



ELSEVIER

Available online at [www.sciencedirect.com](http://www.sciencedirect.com)

SCIENCE @ DIRECT®

Computers & Operations Research 32 (2005) 1477–1497

computers &  
operations  
research

[www.elsevier.com/locate/dsw](http://www.elsevier.com/locate/dsw)

## Finding the $K$ shortest hyperpaths

Lars Relund Nielsen<sup>a</sup>, Kim Allan Andersen<sup>b,\*</sup>, Daniele Pretolani<sup>c</sup>

<sup>a</sup>*Department of Operations Research, University of Aarhus, Ny Munkegade, building 530,  
DK-8000 Aarhus C, Denmark*

<sup>b</sup>*Department of Management Science and Logistics, Aarhus School of Business, Fuglesangs Allé 4,  
DK-8210 Aarhus V, Denmark*

<sup>c</sup>*Dipartimento di Matematica e Informatica, Università di Camerino, Via Madonna delle Carceri,  
I-62032 Camerino (MC), Italy*

### Abstract

The  $K$  shortest paths problem has been extensively studied for many years. Efficient methods have been devised, and many practical applications are known. Shortest hyperpath models have been proposed for several problems in different areas, for example in relation with routing in dynamic networks. However, the  $K$  shortest hyperpaths problem has not yet been investigated.

In this paper we present procedures for finding the  $K$  shortest hyperpaths in a directed hypergraph. This is done by extending existing algorithms for  $K$  shortest loopless paths. Computational experiments on the proposed procedures are performed, and applications in transportation, planning and combinatorial optimization are discussed.

© 2003 Elsevier Ltd. All rights reserved.

*Keywords:* Network programming; Directed hypergraphs;  $K$  shortest hyperpaths;  $K$  shortest paths

### 1. Introduction

One classical problem encountered in the analysis of networks is the ranking of paths in non-decreasing order of length, known as  $K$  shortest paths. As early as 1959 attention was drawn to this problem [1]. Usually, two different situations are distinguished.

In the general (unrestricted) problem the paths are allowed to be *looping*, i.e. to contain cycles. Several techniques, based e.g. on dynamic programming or sophisticated data structures, have been applied to this problem, obtaining algorithms that are fast from a practical as well as theoretical point of view; see for example the recent results in [2,3].

\* Corresponding author. Fax: +45-894-866-60.

E-mail addresses: [kia@asb.dk](mailto:kia@asb.dk) (K.A. Andersen), [daniele.pretolani@unicam.it](mailto:daniele.pretolani@unicam.it) (D. Pretolani).

The restricted problem where only *loopless* paths are accepted is considered to be harder to solve. In practice, solution methods proposed so far are based on the branching approach by Yen [4], later discussed by Lawler [5] in the more general framework of finding the  $K$  best solutions to a discrete optimization problem.

The applications of the  $K$  shortest paths problem are numerous. First, practical problems often include constraints which are hard to specify formally or hard to optimize. Here an optimal solution can be found by enumerating suboptimal paths until a path satisfying the hard constraints is found. Second, by computing more than one shortest path, one can to a certain extent determine how sensitive the optimal solution is to variations of the parameters in the model. Last but not least, the  $K$  shortest paths problem often appears as a subproblem within algorithms for the bicriterion shortest path problem, see for example [6,7]. A complete survey of the existing literature on  $K$  shortest paths does not fall into the scope of this paper; the interested reader is referred to the work of Eppstein [2].

Directed hypergraphs are an extension of directed graphs, and have often been used in several areas as a modelling and algorithmic tool. A technical as well as historical introduction to directed hypergraphs has been given by Gallo et al. [8]. Hyperpaths in hypergraphs are a non-trivial extension of directed paths whose expressive power allows us to deal with more complex situations. In fact, several applications of shortest hyperpath methods are known. Similar to what is discussed above for  $K$  shortest paths, applications and solution methods based on shortest hyperpaths would take advantage of the availability of alternate optimal or sub-optimal solutions. However, to the authors' knowledge, no one has considered the problem of finding the  $K$  shortest hyperpaths. Possible applications of  $K$  shortest hyperpath algorithms are numerous.

A shortest hyperpath model has been proposed for routing problems in discrete *random time-dependent networks (RTD networks)*, where the travel time through an arc is a random variable whose distribution depends on the departure time. Problems on random time-dependent networks related to applications such as hazardous material transportation or packet routing in congested communication networks have recently attracted a growing attention [9,10]. In these contexts, it is relevant to provide alternate solutions to allow for real-time routing decisions. Hall [11] introduced the problem of finding the minimum expected travel time (*MET*) through a RTD network. He pointed out that the best route does not necessarily correspond to an origin-destination path. Instead, a *strategy* must be found that assigns optimal successors to a node as a function of time. As shown in [12], directed hypergraphs can be used to model discrete RTD networks, and the MET problem can be reduced to solving a shortest hyperpath problem in a suitable acyclic *time-expanded* hypergraph. A deep computational analysis of hypergraph algorithms for the MET problem can be found in [9]. In addition, the hypergraph model shows a high degree of flexibility. Optimal strategies under different objectives, such as min–max travel time, min expected cost and min–max cost, can be found by using suitable weights and weighting functions [12]. Often in a real application hard constraints not intercepted by the model may occur. In this case, an optimal strategy may be found by enumerating suboptimal strategies until the hard constraints are satisfied. Here the  $K$  best strategies can be obtained by finding the  $K$  shortest hyperpaths in the time-expanded hypergraph.

Algorithms based on  $K$  shortest paths procedures have been proposed for the *bicriterion* shortest path problem. Assume that two *criteria*, e.g. time and cost, are associated with each arc of a graph. In general, there does not exist a path that is optimal for both criteria. Instead, a decision maker would be interested in finding *efficient* paths, that is solutions where the cost (respectively, time) criterion cannot be improved without getting a worse time (respectively, cost). A possible

approach to these problems is based on a *two-phase* method [13]. Here first phase finds a subset of the efficient paths (“supported” paths). In the second phase, a  $K$  shortest paths procedure is used to find all of (or some of) the remaining efficient paths. Clearly, the bicriterion problem can be extended to hypergraphs, and solved by a two-phases approach. In this case, a  $K$  shortest hyperpaths procedure would be used in the second phase. The bicriterion shortest hyperpath problem may be quite relevant in the context of RTD networks. For instance, in hazardous material transportation one may be interested in minimizing both expected travel time and expected risk. These problems have recently been investigated in [14]; a related problem has been considered in [10].

