DIRECTED HYPERGRAPHS AND APPLICATIONS[(*)]

Giorgio Gallo[1] Giustino Longo[1]

Sang Nguyen[2] Stefano Pallottino[1]

Abstract

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 data bases and transportation analysis are presented.

January 1990

Revised, October 1992

(*) This research has been supported in part by the “Comitato Nazionale Scienza e Tecnologia dell'Informazione”, National Research Council of Italy, under Grant n.89.00208.12, and in part by research grants from the National Research Council of Canada.

[1] Dipartimento di Informatica, Università di Pisa, Italy

[2] Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Canada

INTRODUCTION

Hypergraphs, a generalization of graphs, have been widely and deeply studied in Berge (1973, 1984, 1989), 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 introduced in the literature as a way to deal with particular problems arising in Computer Science and in Combinatorial Optimization (see, for example, Nilsson (1971), Martelli and Montanari (1973), Levi and Sirovich (1976), Boley (1977), Furtado (1978), Maier (1980), Nilsson (1980), Gnesi, Montanari and Martelli (1981), Ullman (1982), Nguyen and Pallottino (1989)).

Directed hypergraphs have also been explicitly introduced in Torres and Aráoz (1988), Longo (1989), Gallo, Longo, Nguyen and Pallottino (1989). In addition, particular instances of directed hypergraphs can be found in Dowling and Gallier (1984), Ausiello, D'Atri and Saccà (1985, 1986), Nguyen and Pallottino (1988), Gallo and Urbani (1989), Ausiello, Italiano and Nanni (1990), Italiano and Nanni (1989).

The remaining of the paper is organised 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 (Dowling and Gallier (1984), Gallo and Urbani (1989)), And-Or graphs (Martelli and Montanari (1973), Levi and Sirovich (1976), Nilsson (1980), Gnesi, Montanari and Martelli (1981)), data bases (Maier (1980), Ullman (1982), Ausiello, D'Atri and Saccà (1983, 1985, 1986)), and urban transportation (Nguyen and Pallottino (1986, 1988, 1989)).

2. DIRECTED HYPERGRAPHS

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

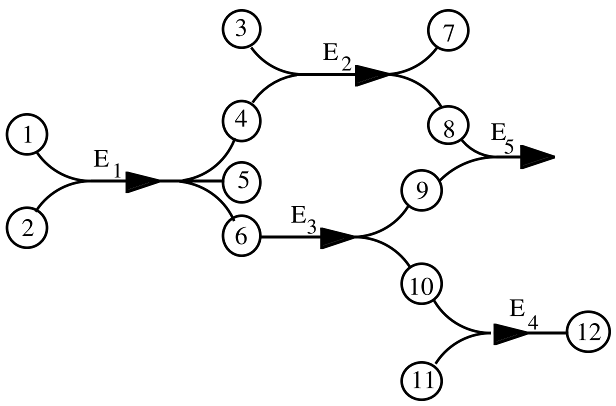
While the size of a standard graph is uniquely defined by n and m, the size of a hypergraph depends also on the cardinality of its hyperedges; we define the *size* of $\mathcal{H}$ as the sum of the cardinalities of its hyperedges:

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

It is worth noting that there is a one-to-one correspondence between hypergraphs and Boolean matrices. Indeed, any nxm 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 $\mathrm{T}(E)$ and $\mathrm{H}(E)$, 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.



	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.

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

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

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 $E = (\mathrm{T}(E), \mathrm{H}(E))$ with $|\mathrm{H}(E)|=1$ (Fig. 2a). A *Forward* hyperarc, or simply *F-arc*, is a hyperarc $E = (\mathrm{T}(E), \mathrm{H}(E))$ with $|\mathrm{T}(E)|=1$ (Fig. 2b).

Fig. 2 - A B-arc (a) and a F-arc (b).

A *B-graph* (or B-hypergraph) is a hypergraph whose hyperarcs are B-arcs. A *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 $\mathrm{H}=(\mathrm{V},\mathrm{E})$, we define its *symmetric image* the hypergraph $\tilde{\mathrm{H}} =(\mathrm{V},\tilde{\mathrm{E}})$ where $\tilde{\mathrm{E}} =\{(\mathrm{X},\mathrm{Y}): (\mathrm{Y},\mathrm{X})\in \mathrm{E}\}$. Note that the symmetric image of a B-graph is a 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 a F-arc, and thus replacing the hyperarc by one backward and one forward hyperarc (see Fig. 3).

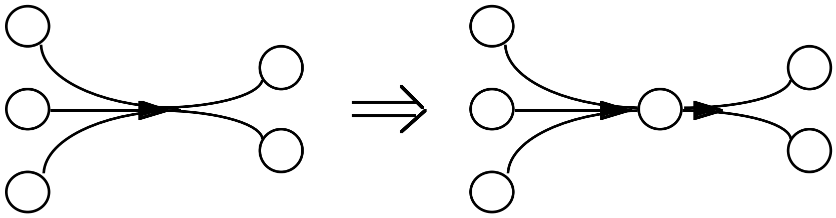


Fig. 3 - Transformation of a hyperarc into a B-arc and a F-arc.

