INFORMATIQUE THÉORIQUE ET APPLICATIONS

S. NGUYEN
D. PRETOLANI
L. MARKENZON

On some path problems on oriented hypergraphs

Informatique théorique et applications, tome 32, nº 1-3 (1998), p. 1-20.
<http://www.numdam.org/item?id=ITA_1998__32_1-3_1_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 32, n° 1-2-3, 1998, pp. 1-20)

ON SOME PATH PROBLEMS ON ORIENTED HYPERGRAPHS (*)

by S. NGUYEN (1), D. PRETOLANI (2), L. MARKENZON (3)

Communicated by G. AUSIELLO

Abstract. – *The BF-graphs form a particular class of Directed Hypergraphs. For this important family, different applications are known in data bases and artificial intelligence domain. They may also be used to describe the behaviour of concurrent systems. We present here a theoretical analysis of several hyperpath problems in BF-graphs, with emphasis on the acyclic BF-graphs. After briefly exposing the basic concepts of directed hypergraphs, we present an algorithm for finding a BF-path. We next discuss the problem of finding a hyperpath cover, and present a polynomial solution for two constrained hyperpath problems.* © Elsevier, Paris

Key-words: *directed hypergraphs, hyperpaths; constrained path problems.*

Résumé. – *Les BF-graphes forment une classe particulière d'hypergraphes orientés. Pour cette importante famille, nous connaissons des applications dans les domaines des bases de données et de l'intelligence artificielle. Ils peuvent aussi être utilisés pour modéliser les systèmes concurrents. Nous présentons une analyse théorique de quelques problèmes d'hyperchemin dans les BF-graphes, surtout pour les BF-graphes acycliques. Tout d'abord, nous décrivons les concepts fondamentaux des hypergraphes orientés, puis nous donnons un algorithme pour la recherche d'un BF-chemin. Ensuite, nous étudions le problème de recouvrement par des hyperchemins et, finalement, nous présentons des solutions polynomiales pour deux problèmes d'hyperchemin avec contraintes.* © Elsevier, Paris

Mots clés: *hypergraphes orientés, hyperchemins ; problèmes de chemin avec contraintes.*

1. INTRODUCTION

A commonly used approach in software validation consists in describing the structure of a program by a directed graph. The generation of *program*

(*) Received July 1993; revised October 1994; accepted October 1994.

(1) Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal (Québec) H3C 3J7.

(2) Dipartimento di Matematica e Fisica, Università di Camerino, via Madonna delle Carceri, 62032 Camerino (MC), Italy.

(3) Universidade Federal do Rio de Janeiro, Nucleo de Computação Electrônica, Caixa Postal 2324, Rio de Janeiro, Brazil.

0988-5004/98/01-02-03/

test paths gives rise to path problems, which are of interest from both theoretical and practical standpoints. For example, Gabow, Maheshwari and Osterweil (1976) and Ntafos and Hakimi (1979) discussed the search of a minimum path cover and the search of constrained paths. Ntafos and Hakimi (1981) proposed the use of structured graphs in structured programming, and presented new theoretical results. Ntafos and Gonzalez (1984) gave a review of known results for directed acyclic graphs.

Graphs can also be used to describe the behavior of concurrent systems (see for example Sahner and Trivedi (1987), El-Rewini and Lewis (1990) and Kapelnikov, Muntz and Ercegovac (1989)). The level of details varies in each model: a node in the graph can represent a simple machine instruction or a whole task. For this latter case, the *BF-graphs* –a particular type of *directed hypergraphs* (see Gallo *et al.*, 1993)– provide a powerful descriptive tool, since they offer a most natural way of modeling several features of parallel environments, such as multiple task precedence or simultaneous execution in a synchronous sytem.

Kapelnikov, Muntz and Ercegovac (1989) presented a new model for the performance evaluation of concurrent systems, in particular, for obtaining an approximation of the average execution time of parallel computation programs. To this aim, they defined a *Computation Control Graph* (CCG), where nodes represent tasks and arcs model precedence constraints between tasks. The existence of rules for classifying different types of arcs in a CCG in terms of the precedence relations, shows an interesting possible application of BF-graphs. Indeed, the above precedence relations can be easily modeled with BF-graphs.

Even though it is not our purpose to define a hypergraph-based model for parallel computations, we observe that in this framework several hyperpath problems may arise in the generation of *test hyperpaths* for parallel programs validation, or for the static performance evaluation. The above applications motivate the present analysis of several hyperpath problems. Our main goal is to develop efficient algorithms for polynomially solvable problems, and also to contrast our results with that obtained with directed graphs. In Section 2, we present an overview of the basic concepts of directed hypergraphs and hyperpaths. An algorithm for finding BF-paths is given in Section 3, while Section 4 considers some more difficult variants, namely the search of a BF-path cover. Finally, BF-paths with constraints are discussed in Section 5; polynomial algorithms for particular cases are given.

2. DEFINITIONS, NOTATION AND TERMINOLOGY

This Section introduces the hypergraph notation used throughout the paper. A more detailed introduction to directed hypergraphs can be found in Gallo *et al.* (1993).

A *directed hypergraph* is a pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of *hyperarcs*. A hyperarc $a \in \mathcal{E}$ is a pair $(Tail(a),$ $Head(a))$, where $Tail(a)$ and $Head(a)$ are disjoint subsets of $\mathcal{V}$. A hyperarc a is a *B-arc* (respectively an *F-arc*) if $|Head(a)| \leq 1$ (respectively if $|Tail(a)| \leq 1$). A *B-graph* (respectively an *F-graph*) is a hypergraph whose hyperarcs are B-arcs (respectively F-arcs). A *BF-graph* admits both B-arcs and F-arcs. Every hypergraph $\mathcal{H}$ can be transformed into a BF-graph by replacing each hyperarc a by the hyperarcs $(Tail(a), \{n_a\})$ and $(\{n_a)\}, Head(a))$ where n_a is a new node.

We denote by $BS(v) = \{a \in \mathcal{E} \mid v \in Head(a)\}$ and

$$FS(v) = \{a \in \mathcal{E} \mid v \in Tail(a)\}$$

respectively the *backward star* and the *forward star* of node u.