Propositional satisfiability problems represent another research area where directed hypergraphs are widely used, see e.g. [8,15]. For instance, the *maximum satisfiability* problem for Horn formulas (Max Horn SAT) turns out to be equivalent to the problem of finding a *minimum cut* in a directed hypergraph (MCH) [15]. Since Max Horn SAT is NP-hard, also MCH is, opposed to the well-known minimum cut problem in graphs. A *branch and cut* algorithm for MCH has been devised in [15]. This algorithm is based on a particular *cut generation* technique, that requires to find hyperpaths with a *cost* less than one, where the *cost* of a hyperpath is the sum of the weights of its hyperarcs. It has been proved in [16] that the problem of finding a minimum cost  $s$ - $t$  hyperpath is strongly NP-hard. Therefore, finding hyperpaths with a cost less than one is in general hard. In order to overcome this difficulty, a quite simple heuristic is adopted in [15], that consists in computing shortest hyperpaths for the *sum* and *distance* weighting functions; as we shall see, this can be done in polynomial time. Even though this heuristic performs well for some classes of instances, more sophisticated techniques seem to be necessary to improve the effectiveness of the algorithm. To this aim,  $K$  shortest hyperpaths procedures can be used. Since the sum gives an upper bound on the cost, one may enumerate the hyperpaths with sum less than one. This defines a (possibly empty) set of cuts that can be generated “easily”. Furthermore, the distance function gives a lower bound on the cost. Hence finding all the hyperpaths with a distance less than one provides a superset of the valid cuts.

Besides the specific application to cut generation discussed above, it is apparent that a  $K$  shortest hyperpaths procedure can be used to find a minimum cost  $s$ - $t$  hyperpath. More precisely, we can rank hyperpaths by distance, keeping track of the minimum cost hyperpath generated in the process. The procedure terminates as soon as the distance function of the next ranked hyperpath is greater than or equal to the minimum cost found so far. Here we exploit again the fact that the distance is a lower bound on the cost.

Algorithms for finding the  $K$  shortest hyperpaths may also be used to solve *minimum makespan assembly problems*. As shown in [17], an assembly line can be represented by a suitable acyclic directed hypergraph, where each hyperarc represents a machine operation linking two or more *sub-assemblies* together. A hyperpath thus represents a particular *assembly plan*. Assuming that each operation has a cost as well as an execution time, shortest hyperpaths with respect to the sum and distance weighting functions give assembly plans with minimum total cost or minimum execution time (with an unlimited number of machines) respectively. Observe that a “good” assembly plan should represent a trade-off between execution time and cost; clearly, this is related to the aforementioned bicriterion shortest hyperpath problem. In general, scheduling a given assembly plan on a fixed number of machines in order to minimize its *makespan* is a hard problem; approximated methods are discussed in [17]. A possible approach for refining these methods would be to generate several candidate assembly plans; an “optimal” plan would then be chosen according to its approximated minimum makespan scheduling, possibly taking into account other objectives.

A further application deserves to be mentioned, related to the hypergraph model for *transit networks* proposed by Nguyen and Pallottino [8,18]. Transit networks consist of a set of *bus lines* connected to *stop nodes* where passengers board or unboard buses. In the hypergraph model, a hyperarc represents the set of *attractive* bus lines for a passenger waiting at a stop node; a shortest hyperpath represents a set of attractive origin-destination routes. The hypergraph model is embedded within a *traffic assignment* model, based on Wardrop's equilibrium, where passengers are assumed to travel along their shortest available hyperpaths. In the context of iterative methods for traffic assignment, it may be computationally useful to identify alternate optimal hyperpaths by using a  $K$  shortest hyperpath procedure.

In this paper we propose algorithms for the  $K$  shortest hyperpaths problem. Moreover computational test are performed on these algorithms. Since hyperpaths in our context are acyclic, we extend to directed hypergraphs Yen's method for loopless paths; as we shall see, this extension is not straightforward.

The paper is organized as follows. Directed hypergraphs are introduced in Section 2. In Section 3 different procedures to find the  $K$  shortest hyperpaths are developed. Computational results are reported in Section 4. Finally, we summarize original contributions and topics for further research in Section 5.

## 2. Directed hypergraphs

A *directed hypergraph* is a pair  $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ , where  $\mathcal{V} = (v_1, \dots, v_n)$  is the set of nodes, and  $\mathcal{E} = (e_1, \dots, e_m)$  is the set of *hyperarcs*. A hyperarc  $e \in \mathcal{E}$  is a pair  $e = (T(e), h(e))$ , where  $T(e) \subset \mathcal{V}$  denotes the *tail nodes* and  $h(e) \in \mathcal{V} \setminus T(e)$  denotes the *head node*. The *cardinality* of a hyperarc  $e$  is the number of nodes it contains, i.e.  $|e| = |T(e)| + 1$ . If  $|e| = 2$ , hyperarc  $e$  is an *arc*. The *size* of  $\mathcal{H}$  is the sum of the cardinalities of its hyperarcs:

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

Without loss of generality, we assume  $size(\mathcal{H}) > n$ . We denote by

$$FS(u) = \{e \in \mathcal{E} \mid u \in T(e)\}, \quad BS(u) = \{e \in \mathcal{E} \mid u = h(e)\},$$

the *forward star* and the *backward star* of node  $u$ , respectively. A hypergraph  $\tilde{\mathcal{H}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$  is a *sub-hypergraph* of  $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ , if  $\tilde{\mathcal{V}} \subseteq \mathcal{V}$  and  $\tilde{\mathcal{E}} \subseteq \mathcal{E}$ . This is written  $\tilde{\mathcal{H}} \subseteq \mathcal{H}$  or we say that  $\tilde{\mathcal{H}}$  is *contained* in  $\mathcal{H}$ . A *path*  $P_{st}$  in a hypergraph  $\mathcal{H}$  is a sequence of nodes and hyperarcs in  $\mathcal{H}$ :

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

where, for  $i = 1, \dots, q$ ,  $v_i \in T(e_i)$  and  $v_{i+1} = h(e_i)$ . A node  $t$  is connected to node  $s$  if a path  $P_{st}$  exists in  $\mathcal{H}$ . A *cycle* is a path  $P_{st}$ , where  $t \in T(e_1)$ . This is in particular true if  $t = s$ . If  $\mathcal{H}$  contains no cycles, it is *acyclic*.

**Definition 1.** Let  $\mathcal{H} = (\mathcal{V}, \mathcal{E})$  be a hypergraph. A *valid ordering* in  $\mathcal{H}$  is a topological ordering of the nodes

$$V = \{u_1, u_2, \dots, u_n\}$$

such that, for any  $e \in \mathcal{E}$ :  $(u_j \in T(e)) \wedge (h(e) = u_i) \Rightarrow j < i$ .

Notice that, in a valid ordering any node  $u_j \in T(e)$  precedes node  $h(e)$ . The next theorem is a generalization of a similar result for acyclic directed graphs, see [19].

**Theorem 1.**  $\mathcal{H}$  acyclic  $\Leftrightarrow$  A valid ordering of the nodes in  $\mathcal{H}$  is possible.

Theorem 1 is proven in [8], where an  $O(\text{size}(\mathcal{H}))$  algorithm finding a valid ordering of the nodes in an acyclic hypergraph is presented. It should be noticed that a valid ordering in general is not unique, which is also the case for acyclic directed graphs.

### 2.1. Hyperpaths and hypertrees

Consider a hypergraph  $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ . A hyperpath  $\pi_{st}$  of origin  $s$  and destination  $t$ , is an acyclic minimal hypergraph (with respect to deletion of nodes and hyperarcs)  $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$  satisfying the following conditions:

1.  $\mathcal{E}_\pi \subseteq \mathcal{E}$
2.  $s, t \in \mathcal{V}_\pi = \bigcup_{e \in \mathcal{E}_\pi} (T(e) \cup \{h(e)\})$
3.  $u \in \mathcal{V}_\pi \setminus \{s\} \Rightarrow u$  is connected to  $s$  in  $\mathcal{H}_\pi$ .