Let $\mathrm{FS}(v)=\{E\in \mathcal{E}: v\in \mathrm{T}(E)\}$ and $\mathrm{BS}(v)=\{E\in \mathcal{E}: v\in \mathrm{H}(E)\}$ 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 (1984) and Gallo and Urbani (1989) to represent Horn-formulae, are B-graphs; B-graphs have been introduced as tools to analyze *deductive data bases* (Ausiello, D'Atri and Saccà (1985, 1986), Ausiello, Italiano and Nanni (1990), Italiano and Nanni (1989)) and to study *Leontiev substitution matrices* and *Leontiev flow problems* (Jeroslow, Martin, Rardin and Wang (1989)); F-graphs have been studied in the context of urban transit problems (Nguyen and Pallottino (1988, 1989)) and applications of F-graphs to the analysis of And-Or graphs are reported in Gallo, Longo, Nguyen and Pallottino (1989).

Torres and Aráoz (1988) introduced hypergraphs and B-graphs, called directed hypergraphs and rule hypergraphs respectively, to represent deduction properties in data bases as paths in hypergraphs.

3. PATHS, HYPERPATHS AND CONNECTION

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

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

Nodes s and t are the *origin* and the *destination* of P_{st}, respectively, and we say that t is *connected* to s. If $t\in \mathrm{T}(E_{i_1})$, 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.

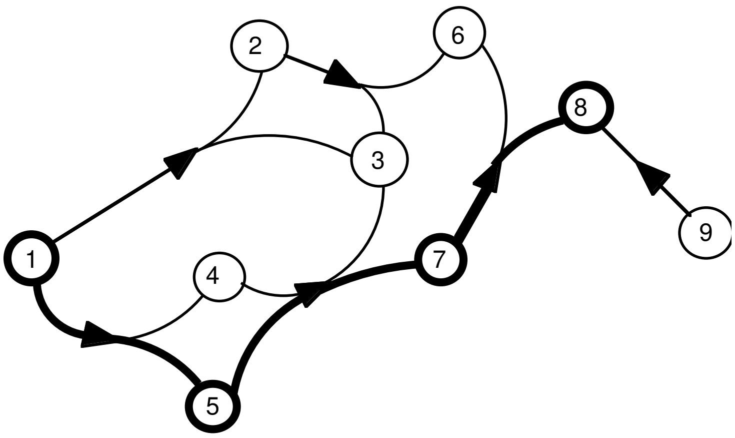


Fig. 4 - A path $P_{1,8}$.

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

i) $\mathrm{E}_\Pi \subseteq \mathrm{E}$;

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

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

We say that $\mathrm{H}_\Pi=(\mathrm{V}_\Pi,\mathrm{E}_\Pi)$ is a *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 a 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 H.

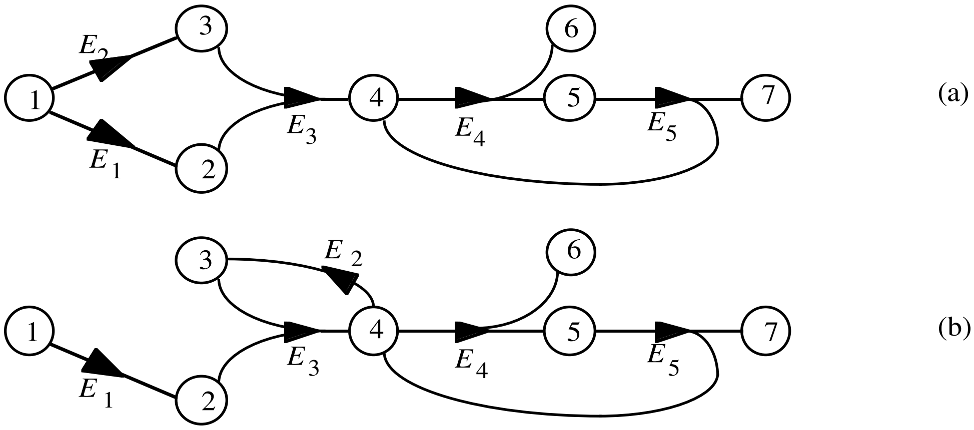


Fig. 5 - A B-path (a) and a B-graph which is not a B-path (b).

The hypergraph in Fig. 5a 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. 5b 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 1 - Given a B-path Π_{st} and a hyperarc $E \in \mathrm{E}_{\Pi}$, 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 H is the B-graph $\mathrm{H_B}$ obtained from H by replacing each hyperarc by one of its B-reductions. Clearly, a hypergraph may have many B-reductions; we shall denote by $\mathrm{B(H)}$ the set of all the B-reductions of 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 H if y is B-connected to x in any B-reduction $\mathrm{H_B}$ of 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.

