that in B-graphs the concepts of B-connection and super-connection coincide.
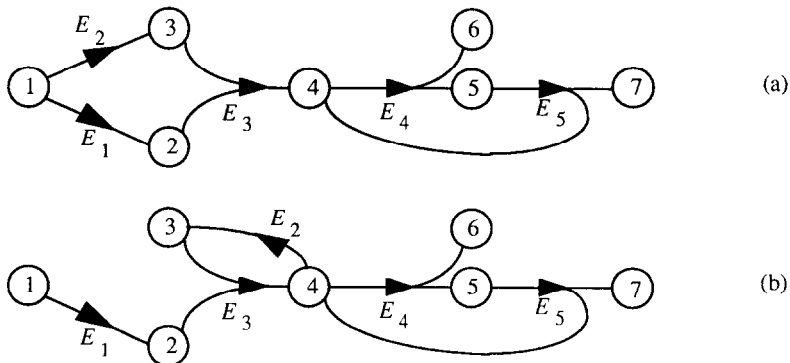

Fig. 5. (a) A B-path. (b) A B-graph which is not a B-path.

The definitions of connection introduced above can be generalized as follows: given a set S of nodes, we say that node y is B-connected (F-connected, BF-connected, super-connected) to S in the hypergraph $\mathcal{H}$ if y is B-connected (F-connected, BF-connected, super-connected) to s in the hypergraph $\mathcal{H}'$ obtained from $\mathcal{H}$ by addition of a new origin node s and an arc (s, x) for each $x \in S$. Similarly, we can define the connection of a set of nodes T to a single origin node x.

4. Cuts and cutsets

Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph and s and t be two distinguished nodes, the *source* and the *sink* respectively.

A *cut* $\mathcal{T}_{st} = (\mathcal{V}_s, \mathcal{V}_t)$ is a partition of $\mathcal{V}$ into two subsets $\mathcal{V}_s$ and $\mathcal{V}_t$ such that $s \in \mathcal{V}_s$ and $t \in \mathcal{V}_t$. Given the cut $\mathcal{T}_{st}$, its *cutset* $\mathcal{E}_{st}$ is the set of all hyperarcs E such that $T(E) \subseteq \mathcal{V}_s$ and $H(E) \subseteq \mathcal{V}_t$. Such a cutset may be empty; see for instance the cut $(\{1, 2\}, \{3, 4, 5, 6, 7\})$ in the B-graph of Fig. 5(b).

The *cardinality* of a cut is the cardinality of its cutset. In Fig. 6 three cuts are indicated; the cardinality of $\mathcal{T}_{st}^1$ is 2, while $\mathcal{T}_{st}^2$ and $\mathcal{T}_{st}^3$ have cardinality 1. Note that t is not necessarily disconnected from s by removing the hyperarcs of a cutset. For example, in Fig. 6 by removing the cutset of $\mathcal{T}_{st}^1$ we disconnect t from s, by removing the cutset of $\mathcal{T}_{st}^2$ only the B-connection of t to s is lost, while t remains both connected and B-connected to s when we remove the cutset of $\mathcal{T}_{st}^3$.

The following two theorems relate cuts to connection in hypergraphs.

Theorem 4.1. *In a B-graph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, *a cut* $\mathcal{T}_{st}$ *of cardinality* 0 *exists if and only if* t *is not B-connected to* s.

Proof. ($\Rightarrow$) Assume that a cut $\mathcal{T}_{st}$ with an empty cutset $\mathcal{E}_{st}$ exists and there is a node $v \in \mathcal{V}_t$ B-connected to s. Then a B-arc $E = (T(E), \{v\})$ must exist with the property that every node $x \in T(E)$ be B-connected to s (see Proposition 3.1). Clearly, as $\mathcal{E}_{st}$ is empty, at least one node $u \in T(E)$ must belong to $\mathcal{V}_t$. By repeating the same argument on u, we may eventually conclude that s also belongs to $\mathcal{V}_t$, which is a contradiction.
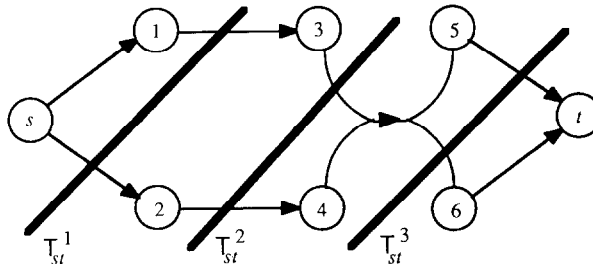

Fig. 6. Only cut $\mathcal{T}_{st}^1$ disconnects source s and sink t.

(⇐) Now assume that t is not B-connected to s. Define $\mathscr{V}_s$ as the set of all the nodes B-connected to s and $\mathscr{V}_t = \mathscr{V} \setminus \mathscr{V}_s$. $\mathscr{T}_{st}$ is necessarily a cut of cardinality 0, for the existence of a B-arc $E = (T(E), \{v\})$ in the cut, being $T(E) \subseteq \mathscr{V}_s$ and $v \in \mathscr{V}_t$, imply the B-connection of v to s. $\square$

Theorem 4.2. *In a hypergraph* $\mathscr{H} = (\mathscr{V}, \mathscr{E})$ *a cut* $\mathscr{T}_{st}$ *of cardinality 0 exists if and only if* t *is not super-connected to* s.

Proof. (⇒) Let $\mathscr{T}_{st} = (\mathscr{V}_s, \mathscr{V}_t)$ be a cut of cardinality 0. Consider the B-reduction $\mathscr{H}_B$ of $\mathscr{H}$ obtained by replacing each hyperarc E with a B-arc $(T(E), \{v\})$ with the condition that if $T(E) \subseteq \mathscr{V}_s$ then also $v \in \mathscr{V}_s$. This reduction is always possible since for any hyperarc E with $T(E) \subseteq \mathscr{V}_s$ at least one node in its head must belong to $\mathscr{V}_s$, otherwise E belongs to the cutset which, by hypothesis, is empty. By Theorem 4.1, t is not B-connected to s in $\mathscr{H}_B$ and therefore t is not super-connected to s in $\mathscr{H}$.

(⇐) If t is not super-connected to s, then a B-reduction exists such that t is not B-connected to s in it, and, by Theorem 4.1, the proof is completed. $\square$

Theorems 4.1 and 4.2 generalize to hypergraphs the property holding for standard digraphs that the removal of all the arcs of a cutset disconnects the sink from the source. Unfortunately, other nice properties do not hold for hypergraphs, even if we restrict our attention to B-graphs. In particular, it is well known that the two following equivalent facts hold for standard digraphs:

P_1: the minimum cardinality of an s–t path in a digraph is equal to the maximum number of disjoint s–t cutsets.

P_2: the minimum cardinality of an s–t cut in a digraph is equal to the maximum number of disjoint s–t paths.

Such properties do not hold for hypergraphs, although the following theorems show that they hold in a weaker form for B-graphs.

Theorem 4.3. *In a B-graph* $\mathscr{H} = (\mathscr{V}, \mathscr{E})$ *the following inequalities hold*:

$$\min\{|\Pi_{st}|: \Pi_{st} \text{ is an } s\text{–}t \text{ B-path}\} \geq \text{maximum number of disjoint } s\text{–}t \text{ cutsets}$$

$$\geq \min\{|P_{st}|: P_{st} \text{ is an } s\text{–}t \text{ path}\}.$$

Proof. The first inequality follows directly from the fact that, due to Theorem 4.1, a cutset must contain at least a B-arc of every B-path, then the number of disjoint s–t cutsets cannot exceed the cardinality of any B-path.