Let n and m be the number of nodes and hyperarcs in a BF-graph $\mathcal{H}$. We define the *size* of a BF-graph as follows:

$$size(\mathcal{H}) = |\mathcal{V}| + \sum_{a \in \mathcal{E}} (|Tail(a)| + |Head(a)|).$$

A BF-graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{a, b, c, d, 1, 2, 3, 4, s, t\}$ is shown in Figure 1.

A *path* P_{st} in a BF-graph is a sequence of the form:

$$P_{st} = (v_1 = s, e_1, v_2, e_2, \ldots, e_q, v_{q+1} = t)$$

where

- $v_1, v_2, \ldots, v_{q+1}$ are nodes in $\mathcal{V}$;
- $e_1, e_2, \ldots, e_q$ are hyperarcs in $\mathcal{E}$;
- $v_1 \in Tail(e_1)$, $v_{q+1} \in Head(e_q)$;
- $v_i \in Head(e_{i-1}) \cap Tail(e_i), i = 2, \ldots, q$.

A node v is *connected* to node u if there exists in $\mathcal{H}$ a path P_{uv}. A path is *simple* if all its hyperarcs are different; a path is *elementary* if all its nodes are different. If we have $s = t$, P_{st} is a *cycle*; if the BF-graph $\mathcal{H}$

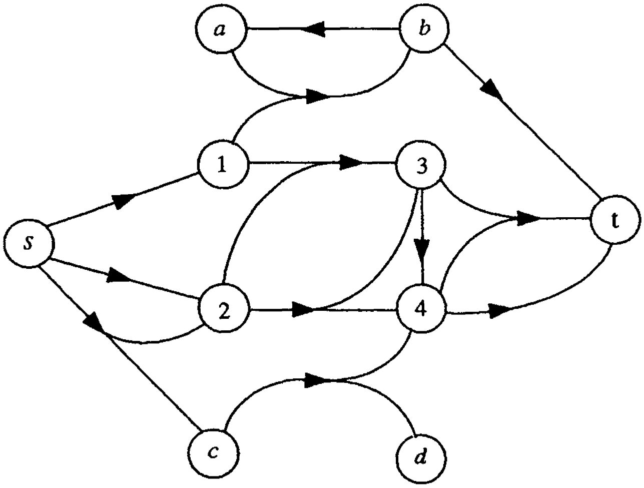


Figure 1. – A BF-graph.

contains no cycles, it is *acyclic*. A path P_{st} is called *backward cycle-free* if the following condition holds:

$$v_i \in Tail(e_j) \Rightarrow j \geq i;$$

and is called *forward cycle-free* if the following condition holds:

$$v_i \in Head(e_j) \Rightarrow j < i.$$

Given a BF-graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, it is possible to define different types of hyperpath. Gallo *et al.* (1993) distinguish the B-paths, the F-paths and the BF-paths.

A *B-hyperpath* π_{st} (or simply a *B-path*), of *origin* s and *destination* t, is a minimal (in the inclusion sense) BF-graph $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ satisfying the following conditions:

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

(ii) $s, t \in \mathcal{V}_\pi = \bigcup_{e \in \mathcal{E}_\pi} Tail(e) \cup Head(e)$;

(iii) $v \in \mathcal{V}_\pi \Rightarrow$ there exists in $\mathcal{H}_\pi$ a backward cycle-free path P_{sv}.

An *F-hyperpath* π_{st} (or simply an *F-path*), of *origin* s and *destination* t, is a minimal BF-graph $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ satisfying conditions (i) and (ii) together with the following one:

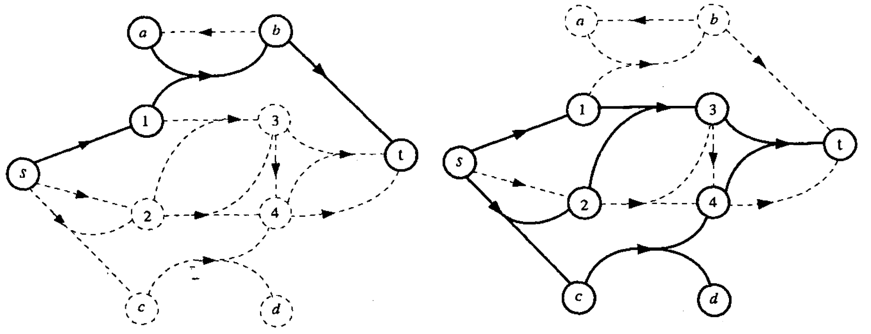
(iv) $v \in \mathcal{V}_\pi \Rightarrow$ there exists in $\mathcal{H}_\pi$ a forward cycle-free path P_{vt}.

A *BF-hyperpath* π_{st} (or *BF-path*), of *origin* s and *destination* t, is a minimal BF-graph $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ satisfying conditions (i)-(iv) above. We say that a node u is *B-connected* (*F-connected*, *BF-connected*) to node v in $\mathcal{H}$ if there exists in $\mathcal{H}$ a B-path (F-path, BF-path) π_{st}.

Consider the BF-graph in Figure 1; there is only one path connecting node a to s, namely $P_{sa} = (s, (\{s\}, \{1\}), 1, (\{1, a\}, \{b\}), b, (\{b\}, \{a\}), a)$, which is not backward acyclic. Therefore, a is not B-connected to s, nor is node b, as a consequence. Clearly, t is not F-connected to d. In Figure 2(a) and Figure 2(b) respectively, we show an F-path and a B-path connecting node t to node s in the BF-graph of Figure 1.