Note that condition 3 implies that, for each  $u \in \mathcal{V}_\pi \setminus \{s\}$ , there exists a hyperarc  $e \in \mathcal{E}_\pi$ , such that  $h(e) = u$ . It follows from minimality that  $e$  is unique; hyperarc  $e$  is the predecessor of  $u$  in  $\pi_{st}$ . Conversely, condition 3 can be replaced by condition 4 below, where  $\mathcal{N} = \mathcal{V}_\pi \setminus \{s\}$ . Minimality also implies that, for any node  $u \in \mathcal{V}_\pi \setminus \{t\}$ , there is at least one hyperarc  $h \in \mathcal{E}_\pi$  such that  $u \in T(h)$ , thus there is a  $u$ - $t$  path in  $\pi_{st}$ . We say that node  $t$  is hyperconnected to  $s$  in  $\mathcal{H}$  if there exists in  $\mathcal{H}$  a hyperpath  $\pi_{st}$ .

Let  $s \in \mathcal{V}$  be a given root node and let  $\mathcal{N} \subseteq \mathcal{V} \setminus \{s\}$  be a set of nodes hyperconnected to  $s$ .

**Definition 2.** A directed hypertree with rootnode  $s$  is an acyclic hypergraph  $\mathcal{T}_s = (\{s\} \cup \mathcal{N}, \mathcal{E}_{\mathcal{T}})$  with  $s \notin \mathcal{N}$  such that:

4.  $BS(s) = \emptyset; |BS(v)| = 1 \ \forall v \in \mathcal{N}$ .

Note that a directed hypertree is the union of hyperpaths from  $s$  to all nodes in  $\mathcal{N}$ . A hypertree  $\mathcal{T}_s = (\{s\} \cup \mathcal{N}, \mathcal{E}_{\mathcal{T}})$  in a hypergraph  $\mathcal{H}$  is defined by a predecessor function  $p: \mathcal{V} \rightarrow \mathcal{E}$ ; for each  $u \in \mathcal{N}$ ,  $p(u)$  is the unique hyperarc in  $\mathcal{T}_s$  which has node  $u$  as the head. A sub-hypertree (or simply a subtree) of a hypertree  $\mathcal{T}_s$  is a hypertree with root  $s$  contained in  $\mathcal{T}_s$ . Note that any hyperpath is a hypertree, in particular it can be defined by a predecessor function. Moreover, different hypertrees can share the same hyperpath  $\pi_{st}$  as a subtree.

We point out some relevant differences between paths in directed graphs and hyperpaths in hypergraphs. Assume that a path  $P_{st}$  from node  $s$  to node  $t$  in a directed graph  $G = (N, A)$  is known:

$$P_{st} = (s, a_1, u_1, a_2, \dots, a_q, u_q = t).$$

Clearly,  $P_{st}$  is the concatenation of a subpath  $P_{su_i}$  from node  $s$  to node  $u_i$  and a subpath  $P_{u_it}$  from node  $u_i$  to node  $t$ , where

$$P_{su_i} = (s, a_1, u_1, a_1, \dots, a_i, u_i),$$

$$P_{u_it} = (u_i, a_{i+1}, \dots, a_q, u_q = t).$$

In other words, for each  $1 \leq i \leq q + 1$  we can *split* path  $P_{st}$  into the subpaths  $P_{su_i}$  and  $P_{u_it}$ .

Unfortunately, this need not be so for hyperpaths. In general a hyperpath  $\pi_{st}$  is not the concatenation of two hyperpaths  $\pi_{su}$  and  $\pi_{ut}$  (consider e.g.  $\pi_{st}$  in Example 1). However, we can define a “splitting” operation on a hyperpath  $\pi_{st} = (\mathcal{V}_\pi, \mathcal{E}_\pi)$  as follows. Recall that  $\pi_{st}$  is a hypertree  $\mathcal{T}_s$  defined by a predecessor function  $p$ .

**Definition 3.** Let  $V = \{s, u_1, u_2, \dots, u_{q+1} = t\}$  be a valid ordering of hyperpath  $\pi_{st}$ . For each  $1 \leq i \leq q$ , the *splitting around  $u_i$*  of  $\pi_{st}$  defines two hypergraphs  $\tau^i$  and  $\eta^i$  where:

$$\tau^i = (\mathcal{V}_\tau, \mathcal{E}_\tau), \quad \mathcal{V}_\tau = \{s, u_1, \dots, u_i\}, \quad \mathcal{E}_\tau = \{p(u_1), \dots, p(u_i)\},$$

$$\eta^i = (\mathcal{V}_\eta, \mathcal{E}_\eta), \quad \mathcal{E}_\eta = \{p(u_{i+1}), \dots, p(u_{q+1})\}, \quad \mathcal{V}_\eta = \bigcup_{e \in \mathcal{E}_\eta} T(e) \cup \{h(e)\}.$$

Note that  $\mathcal{V}_\pi = \mathcal{V}_\tau \cup \mathcal{V}_\eta$ ,  $\mathcal{E}_\pi = \mathcal{E}_\tau \cup \mathcal{E}_\eta$  and  $\mathcal{E}_\tau \cap \mathcal{E}_\eta = \emptyset$ . Clearly,  $\tau^i$  is a subtree of  $\mathcal{T}_s = \pi_{st}$ . On the contrary,  $\eta^i$  is not a hypertree in general; we call  $\eta^i$  an *end-tree*. Observe that for any node  $u \neq t$  in  $\eta^i$  there is a  $u$ - $t$  path in  $\eta^i$ . The above splitting operation will be exploited in our algorithms for  $K$ -shortest hyperpaths.

**Example 1.** A hypergraph  $\mathcal{H} = (\mathcal{V}, \mathcal{E})$  is shown in Fig. 1(a). Below we give two hyperpaths in  $\mathcal{H}$ , namely a hyperpath from  $s$  to  $t$  and a hyperpath from  $s$  to 4.

$$\pi_{st} = (\{s, 1, 2, t\}, \{e_1, e_2, e_3\}) \quad \pi_{s4} = (\{s, 2, 3, 4\}, \{e_2, e_4, e_5\}).$$

The hypergraph  $\mathcal{H}$  becomes acyclic when hyperarc  $e_9$  is deleted and has a unique valid ordering, namely  $V = (\{s, 1, 2, 3, 4, t\})$ . A hypertree  $\mathcal{T}_s$  in  $\mathcal{H}$  is shown with solid lines in Fig. 1(b). It is the union of the two hyperpaths given above. Several valid orderings for  $\mathcal{T}_s$  exist; one of them is  $V = (\{s, 1, 2, t, 3, 4\})$ . According to  $V$ , the subtree  $\tau^3$  corresponds to hyperpath  $\pi_{st}$  above.