The second inequality can be proved as follows. Let V_k denote the set of nodes $\{i\}$ for which there exists a path P_{si} with cardinality $\leq k$. Clearly, if h is the minimum cardinality of the s–t paths, then we have $\{s\} = V_0 \subset V_1 \subset \cdots \subset V_h \subseteq \mathscr{V}$; then $(V_0, \mathscr{V} \setminus V_0), (V_1, \mathscr{V} \setminus V_1), \ldots, (V_{h-1}, \mathscr{V} \setminus V_{h-1})$ are s–t cuts with disjoint cutsets,

for no B-arc with a tail node in V_i and the head in V_j with $j \geq i + 2$ may exist, and thus, no B-arc can belong to more than one cutset. This completes the proof. □

Theorem 4.4. *In a B-graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ the following inequalities hold*:

maximum number of disjoint s–t paths

$$\geq \min\{|\mathcal{E}_{st}| : \mathcal{E}_{st} \text{ is an s–t cutset}\}$$

$$\geq \text{maximum number of disjoint s–t B-paths.}$$

Proof. Transform $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ into a standard digraph $G = (\mathcal{V}, A)$ where for each B-arc (X, y) there is a unique arc $(x, y) \in A$, with $x \in X$. The choice of $x \in X$ is arbitrary. It is easy to check that to any s–t cutset $\mathcal{E}_{st}$ in $\mathcal{H}$ corresponds an s–t cutset C_{st} in G with $|C_{st}| \geq |\mathcal{E}_{st}|$; moreover, any set of k disjoint paths in G corresponds to a set of k disjoint paths in $\mathcal{H}$, then the maximum number of disjoint paths in G is not larger than the maximum number of disjoint paths in $\mathcal{H}$. Hence, from the well-known max flow-min cut theorem for digraphs one has:

maximum number of disjoint s–t paths in $\mathcal{H}$

$$\geq \text{maximum number of disjoint s–t paths in } G$$

$$= \min\{|C_{st}| : C_{st} \text{ is an s–t cutset in } G\}$$

$$\geq \min\{|\mathcal{E}_{st}| : \mathcal{E}_{st} \text{ is an s–t cutset in } \mathcal{H}\}.$$

The second inequality follows directly from the fact that, due to Theorem 4.1, any cutset must contain one B-arc from each B-path at least, and this completes the proof. □

The following examples show that strict inequalities may hold in all cases.

In Fig. 7, a B-graph is presented for which the minimum cardinality of s–t B-paths is 5, the maximum number of disjoint s–t cutsets is only 4 and the minimum cardinality of s–t paths is 3.

In the B-graph of Fig. 8, the maximum number of disjoint s–t paths is 3, the minimum cardinality of s–t cuts is 2, and the maximum number of disjoint s–t B-paths is 1.

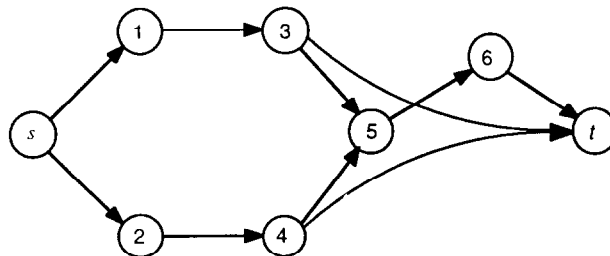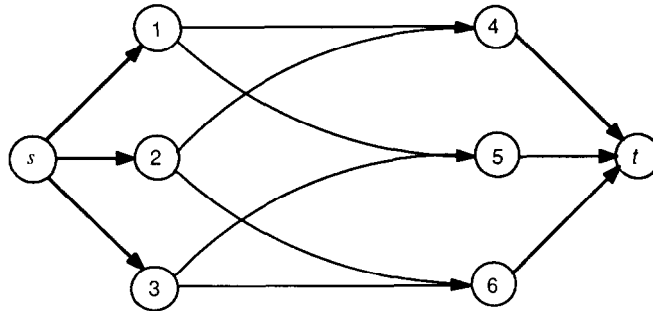Fig. 7.

Fig. 8.

In Section 7.1 we will show that the problem of finding the minimum cardinality *s–t* cut is NP-hard also in the case of B-graphs.

5. Visit of a hypergraph

Here we consider the problem of visiting a hypergraph starting from an origin node *r*, i.e., of finding all the nodes which are connected (B-connected, super-connected) to *r*.

The simplest case is to find in a hypergraph all the nodes which are connected to *r*. Procedure Visit described below finds all such nodes and returns a set of paths connecting them to *r*. Such paths, which define a *tree* rooted at *r*, are described by two predecessor functions: $Pe(E)$ points to the node $i \in T(E)$ which precedes hyper-arc E in the path, $Pv(i)$ points to the arc $E \in BS(i)$ which precedes node i in the path.

Procedure Visit($r, \mathcal{H}$).
  begin
    for each $i \in \mathcal{V}$ **do** $Pv[i] := 0$;
    for each $E_j \in \mathcal{E}$ **do** $Pe[E_j] := 0$;
    $Pv[r] := nil$; $Q := \{r\}$;
    repeat
      select and remove $i \in Q$;
      for each $E_j \in FS(i)$ **such that** $Pe[E_j] = 0$ **do**
        begin
          $Pe[E_j] := i$;
          for each $h \in H(E_j)$ **such that** $Pv[h] = 0$ **do**
            begin
              $Pv[h] := E_j$; $Q := Q \cup \{h\}$
            end-for
        end-for
    until $Q = \emptyset$
  end-procedure.

It is easy to check that Procedure Visit runs in O($size(\mathcal{H})$) time. In fact, the initialization phase runs in O($n + m$) time and, since each node is inserted and removed from the candidate set Q at most once, each hyperarc is examined only once, i.e., the first time the hyperarc is selected.

Now, consider the case of B-connection. Procedure B-Visit returns a set of B-paths containing all the nodes B-connected to r. Such B-paths define a *B-tree* rooted at r.

Notice that in this case only one predecessor function, Pv, is necessary. In fact, by the definition of B-path, if hyperarc E belongs to a B-path, then all the nodes of its tail must belong to the same B-path. Nevertheless, we shall maintain the use of the second predecessor function, Pe. Such a function defines a particular tree among the trees contained in the B-tree returned by the procedure. In connection with the function Pe, a node function, v, is introduced which, for each node h, gives the *cardinality* of the path from r to h in the tree defined by Pe and Pv. The motivation of introducing such a function will be made clear in the next section.

A counter k_j is used to provide for each hyperarc E_j the number of nodes of its tail already removed from Q. Note that functions Pe and v are not essential to the computation of the B-tree.