3. HYPERPATHS IN BF-GRAPHS

We consider first the problem of checking B-connection and F-connection. Following is a simplified version of procedure $Bvisit(s, \mathcal{H})$ (Gallo *et al.*, 1993) that determines the set of nodes B-connected to the origin node s in the BF-graph $\mathcal{H}$. For each node $u \in \mathcal{H}$, the label $blabel(u)$ is set to *true* if u has been visited, i.e. it is connected to s.



a b

Figure 2. – An F-path and a B-path.

```
procedure Bvisit(s, H)
   for_each u ∈ V do    blabel(u) := false;
   for_each e ∈ E do    T(e) := 0;
   Q := {s}; blabel(s) := true;
   while Q ≠ ∅ do
      select and remove u ∈ Q;
      for_each e ∈ FS(u) do
         T(e) := T(e) + 1;
         if T(e) = | Tail(e) |
         then for_each v ∈ Head(e) do
                  if blabel(v) = false
                  then blabel(v) := true;
                       Q := Q ∪ {v};
                  end_if
      end_for_each
   end_while
end_procedure
```

One can show that the complexity of $Bvisit(s, \mathcal{H})$ is $O(size(\mathcal{H}))$. Indeed, for each node $u \in Q$, the hyperarcs $e \in FS(u)$ will be examined. If e is a B-arc, then $T(e)$ will be updated at most $|Tail(e)|$ times; if e is an F-arc, $T(e)$ will be updated at most once, but then $|Head(e)|$ nodes will be examined.

In order to check F-connection, we can follow the same approach introduced for B-connection. Procedure $Fvisit(t, \mathcal{H})$ finds the set of nodes to which the destination t is F-connected. In this case, the visit of the BF-graph starts from the node t, and proceeds in a backward fashion. The label *flabel*(u) indicates that node u has been visited. Transforming procedure $Bvisit$ into $Fvisit$ is straightforward; similar to the previous procedure, the complexity of $Fvisit(t, \mathcal{H})$ is $O(size(\mathcal{H}))$.

Note that when the BF-graph $\mathcal{H}$ is a directed graph, $Bvisit$ and $Fvisit$ respectively perform a direct and reverse Breadth-First search, if the set of nodes Q is implemented as a queue. The traditional linear complexity is therefore achieved.

In order to devise a method for determining BF-paths, we must first introduce the notion of *frontier graph.* A frontier graph $\mathcal{H}' = (\mathcal{V}', \mathcal{E}', s, t)$ is a BF-graph in which:

– s and t are the *origin* and *destination* nodes;

– $v \in \mathcal{V}' \Rightarrow v$ is B-connected to s, and t is F-connected to v in $\mathcal{H}'$.

Given a BF-graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, and two nodes $s, t \in \mathcal{V}$, the *frontier graph of* $\mathcal{H}$ is a maximal (in the inclusion sense) frontier graph $\mathcal{H}' = (\mathcal{V}', \mathcal{E}', s, t)$ contained in $\mathcal{H}$.

Note that the frontier graph of a given BF-graph $\mathcal{H}$ is unique, and can be empty. The frontier graph of the BF-graph in Figure 1 is shown in Figure 3.

The following procedure *frontier*$(\mathcal{H}, \mathcal{H}', s, t)$ finds the frontier graph $\mathcal{H}' = (\mathcal{V}', \mathcal{E}', s, t)$ of a given BF-graph $\mathcal{H}$. At the beginning, one has $\mathcal{H}' = \mathcal{H}$. At each iteration, procedures $Bvisit$ and $Fvisit$ are executed. Each node u such that *blabel*(u) = *false* or *flabel*(u) = *false* is deleted, together with the hyperarcs in $FS(u) \cup BS(u)$. The procedure iterates until there is no more node to delete. If node s or node t is deleted, then the frontier graph is empty.

```
procedure frontier(H, H', s, t)
   H' := H; change := true;
   while change = true do
      change := false ;
      Bvisit(s, H'); Fvisit(t, H');
      for_each v ∈ V'
         if blabel(v) = false or flabel(v) = false
         then V' := V' − {v}; E' := E' − FS(v) − BS(v);
                change := true;
         end_if
      if s ∉ V' or t ∉ V'
      then { frontier graph empty}
            H' := ∅; change := false ;
      end_if
   end_while
end_procedure
```

In the worst case, only one node will be deleted at each iteration. Since the procedures $Bvisit$ and $Fvisit$ take $O(size(\mathcal{H}))$, and may be called at most n times, the complexity of procedure *frontier*$(\mathcal{H}, \mathcal{H}', s, t)$ is $O(n\, size(\mathcal{H}))$.

The detection of the frontier graph is a crucial step in the search of a BF-path. Observe that $\mathcal{H}'$ satisfies condition (i)-(iv) of the definition of BF-path, and hence, a BF-path from s to t, if it exists, is contained

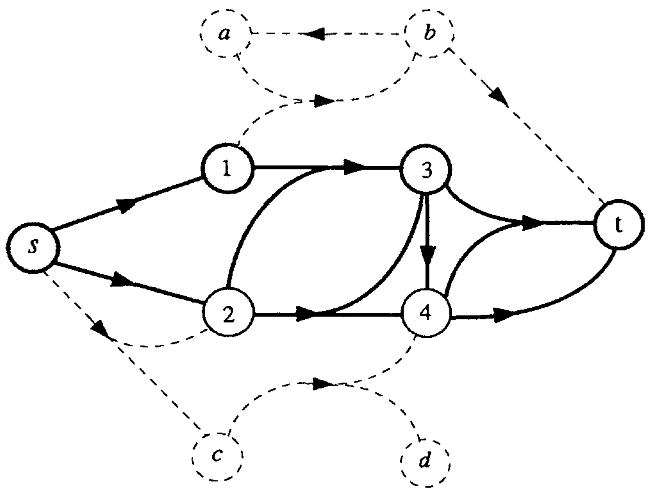


Figure 3. – A frontier graph.

in $\mathcal{H}'$. However, a non-empty frontier graph is not necessarily minimal, and finding a BF-path π_{st} may require a certain reduction by deleting nodes or hyperarcs. To this aim, one can proceed as follows. Let $\mathcal{H}''$ the BF-graph obtained from $\mathcal{H}'$ by deleting a node u. If $\mathcal{H}''$ contains a non-empty frontier graph $\mathcal{F}$, $\mathcal{H}'$ can be replaced by $\mathcal{F}$; otherwise, node u must be kept in $\mathcal{H}'$. This step is repeated for all nodes to produce a node minimal frontier graph $\mathcal{H}'$. Clearly, this takes an overall $O(n^2\, size(\mathcal{H}))$ time.