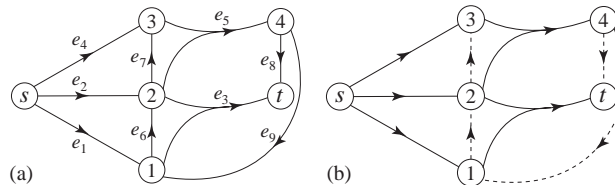


Fig. 1. The running example hypergraph. (a) A hypergraph  $\mathcal{H}$  with a cycle. (b) A hypertree in  $\mathcal{H}$ .

### 2.2. Weighted hypergraphs and shortest hyperpaths

A *weighted hypergraph* is a hypergraph where each hyperarc  $e$  is assigned a real weight  $w(e)$ . In this paper we shall assume that all weights are non-negative. Given a hyperpath  $\pi_{st}$ , a *weighting function*  $W$  is a node function assigning weights  $W(u)$  to all nodes in  $\pi_{st}$ . The weight of hyperpath  $\pi_{st}$  is  $W(t)$ . We shall restrict ourselves to *additive* weighting functions, defined by the recursive equations:

$$W(u) = \begin{cases} w(p(u)) + F(p(u)) & u \in \mathcal{V}_\pi \setminus \{s\} \\ 0 & u = s \end{cases}$$

where  $F = (e)$  is a non-decreasing function of the weights of the nodes in  $T(e)$ . We shall consider two particular weighting functions, namely the *distance* and the *value*. The *distance* function is obtained by defining  $F(e)$  as follows:

$$F(e) = \max_{v \in T(e)} \{W(v)\}$$

and the *value* function is obtained as follows:

$$F(e) = \sum_{v \in T(e)} a_e(v)W(v)$$

where  $a_e(v)$  is a non-negative multiplier defined for each hyperarc  $e$  and node  $v \in T(e)$ . With respect to the value function there are two interesting cases which may arise:

- If  $a_e(v) = 1, \forall e \in \mathcal{E}, \forall v \in T(e)$ , then the weighting function is called the *sum* function.
- If  $\sum_{v \in T(e)} a_e(v) = 1, \forall e \in \mathcal{E}$ , then the weighting function is called the *mean* function.

The *shortest hyperpath problem* consists in finding the minimum weight hyperpaths (with respect to a particular weighting function) from an origin  $s$  to all nodes in  $\mathcal{H}$  hyperconnected to  $s$ . The result is a *shortest hypertree*  $\mathcal{T}_s$  containing minimum weight hyperpaths to all hyperconnected nodes (see [8]).

**Example 1 (continued).** Consider again the hypergraph in Fig. 1(a), and suppose all edge weights are equal to 1. In this case, Fig. 1(b) shows the shortest hypertree with respect to the sum as well as to the distance weighting function.

The shortest hyperpath problem has been shown in [8] to be polynomially solvable provided that the weighting function is additive, the weights are non-negative and that all cycles are *non-decreasing*. According to the sufficient conditions given in [8], cycles are guaranteed to be non-decreasing for the sum and distance weighting functions; obviously, the condition holds true for every additive weighting function if the hypergraph is acyclic.

A general shortest hyperpath algorithm was proposed in [8]: namely, procedure  $SBT(s, \mathcal{H})^1$ , that finds a shortest hypertree rooted at  $s$  in hypergraph  $\mathcal{H}$ . A particular version of  $SBT$ , called  $SBT$ -heap

---

<sup>1</sup> Abbreviation for *shortest B-tree*, which is denoted a hypertree in this paper.

in [8], generalizes Dijkstra's algorithm for shortest paths, and takes  $O(m \log n + \text{size}(\mathcal{H}))$  time. It can be shown that the order in which nodes are processed in *SBT-heap* gives a valid ordering for the nodes in the shortest hypertree. We assume that *SBT-heap* and the corresponding valid orderings are used in the  $K$  shortest hyperpath algorithms described in the next section.

For the particular case where the hypergraph is acyclic, a simpler and faster procedure exists denoted procedure *SBT\_acyclic*( $s, \mathcal{H}$ ) (see [20]). In procedure *SBT\_acyclic*, nodes are processed according to a valid ordering  $V$  of  $\mathcal{H}$  and when a node is processed, the shortest hyperpaths to all the nodes preceding it in  $V$  are known. In this case, the computational complexity is linear, i.e.  $O(\text{size}(\mathcal{H}))$ .

### 3. Finding the $K$ shortest hyperpaths

The  $K$  shortest hyperpaths problem addressed in this paper is as follows: given a hypergraph  $\mathcal{H}$ , an origin node  $s$  and a destination node  $t$ , generate the  $K$  shortest  $s$ - $t$  hyperpaths in  $\mathcal{H}$  in non-decreasing order of weight; hyperpaths with the same weight can be generated in arbitrary order. Obviously, a problem is characterized by the chosen weighting function. In the following we shall consider the *sum* and *distance* weighting functions on general and acyclic hypergraphs. Moreover, we consider the *mean* function on acyclic hypergraphs. Our algorithms extend the  $K$  shortest loopless paths procedure by Yen [4]; therefore we briefly recall this procedure first. The extension to the hyperpath case is then discussed in detail and some improvements are proposed.

In general terms, Yen's algorithm is an implicit enumeration method, where the set of solutions is partitioned into smaller sets by recursively applying a branching step. Given a graph  $G = (N, A)$ , and two nodes  $s, t \in N$ , denote by  $\mathcal{P}$  the set of paths from  $s$  to  $t$  in  $G$ . Assume that a shortest  $s$ - $t$  path  $P_{st}$  is known, where

$$P_{st} = (s = u_1, a_1, u_2, a_2, \dots, a_q, u_{q+1} = t).$$

In the branching step, the set  $\mathcal{P} \setminus \{P_{st}\}$  is partitioned into  $q$  subsets  $\mathcal{P}^i$ ,  $1 \leq i \leq q$ . Each set  $\mathcal{P}^i$  contains the *deviations from  $P_{st}$  at  $i$* , that is: each  $s$ - $t$  path in  $\mathcal{P}^i$  is the concatenation of  $P_{su_i}$  (the subpath of  $P_{st}$  from  $s$  to  $u_i$ ) and a path from  $u_i$  to  $t$  not containing arc  $a_i$  (some of the sets  $\mathcal{P}^i$  may be empty).

The shortest  $s$ - $t$  path in each subset  $\mathcal{P}^i$  can be found by a standard shortest path procedure. Indeed, it suffices to find a shortest path from  $u_i$  to  $t$  in a subgraph  $G^i$ , obtained from  $G$  by deleting each node  $u_j$  in  $P_{su_i}$  except  $u_i$ , and deleting arc  $a_i$  as well. That is, each set  $\mathcal{P}^i$  can be represented by a pair  $(P_{su_i}, G^i)$ . Yen's algorithm maintains a list of such pairs  $(P', G')$ , where  $P'$  is a path from  $s$  to a node  $u \neq t$ , and a shortest  $s$ - $t$  path  $P'_{st}$  is defined by the concatenation of  $P'$  and the shortest  $u$ - $t$  path in  $G'$ . Initially, the list contains a single pair with  $P' = (s)$  and  $G' = G$ ; here,  $(s)$  is a path containing the single node  $s$ . At step  $k$ , the shortest  $s$ - $t$  path in the list is ranked as the  $k^{\text{th}}$  shortest path. A branching step is then applied on the shortest  $s$ - $t$  path, obtaining a set of pairs that replace  $(P', G')$  in the list. The algorithm terminates when the  $K$  shortest paths are found, or when the list is empty.