Procedure B-Visit(r, $\mathcal{H}$).
  begin
    for each $i \in \mathcal{V}$ **do begin** $Pv[i] := 0$; $v[i] := \infty$ **end-for**;
    for each $E_j \in \mathcal{E}$ **do** $Pe[E_j] := k_j := 0$;
    $Pv[r] := nil$; $Q := \{r\}$; $v[r] := 0$;
    repeat
      select and remove $i \in Q$;
      for each $E_j \in FS(i)$ **do**
        begin
          $k_j := k_j + 1$;
          if $k_j = |T(E_j)|$ **then**
            begin
              $Pe[E_j] := i$;
              for each $h \in H(E_j)$ such that $Pv[h] = 0$ **do**
                 begin
                    $Pv[h] := E_j$; $Q := Q \cup \{h\}$; $v[h] := v[Pe[E_j]] + 1$
                 end-for
            end-if
        end-for
      until $Q = \emptyset$
  end-procedure.

Procedure B-Visit runs in O($size(\mathcal{H})$) time. In fact, each hyperarc E is selected

at most $|T(E)|$ times and only the last time its head is examined. Moreover, each node is inserted and removed from Q at most once.

In a similar way it is possible to define a Procedure F-Visit which finds a set of F-paths which have r as terminal node and all the nodes to which r is F-connected. Note that while Procedure B-Visit starts from the origin of the B-paths, Procedure F-Visit must start from the destination of the F-paths to retain a linear time complexity.

One can also define a Procedure BF-Visit. Unfortunately, the problem of performing such a visit is not an easy one unless the hypergraph is either a B-graph or an F-graph; in the former case a BF-path is simply a B-path, while in the latter it is an F-path.

The following Procedure SuperVisit checks whether a node t is super-connected to a node s on a general hypergraph.

> **Procedure SuperVisit($s, t, \mathcal{H}$).**
>   **begin**
>   superconnected := *true*;
>   **while** superconnected **and** $\mathcal{B}(\mathcal{H}) \neq \emptyset$ **do**
>   **begin**
>   select and remove $\mathcal{H}_B$ from $\mathcal{B}(\mathcal{H})$; B-Visit($s, \mathcal{H}_B$);
>   **if** $Pv[t] = 0$ **then** superconnected := *false*
>   **end-while**
>   **end-procedure.**

Procedure SuperVisit runs in $O(size(\mathcal{H}) \cdot |\mathcal{B}(\mathcal{H})|)$ time, where $|\mathcal{B}(\mathcal{H})| = \prod_{E_j \in \mathcal{E}} |H(E_j)|$ is the number of all possible B-reductions of $\mathcal{H}$.

A quite efficient branch and bound scheme to solve this problem can be easily derived from the second algorithm for the satisfiability problem presented in [14].

6. Weighted hypergraphs

6.1. Weighting functions

A *weighted hypergraph* is one in which each hyperarc E is assigned a real weight vector $w(E)$. Depending on the particular application, the components of $w(E)$ may represent *costs, lengths, capacities*, etc. For the sake of simplicity, in the following we shall consider only scalar weights.

Given a B-path $\Pi = (\mathcal{V}_\Pi, \mathcal{E}_\Pi)$ from s to t, by *weighting function* we mean a node function W_Π which assigns weights to all its nodes depending on the weights of its hyperarcs. $W_\Pi(t)$ is the *weight* of the B-path Π under the chosen weighting function.

We shall restrict ourselves to weighting functions for which $W_\Pi(s) = 0$ and

$W_{\Pi}(y)$, for each $y \neq s$, depends only on the hyperarcs which *precede* y in the B-path Π, i.e., the hyperarcs belonging to all B-paths from s to y contained in Π.

A typical example of this kind of weighting function is *cost*, C_{Π}, defined as the sum of the weights of all the hyperarcs preceding node y in Π:

$$C_{\Pi}(s) = 0;$$

$$C_{\Pi}(y) = \sum_{E \in \{\mathscr{E}_{\Pi_{sy}} : \Pi_{sy} \subseteq \Pi\}} w(E), \quad y \in \mathcal{V}_{\Pi} \setminus \{s\}.$$

Clearly, $C_{\Pi}(t) = \sum_{E \in \mathscr{E}_{\Pi}} w(E)$ is the *cost* of Π. This function is the usual cost in the graph setting, and the problem of finding a minimum cost B-path is a natural generalization of the minimum cost path problem. Note that when the weights are all equal to 1, the cost of Π is its *cardinality*.

A relevant class of weighting functions is the one in which the weight of node y can be written as a function of both the weights of the hyperarcs entering into y and that of the nodes in their tails:

$$W_{\Pi}(y) = \min\{w(E) + F_{\Pi}(T(E)): E \in \mathscr{E}_{\Pi} \cap BS(y)\}, \quad y \in \mathcal{V}_{\Pi} \setminus \{s\}, \qquad (1)$$

where $F_{\Pi}(T(E))$ is a function of the weights of the nodes in $T(E)$:

$$F_{\Pi}(T(E)) = F(\{W_{\Pi}(x): x \in T(E)\}), \quad E \in \mathscr{E}_{\Pi}, \qquad (2)$$

where F is a nondecreasing function of $W_{\Pi}(x)$ for each $x \in T(E)$. Such weighting functions will be called *additive weighting functions*.

In the particular case of B-graphs, the B-paths have the property that there is only one B-arc E entering into every node $y \neq s$; in this case (1) becomes:

$$W_{\Pi}(y) = w(E) + F_{\Pi}(T(E)), \quad y \in \mathcal{V}_{\Pi} \setminus \{s\}. \qquad (1')$$

Two particular additive weighting functions which have been presented in the literature in the context of some relevant applications of hypergraphs are the *distance* and the *value*.

Given an s–t B-path $\Pi = (\mathcal{V}_{\Pi}, \mathscr{E}_{\Pi})$, the *distance* in Π from s to all the nodes $y \in \mathcal{V}_{\Pi} \setminus \{s\}$ which are B-connected to s, $D_{\Pi}(y)$, is defined by the following recursive equations:

$$D_{\Pi}(s) = 0;$$

$$D_{\Pi}(y) = \min\{l(E) + \max\{D_{\Pi}(x): x \in T(E)\}: E \in \mathscr{E}_{\Pi} \cap BS(y)\}, \qquad (3)$$

$$y \in \mathcal{V}_{\Pi} \setminus \{s\};$$

where $l(E)$ is the *length* of hyperarc E.

For B-graphs, equation (3) becomes:

$$D_{\Pi}(y) = l(E) + \max\{D_{\Pi}(x): x \in T(E)\}, \quad y \in \mathcal{V}_{\Pi} \setminus \{s\}. \qquad (3')$$

In the case of unit hyperarc lengths, i.e., $l(E) = 1 \ \forall E \in \mathscr{E}$, the distance will be called *depth*. Gallo and Urbani [14] have introduced the depth function on B-graphs in the context of the satisfiability analysis of propositional Horn formulae. Note

that, in this case, Procedure B-Visit, with the use of function v and a breadth-first search strategy, finds the minimum depth B-tree in $O(size(\mathcal{H}))$ time.

The *value*, V_Π, defined by Jeroslow, Martin, Rardin and Wang [20] in the context of the Leontiev flow problem for the case of B-graphs, is the solution of the following recursive equations:

$$V_\Pi(s) = 0;$$
$$V_\Pi(y) = c(E) + \sum_{x \in T(E)} a(x, E) V_\Pi(x), \quad E \in \mathcal{E}_\Pi \cap BS(y), \; y \in \mathcal{V}_\Pi \setminus \{s\}; \tag{4}$$

where $c(E)$ is the *cost* of B-arc E and, for each E and each $x \in T(E)$, $a(x, E)$ is a nonnegative real coefficient.