A similar technique can be adopted to produce an arc minimal frontier graph from $\mathcal{H}'$. A hypergraph $\mathcal{H}''$ is obtained by deleting hyperarc $e \in \mathcal{E}'$, and procedures $Bvisit$ and $Fvisit$ are then applied. If for some node $u \in \mathcal{V}'$ $blabel(u) = false$ or $flabel(u) = false$, then clearly e must be kept in $\mathcal{E}'$; otherwise, it can be deleted from $\mathcal{E}'$. This step is performed for all hyperarcs, with an overall $O(m\; size(\mathcal{H}))$ complexity, until one obtains a node minimal and arc minimal frontier graph $\mathcal{H}' = (\mathcal{V}', \mathcal{E}', s, t)$, corresponding to a BF-path π_{st}. Observe that depending on the order in which nodes and hyperarcs are considered different BF-paths π_{st} are identified.

The following procedure $BFpath(\mathcal{H})$ finds a BF-path in a BF-graph $\mathcal{H}$ (if one exists) in $O((n^2 + m)size(\mathcal{H}))$ time.

```
procedure BFpath(H)
   frontier(H, P, s, t);
   if P = ∅
   then { frontier graph empty}
         return(P);
   for_each u ∈ V such that u ≠ s, t do
      P' := P − {u};
      frontier(P', P'', s, t);
      if P'' ≠ ∅ then P := P'';
   end_for_each
   for_each e ∈ E such that e ∈ P do
      P' := P − {e};
      Bvisit(P'); Fvisit(P');
      if ∀u ∈ P' : blabel(u) = flabel(u) = true
      then P := P';
   end_for_each
   return(P);
end_procedure
```

In Figure 4 we show the four BF-paths from s to t contained in the BF-graph of Figure 1, that is, in the frontier graph of Figure 3.

For example, the BF-path in Figure 4(c) is obtained as follows: first, node 1 is deleted, obtaining a node minimal frontier graph; then, arc minimality is obtained by deleting hyperarcs ({3},{4}) and ({4},{t}).

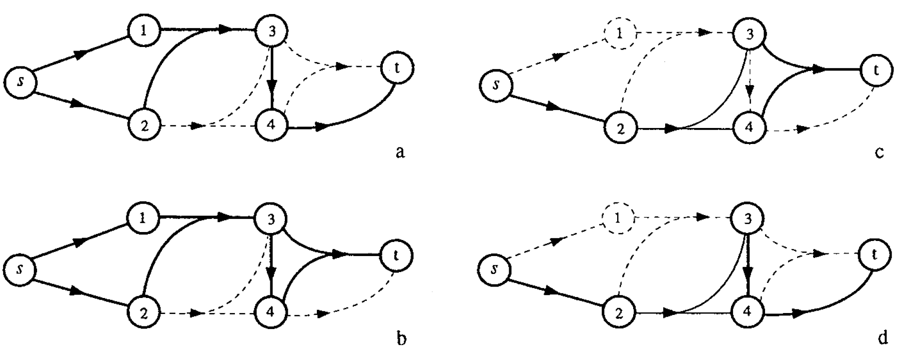


Figure 4. – BF-paths.

3.1. BF-paths in Acyclic BF-graphs

If the BF-graph $\mathcal{H}$ is acyclic, a significant simplification can be introduced into procedure *frontier*$(\mathcal{H}, \mathcal{H}', s, t)$. Indeed, if all the nodes in $\mathcal{H}$ are connected to s, then they are also B-connected to s, and similarly, if t is connected to all nodes, it is also F-connected to all of them. Hence, it is not necessary to apply procedures *Bvisit* and *Fvisit*: it suffices to delete each node u such that $FS(u) = \emptyset$, i.e. node t is not connected to u, and each node u such that $BS(u) = \emptyset$, i.e. u is not connected to s. After deleting the hyperarcs incident to these nodes, it is possible that additional nodes become disconnected. However, each node and hypearc will be considered at most once.

The proposed simplification is described in the following procedure *A_frontier.* The counters $nf(u)$ and $nb(u)$ give the current number of hyperarcs in $FS(u)$ and $BS(u)$, respectively.

```
procedure A_frontier(H, H', s, t)
   H' := H; Q := ∅;
   for_each u ∈ V do
      nf(u) :=| FS(u) |; nb(u) :=| BS(u) |;
      if ( nf(u) = 0  ∧ u ≠ t) or (nb(u) = 0  ∧ u ≠ s)
      then Q := Q ∪ {u}; V' := V' − {u};
      end_if
   end_for_each
   while Q ≠ ∅ do
      select and remove u ∈ Q;
      for_each e ∈ FS(u) ∪ BS(u) such that e ∈ E' do
         E' := E' − {e};
         for_each v ∈ Tail(e) do
            nf(v) := nf(v) − 1;
            if nf(v) = 0 and v ∈ V' − {t}
            then Q := Q ∪ {v}; V' := V' − {v};
            end_if
         end_for_each
         for_each v ∈ Head(e) do
            nb(v) := nb(v) − 1;
            if nb(v) = 0 and v ∈ V' − {s}
            then Q := Q ∪ {v}; V' := V' − {v};
            end_if
         end_for_each
         if s ∉ V' or t ∉ V'
         then H' := ∅; Q := ∅;
         end_if
```

```
    end_for_each
  end_while
end_procedure
```