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 H if y is B-connected (F-connected, BF-connected, super-connected) to s in the hypergraph H' obtained from H by addition of the 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 $\mathrm{H}=(\mathrm{V},\mathrm{E})$ be a hypergraph and s and t be two distinguished nodes, the *source* and the *sink* respectively.

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

The *cardinality* of a cut is the cardinality of its cutset. In Fig. 6 three cuts are indicated; the cardinality of T_{st}^1 is 2, while T_{st}^2 and 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 T_{st}^1 we disconnect t from s, by removing the cutset of 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 T_{st}^3 .

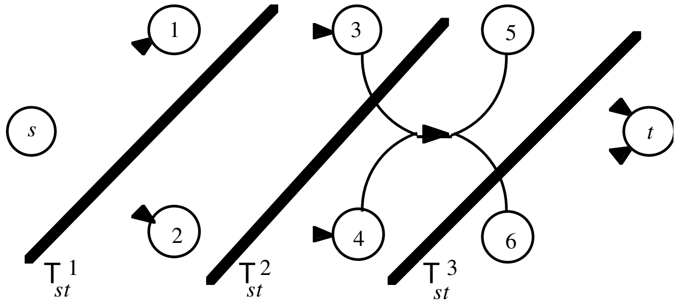


Fig. 6 - Only cut T_{st}^1 disconnects source s and sink t.

The following two theorems relate cuts to connection in hypergraphs.

Theorem 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 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.

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

Theorem 2 - In a hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{E})$ a cut $\mathcal{T}_{st}$ of cardinality 0 exists if and only if t is not super-connected to s.

Proof: ($\Rightarrow$) Let $\mathcal{T}_{st}=(\mathcal{V}_s,\mathcal{V}_t)$ be a cut of cardinality 0. Consider the B-reduction $\mathcal{H}_B$ of $\mathcal{H}$ obtained by replacing each hyperarc E with a B-arc $(T(E),\{v\})$ with the condition that if $T(E) \subseteq \mathcal{V}_s$ then also $v \in \mathcal{V}_s$. This reduction is always possible since for any hyperarc E with $T(E) \subseteq \mathcal{V}_s$ at least one node in its head must belong to $\mathcal{V}_s$, otherwise E belongs to the cutset which, by hypothesis, is empty. By Theorem 1, t is not B-connected to s in $\mathcal{H}_B$ and therefore t is not super-connected to s in $\mathcal{H}$.

($\Leftarrow$) 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 1, the proof is completed.♦

Theorems 1 and 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 3 - In a B-graph $\mathcal{H}=(\mathcal{V},\mathcal{E})$ the following inequalities hold:

min$\{|\Pi_{st}|$: Π_{st} is a s-t B-path$\}$ $\geq$ maximum number of disjoint s-t cutsets $\geq$ min$\{|P_{st}|$: P_{st} is a s-t path$\}$.

Proof: The first inequality follows directly from the fact that, due to Theorem 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 $\{s\}= V_0 \subset V_1 \subset \ldots \subset V_h \subseteq \mathcal{V}$; then $(V_0,\mathcal{V} \setminus V_0)$, $(V_1,\mathcal{V} \setminus V_1)\ldots$ $(V_{h-1},\mathcal{V} \setminus V_{h-1})$ 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 - In a B-graph $\mathrm{H}=(\mathrm{V},\mathrm{E})$ the following inequalities hold:

max-number of disjoint s-t paths $\geq \min\{|\mathrm{E}_{st}|: \mathrm{E}_{st}$ is a s-t cutset$\} \geq$ max-number of disjoint s-t B-paths.

Proof: Transform $\mathrm{H}=(\mathrm{V},\mathrm{E})$ into a standard digraph $G=(\mathrm{V},A)$ 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 E_{st} in H corresponds a s-t cutset C_{st} in G with $|C_{st}| \geq |\mathrm{E}_{st}|$; moreover, any set of k disjoint paths in G corresponds to a set of k disjoint paths in H, then the maximum number of disjoint paths in G is not larger than the maximum number of disjoint paths in H. Hence, from the well known max flow - min cut theorem for digraphs one has:

max-number of disjoint s-t paths in H $\geq$ max-number of disjoint s-t paths in G =

= $\min\{|C_{st}|: C_{st}$ is a s-t cutset in $G\}$ $\geq$ $\min\{|\mathrm{E}_{st}|: \mathrm{E}_{st}$ is a s-t cutset in $\mathrm{H}\}$.

The second inequality follows directly from the fact that, due to Theorem 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.

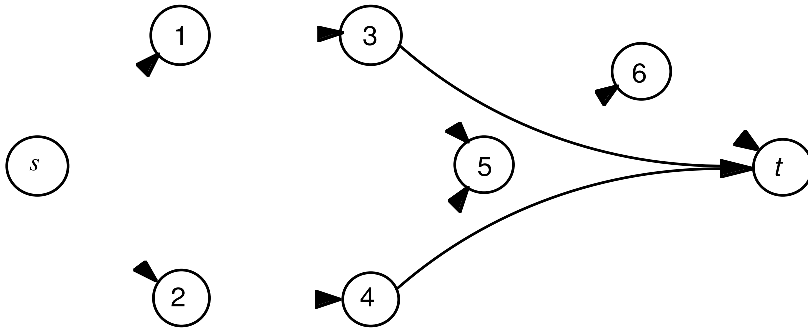


Fig. 7

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.