At the beginning of step  $k + 1$ , the list of pairs represents a partition of  $\mathcal{P} \setminus \{P^1, \dots, P^k\}$ , where  $\{P^1, \dots, P^k\}$  are the previously found  $k$  shortest paths. When a pair  $(P', G')$  is inserted in the list, the optimal  $s$ - $t$  path must be computed. Since  $O(n)$  pairs can be generated at each branching step, Yen's algorithms must solve  $O(Kn)$  shortest path problems.

Finally, note that Yen's algorithm follows a "forward branching" approach, since we process the arcs in  $P_{st}$  from  $s$  to  $t$ , i.e. paths in  $\mathcal{P}^i$  must contain the subpath  $P_{su_i}$ . However, the "forward branching" described above can be replaced by a "backward branching", where path  $P_{st}$  is processed from  $t$  to  $s$ . In the latter case, each path in the set  $\mathcal{P}^i$  would be the concatenation of a path from  $s$  to  $u_{i+1}$ , not containing arc  $a_i$ , and the subpath of  $P_{st}$  from  $u_{i+1}$  to  $t$ . From a theoretical point of view, the two approaches are equivalent; as we shall see, this symmetry does no longer hold when hypergraphs are considered.

### 3.1. Branching on hyperpaths

In order to extend Yen's algorithm to hypergraphs, we need to devise a suitable branching rule, i.e. a partition technique so that finding the best hyperpath in each subset is easy, in particular solved by procedure SBT.

Consider a hypergraph  $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ , and two nodes  $s, t \in \mathcal{V}$ . In the following we let  $\Pi$  denote the set of hyperpaths from  $s$  to  $t$  in  $\mathcal{H}$ . We assume that a shortest  $s$ - $t$  hyperpath  $\pi_{st} = (\mathcal{V}_\pi, \mathcal{E}_\pi)$  is known, and is defined by a predecessor function  $p: \mathcal{V} \setminus \{s\} \rightarrow \mathcal{E}$ ; moreover, a valid ordering:

$$V_\pi = (s = u_1, u_2, \dots, u_q, u_{q+1} = t)$$

for the nodes in  $\pi_{st}$  is at hand.

Since a hyperpath is not in general the concatenation of two hyperpaths, a direct application of Yen's branching technique is not possible. However, we may still follow a forward branching approach, where we split a hyperpath around each node  $u_i$  (see Definition 3) according to the given valid ordering. That is, we partition the set  $\Pi \setminus \{\pi_{st}\}$  into  $q$  subsets  $\Pi^i$ ,  $1 \leq i \leq q$ ; each  $s$ - $t$  hyperpath in  $\Pi^i$  must contain the subtree  $\tau^i$  of  $\pi_{st}$  spanning the first  $i$  nodes, and cannot contain hyperarc  $p(u_{i+1})$ .

Unfortunately, the forward branching approach turns out to be computationally intractable. Indeed, finding the shortest hyperpath  $\pi^i$  in  $\Pi^i$  corresponds to solving a *Subtree Constrained hyperpath* problem (SCH), which is NP-hard; a proof of this claim, and the definition of (SCH), are given in the Appendix. Here we state the following theorem.

**Theorem 2.** *The problem of finding the shortest hyperpath  $\pi^i$  in  $\Pi^i$ , using the forward branching approach, is  $\mathcal{NP}$ -hard.*

We shall therefore follow a different approach, in particular one based on *backward branching*.

**Definition 4** (backward branching). Given the shortest hyperpath  $\pi_{st}$  in  $\Pi$  and the valid ordering  $V_\pi$ , the set  $\Pi \setminus \{\pi_{st}\}$  can be partitioned into  $q$  subsets  $\Pi^i$ ,  $1 \leq i \leq q$  as follows:

- Hyperpaths in  $\Pi^q$  do not contain hyperarc  $p(u_{q+1})$ , that is  $p(t)$ ;
- For  $1 \leq i < q$ , hyperpaths in  $\Pi^i$  contain the end-tree  $\eta^{i+1}$ , and do not contain hyperarc  $p(u_{i+1})$ .

It is easy to see that the above sets  $\Pi^i$  are disjoint, and give a partition of  $\Pi$ . Now consider the problem of finding the shortest hyperpath  $\pi^i \in \Pi^i$ . This problem reduces to solving a shortest hyperpath problem on a hypergraph  $\mathcal{H}^i$  obtained from  $\mathcal{H}$  as follows:

- for each node  $u_j$ ,  $i + 1 < j \leq q + 1$ , remove each hyperarc in  $BS(u_j)$  except  $p(u_j)$ ;
- remove hyperarc  $p(u_{i+1})$  from  $BS(u_{i+1})$ .

We say that in  $\mathcal{H}^i$  each hyperarc  $p(u_j)$ ,  $i + 1 < j \leq q + 1$ , is *fixed*, while  $p(u_{i+1})$  is *deleted*. Note that for  $1 \leq i < q$  hypergraph  $\mathcal{H}^i$  contains the hyperarcs in the end-tree  $\eta^{i+1}$ .

**Theorem 3.** *Each hyperpath in  $\Pi^i$  is also an  $s$ - $t$  hyperpath in  $\mathcal{H}^i$ ; conversely, each  $s$ - $t$  hyperpath in  $\mathcal{H}^i$  belongs to  $\Pi^i$ .*

**Proof.** The first claim is trivial, since a hyperpath in  $\Pi^i$  contains hyperarcs that also belong to  $\mathcal{H}^i$ . The converse claim is trivial for  $i = q$ , since  $p(t)$  is deleted in  $\mathcal{H}^q$ . In order to prove the converse claim for  $1 \leq i < q$ , we have to show that each  $s$ - $t$  hyperpath  $\pi$  in  $\mathcal{H}^i$  contains the end-tree  $\eta^{i+1}$ , that is, contains hyperarc  $p(u_j)$  for  $i + 1 < j \leq q + 1$ . This can be done by induction. Observe that  $\pi$  must contain  $p(t)$ , since  $p(t)$  is fixed in  $\mathcal{H}^i$ , i.e.  $BS(t) = \{p(t)\}$ . Now assume that  $\pi$  contains  $p(u_k)$  for each  $j < k \leq q + 1$ , and consider node  $u_j$ , with  $i + 1 < j$ . Since  $\eta^{i+1}$  contains a path from  $u_j$  to  $t$ ,  $u_j$  belongs to the tail of  $p(u_k)$  for some  $k > j$ , thus  $u_j$  is in  $\pi$ ; since  $p(u_j)$  is fixed in  $\mathcal{H}^i$ ,  $p(u_j)$  must be in  $\pi$ .  $\square$

Theorem 3 provides us with the following useful result.