Procedure *A_frontier* runs in $O(size(\mathcal{H}))$ time, and thus reduces the complexity by a factor $O(n)$. Therefore, it can be used to obtain a node minimal frontier graph $\mathcal{H}'$ in $O(n\ size(\mathcal{H}))$ time. To obtain a BF-path π_{st} from $\mathcal{H}'$, one can proceed as follows. Each hypearc $e \in \mathcal{H}'$ is examined, and is deleted from the BF-graph, unless it is the only hyperarc in $BS(u)$ or $FS(u)$ for some node u. Clearly, this produces an arc minimal frontier graph. The following procedure *A_minimal* performs the above simplification:

```
procedure A_minimal(H)
  H' := H; Q := ∅;
  for_each u ∈ V do
     nf(u) :=| FS(u) |; nb(u) :=| BS(u) |;
  end_for_each
  for_each e ∈ E do
     if (∀u ∈ Tail(e) :  nf(u) > 1) and (∀u ∈ Head(e) :  nb(u) > 1)
     then E := E − {e};
          for_each v ∈ Tail(e) do nf(v) := nf(v) − 1;
          for_each v ∈ Head(e) do nb(v) := nb(v) − 1;
     end_if
  end_for_each
end_procedure
```

It is easy to see that procedure *A_minimal* runs in linear $O(size(\mathcal{H}))$ time. In conclusion, the search of a BF-path in an acyclic BF-graph can be performed by a simplified version of procedure $BFpath$, that uses procedure *A_frontier* instead of *frontier*, and procedure *A_minimal*. The complexity of the resulting procedure is $O(n\ size(\mathcal{H}))$.

4. OTHER HYPERPATH PROBLEMS IN BF-GRAPHS

In this section we consider some polynomially solvable path problems on graphs, and show that the corresponding hyperpath problems are NP-complete.

Given a directed graph $G = (N, A)$, one may be interested in finding a path P_{st} containing a given set of nodes $V^* \in N$. This problem can be solved in linear time (see Gabow *et al.*, 1976), provided that P_{st} is not required to be elementary. If an elementary path is required, then the resulting problem is NP-complete, since it is equivalent to the Hamiltonian Path Problem (see Garey and Johnson, 1979). Observe that a BF-path in a graph is by definition elementary, since it is required to be *minimal.* It follows immediately that finding a BF-path containing a given set of nodes is an NP-complete problem.

Another well known graph problem is that of finding a *minimum path cover*, i.e. a minimum cardinality set of paths $P = \{p_1, p_2, \ldots, p_q\}$, of origin s and destination t, such that each node in the graph belongs to at least one path. Ntafos and Hakimi (1979) presented polynomial solutions for this problem. In the following, we will restrict ourselves to frontier graphs, in which by definition each node belongs to at least one BF-path from the origin to the destination. We shall prove that the problem of finding a minimum cardinality set of BF-paths covering a frontier graph $\mathcal{H}$ (*BF-path Cover*) is NP-complete, even if $\mathcal{H}$ is a B-graph.

Clearly, the problem is in the class NP; in order to prove the NP-completeness, we show a reduction from the *Vertex Cover* problem (see Garey and Johnson, 1979). Given an undirected graph $G = (N, A)$, a *cover* of G is a set $V \subseteq N$ such that every arc $(u, v) \in A$ has at least one endpoint in V. Vertex Cover may be formulated as follows: given G and an integer k, answer "yes" if and only if there exists a cover of G of cardinality not greater than k. Let $AD(u)$ be the set of arcs incident to node u in G, we can associate with G a B-graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ where:

– $\mathcal{V} = \{v_a | a \in A\} \cup \{s, t\}$;

– $\mathcal{E} = \{(\{s\}, \{v_a\}) | a \in A\} \cup \{a_u \ : \ u \in N\}$,

such that $a_u = (\{v_a \ : \ a \in AD(u)\}, \{t\})$;

Due to the particular structure of $\mathcal{H}$, a BF-path π_{st} in $\mathcal{H}$ contains a single hyperarc a_u, and the hyperarcs connecting the nodes in $Tail(a_u)$ to s. Hence, each cover V of G corresponds to a unique set of BF-paths covering the nodes in $\mathcal{H}$. The NP-completeness of *BF-path cover* follows immediately from the above observations.

A graph G and the corresponding B-graph $\mathcal{H}$ are shown in Figure 5. A minimum vertex cover in G is given by the set b, c and a minimum BF-path cover of $\mathcal{H}$ is given by two BF-paths containing the B-arcs b and c respectively.

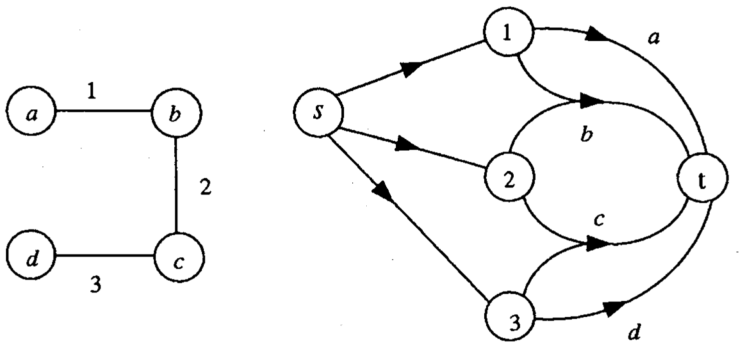


Figure 5. – Graph G and B-graph $\mathcal{H}$.

5. CONSTRAINED BF-PATHS

Let $G = (N, A)$ be a directed graph, and C a set of k unordered pairs (u, v), $u, v \in N$. Consider the following two path problems:

Impossible Pair Problem (IP): is there a path from an origin s to a destination t that contains at most one node of every pair $(u, v) \in C$?

Must Pair Problem (MP): is there a path from an origin s to a destination t that contains either both nodes or none of every pair $(u, v) \in C$?

For these problems, some results are known. Gabow *et al.* (1976) showed that the Impossible Pair problem is NP-complete even for acyclic directed graphs. The Must Pair problem is also NP-complete, even for structured graphs (Ntafos and Hakimi, 1981). Polynomial solution algorithms are known only for trees (Gabow *et al.*, 1976). Nevertheless, for particular cases of pairs, a polynomial algorithm can be devised.