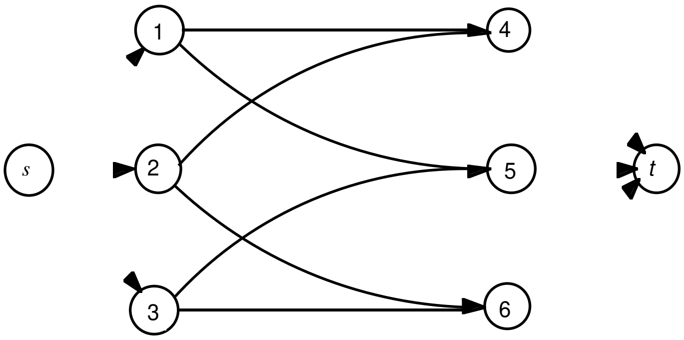


Fig. 8

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 hyperarc E in the path, $Pv(i)$ points to the arc $E \in BS(i)$ which precedes node i in the path.

```
Procedure Visit(r,H):
 begin
   for each i ∈ V do Pv[i]:= 0;
   for each Ej ∈ E do Pe[Ej]:= 0;
   Pv[r]:= nil;   Q:= {r};
   repeat
     select and remove i∈Q;
     for each Ej∈FS(i) such that Pe[Ej]=0 do
       begin
         Pe[Ej] := i ;
         for each h∈H(Ej) such that Pv[h]=0 do begin  Pv[h] := Ej; Q := Q ∪ {h} end-for
       end-for
   until  Q = Ø
 end-procedure.
```

It is easy to check that **Visit** runs in $\mathbf{O}(size(\mathcal{H}))$ time. In fact, the initialization phase runs in $\mathbf{O}(n+m)$ 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 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, ν, 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. To stress the fact that functions *Pe* and ν are not essential to the computation of the B-tree, the statements involving them have been written in italics.

```
Procedure B-Visit(r,H):
 begin
   for each i ∈ V do begin Pv[i] := 0; v[i] := ∞ end-for;
   for each Ej ∈ E do Pe[Ej] := kj := 0;
   Pv[r] := nil; Q := {r}; v[r] := 0;
   repeat
     select and remove i∈Q;
     for each Ej∈FS(i) do
       begin
         kj := kj + 1;
         if kj = |T(Ej)| then
           begin
             Pe[Ej] := i;
               for each h∈H(Ej) such that Pv[h]=0 do
                 begin Pv[h] := Ej; Q := Q ∪ {h}; v[h]:= v[Pe[Ej]]+1 end-for
           end-if
       end-for
   until Q = Ø
 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 the **B-Visit** starts from the origin of the B-paths, the **F-Visit** must start from the destination of the F-paths to retain a linear time complexity.

One can also define a **BF-Visit**. Unfortunately, the problem of performing such a visit is not an easy one unless the hypergraph is either a B-graph or a F-graph; in the former case a BF-path is simply a B-path, while in the latter it is a 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,H):
 begin
   superconnected := true;
   while superconnected and B(H) ≠ Ø do
     begin
       select and remove HB from B(H); B-Visit(s, HB);
       if Pv[t]=0 then superconnected := false
     end-while
 end-procedure.
```

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

A quite efficient Branch&Bound scheme to solve this problem can be easily derived from the second algorithm for the satisfiability problem presented in Gallo and Urbani (1989).

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 $\Pi=(\mathrm{V}_\Pi,\mathrm{E}_\Pi)$ 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 the *cost*, C_Π, defined as the sum of the weights of all the hyperarcs preceding node y in Π:

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

$$C_\Pi(y) = \sum_{E\in\{\mathrm{E}_{\Pi_{sy}}:\ \Pi_{sy}\subseteq\Pi\}} w(E), \qquad y\in \mathrm{V}_\Pi\setminus\{s\},$$

Clearly, $C_\Pi(t)=\sum_{E\in \mathrm{E}_\Pi} w(E)$ 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:

$$W_\Pi(y) = \min\{w(E) + \mathrm{F}_\Pi(\mathrm{T}(E)):\ E\in \mathrm{E}_\Pi\cap\mathrm{BS}(y)\}, \quad y\in \mathrm{V}_\Pi\setminus\{s\}, \tag{1}$$

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

$$\mathrm{F}_\Pi(\mathrm{T}(E)) = \mathrm{F}(\{W_\Pi(x):\ x\in \mathrm{T}(E)\}), \quad E\in \mathrm{E}_\Pi, \tag{2}$$

where F is a non-decreasing function of $W_\Pi(x)$ for each $x\in \mathrm{T}(E)$. 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:

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

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

$$\begin{aligned} &D_\Pi(s) = 0;\\ &D_\Pi(y) = \min\{l(E) + \max\{D_\Pi(x):\ x\in \mathrm{T}(E)\}:\ E\in \mathrm{E}_\Pi\cap\mathrm{BS}(y)\}, \quad y\in \mathrm{V}_\Pi\setminus\{s\}; \end{aligned} \tag{3}$$

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

For B-graphs, equation (3) becomes:

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

In the case of unit hyperarc lengths, i. e. $l(E)=1\ \forall\ E\in \mathrm{E}$, the distance will be called *depth*. Gallo and Urbani (1989) 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 ν and a breadth-first search strategy, finds the minimum depth B-tree in $\mathbf{O}(size(\mathrm{H}))$ time.

The *value*, V_Π, defined by Jeroslow, Martin, Rardin and Wang (1989) in the context of the Leontiev flow problem for the case of B-graphs, is the solution of the following recursive equations:

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

where $c(E)$ is the *cost* of B-arc E and, for each E and each $x\in \mathrm{T}(E)$, $a(x,E)$ is a non-negative 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 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 (1989) 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 non-negative and that all cycles are non-decreasing. A *non-decreasing 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:

$$W(E_r) + \mathrm{F}_{(v_r)}\big(W(E_{r\text{-}1}) + \mathrm{F}_{(v_{r\text{-}1})}\big(\ldots+ \mathrm{F}_{(v_2)}\big(W(E_1) + \mathrm{F}_{(v_1)}(z)\big)\ldots\big)\big) \geq z, \tag{5}$$

where, for each E_i, $\mathrm{F}_{(v_i)}(w)$ is the restriction of $\mathrm{F}_C(\mathrm{T}(E_i))$ to the case in which all the nodes of $\mathrm{T}(E_i)$ 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 non-negative 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 $\mathrm{F}_C(\mathrm{T}(E_i))$ is the maximum among the weights of the nodes belonging to $\mathrm{T}(E_i)$, $\mathrm{F}_{(v_i)}(w) = w$, 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 non-negativity 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:

$$W(E_r)+a(v_r,E_r)\Big(W(E_{r-1}) + a(v_{r-1},E_{r-1})\Big(W(E_{r-2}) + \dots + a(v_2,E_2)\Big(W(E_1) + a(v_1,E_1)z\Big) \dots \Big)\Big) \geq z,$$

and

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

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 Jeroslow, Martin, Rardin and Wang (1989).

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 strictly related to that of finding a solution to the following *Generalized Bellman's Equations*:

$$\begin{aligned} &W(r) = 0; \\ &W(y) = \min\{w(E) + \mathrm{F}(\{W(x): x\in \mathrm{T}(E)\}): E\in \mathrm{BS}(y)\}, \qquad y\in \mathcal{V} \setminus\{r\}. \end{aligned} \tag{6}$$

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, **SBT** returns $W(y) = +\infty$. As in **B-Visit**, the B-tree computed by **SBT** is described by the predecessor function Pv.

```
Procedure SBT(r,H):
 begin
   for each i∈V do W(i) := +∞;
   for each Ej∈E do kj := 0;
   Q := {r};   W(r)=0;
   repeat
     select and remove u∈Q;
     for each Ej∈FS(u) do
       begin
         kj := kj + 1;
         if kj=|T(Ej)| then
           begin
             f := F(T(Ej));
             for each y∈H(Ej) such that W(y)>w(Ej)+f do
               begin
                 if y∉Q then
                   begin
                     Q := Q ∪ {y};
                     if W(y)<+∞ then for each Eh∈FS(y) do kh:=kh-1
                   end-if;
                 W(y) := w(Ej)+f;   Pv[y] := Ej
               end-for
           end-if
       end-for
   until Q = Ø
 end-procedure.
```

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 **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 **SBT** depends on the implementation of the candidate set Q and on the cost needed to evaluate the function F.

For the sake of simplicity, we shall assume that F(T(E)) can be computed in **O**(|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 Gallo and Pallottino (1986) we shall call the corresponding versions of **SBT**: **SBT-queue**, **SBT-Dijkstra** and **SBT-heap**, respectively.

Consider first **SBT-queue**. The cost of the 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 $H(E)$ are only examined when all the nodes in $T(E)$ no longer belong to Q. The scanning of $H(E)$ costs $\mathbf{O}(|T(E)|)$ time for the evaluation of $F(T(E))$ and $\mathbf{O}(|H(E)|)$ time for the testing of condition $W(y)>w(E)+F(T(E))$, for each $y\in H(E)$. Thus, algorithm **SBT-queue** runs in $\mathbf{O}(n\cdot size(\mathcal{H}))$ time.

It is worth noting that condition (5) on non-decreasing cycles is tighter than what is actually needed; in fact, for the correctness of **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, **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 non-negative arc weights for standard digraphs in Dijkstra Theorem can be generalized to:

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

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

Theorem 5 - 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 $u\in \mathcal{V}$ is removed from Q only once.