**Corollary 1.** *Finding the shortest hyperpath  $\pi^i \in \Pi^i$ ,  $i = 1, \dots, q$ , reduces to solving a shortest hyperpath problem on  $\mathcal{H}^i$ .*

As a consequence, each set  $\Pi^i$  is represented by the corresponding hypergraph  $\mathcal{H}^i$ . In other words, a *backward branching* operation on  $\pi_{st}$  returns the set of hypergraphs  $\mathcal{B}(\mathcal{H}) = \{\mathcal{H}^i : 1 \leq i \leq q\}$ , representing the partition  $\{\Pi^i : 1 \leq i \leq q\}$  of  $\Pi \setminus \{\pi_{st}\}$ . Recall that a branching operation is related to an underlying valid ordering  $V_\pi$ ; a different order may result in a different set of hypergraphs.

Adopting the backward branching technique, we can extend Yen’s algorithm to hypergraphs. The algorithm maintains a list  $L$  of *subproblems*. Each subproblem is represented by the pair  $(\tilde{\mathcal{H}}, \tilde{\pi})$ , where  $\tilde{\pi}$  is a shortest  $s$ - $t$  hyperpath in  $\tilde{\mathcal{H}}$ . Note that we assume  $t$  hyperconnected to  $s$  in each subproblem. Initially,  $L$  contains the pair  $(\mathcal{H}, \pi_{st})$ . In iteration  $k$  we remove from  $L$  a pair  $(\tilde{\mathcal{H}}, \tilde{\pi})$ , such that  $\tilde{\pi}$  has minimum weight among the subproblems in  $L$ ;  $\tilde{\pi}$  is the  $k$ ’th shortest hyperpath. Then backward branching is applied to  $\tilde{\pi}$ , and for each sub-hypergraph  $\tilde{\mathcal{H}}^i \in \mathcal{B}(\tilde{\mathcal{H}})$ , the shortest hypertree rooted at  $s$  is computed; if  $t$  is hyperconnected to  $s$  in  $\tilde{\mathcal{H}}^i$ , the pair  $(\tilde{\mathcal{H}}^i, \tilde{\pi}^i)$  is inserted into  $L$ , where  $\tilde{\pi}^i$  denotes the shortest  $s$ - $t$  hyperpath in  $\tilde{\mathcal{H}}^i$ . Otherwise,  $\tilde{\mathcal{H}}^i$  is discarded. The algorithm terminates when  $L$  becomes empty or at the end of iteration  $K$ .

Procedure *Yen*, given below, formally describes our  $K$  shortest hyperpaths algorithm. Here  $W(\pi)$  denotes the weight of hyperpath  $\pi$ , and we assume  $W(\tilde{\pi}^i) = +\infty$ , if  $t$  is not hyperconnected to  $s$  in  $\tilde{\mathcal{H}}^i$ .

**Procedure**  $Yen(\mathcal{H}, s, t, K)$

- Step 0**  $L = \{(\mathcal{H}, \pi_{st})\}$ ;  $k = 1$ ;
- Step 1** if  $L = \emptyset$  then STOP; otherwise, let  $L = L \setminus \{(\tilde{\mathcal{H}}, \tilde{\pi})\}$  where  $W(\tilde{\pi}) = \min_{(\mathcal{H}', \pi') \in L} W(\pi')$ ;
- Step 2** OUTPUT  $\tilde{\pi}$ ;  $k = k + 1$ ; if  $k > K$  then STOP;
- Step 3** for each  $\tilde{\mathcal{H}}^i$  in  $\mathcal{B}(\tilde{\mathcal{H}})$  do:
  - (a) apply procedure  $SBT(s, \tilde{\mathcal{H}}^i)$ ;
  - (b) if  $W(\tilde{\pi}^i) < +\infty$  then  $L = L \cup \{(\tilde{\mathcal{H}}^i, \tilde{\pi}^i)\}$ ;
 go to Step 1.

In Step 3  $q$  sub-hypergraphs are generated, i.e.  $q$  shortest hyperpath problems must be solved. In practice, however, it is not always necessary to process all the hypergraphs in  $\mathcal{B}(\tilde{\mathcal{H}})$ . Consider the case where, in order to obtain  $\tilde{\mathcal{H}}^i$ , we must delete a hyperarc  $p(u)$  that has been fixed in a previous branching operation. As follows from Theorem 3, no  $s$ - $t$  hyperpath exists in  $\tilde{\mathcal{H}}^i$ . In this case, we may assume that  $\tilde{\mathcal{H}}^i$  is not generated by the algorithm.

The correctness for Procedure  $Yen$  follows easily from Theorem 3, and from the fact that backward branching provides a partition of  $\Pi \setminus \{\pi_{st}\}$ . In order to evaluate the computational complexity, observe that at most  $n$  subproblems are generated in Step 3 at each iterations. Assuming that procedure  $SBT$ -heap is used in Step 3, we have the following theorem:

**Theorem 4.** Procedure  $Yen$  finds the  $K$  shortest hyperpaths in  $O(Kn(m \log n + size(\mathcal{H})))$  time.

**Example 2.** Assume that we want to find the  $K = 3$  shortest hyperpaths, with respect to the sum function, in the hypergraph of Fig. 1(a), where we assume unit weights on the hyperarcs. The shortest hypertree has been emphasized in Fig. 1(b). The shortest hyperpath from  $s$  to  $t$ , with weight 3, is the one given in Example 1, that is  $\pi_{st} = (\{t, 1, 2, s\}, \{e_1, e_2, e_3\})$ .

A valid ordering of the nodes in  $\pi_{st}$  is  $V = \{s, 1, 2, t\}$ . Applying backward branching, three sub-hypergraphs  $\mathcal{H}^3$ ,  $\mathcal{H}^2$  and  $\mathcal{H}^1$  are created from  $\mathcal{H}$ , as shown in Fig. 2;  $\mathcal{H}^3$  is obtained by deleting  $p(t) = e_3$ ;  $\mathcal{H}^2$  is created by fixing  $e_3$  and deleting  $e_2$ ;  $\mathcal{H}^1$  is created by fixing  $e_3$  and  $e_2$ , and by deleting  $e_1$ . Subhypergraphs  $\mathcal{H}^3$ ,  $\mathcal{H}^2$  and  $\mathcal{H}^1$  are shown in Figs. 3(a)–(c) with shortest hypertree shown with solid lines emphasized; the shortest hyperpath weight appears close to  $t$ . In each figure the fixed hyperarcs are marked with thick lines. Observe that arc  $e_6 = (\{1\}, 2)$  has been removed from  $\mathcal{H}^1$ , since  $e_2 = (\{s\}, 2)$  has been fixed.

Assume that we select the shortest hyperpath  $\pi_{st}^2$  in  $\mathcal{H}^2$  next. Again,  $V = \{s, 1, 2, t\}$  is a valid ordering. Since  $BS(t)$  and  $BS(2)$  have cardinality one, only hyperarc  $p(1) = e_1$  can be deleted. This gives subhypergraph  $\mathcal{H}^{21}$ , shown in Fig. 3(d), where no hyperpath from  $s$  to  $t$  exists. The third shortest hyperpath thus becomes the hyperpath in  $\mathcal{H}^3$ , with weight 4.