Two nodes $u, v \in G$ are *independent* if G does not contain any path P_{uv} or P_{vu}. Node u is *strong for* v if u belongs to each path connecting v to the origin s, or connecting the destination t to v. A pair $u, v \in C$ is *stable* if either u and v are independent, or one of them is strong for the other. As observed by Markenzon and Szwarcfiter (1987) IP and MP are solvable in polynomial time if all the pairs in C are stable. Intuitively, it is easy to see that in an IP problem one can ignore a pair of independent nodes, while if node u is strong for node v then v must be deleted. In MP, two independent nodes must be deleted, and a pair such that each node is strong for the other can be ignored. Moreover, if only one of the nodes in a pair is strong for the other, it is possible to simplify G, without affecting the solution, so that each node becomes strong for the other. Observe that, due to the above simplification, a non-stable pair $(u, v) \in C$ can become stable. It is therefore possible to devise an algorithm which repeatedly searches and deletes a stable pair $(u, v) \in C$, possibly simplifying G. This algorithm is *partial*, in the sense that it either finds a solution or halts with C containing only non-stable pairs.

In the following, we generalize IP and MP to BF-paths in BF-graphs. We extend the definition of stable pairs to BF-graphs, devise solution techniques for stable pairs, and propose a polynomial partial algorithm. The proposed approach is then adapted for directed graphs.

5.1. Stability in BF-graphs

For the sake of simplicity, we formulate constrained BF-path problems on frontier graphs; the extension to general BF-graphs is immediate. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E}, s, t)$ be a frontier graph, and C a set of k unordered pairs (u, v), $u, v \in \mathcal{V}$. The following two problems can be formulated on $\mathcal{H}$:

Impossible Pair BF-path (IPBF): is there a BF-path π_{st} containing at most one node of every pair $(u, v) \in C$?

Must Pair BF-Path (MPBF): is there a BF-path π_{st} containing either both nodes or none of every pair $(u, v) \in C$?

The above two problems are NP-complete, since they correspond to IP and MP when $\mathcal{H}$ is a directed acyclic graph. To give a definition of *stable pairs* in BF-graphs, we introduce the concept of *necessity* and *independency*.

Denote by $\mathcal{H}_u$ the BF-graph obtained from $\mathcal{H}$ by deleting node u and the hyperarcs in $FS(u) \cup BS(u)$. Let $\mathcal{F}_u$ be the frontier graph of $\mathcal{H}_u$, obtained applying procedure *frontier*$(\mathcal{H}_u, \mathcal{F}_u, s, t)$. A node $u \in \mathcal{V}$ is *necessary for* node v if v does not belong to $\mathcal{F}_u$. Observe that if u is necessary for v then u belongs to each BF-path π_{st} containing v. Moreover, node u remains necessary for v if we delete some nodes and hyperarcs from $\mathcal{H}$: we say that the necessity property is preserved under the deletion operation.

We say that $\mathcal{H}$ is *splittable* if there exist $I (\geq 2)$ BF-graphs $\mathcal{H}^i = (\mathcal{V}^i, \mathcal{E}^i, s, t), 1 \leq i \leq I$, satisfying the following properties:

– $\mathcal{V} = \bigcup_{1 \leq i \leq I} \mathcal{V}^i$;

– $\mathcal{E} = \bigcup_{1 \leq i \leq I} \mathcal{E}^i$;

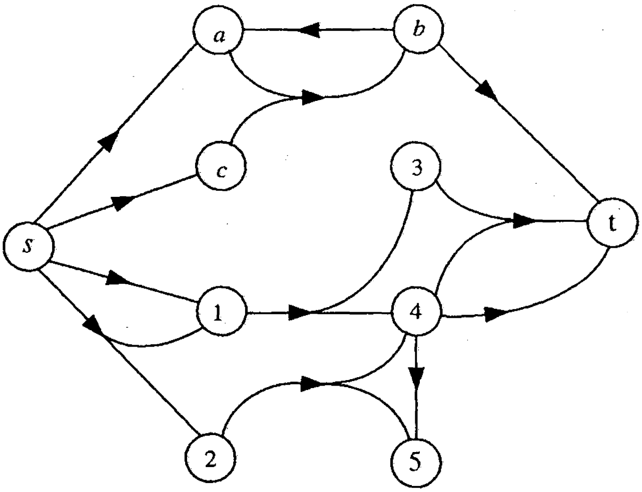


Figure 6. – Independent graphs.

– $\mathcal{V}^i \cap \mathcal{V}^j = \{s, t\}, 1 \leq i < j \leq I.$

The BF-graphs $\mathcal{H}^i$ are called *independent graphs*; two nodes are *independent* if they belong to different independent graphs of $\mathcal{H}$. Observe that, due to the minimality of BF-paths, it cannot exist a BF-path π_{st} containing a pair of independent nodes. More precisely, each BF-path π_{st} is contained in one independent graph. The independency property is preserved under the deletion operation as well. Indeed, if nodes or hyperarcs are deleted from $\mathcal{H}$, then a further splitting of some independent graphs may be possible, but independent nodes will remain independent.

An example of splittable BF-graph is given in Figure 6. The two independent graphs contain the set of nodes $\{s, t, a, b, c\}$ and $\{s, t, 1, 2, 3, 4, 5\}$ respectively.

The independent graphs of a BF-graph can be easily identified as follows. Initially, a node $u \in \mathcal{H}$ is inserted into a new independent graph $\mathcal{H}^i$; for each node $v \neq s$, t inserted into $\mathcal{H}^i$, the hyperarcs in $FS(v) \cup BS(v)$ and the nodes in these hyperarcs are inserted into $\mathcal{H}^i$. This process terminates when no more nodes can be inserted into $\mathcal{H}^i$, and is repeated until all nodes have been inserted into some independent graphs. The following procedure $IGraphs(\mathcal{H}, s, t)$ implements the above method, the label $In(u)$ takes the value i if node u is inserted into the independent graph $\mathcal{H}^i$.

```
procedure IGraphs(H, s, t)
   i = 0;
   for_each u ∈ V do In(u) := 0;
   In(s) := In(t) := n;
   S := V − {s, t}
   while S ≠ ∅ do
      select and remove u ∈ S;
      i = i + 1; In(u) := i;
      V^i = {u, s, t}; E^i = ∅;
      Q = {u};
      while Q ≠ ∅ do
         select and remove v ∈ Q;
         E^i := E^i ∪ FS(v) ∪ BS(v);
         for_each e ∈ FS(v) ∪ BS(v) do
            for_each v ∈ Tail(e) ∪ Head(e) do
               if In(v) = 0
               then V^i := V^i ∪ {v}; In(v) := i;
                      Q := Q ∪ {v}; S := S − {v};
               end_if
      end_while
   end_while
end_procedure
```

$IGraphs(\mathcal{H}, s, t)$ identifies the independent graphs of $\mathcal{H}$ in $O(size(\mathcal{H}))$ time.