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

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

- Algorithm **SBT-Dijkstra** runs in $\mathbf{O}(\max\{n^2, size(\mathcal{H})\})$ time, as the total cost of node selections and removals from Q is $\mathbf{O}(n^2)$ and the total cost of processing all hyperarcs E (evaluation of $F(T(E))$ and scanning of $H(E)$) is $\mathbf{O}(size(\mathcal{H}))$.
- Algorithm **SBT-heap** runs in $\mathbf{O}(size(\mathcal{H})\cdot\log n)$ time, as each time the value $W(y)$ of a node y is updated the heap must be updated at cost $\mathbf{O}(\log n)$.
- In the case of B-graphs, algorithm **SBT-heap** runs in $O(\max\{m\log n, size(\mathcal{H})\})$ time, as each B-arc produces at most one weight improvement, thus the overall cost of updating the heap is $\mathbf{O}(m\log n)$.

Jeroslow, Martin, Rardin and Wang (1989) 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 $\mathbf{O}(n\cdot size(\mathcal{H}))$.It is as fast as **SBT-queue** and slower than **SBT-Dijkstra** and **SBT-heap**.

7. APPLICATION OF HYPERGRAPHS

7.1. Satisfiability

Let $\mathcal{P}$ 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 $\mathcal{C}$ be a set of m *clauses*, each of the form:

$$p_1 \vee p_2 \vee \ldots \vee p_r \leftarrow p_{r+1} \wedge p_{r+2} \wedge \ldots \wedge p_q, \qquad (7)$$

where, for i=1,...,q, $p_i \in \mathrm{P}$. 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 $p_1 \vee p_2 \vee \ldots \vee p_r$ is also called the *consequence* of the clause, while the conjunction $p_{r+1} \wedge p_{r+2} \wedge \ldots \wedge p_q$ 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*:

$$p_1 \vee p_2 \vee \ldots \vee p_r \vee \neg p_{r+1} \vee \neg p_{r+2} \vee \ldots \vee \neg p_q,$$

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

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

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

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

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

It is well known that *SAT* is *NP*-complete (Cook (1971), Garey and Johnson (1979)). Either *NP*-complete, or *NP*-hard, are also most of its variants such as *k-SAT* (each clause contains $k \geq 3$ atomic propositions at most) and *Max-SAT* (the maximization of the number of satisfied clauses, or equivalently, the minimisation of the number of clauses to be dropped in order to make the remaining clauses satisfiable). A notable exception is the case in which C contains only Horn clauses. In this case the satisfiability problem (*HORN-SAT*) is polynomial: in fact it can be solved in linear time (Itai and Makowsky (1982), Dowling and Gallier (1984)). Unfortunately, *Max-HORN-SAT* remains *NP*-hard (Jaumard and Simeone (1987)).

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

To any given instance $\pi \in SAT$ we can associate the hypergraph H_π with one node for each element of $\mathrm{P} \cup \{f,t\}$ 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 $p_1 \vee p_2 \vee \ldots \vee p_r \leftarrow p_{r+1} \wedge p_{r+2} \wedge \ldots \wedge p_q$. Clearly, from the definition, if $\pi \in$ *HORN-SAT* then H_π is a B-graph. Note that the labelled graphs introduced by Dowling and Gallier (1984) to represent *HORN-SAT* instances have a direct interpretation as B-graphs.

Theorem 6 - An instance $\pi \in SAT$ is satisfiable if and only if the associated hypergraph H_π has a cut T_{tf} with cardinality 0.

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

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

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

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

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

is a truth assignment which makes all the clauses of π *true*.♦

A direct consequence of Theorem 6 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, **B-Visit** can solve any instance of *HORN-SAT* in linear time. Actually, **B-Visit** bears a strong resemblance with the linear algorithm for *HORN-SAT* proposed by Dowling and Gallier (1984).

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

Another interesting consequence of Theorem 6 is that:

Theorem 7 - *Max-SAT* can be solved by finding a minimum cardinality *t-f* cut on the corresponding hypergraph.

Proof: The proof follows directly from Theorem 6 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*.♦

Since *Max-SAT* is *NP*-hard, Theorem 7 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. $\mathrm{T}(a)=\mathrm{T}(b)$.

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 (1971), 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 (1973), 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 a F-path in the corresponding F-graph.