6.2. Minimum weight B-paths

This section addresses the problem of finding a minimum weight B-path in a weighted hypergraph. Such a problem can be viewed as a natural generalization of the shortest path problem for standard digraphs.

Unfortunately, at least in general, the minimum weight B-path problem on hypergraphs is NP-hard. In fact, Italiano and Nanni [18] have proved that the particular problem of finding minimum cardinality B-paths in a B-graph is NP-hard. Nevertheless, many particular cases exist for which the problem is easy to solve. One example is when the weighting functions are additive, and this is exactly the case of the standard shortest path problem in digraphs.

From now on, we shall restrict ourselves to the case of additive weighting functions. Furthermore, we shall assume throughout that arc weights are nonnegative and that all cycles are nondecreasing. A *nondecreasing cycle* is a cycle $C = \{v_1, E_1, v_2, E_2, \ldots, v_r, E_r, v_1\}$ such that, for any real z:

$$W(E_r) + F_{(v_r)}(W(E_{r-1}) + F_{(v_{r-1})}(\cdots + F_{(v_2)}(W(E_1) + F_{(v_1)}(z))\cdots)) \geq z, \tag{5}$$

where, for each E_i, $F_{(v_i)}(w)$ is the restriction of $F_C(T(E_i))$ to the case in which all the nodes of $T(E_i)$ have weight zero except node v_i which has weight w.

Condition (5) ensures that no node weight can be decreased through a cycle, and plays the same role as the nonnegative cycles condition in digraphs.

To provide a deeper understanding of condition (5) we will apply it to both the distance function and to the value function below. In the first case, since $F_C(T(E_i))$ is the maximum among the weights of the nodes belonging to $T(E_i)$, $F_{(v_i)}(w) = w$, and condition (5) becomes:

$$\sum_{i=1}^{r} W(E_i) + z \geq z,$$

from which we get

$$\sum_{i=1}^{r} W(E_i) \geq 0.$$

We have thus derived the nonnegativity condition for cycle weights, which is a standard assumption when dealing with shortest paths in digraphs.

Quite different is the case of the value function. Here we get:

$$W(E_r) + a(v_r, E_r)(W(E_{r-1}) + a(v_{r-1}, E_{r-1})(W(E_{r-2}) + \cdots$$

$$+ a(v_2, E_2)(W(E_1) + a(v_1, E_1)z)\cdots)) \geq z,$$

and

$$W(E_r) + \sum_{h=1}^{r-1} \left(W(E_{r-h}) \prod_{l=0}^{h-1} a(v_{r-l}, E_{r-l}) \right) + z \prod_{i=1}^{r} a(v_i, E_i) \geq z,$$

which is true for any real z if:

$$\prod_{i=1}^{r} a(v_i, E_i) \geq 1.$$

We have thus obtained the "gain-free condition" stated in [20].

Now, consider the problem of finding a set of minimum weight B-paths from origin r to all the nodes y which are B-connected to r. This is the generalization of the well-known shortest path tree problem. Such problem is equivalent to that of finding a solution to the following *generalized Bellman's equations*:

$$W(r) = 0;$$

$$W(y) = \min\{w(E) + F(\{W(x): x \in T(E)\}): E \in BS(y)\}, \quad y \in \mathcal{V} \setminus \{r\}. \tag{6}$$

The following procedure SBT finds a solution of (6) together with a minimum weight *B-tree* rooted at r, i.e., a cycle-free set of minimum weight B-paths connecting r to all the nodes y which are B-connected to it. If y is not B-connected to r, Procedure SBT returns $W(y) = +\infty$. As in Procedure B-Visit, the B-tree computed by Procedure SBT is described by the predecessor function Pv.

The counter k_j, for each hyperarc E_j, represents the number of nodes belonging to $T(E_j)$ which have been removed from Q at a previous iteration and are currently out of Q. The use of the counter permits to reduce substantially the number of updating operations (**for each** $y \in H(E_j)$...); in fact, for each E_j, instead of checking the values $W(y)$ of the nodes belonging to $H(E_j)$ every time a node $u \in T(E_j)$ is selected from Q, this is done only when the last node $u \in T(E_j)$ is removed from Q, i.e., when $k_j = |T(E_j)|$.

The correctness of Procedure SBT directly follows from the fact that, at termination, equations (6) are satisfied; moreover, the number of iterations is finite since:

(i) each time a weight is updated, a new B-tree is found, and no B-tree can be found twice;

(ii) the number of consecutive iterations which do not lead to a change in the node weights is bounded by n.

Clearly, the complexity of Procedure SBT depends on the implementation of the candidate set Q and on the cost needed to evaluate the function F.

Procedure SBT($r, \mathcal{H}$).
  begin
    for each $i \in \mathcal{V}$ **do** $W(i) := +\infty$;
    for each $E_j \in \mathcal{E}$ **do** $k_j := 0$;
    $Q := \{r\}$; $W(r) = 0$;
    repeat
      select and remove $u \in Q$;
      for each $E_j \in \mathrm{FS}(u)$ **do**
        begin
          $k_j := k_j + 1$;
          if $k_j = |\mathrm{T}(E_j)|$ **then**
            begin
              $f := \mathrm{F}(\mathrm{T}(E_j))$;
              for each $y \in \mathrm{H}(E_j)$ such that $W(y) > w(E_j) + f$ **do**
                begin
                  if $y \notin Q$ **then**
                    begin
                      $Q := Q \cup \{y\}$;
                      if $W(y) < +\infty$ **then for each** $E_h \in \mathrm{FS}(y)$ **do** $k_h := k_h - 1$
                  end-if;
                $W(y) := w(E_j) + f$; $Pv[y] := E_j$
              end-for
            end-if
        end-for
    until $Q = \emptyset$
  end-procedure.

For the sake of simplicity, we shall assume that $\mathrm{F}(\mathrm{T}(E))$ can be computed in $O(|\mathrm{T}(E)|)$ time, which is the case in most applications. As for Q, we shall consider three different implementations: the *queue*, with a FIFO selection policy, the *unordered list*, and the *heap*, both with the selection of the minimum weight element. According to the notation introduced in [13] we shall call the corresponding versions of Procedure SBT: SBT-queue, SBT-Dijkstra and SBT-heap, respectively.

Consider first Procedure SBT-queue. The cost of initialization is $O(n + m)$ time. Each operation of selection and removal from Q and insertion into Q has unit cost. As in the classical shortest path algorithms, one can easily prove that if Q is implemented as a queue then each node is selected and processed at most n times. Also each hyperarc E is examined at most n times; this is due to the fact that the nodes of $\mathrm{H}(E)$ are only examined when all the nodes in $\mathrm{T}(E)$ no longer belong to Q. The scanning of $\mathrm{H}(E)$ costs $O(|\mathrm{T}(E)|)$ time for the evaluation of $\mathrm{F}(\mathrm{T}(E))$ and $O(|\mathrm{H}(E)|)$ time for the testing of condition $W(y) > w(E) + \mathrm{F}(\mathrm{T}(E))$, for each $y \in \mathrm{H}(E)$. Thus, Procedure SBT-queue runs in $O(n \cdot size(\mathcal{H}))$ time.

It is worth noting that condition (5) on nondecreasing cycles is tighter than what

is actually needed; in fact, for the correctness of Procedure SBT-queue, it is enough that during its operations no negative cycle is detected, where by negative cycle we mean a decreasing cycle which actually leads to cyclic improvements of its node weights. Note that Procedure SBT-queue can be easily modified in order to detect such negative cycles by simply bounding the number of improvements on the weight of a single node.