3.2. An improved algorithm

The main drawback of Yen’s algorithm for  $K$ -shortest paths is that an SBT problem must be solved for each  $G^i$  generated by branching. The number of SBT problems to solve is therefore

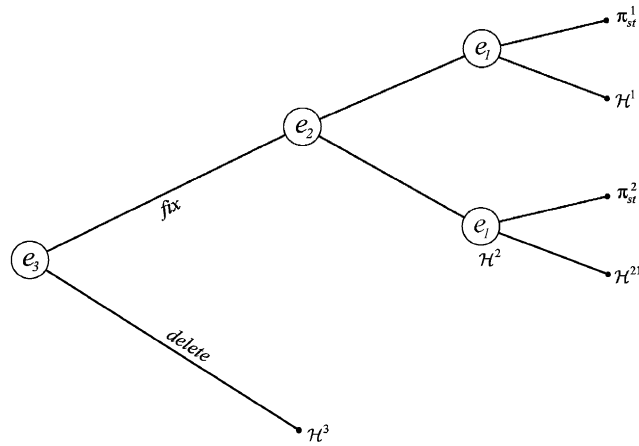


Fig. 2. The branching tree of  $\mathcal{H}$ .

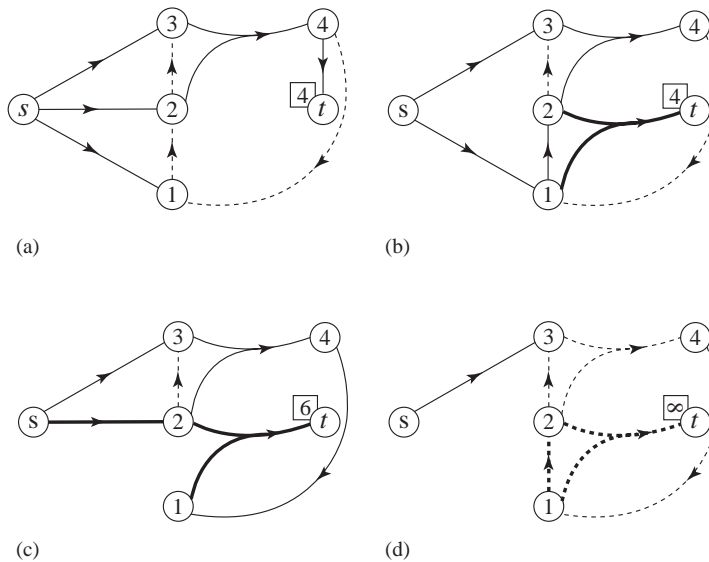


Fig. 3. Subhypergraphs generated during the procedure. (a) Hypergraph  $\mathcal{H}^3$ . (b) Hypergraph  $\mathcal{H}^2$ . (c) Hypergraph  $\mathcal{H}^1$ . (d) Hypergraph  $\mathcal{H}^{21}$ .

much larger than  $K$ , and possibly proportional to  $Kn$ . The efficiency of the branching phase can be improved by applying *reoptimization* techniques (see e.g. [21]). These techniques cannot be directly extended to hypergraphs, and are not discussed here. Instead, we propose an improved version of procedure *Yen*, based on a new strategy. Our goal is to *delay* the computation of shortest hypertrees. In particular, hypertrees are computed when a subproblem is *selected* from the list  $L$ ; the selection order is based on a lower bound on the shortest  $s$ - $t$  hyperpath weight. The selected subproblems provide a superset of the  $K$  shortest hyperpaths. Clearly, this technique is effective if the number of selections is small (e.g. close to  $K$ ) compared to the total size of  $L$ , which is  $O(Kn)$ .

In order to compute a tight lower bound on hyperpath weight, we take advantage of some properties that are known to hold for graphs, and can be extended to hypergraphs. Let the predecessor function  $p$  define the shortest hypertree  $\mathcal{T}_s$  in hypergraph  $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ . Moreover, denote by  $W$  the vector of minimum weights, i.e.  $W(u)$  is the minimum weight of an  $s$ - $u$  hyperpath, and let  $F(W, e)$  denote the value of the weighting function  $F$  on hyperarc  $e$  with respect to the weights  $W$ . For example, for the distance function we would have:

$$F(W, e) = \max_{v \in T(e)} \{W(v)\}.$$

Given node  $v \neq s$ , suppose that  $p(v)$  is removed from  $\mathcal{H}$ , obtaining a sub-hypergraph  $\mathcal{H}'$ ; let  $BS'(v) \neq \emptyset$  be the backward star of  $v$  in  $\mathcal{H}'$ . Compute the value  $\bar{W}(v)$  as follows:

$$\bar{W}(v) = \min_{e \in BS'(v)} \{F(W, e) + w(e)\}$$

and let  $\bar{p}$  be the predecessor function obtained from  $p$  by setting:

$$\bar{p}(v) = \arg \min_{e \in BS'(v)} \{F(W, e) + w(e)\}.$$

Note that  $\bar{p}$  does not necessarily define a hypertree.

**Theorem 5.** *The value  $\bar{W}(v)$  is a lower bound on the minimum weight  $W'(v)$  of an  $s$ - $v$  hyperpath in  $\mathcal{H}'$ . Moreover, if  $\bar{p}$  defines a hypertree,  $\bar{W}(v) = W'(v)$ .*

**Proof.** Denote by  $W'$  the vector of minimum weights in  $\mathcal{H}'$ ; clearly,  $W' \geq W$ , and thus  $F(W', e) \geq F(W, e)$  for each  $e$  in  $\mathcal{H}'$ . This implies the first claim:

$$\bar{W}(v) = \min_{e \in BS'(v)} \{F(W, e) + w(e)\} \leq \min_{e \in BS'(v)} \{F(W', e) + w(e)\} = W'(v).$$

To prove the second claim, assume that  $\bar{p}$  defines a hypertree  $\mathcal{T}'$ , and let

$$V' = (s = u_1, u_2, \dots, v = u_i, \dots, t = u_n)$$

be a valid ordering for  $\mathcal{T}'$ . Since  $p(u_j) = \bar{p}(u_j)$  for each  $j < i$ , it follows that  $W'(u) = W(u)$  for each node  $u$  in  $\mathcal{T}'^{i-1}$ . Thus we have  $F(W', \bar{p}(v)) = F(W, \bar{p}(v))$ , which implies  $W'(v) \leq \bar{W}(v)$ , and the claim follows.  $\square$

Let  $V = (s = u_1, u_2, \dots, v = u_{i+1}, \dots, t = u_{q+1})$  be a valid ordering for the shortest hyperpath  $\pi_{st}$  in  $\mathcal{H}$ , and consider the sub-hypergraph  $\mathcal{H}^i$  in the branching step. Recall that  $p(v)$  is deleted in  $\mathcal{H}^i$ . As follows from Theorem 5,  $\bar{W}(v)$  is a lower bound for the minimum weight of an  $s$ - $v$  hyperpath in  $\mathcal{H}^i$ . We can extend the node function  $\bar{W}$  to all the nodes in  $\pi_{st}$  as follows:

1. let  $\bar{W}(u_j) = W(u_j)$  for  $1 \leq j \leq i$ ;
2. for  $i + 1 < j \leq q + 1$ , let  $\bar{W}(u_j) = F(\bar{W}, p(u_j)) + w(p(u_j))$ .