Both IPBF and MPBF problems can be solved efficiently when they involve pairs of independent or necessary nodes. Consider the IPBF problem first. If for a pair $(u, v) \in C$ node u is necessary for v, then node v can be deleted, since it does not exist a BF-path π_{st} containing v and not containing u. Both nodes can be deleted if each of them is necessary for the other. Moreover, if u and v are independent, the pair (u, v) can be ignored, since u and v will not belong to the same BF-path. Similarly, for the MPBF problem, a pair $(u, v) \in C$ can be ignored if each node in the pair is necessary for the other. Moreover, if u and v are independent, then they can be deleted, since it does not exist a BF-path containing both of them.

The above observation suggests a possible extension of the concept of stable pairs to BF-graphs. However, the case in which only one node in a pair is necessary for the other is treated in a different way for IPBF and MPBF. As a consequence, the definition of stable pair in a BF-graph depends on the problem considered. In particular, we give the following two definitions:

- a pair (u, v) is *stable for IPBF problem* if either u and v are independent, or (at least) one of them is necessary for the other;
- a pair (u, v) is *stable for MPBF problem* if either u and v are independent, or each one of them is necessary for the other.

5.2. An Algorithm for Stable Pairs

The above definitions allow the extension to BF-graphs of the solution approach for stable pairs in graphs. A partial algorithm for the IPBF or MPBF problem may be described as follows:

Algorithm Stable-Pairs

1. If C is empty, go to Step (4); otherwise, if C does not contain a stable pair then return "not solvable";
2. Let S be the set of stable pairs in C; set $C = C - S$, and find the set $\mathcal{V}_S$ of nodes to be deleted from $\mathcal{H}$;
3. Delete from $\mathcal{H}$ the nodes in $\mathcal{V}_S$, and replace $\mathcal{H}$ with its frontier graph; if $\mathcal{H}$ is empty, return "no"; otherwise go to Step (1);
4. If t is BF-connected to s in $\mathcal{H}$ then return "yes"; else return "no".

Note that the definiton of stable pair, and the set of nodes $\mathcal{V}_S$ found in Step 2, depend on the particular problem considered. Note also that a pair $(u, v) \in C$ can be considered stable if either u or v has been deleted. For each pair $(u, v) \in S$, the nodes inserted in $\mathcal{V}_S$ can be determined as follows:

Problem IPBF

- if u is necessary fo v, then $v \in \mathcal{V}_S$;
- if v is necessary for u, then $u \in \mathcal{V}_S$.

Problem MPBF

- if (u, v) are independent, then $u \in \mathcal{V}_S$ and $v \in \mathcal{V}_S$;
- if $v \notin \mathcal{H}$ then $u \in \mathcal{V}_S$; if $u \notin \mathcal{H}$ then $v \in \mathcal{V}_S$.

Recall that each stable pair remains stable after the execution of Step 3, therefore, the algorithm correctly solves the problem when the set C contains only stable pairs. If the answer is "yes", a valid BF-path π_{st} can be found in the resulting BF-graph $\mathcal{H}$.

The following proposition establishes the complexity of algorithm *Stable-Pairs*, where $k = |C|$.

PROPOSITION 1:
Algorithm Stable-Pairs terminates in $O(min(n,k)n\ size(\mathcal{H}))$ time.

Proof:

First, observe that each step is performed at most $min\{k,n\}$ times, and that Step 1 and Step 2 can be implemented in $O(k)$ time if pairs of necessary and independent nodes are known. Moreover, Step 3 takes an overall $O(n\ size(\mathcal{H}))$ time. Indeed, in the whole sequence of updating of $\mathcal{H}$, procedures $Bvisit(s,\mathcal{H})$ and $Fvisit(t,\mathcal{H})$ may be applied at most n times. The crucial part of the algorithm is the detection of necessary and independent nodes, we now show that this task can be performed with an overall $O(min(n,k)n\ size(\mathcal{H}))$ complexity.

Consider first the search of independent nodes. The independent graphs of $\mathcal{H}$ can be found in $O(size(\mathcal{H}))$ time, and this search needs to be repeated at most $min\{k,n\}$ times. The resulting overall cost is $O(min\{k,n\}\ size(\mathcal{H}))$.

In order to check that node u is necessary for node v, we build the frontier graph $\mathcal{F}_u$ of the hypergraph $\mathcal{H}_u$ obtained from $\mathcal{H}$ by deleting u. When $\mathcal{H}$ is modified in Step 3, we can directly delete the nodes in $\mathcal{V}_S$ from $\mathcal{F}_u$, and then replace $\mathcal{F}_u$ by its frontier graph. The sequence of updating of $\mathcal{F}_u$ takes an overall $O(n\ size(\mathcal{H}))$ time. If two hypergraphs $\mathcal{F}_u$ and $\mathcal{F}_v$ are built for each pair $(u,v) \in C$, we can determine necessary nodes in $O(kn\ size(\mathcal{H}))$. Actually, it is not necessary to build more than n BF-graphs, one for each node in $\mathcal{H}$, therefore, we can find pairs of necessary nodes with an overall $O(min\{n,k\}n\ size(\mathcal{H}))$ cost.

Since we can assume k is $O(n^2)$, and $size(\mathcal{H}) \geq n$, the cost of finding necessary nodes dominates the cost $O(min\{n,k\}(k + size(\mathcal{H})))$ of the remaining operations, and this completes the proof. $\square$

Observe that the complexity of *Stable-Pairs* is dominated by the cost of finding a BF-path: $O((n^2 + m)size(\mathcal{H}))$.