Now, consider the case in which at each iteration a node u such that $W(u) = \min\{W(x): x \in Q\}$ is selected. In this case, the well-known assumption of nonnegative arc weights for standard digraphs in the Dijkstra Theorem can be generalized to:

$$w(E) + F(\{W(x): x \in T(E)\}) \geq W(x), \quad x \in T(E), \ E \in \mathscr{E}.$$

Under this additional assumption the Dijkstra Theorem can be easily extended to hypergraphs:

Theorem 6.1. *If* $W(u) = \min\{W(x): x \in Q\}$, *then* $W(u)$ *is the minimum among the weights of the B-paths from r to u.*

Corollary. *Each node $u \in \mathscr{V}$ is removed from Q at most once.*

A consequence of the above Corollary is that statement "**if** $W(y) < +\infty$ **then for each** $E_h \in \mathrm{FS}(y)$ **do** $k_h := k_h - 1$" can be dropped since it is no longer necessary to decrease the counters.

The complexity for Procedure SBT-Dijkstra and for Procedure SBT-heap directly follow from the Corollary:

- Procedure SBT-Dijkstra runs in $O(\max\{n^2, size(\mathscr{H})\})$ time, as the total cost of node selections and removals from Q is $O(n^2)$ and the total cost of processing all hyperarcs E (evaluation of $F(T(E))$ and scanning of $H(E)$) is $O(size(\mathscr{H}))$.

- Procedure SBT-heap runs in $O(size(\mathscr{H}) \cdot \log n)$ time, as each time the value $W(y)$ of a node y is updated the heap must be updated at cost $O(\log n)$.

- In the case of B-graphs, Procedure SBT-heap runs in $O(\max\{m \log n, size(\mathscr{H})\})$ time, as each B-arc produces at most one weight improvement, thus the overall cost of updating the heap is $O(m \log n)$.

Jeroslow, Martin, Rardin and Wang [20] presented an algorithm to find the optimal values of $V(y)$ for each node y in a B-graph. This algorithm generalizes the Bellman–Ford–Moore algorithm and runs in $O(n \cdot size(\mathscr{H}))$. It is as fast as Procedure SBT-queue and slower than Procedures SBT-Dijkstra and SBT-heap.

7. Application of hypergraphs

7.1. Satisfiability

Let $\mathscr{P}$ be a set of n atomic *propositions*, which can be either *true* or *false*, and

denote by t a proposition which is always *true*, and by f a proposition which is always *false*. Let $\mathscr{C}$ be a set of *m clauses*, each of the form:

$$p_1 \vee p_2 \vee \cdots \vee p_r \leftarrow p_{r+1} \wedge p_{r+2} \wedge \cdots \wedge p_q, \tag{7}$$

where, for $i = 1, \ldots, q$, $p_i \in \mathscr{P}$. The meaning of (7) is that at least one of the propositions $p_1, \ldots, p_r$ must be *true* when all the propositions $p_{r+1}, \ldots, p_q$ are *true*. If this is the case, the clause is *true*; otherwise ($p_1, \ldots, p_r$ are all *false*, and $p_{r+1}, \ldots, p_q$ are *true*) the clause is *false*. The disjunction $p_1 \vee p_2 \vee \cdots \vee p_r$ is also called the *consequence* of the clause, while the conjunction $p_{r+1} \wedge p_{r+2} \wedge \cdots \wedge p_q$ is called the *implicant*. We allow for $r = 0$, in which case the consequence is replaced by f, and for $r = q$, in which case the implicant is replaced by t.

Clause (7) can be easily converted into *disjunctive form*:

$$p_1 \vee p_2 \vee \cdots \vee p_r \vee \neg p_{r+1} \vee \neg p_{r+2} \vee \cdots \vee \neg p_q.$$

A *truth evaluation* is a function $v : \mathscr{P} \to \{false, true\}$. If there is a truth evaluation which makes all the clauses *true*, then $\mathscr{C}$ is said to be *satisfiable*, otherwise it is *unsatisfiable*.

The *satisfiability problem* (SAT) is defined as follows:

Input: A set $\mathscr{P}$ of n propositions, and a set $\mathscr{C}$ of m clauses over $\mathscr{P} \cup \{f, t\}$;
Output: "**yes**" if $\mathscr{C}$ is satisfiable, "**no**" otherwise.

Most often, in the case of *yes-instances*, a truth evaluation which satisfies $\mathscr{C}$ is also desired.

A particularly important case is when a clause contains only one atomic proposition, i.e., $r \le 1$ in (6). Such clause is called a *Horn clause*.

It is well known that SAT is NP-complete [9, 15]. Either NP-complete, or NP-hard, are also most of its variants such as k-SAT (each clause contains $k \ge 3$ atomic propositions at most) and Max-SAT (the maximization of the number of satisfied clauses, or equivalently, the minimization of the number of clauses to be dropped in order to make the remaining clauses satisfiable). A notable exception is the case in which $\mathscr{C}$ contains only Horn clauses. In this case the satisfiability problem (HORN-SAT) is polynomial: in fact it can be solved in linear time [10, 17]. Unfortunately, Max-HORN-SAT remains NP-hard [19].

HORN-SAT is the set of the instances of SAT whose clauses are Horn clauses.

To any given instance $\pi \in \mathrm{SAT}$ we can associate the hypergraph $\mathscr{H}_\pi$ with one node for each element of $\mathscr{P} \cup \{f, t\}$ and one hyperarc E with $\mathrm{H}(E) = \{p_1, p_2, \ldots, p_r\}$ and $\mathrm{T}(E) = \{p_{r+1}, p_{r+2}, \ldots, p_q\}$ for each clause $p_1 \vee p_2 \vee \cdots \vee p_r \leftarrow p_{r+1} \wedge p_{r+2} \wedge \cdots \wedge p_q$. Clearly, from the definition, if $\pi \in \mathrm{HORN\text{-}SAT}$ then $\mathscr{H}_\pi$ is a B-graph. Note that the labelled graphs introduced by Dowling and Gallier [10] to represent HORN-SAT instances have a direct interpretation as B-graphs.

Theorem 7.1. *An instance $\pi \in \mathrm{SAT}$ is satisfiable if and only if the associated hypergraph $\mathscr{H}_\pi$ has a cut $\mathscr{T}_{tf}$ with cardinality 0.*

Proof. ($\Rightarrow$) If π is satisfiable, then a truth assignment v exists which makes all the clauses in π *true*. Consider the cut $\mathcal{T}_{tf} = (\mathcal{V}_t, \mathcal{V}_f)$ with:

$$\mathcal{V}_t = \{p : v(p) = true\} \cup \{t\} \quad \text{and} \quad \mathcal{V}_f = \{p : v(p) = false\} \cup \{f\}.$$

We claim that $\mathcal{T}_{tf}$ has cardinality 0; in fact the existence of a hyperarc E with $T(E) \subseteq \mathcal{V}_t$ and $H(E) \subseteq \mathcal{V}_f$ would imply the existence of a clause made *false* by v.