Nilsson (1971), Martelli and Montanari (1973), Levi and Sirovich (1976) and Gnesi, Martelli and Montanari (1981) 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 a 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 $\mathrm{H}=(\mathrm{V},\mathrm{E})$ corresponding to the And-Or graph is pre-processed in order to re-number its nodes in inverse topological order such that:

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

Such node pre-ordering can be accomplished by the following procedure **F-Acyclic**, a generalization of the classical procedure described in Knuth (1968), proposed by Longo (1989).

```
Procedure F-Acyclic(H):
 begin
   for each i ∈ V do r_i := 0 ;
   for each E=({i},H(E)) ∈ E do r_i := r_i + |T(E)|;
   k:= 0; Q:= Ø;
   for each i ∈ V do if r_i = 0 then Q:= Q ∪ {i};
   while Q ≠ Ø do
     begin
       select and remove u∈ Q;
       k:= k+1; e_u:= k;
       for each E=({i},H(E))∈ BS(u) do
         begin r_i := r_i - 1; if r_i = 0 then Q:= Q ∪ {i} end-for
     end-while;
   if k = n then return "H is acyclic" else return "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,H):
 begin
   for each i ∈ V do
     begin
       Pv[i] := 0;
       if FS(i)=ø then W(i):=0 else W(i):=∞
     end-for
   for each E_j ∈ E do k_j := 0;
   for i = 1 to |V|-1 do for each E_j=({y},H(E_j))∈ 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. A F-arc $E=(\{y\},\mathrm{H}(E))$ 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 $\mathrm{H}(E)$. Thus, each node and each F-arc are selected at most once leading to an overall complexity of $\mathbf{O}(size(\mathcal{H}))$.

7.3. Relational data bases

In the last years a substantial amount of research has been devoted to the analysis of relational data bases using graph related techniques (Martin (1977), Maier (1980), Ullman (1982), Ausiello, D'Atri and Saccà (1983, 1985), Smith (1985), Yang (1989)).

A *Relational Data Base* (*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 a 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 data base. Typical inference rules are (see Yang (1986, 1989)):

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, $\mathcal{F}$, we might need to solve problems such as:

a) find whether a given Functional Dependency $F(X,Y)\notin\mathcal{F}$ can be *derived* from $\mathcal{F}$ based on inference rules;
b) given a set of attributes $X\in\mathcal{F}$, find its *closure* with respect to $\mathcal{F}$, i.e. find the largest set X^* such that $F(X,X^*)$ either belongs to or can be derived from $\mathcal{F}$.

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 $\mathcal{F}$ of Functional Dependencies on the attribute set N can be represented by a hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{E})$, with $\mathcal{V}=N$ and $\mathcal{E}=\{(X,Y\backslash X)\colon F(X,Y)\in\mathcal{F},\ Y\not\subseteq X\}$. It is easy to see that a B-path on $\mathcal{H}$ 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.

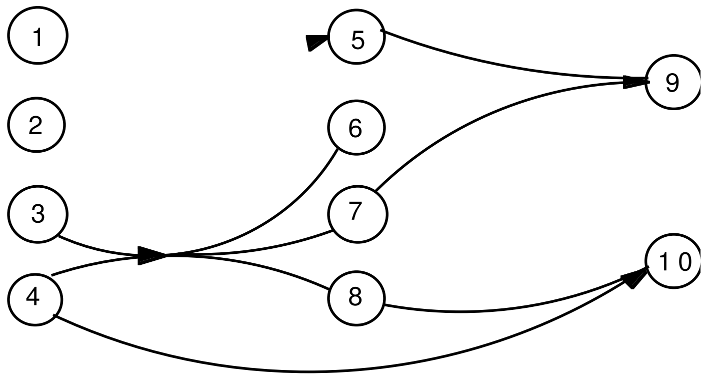


Fig. 9 - A B-path representation of a sequence of implications.

Procedure **B-Visit** solves problems (a) and (b) in $\mathbf{O}(size(\mathcal{H})) = \mathbf{O}(size(\mathcal{F}))$ time. In both cases, the set Q used in **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 $\mathcal{F}$ if and only if $Y \subseteq X'$, while in problem (b) the answer is $X^*=X'$.

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 Ausiello, D'Atri and Saccà (1985, 1986), Ausiello, Italiano and Nanni (1990) and Italiano and Nanni (1989), 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 (Nguyen and Pallottino (1986, 1988, 1989)).

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.

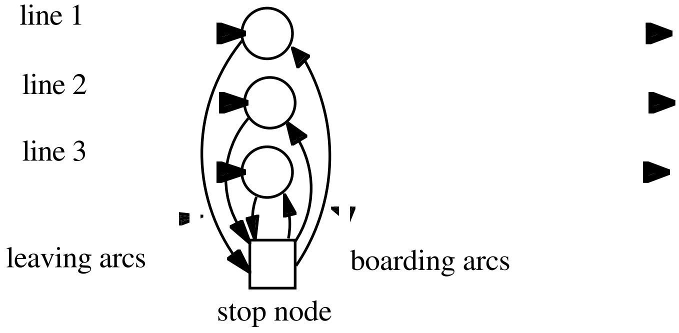


Fig. 10 - A stop served by three lines.

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;
- $w(L_i')$ 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:

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

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

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

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 non-decreasing 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 (Nguyen and Pallottino (1986, 1988)).

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 a F-arc $E=(\{i\},\mathrm{H}(E))$, then each $E'=(\{i\},\mathrm{H}(E'))$ 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 $\mathrm{H}=(\mathrm{V},\mathrm{E})$ 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 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 V_S be the set of stops, then:

$$d_s(s) = 0;$$

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

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

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

Similar to procedures **SBT**, *Shortest F-Tree* procedures (**SFT**) have been developed to solve the above equations. Both type of **SFT-queue** and **SFT-Dijkstra** procedures are described in Nguyen and Pallottino (1988, 1989).

REFERENCES

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

Ausiello, G., A. D'Atri and D. Saccà: *Strongly equivalent directed hypergraphs*, in: **Analysis and Design of Algorithms for Combinatorial Problems** (G. Ausiello e M. Lucertini, eds.), **Annals of Discrete Mathematics, 25** (1985), 1-25.

Ausiello, G., A. D'Atri and D. Saccà: *Minimal representation of directed hypergraphs*, **SIAM J. Comput., 15** (1986), 418-431.

Ausiello, G., G.F. Italiano and U. Nanni: *Dynamic maintenance of directed hypergraphs*, **Theor. Comp. Sci. 72** (1990), 97-117.

Berge, C.: **Graphs and Hypergraphs**, North-Holland, Amsterdam (1973).

Berge, C.: *Minimax theorems for normal hypergraphs and balanced hypergraphs - a survey*, **Annals of Discrete Mathematics, 21** (1984), 3-19.

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

Boley, H.: *Directed recursive labelnode hypergraphs: a new representation language*, **Artificial Intelligence, 9** (1977), 49-85.

Cook, S.: *The complexity of theorem-proving procedures*, **Proc. 3-th ACM Symp. on Theory of Computing** (1971), 151-158.

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

Furtado, A.L.: *Formal aspects of the relational model*, **Inform. Systems, 3** (1978), 131-140.

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

Gallo, G. and S. Pallottino: *Shortest path methods: a unifying approach*, **Math. Progr. Study, 26** (1986), 38-64.

Gallo, G. and G. Urbani: *Algorithms for testing the satisfiability of propositional formulae*, **J. of Logic Programming, 7** (1989), 45-61.

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

Gnesi, S., U. Montanari and A. Martelli: *Dynamic programming as graph searching: an algebraic approach*, **J. Assoc. Comp. Mach., 28** (1981), 737-751.

Itai, A. and J. Makowsky: *On the complexity of Herbrand's theorem*, **Tech. Rept. 243**, Dept. Comp. Sci., Israel Inst. of Technology (1982).

Italiano, G.F. and U. Nanni: *On line maintenance of minimal directed hypergraphs*, **Proc. 3º Convegno Italiano di Informatica Teorica**, Mantova, World Science Press (1989), 335-349.

Jaumard, B. and B. Simeone: *On the complexity of the maximum satisfiability problem for Horn formulas*, **Inf. Proc. Letters, 26** (1987), 1-4.

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

Knuth, D.E.: **The Art of Computer Programming**, Addison-Wesley, Reading, MA (1968).

Levi, G. and F. Sirovich: *Generalized And/Or graphs*, **Artificial Intelligence, 7** (1976), 243-259.

Longo, G.: *Per una nuova teoria degli ipergrafi orientati*, **tesi di laurea**, Dip. Informatica, Univ. Pisa (1989).

Maier, D.: *Minimum covers in the relational data base model*, **J. Assoc. Comp. Mach., 27** (1980), 664-674.

Martelli, A. and U. Montanari: *Additive AND/OR graphs*, **Proc. IJCAI, 3** (1973), 1-11.

Martin, J.: **Computer Data-Base Organization**, Prentice-Hall, Englewood Cliffs, NJ (1977).

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

Nguyen, S. and S. Pallottino: *Equilibrium traffic assignment for large scale transit network*, **Eur. J. of Oper. Res., 37** (1988), 176-186.

Nguyen, S. and S. Pallottino: *Hyperpaths and shortest hyperpaths*, in: **Combinatorial Optimization** (B. Simeone, ed.)**, Lecture Notes in Mathematics, 1403**, Springer-Verlag, Berlin (1989), 258-271.

Nilsson, N.J.: **Problem Solving Methods in Artificial Intelligence**, McGraw-Hill, New York, NY (1971).

Nilsson, N.J.: **Principles of Artificial Intelligence**, Morgan Kaufmann, Los Altos, CA (1980).

Smith, H.C.: *Database design: composing fully normalized tables from a rigourous dependency diagram*,**Commun. ACM, 28** (1985), 826-838.

Torres, A.F. and J.D. Aráoz: *Combinatorial models for searching in knowledge bases*, **Mathematicas, Acta Cientifica Venezolana, 39** (1988) 387-394.

Ullman, J.D.: **Principles of Database Systems**, Computer Science Press, Rockville, MD (1982).

Yang, C.C.: **Relational Databases**, Prentice-Hall, Englewood Cliffs, NJ (1986).

Yang, C.C.: *Deduction graphs: an algorithm and applications*, **IEEE Tans. on Software Engng., 15** (1989) 60-67.