5.3. Acyclic BF-graphs and Directed Graphs

When acyclic BF-graphs are considered, a better time bound can be obtained. In fact, by adapting the technique used in procedure *A_frontier*, we can maintain $\mathcal{H}$, and each frontier graph $\mathcal{F}_u$, with an $O(size(\mathcal{H}))$ cost. This gives an overall $O(n\ size(\mathcal{H}))$ cost for finding necessary nodes and leads to an $O(min\{n,k\}(k + size(\mathcal{H})))$ complexity.

The approach of algorithm *Stable-Pairs* can be applied to directed graphs. Assume that the BF-graph $\mathcal{H}$ is a graph, it is easy to see that a node u is strong for v in $\mathcal{H}$ if and only if u is necessary for v. Hence we can find strong nodes with an overall $O(min(n, k)n\,m)$ cost, where $m = |\mathcal{E}|$, since $size(\mathcal{H})$ is $O(m)$. In order to check whether two nodes u, v are independent, we must search two paths P_{uv} and P_{vu}: this can be done in $O(m)$ time by a standard Breadth First visit of the graph (e.g. by procedure $Bvisit$). Observe that, in order to check each pair $(u, v) \in C$, no more than $min\{2k, n\}$ visits are needed. This step must be performed at most $min\{n, k\}$ times, with an overall $O(min\{n, k\}^2 m)$ cost. Hence, the complexity of the graph version of *Stable Pairs* is $O(min\{n, k\}n\,m)$.

A better result can be obtained for acyclic directed graphs. In this case we can find strong nodes with an overall $O(min\{n, k\}m)$ cost. In order to find independent nodes, we can proceed as follows. Let G_u the BF-graph obtained from $\mathcal{H}$ by deleting the nodes not connected to u. Each time a node is deleted from $\mathcal{H}$, we update G_u accordingly, using the same technique adopted in procedure *A_frontier*. If the backward star of a node v in G_u becomes empty, node v is no longer connected to u in $\mathcal{H}$, and is deleted from G_u. It is easy to see that G_u can be maintained with an overall $O(m)$ cost. Again, we need to build at most $min\{n, k\}$ graphs to solve the problem. Therefore, we can implement *Stable Pairs* with an overall $O(min\{n, k\}(k + m))$ complexity.

It is worth noting that for acyclic BF-graphs, when $k > size(\mathcal{H})$ ($k > m$ for graphs) the complexity of the algorithm is determined by the cost of finding the set S in Step 2. This task can be performed with an overall $O(n^2)$ cost by using a *table of pairs*, of size $n \times n$, and by keeping a list of stable pairs instead of scanning the whole set C at each iteration. The details of the implementation are rather straightforward, and are omitted here. As a consequence, we can implement *Stable-Pairs* with an overall $O(min\{k, n\}size(\mathcal{H}))$ complexity ($O(min\{k, n\}m)$ for graphs). Note that the cost of finding a BF-path in an acyclic BF-graph is $O(n\,size(\mathcal{H}))$.

6. CONCLUSIONS

Hypergraphs are promising tools for describing parallel environments. In this paper, we considered some hyperpath problems related to these applications. First, we devised a polynomial algorithm for finding BF-paths in BF-graphs. Then we presented some NP-complete problems, such as *BF-path Cover*, whose analogues for graphs are known to be polynomially solvable. Finally, we considered constrained BF-paths, and extended to BF-graphs some NP-complete constrained path problems. We presented a polynomial algorithm for special cases of constraints; improved versions of this algorithm were devised for acyclic BF-graphs and (general or acyclic) directed graphs.

The problems treated here might be restated in terms of B-graphs (or F-graphs) instead of BF-graphs; our solution techniques can be easily adapted to these cases. For example, consider the problem of finding a B-path π_{st}, where t is a node B-connected to s. A suitable version of procedure *BFpath* finds a B-path π_{st} in $O(nsize(\mathcal{H}))$ time ($O(size(\mathcal{H}))$ for acyclic BF-graphs). Note that this problem was not considered in Gallo *et al.* (1993).

ACKNOWLEDGEMENTS

This research was partially supported by grants from the National Sciences and Engineering Research Council of Canada and a postdoctoral fellowship from the Ministère de l'Enseignement Supérieur et de la Science du Québec. We would like to thank Stefano Pallottino and the referees for their helpful comments that led to important improvements of an earlier version of this paper.

REFERENCES

1. H. El-Rewini and T. G. Lewis, Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing*, *9* (1990), pp. 138-153.
2. H. N. Gabow, S. N. Maleshwari and L. J. Osterweil, On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering*, SE-2 (1976), pp. 227-231.
3. G. Gallo, G. Longo, S. Nguyen and S. Pallottino, Directed hypergraphs and applications. *Discrete Applied Mathematics*, *42*(2-3) (1993), pp. 177-201.
4. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, Ca., 1979.
5. A. Kapelnikov, R. R. Muntz and M. D. Ercegovac, A modeling methodology for the analysis of cuncurrent systems and computations. *Journal of Parallel and Distributed Computing*, *6* (1989), pp. 568-597.
6. L. Markenzon and J. L. Szwarcfiter, Dois problemas de caminhos com restrições. In: *XX Simpósio Brasileiro de Pesquisa Operacional*, page 148, 1987.
7. S. C. Ntafos and T. Gonzalez, On the computational complexity of path cover problems. *Journal of Computer and System Science*, *29* (1984), pp. 225-242.
8. S. C. Ntafos and S. L. Hakimi, On path cover problems in digraphs and applications to program testing. *IEEE Transactions on Software Engineering*, SE-5 (1979), pp. 520-529.
9. S. C. Ntafos and S. L. Hakimi, On structured digraphs and program testing. *IEEE Transactions on Computer*, *c-30* (1981), pp. 67-77.
10. R. A. Sahner and K. S. Trivedi, Performance and reliability analysis using directed acyclic graphs. *IEEE Transactions on Software Engineering*, SE-13 (1987), pp. 1105-1114.