($\Leftarrow$) Let $\mathcal{T}_{tf} = (\mathcal{V}_t, \mathcal{V}_f)$ be a cut with 0 cardinality. It is easy to check that the function:

$$v(p) = \begin{cases} true, & \text{if } p \in \mathcal{V}_t, \\ false, & \text{if } p \in \mathcal{V}_f, \end{cases}$$

is a truth assignment which makes all the clauses of π *true*. $\square$

A direct consequence of Theorem 7.1 and of the results of Sections 4 and 5 is that HORN-SAT is equivalent to the problem of finding a B-path in a B-graph. Then, Procedure B-Visit can solve any instance of HORN-SAT in linear time. Actually, Procedure B-Visit bears a strong resemblance with the linear algorithm for HORN-SAT proposed by Dowling and Gallier [10].

Similarly, as one can easily check, Procedure SuperVisit can be used to solve the instances of SAT.

Another interesting consequence of Theorem 7.1 is that:

Theorem 7.2. Max-SAT *can be solved by finding a minimum cardinality t–f cut on the corresponding hypergraph.*

Proof. The proof follows directly from Theorem 7.1 and from the fact that a minimum cardinality cutset provides the minimum number of hyperarcs to be removed to make f not superconnected to t. $\square$

Since Max-SAT is NP-hard, Theorem 7.2 implies the NP-hardness of the minimum cardinality (capacity) cut in hypergraphs.

7.2. And-Or graphs

An *And-Or graph* is a digraph $G = (N, A)$ where each arc $a \in A$ is assigned a *label* $l(a)$ with the property that if $l(a) = l(b)$ for two arcs $a, b \in A$, then a and b have a common tail, i.e., $T(a) = T(b)$.

An arc a is an *And arc* if it shares its label with some other arc, while an arc a is an *Or arc* if $l(a) \neq l(b)$ for all $b \neq a$.

In the literature, different notations have been used by different authors. Particularly relevant are the work of Nilsson [30], in which the nodes are defined as being *And nodes* or *Or nodes* according to the type of the ingoing arcs, and that of Martelli and Montanari [25], in which the nodes are defined as being *And nodes*

or *Or nodes* according to the type of the outgoing arcs. The definition adopted here is more general and include the others as particular cases.

A *connection* from a node x to a node y in an And-Or graph is a minimal set of arcs A^* such that:

(i) $a \in A^*$ and $l(a') = l(a) \Rightarrow a' \in A^*$;

(ii) $G^* = (N, A^*)$ is the union of paths from x to y.

An And-Or graph can be viewed as an F-graph, with the same set of nodes and one F-arc for each set of arcs with the same label. It is easy to see that a connection on an And-Or graph is an F-path in the corresponding F-graph.

Nilsson [30], Martelli and Montanari [25], Levi and Sirovich [22] and Gnesi, Martelli and Montanari [16] have studied the problem of finding a minimum cost connection between two nodes in an And-Or graph where each arc is assigned a real cost. With respect to the present framework, this is the problem of finding a minimum length F-path on an F-graph considered in Section 6.2.

It is interesting to note that most often the problems considered in the literature lead to acyclic And-Or graphs. In this case the algorithms presented in Section 6.2 can be further simplified if the (acyclic) F-graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ corresponding to the And-Or graph is preprocessed in order to renumber its nodes in inverse topological order such that:

$$E \in \mathcal{E}: \quad (\mathrm{T}(E) = \{i\}) \wedge (j \in \mathrm{H}(E)) \Rightarrow (j < i). \tag{8}$$

Such node preordering can be accomplished by the following Procedure F-Acyclic, a generalization of the classical procedure described in Knuth [21], proposed by Longo [23].

Procedure F-Acyclic($\mathcal{H}$).
  begin
    for each $i \in \mathcal{V}$ **do** $r_i := 0$;
    for each $E = (\{i\}, \mathrm{H}(E)) \in \mathcal{E}$ **do** $r_i := r_i + |\mathrm{T}(E)|$;
    $k := 0$; $Q := \emptyset$;
    for each $i \in \mathcal{V}$ **do if** $r_i = 0$ **then** $Q := Q \cup \{i\}$;
    while $Q \neq \emptyset$ **do**
      begin
        select and remove $u \in Q$;
        $k := k + 1$; $e_u := k$;
        for each $E = (\{i\}, \mathrm{H}(E)) \in \mathrm{BS}(u)$ **do**
          begin $r_i := r_i - 1$; **if** $r_i = 0$ **then** $Q := Q \cup \{i\}$ **end-for**
      end-while;
    if $k = n$ **then** return "$\mathcal{H}$ is acyclic" **else** return "$\mathcal{H}$ is not acyclic"
  end-procedure.

The number of nodes (with repetitions) which follow node i and are not yet scanned is maintained in counter r_i. Initially r_i is equal to the sum of the cardinalities of the

heads of the F-arcs having node i as tail. When $r_i = 0$, then node i can be inserted into the set of candidate nodes Q, implemented as a queue. Procedure F-Acyclic checks whether the F-graph is acyclic or not. In the case it is acyclic, a label e_u satisfying conditions (8) is assigned to each node u.

Since each F-arc is examined only once, the procedure runs in O($size(\mathcal{H})$).

Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be an acyclic F-graph whose nodes satisfy conditions (8). The following Procedure SFT-Acyclic (*Shortest F-Tree for Acyclic F-graphs*) is the adaptation to F-graphs of Procedure SBT described in Section 6.2 in which conditions (8) are exploited; it finds a shortest F-path starting from root node $r = |\mathcal{V}|$, which is the last one in the ordering.

Procedure SFT-Acyclic($r, \mathcal{H}$).
  begin
    for each $i \in \mathcal{V}$ **do**
      begin
        $Pv[i] := 0;$
        if FS(i) $= \emptyset$ **then** $W(i) := 0$ **else** $W(i) := \infty$
      end-for
    for each $E_j \in \mathcal{E}$ **do** $k_j := 0;$
    for $i = 1$ **to** $|\mathcal{V}| - 1$ **do for each** $E_j = (\{y\}, H(E_j)) \in$ BS(i) **do**
      begin
        $k_j := k_j + 1;$
        if $k_j = |H(E_j)|$ **then**
          begin
            $f := F(H(E_j));$
            if $W(y) > w(E_j) + f$ **then**
              begin $W(y) := w(E_j) + f;$ $Pv[y] := E_j$ **end-if**
          end-if
      end-for
    end-procedure.

Procedure SFT-Acyclic selects all the nodes following the inverse topological order. An F-arc $E = (\{y\}, H(E))$ is considered for the improvement of the F-path originating from node y only when a shortest F-path is known for each node belonging to $H(E)$. Thus, each node and each F-arc are selected at most once leading to an overall complexity of O($size(\mathcal{H})$).

7.3. Relational databases

In the last years a substantial amount of research has been devoted to the analysis of relational databases using graph related techniques [1, 2, 24, 26, 32, 34, 36].

A *relational database* (*RDB*) is often represented by a set of relations over a certain domain of attribute values, together with a set of functional dependencies.

Functional dependencies have been studied by means of several types of generalized graphs, such as FD-graphs, implication graphs, deduction graphs, etc.

Let N be the set of attributes of an RDB. A *functional dependency* $F(X, Y)$, with both X and Y subsets of N, defines uniquely the value of the attributes in Y once the value of the attributes in X is given.

A set of functional dependencies together with some *inference rules* allows us to derive new facts from that explicitly stored in the database. Typical inference rules are (see [35, 36]):

(i) *reflexivity*: $F(X, Y)$ if $Y \subseteq X$;

(ii) *transitivity*: $F(X, Z)$ if $F(X, Y)$ and $F(Y, Z)$;

(iii) *conjunction*: $F(X, Y \cup Z)$ if $F(X, Y), F(X, Z)$.

Given a set of functional dependencies, $\mathscr{F}$, we might need to solve problems such as:

(a) find whether a given functional dependency $F(X, Y) \notin \mathscr{F}$ can be *derived* from $\mathscr{F}$ based on inference rules;

(b) given a set of attributes $X \in \mathscr{F}$, find its *closure* with respect to $\mathscr{F}$, i.e., find the largest set X^* such that $F(X, X^*)$ either belongs to or can be derived from $\mathscr{F}$.

Here we show briefly that hypergraphs provide a natural and unifying formalism to deal with most problems arising in the analysis of functional dependencies in RDB.

A set $\mathscr{F}$ of functional dependencies on the attribute set N can be represented by a hypergraph $\mathscr{H} = (\mathscr{V}, \mathscr{E})$, with $\mathscr{V} = N$ and $\mathscr{E} = \{(X, Y \setminus X): F(X, Y) \in \mathscr{F}, Y \nsubseteq X\}$. It is easy to see that a B-path on $\mathscr{H}$ corresponds to a sequence of implications based on rules (i), (ii) and (iii). For example, the B-path of Fig. 9 corresponds to the derivation of $F(\{1, 2, 3, 4\}, \{9, 10\})$ starting from the implication relationships $F(\{2\}, \{5\})$, $F(\{3, 4\}, \{6, 7, 8\})$, $F(\{5, 7\}, \{9\})$ and $F(\{4, 8\}, \{10\})$, where attributes are denoted by natural numbers.

Procedure B-Visit solves problems (a) and (b) in $O(size(\mathscr{H})) = O(size(\mathscr{F}))$ time. In both cases, the set Q used in Procedure B-Visit is initialized to X. Let X' be the set of nodes visited by the procedure, i.e., the set of nodes B-connected to X. In problem (a), the answer is that $F(X, Y)$ is derivable from $\mathscr{F}$ if and only if $Y \subseteq X'$, while in problem (b) the answer is $X^* = X'$.
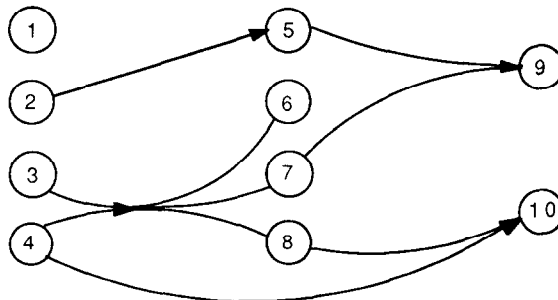

Fig. 9. A B-path representation of a sequence of implications.

When the set Y is a singleton, i.e., the functional dependency is of the type $F(X, y)$ where $y \in N$. The directed hypergraphs representing sets of functional dependencies of this type are B-graphs. This interesting case has been studied in [2–4, 18], where several problems on sets of functional dependencies are defined, and graph algorithms for their solution presented. All these algorithms have a natural interpretation in terms of hypergraph algorithms.

7.4. Urban transit application

The analysis of passenger distribution in a transit system is an interesting application of F-graphs [27–29].

A transit system can be modelled as a special network in which transit lines are superimposed on a ground network. Each transit line is a circuit, i.e., a close alternating sequence of nodes representing the *line stops* and *arcs* representing the *in-vehicle* line segments.

The ground network is formed by nodes representing geographical points (either *stops* or *zone centroids*) in the urban area, and arcs representing *walking paths* between centroids and/or stops.

For each stop node i on the ground network, let L_i be the set of lines which stop at i. Each node i will be connected to the corresponding nodes on the lines belonging to L_i by a *leaving arc* and a *boarding arc*. An example is given in Fig. 10.

From a local standpoint, consider a passenger waiting at a stop i, who wishes to reach his/her destination s with the least expected travel time. The problem consists in determining the optimal subset $L_i^* \subseteq L_i$, the so-called *attractive set*, such that by always boarding the first carrier of these lines arriving at the stop, the expected travel time will be minimized.

Consider the following notation:

- ϕ_j, the frequency of line $l_j \in L_i$;
- $\Phi(L_i')$, the "combined" frequency of the lines set L_i';
- $\pi_j(L_i')$, the probability that a carrier serving line l_j will arrive at stop i before carriers serving other lines of L_i';
- t_j, the expected travel time between stop i and the destination, if line l_j is used, not including the waiting time at i;

line 1

line 2

line 3

leaving arcs   boarding arcs
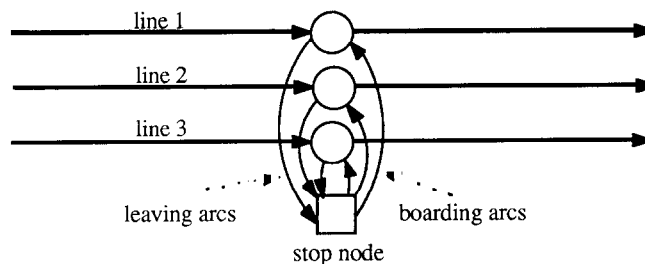
stop node

Fig. 10. A stop served by three lines.

- $w(L_i')$, the average waiting time at stop i.

In general, the travel times t_j are composed of walking times, in-vehicle travel times and waiting times associated with transfers from one line to another which can occur in the sequel of the trip. These times are the lengths of the associated arcs of the network; the lengths of in-vehicle arcs are the corresponding carrier travel times, the lengths of walking arcs are walking times, and the lengths of leaving arcs are set to 0. The waiting times are associated with boarding arcs; the value of a boarding arc (i, j) from a stop i to the corresponding line stop of line l_j depends on the subset of lines L_i' considered. Moreover, all the boarding arcs of lines belonging to L_i' have the same length, which is the *average waiting time* $w(L_i')$.

Under reasonable hypotheses on the distribution of passenger and carrier arrivals at the stops, the following results are obtained:

$$\Phi(L_i') = \sum_{l_j \in L_i'} \phi_j, \qquad w(L_i') = \frac{1}{2\Phi(L_i')}, \qquad \pi_j(L_i') = \frac{\phi_j}{\Phi(L_i')},$$

and the expected travel time between stop i and the destination, when the set L_i' is selected, is:

$$T(L_i') = w(L_i') + \sum_{l_j \in L_i'} t_j \pi_j(L_i') = \frac{1}{2\Phi(L_i')} + \sum_{l_j \in L_i'} \frac{t_j \phi_j}{\Phi(L_i')} = \frac{\frac{1}{2} + \sum_{l_j \in L_i'} t_j \phi_j}{\Phi(L_i')}.$$

The optimal set L_i^* is the subset of L_i which minimizes the expected travel time:

$$T(L_i^*) = \min\{T(L_i'): L_i' \subseteq L_i\}.$$

When travel times t_j for every $l_j \in L_i$ are known, the optimal set L_i^* is easily found with a local greedy algorithm. This algorithm works as follows: first, sort the lines in nondecreasing order of travel times, and then iteratively insert the lines one by one into L_i^* until a line l_j for which $t_j > T(L_i^*)$ is found [27, 28].

The global problem is that of determining the least expected travel times t_r for every origin r and a given destination s. To solve this, the least expected travel times t_j for every $l_j \in L_i$ and the optimal sets L_i^* for all stops i must be computed simultaneously.

For this purpose, F-graphs have been introduced to represent transit networks; boarding arcs corresponding to L_i' may be modelled by a boarding F-arc $E(L_i')$ with length $w(L_i')$. The resulting F-graph is said *full* because if there is an F-arc $E = (\{i\}, H(E))$, then each $E' = (\{i\}, H(E'))$ with $E' \subseteq E$ also exists. E' is called a *contained* F-arc. The contained F-arcs are treated implicitly to keep the size of the F-graph at a reasonable level.

Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be the F-graph in which contained F-arcs are omitted. The problem of finding the least expected travel times for destination s is equivalent to that of finding shortest F-paths terminating at s in F-graph $\mathcal{H}$. In Section 5 we mentioned that F-visits are easy when they are organized from the destination node towards origin nodes; this is also true for shortest F-paths. For the above transportation problem, the following generalized Bellman's equations can be written, in which the

weighted average distances are defined separately for stops and other nodes. Let $\mathcal{V}_S$ be the set of stops, then:

$$d_s(s) = 0;$$

$$d_s(x) = \min\{t_{xy} + d_s(y): (x, y) \in \mathrm{FS}(x)\}, \quad x \in \mathcal{V} \setminus \mathcal{V}_S;$$

$$d_s(x) = \min\left\{ w(L'_x) + \sum_{y_j \in \mathrm{H}(E(L'_x))} d_s(y_j)\pi_j(L'_x): E(L'_x) \in \mathrm{FS}(x) \right\}$$

$$= \min\left\{ \frac{\frac{1}{2} + \sum_{y_j \in \mathrm{H}(E(L'_x))} d_s(y_j)\phi_j}{\Phi(L'_x)}: E(L'_x) \in \mathrm{FS}(x) \right\}, \quad x \in \mathcal{V}_S.$$

Similar to Procedure SBT, *Shortest F-Tree* procedures (SFT) have been developed to solve the above equations. Both types of SFT-queue and SFT-Dijkstra procedures are described in [28, 29].

References

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

[2] G. Ausiello, A. D'Atri and D. Saccà, Strongly equivalent directed hypergraphs, in: G. Ausiello and M. Lucertini, eds., Analysis and Design of Algorithms for Combinatorial Problems, Annals of Discrete Mathematics 25 (North-Holland, Amsterdam, 1985) 1–25.

[3] G. Ausiello, A. D'Atri and D. Saccà, Minimal representation of directed hypergraphs, SIAM J. Comput. 15 (1986) 418–431.

[4] G. Ausiello, G.F. Italiano and U. Nanni, Dynamic maintenance of directed hypergraphs, Theoret. Comput. Sci. 72 (1990) 97–117.

[5] C. Berge, Graphs and Hypergraphs (North-Holland, Amsterdam, 1973).

[6] C. Berge, Minimax theorems for normal hypergraphs and balanced hypergraphs – a survey, in: Annals of Discrete Mathematics 21 (North-Holland, Amsterdam, 1984) 3–19.

[7] C. Berge, Hypergraphs: Combinatorics of Finite Sets (North-Holland, Amsterdam, 1989).

[8] H. Boley, Directed recursive labelnode hypergraphs: a new representation language, Artificial Intelligence 9 (1977) 49–85.

[9] S. Cook, The complexity of theorem-proving procedures, in: Proceedings 3th ACM Symposium on Theory of Computing (1971) 151–158.

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

[11] A.L. Furtado, Formal aspects of the relational model, Inform. Systems 3 (1978) 131–140.

[12] G. Gallo, G. Longo, S. Nguyen and S. Pallottino, Gli ipergrafi orientati: un nuovo approccio per la formulazione e risoluzione di problemi combinatori, in: Atti AIRO 89 (1989) 217–236.

[13] G. Gallo and S. Pallottino, Shortest path methods: a unifying approach, Math. Programming Stud. 26 (1986) 38–64.

[14] G. Gallo and G. Urbani, Algorithms for testing the satisfiability of propositional formulae, J. Logic Programming 7 (1989) 45–61.

[15] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness (Freeman, San Francisco, CA, 1979).

[16] S. Gnesi, U. Montanari and A. Martelli, Dynamic programming as graph searching: an algebraic approach, J. ACM 28 (1981) 737–751.

[17] A. Itai and J. Makowsky, On the complexity of Herbrand's theorem, Tech. Rept. 243, Department of Computer Science, Israel Institute of Technology (1982).

[18] G.F. Italiano and U. Nanni, On line maintenance of minimal directed hypergraphs, in: Proceedings 3° Convegno Italiano di Informatica Teorica, Mantova (World Science Press, 1989) 335–349.

[19] B. Jaumard and B. Simeone, On the complexity of the maximum satisfiability problem for Horn formulas, Inform. Process. Lett. 26 (1987) 1–4.

[20] R.G. Jeroslow, R.K. Martin, R.R. Rardin and J. Wang, Gainfree Leontiev flows problems, Tech. Rept., School of Business, University of Chicago, Chicago, IL (1989).

[21] D.E. Knuth, The Art of Computer Programming (Addison-Wesley, Reading, MA, 1968).

[22] G. Levi and F. Sirovich, Generalized And/Or graphs, Artificial Intelligence 7 (1976) 243–259.

[23] G. Longo, Per una nuova teoria degli ipergrafi orientati, tesi di laurea, Dipartimento di Informatica, Università di Pisa, Pisa (1989).

[24] D. Maier, Minimum covers in the relational data base model, J. ACM 27 (1980) 664–674.

[25] A. Martelli and U. Montanari, Additive AND/OR graphs, Proc. IJCAI 3 (1973) 1–11.

[26] J. Martin, Computer Data-Base Organization (Prentice-Hall, Englewood Cliffs, NJ, 1977).

[27] S. Nguyen and S. Pallottino, Assegnamento dei passeggeri ad un sistema di linee urbane: determinazione degli ipercammini minimi, Ricerca Operativa 38 (1986) 28–47.

[28] S. Nguyen and S. Pallottino, Equilibrium traffic assignment for large scale transit networks, European J. Oper. Res. 37 (1988) 176–186.

[29] S. Nguyen and S. Pallottino, Hyperpaths and shortest hyperpaths, in: B. Simeone, ed., Combinatorial Optimization, Lecture Notes in Mathematics 1403 (Springer, Berlin, 1989) 258–271.

[30] N.J. Nilsson, Problem Solving Methods in Artificial Intelligence (McGraw-Hill, New York, 1971).

[31] N.J. Nilsson, Principles of Artificial Intelligence (Morgan Kaufmann, Los Altos, CA, 1980).

[32] H.C. Smith, Database design: composing fully normalized tables from a rigourous dependency diagram, Comm. ACM 28 (1985) 826–838.

[33] A.F. Torres and J.D. Aráoz, Combinatorial models for searching in knowledge bases, Acta Cient. Venezolana 39 (1988) 387–394.

[34] J.D. Ullman, Principles of Database Systems (Computer Science Press, Rockville, MD, 1982).

[35] C.C. Yang, Relational Databases (Prentice-Hall, Englewood Cliffs, NJ, 1986).

[36] C.C. Yang, Deduction graphs: an algorithm and applications, IEEE Trans. Software Engrg. 15 (1989) 60–67.