Consider the predecessor function  $\bar{p}$  defined above; the following theorem holds true.

**Theorem 6.** *The value  $\bar{W}(t)$  is a valid lower bound on the minimum weight  $W^i(t)$  of an  $s$ - $t$  hyperpath in  $\mathcal{H}^i$ . Moreover, if  $\bar{p}$  defines a hypertree,  $\bar{W}(t) = W^i(t)$ .*

**Proof.** Denote by  $W^i$  the vector of minimum weights in  $\mathcal{H}^i$ , and consider the splitting of  $\pi_{st}$  around  $v$ , i.e.  $\pi_{st} = \tau^{i+1} \cup \eta^{i+1}$ . Since the shortest  $s$ - $t$  hyperpath in  $\mathcal{H}^i$  contains  $\eta^{i+1}$ ,  $W^i(t)$  is computed iteratively similar to  $\bar{W}(t)$ , that is:

$$W^i(u_j) = F(W^i, p(u_j)) + w(p(u_j)), \quad i + 1 < j \leq q + 1.$$

The first claim then follows since, for each  $u_j$  in  $\tau^{i+1}$ , we have  $\bar{W}(u_j) \leq W^i(u_j)$  by Theorem 5. Moreover, if  $\bar{p}$  defines a hypertree, it is  $\bar{W}(u_j) = W^i(u_j)$  for each  $u_j$  in  $\tau^{i+1}$ , and the second claim follows.  $\square$

In light of Theorem 6, we shall use the value  $\bar{W}(t)$  as a lower bound on the shortest hyperpath weight in  $\mathcal{H}^i$ . In our modified Yen’s algorithm, a subproblem can be selected at most twice: the first time to compute the shortest hypertree, and (possibly) the second time to output the shortest  $s$ - $t$  hyperpath and perform branching. Each subproblem is represented by the triple  $(\mathcal{H}, \tilde{lb}, \tilde{\mathcal{T}})$ . Here,  $\tilde{\mathcal{T}}$  denotes a shortest hypertree in  $\mathcal{H}$ , if such hypertree is known, or *nil* otherwise;  $\tilde{lb}$  is a *finite* lower bound on the weight of the shortest  $s$ - $t$  hyperpath  $\tilde{\pi}$  in  $\mathcal{H}$ . Initially,  $L$  contains the triple  $(\mathcal{H}, W(\pi_{st}), \mathcal{T}_s)$ . In each iteration, we remove from  $L$  a subproblem  $(\mathcal{H}, \tilde{lb}, \tilde{\mathcal{T}})$  with minimum value  $\tilde{lb}$ ; The subproblem is then processed according to  $\tilde{\mathcal{T}}$ . Two cases may arise:

- $\tilde{\mathcal{T}} = \text{nil}$ : we compute a shortest hypertree in  $\mathcal{H}$  and assign it to  $\tilde{\mathcal{T}}$ ; if  $t$  is not connected to  $s$  in  $\mathcal{H}$ , we discard the subproblem; otherwise, we assign to  $\tilde{lb}$  the weight  $W(\tilde{\pi})$  of the shortest  $s$ - $t$  hyperpath  $\tilde{\pi}$  in  $\mathcal{H}$ , and reinsert  $(\mathcal{H}, \tilde{lb}, \tilde{\mathcal{T}})$  into  $L$ .
- $\tilde{\mathcal{T}} \neq \text{nil}$ : we output  $\tilde{\pi}$  as the next hyperpath, and we proceed to branching.

In the branching phase, we compute the lower bound  $\bar{W}^i(t)$  for each hypergraph  $\mathcal{H}^i$  into  $\mathcal{B}(\mathcal{H})$ ; if  $\bar{W}^i(t)$  is finite, we insert in  $L$  the subproblem  $(\mathcal{H}^i, \bar{W}^i(t), \text{nil})$ . Procedure *LBZen*, given below, describes the improved algorithm. By  $\tilde{\mathcal{T}} = \text{SBT}(s, \mathcal{H})$  we mean that the shortest hypertree in  $\mathcal{H}$  is stored in  $\tilde{\mathcal{T}}$ .

**Procedure LBZen**( $\mathcal{H}, s, t, K$ )

- Step 0**  $L = \{(\mathcal{H}, W(\pi_{st}), \mathcal{T}_s)\}$ ;  $k = 1$ ;
- Step 1** if  $L = \emptyset$  then STOP; otherwise, let  $L = L \setminus \{(\mathcal{H}, \tilde{lb}, \tilde{\mathcal{T}})\}$  where  $\tilde{lb} = \min_{(\mathcal{H}', lb', \mathcal{T}') \in L} lb'$ ;
- Step 2** if  $\tilde{\mathcal{T}} \neq \text{nil}$  go to Step 4;
- Step 3** set  $\tilde{\mathcal{T}} = \text{SBT}(s, \mathcal{H})$ ; if  $W(\tilde{\pi}) < \infty$  then set  $\tilde{lb} = W(\tilde{\pi})$  and  $L = L \cup \{(\mathcal{H}, \tilde{lb}, \tilde{\mathcal{T}})\}$ ; go to Step 1;
- Step 4** OUTPUT  $\tilde{\pi}$ ;  $k = k + 1$ ; if  $k > K$  then STOP;
- Step 5** for each  $\mathcal{H}^i$  in  $\mathcal{B}(\mathcal{H})$  do:
  - (a) compute the lower bound  $\bar{W}^i(t)$ ;
  - (b) if  $\bar{W}^i(t) < +\infty$  then  $L = L \cup \{(\mathcal{H}^i, \bar{W}^i(t), \text{nil})\}$ ;
 go to Step 1.

The correctness of Procedure *LBZen* follows immediately from the correctness of Procedure *Zen*. Moreover, the computational complexity of the two procedures is the same, since in both cases at most  $Kn$  subproblems are created, and  $Kn$  shortest hyperpath problems are solved. Note that, for

each subproblem, the time needed to compute the lower bound  $\bar{W}(t)$  is linear in the size of the shortest hyperpath, which is  $O(\text{size}(\mathcal{H}))$  and thus dominated by the time needed to solve a shortest hyperpath problem.

**Theorem 7.** *Procedure LBYen finds the  $K$  shortest hyperpaths in  $O(Kn(m \log n + \text{size}(\mathcal{H})))$  time.*

**Example 2** (continued). Assume that procedure *LBYen* is used in Example 2. The lower bound  $\bar{W}(t)$  for sub-hypergraphs  $\mathcal{H}^3$ ,  $\mathcal{H}^2$  and  $\mathcal{H}^1$  is equal to the actual weight. Again, assume we select  $\mathcal{H}^2$  next. In order to compute  $\bar{W}(t)$  for sub-hypergraph  $\mathcal{H}^{21}$ , we must compute  $\bar{W}(1)$  first; the backward star of node 1 in  $\mathcal{H}^{21}$  contains the single arc  $e_9 = (\{4\}, 1)$ , which gives  $\bar{W}(1) = 5$ . We thus obtain  $\bar{W}(2) = 6$  and  $\bar{W}(t) = 12$ , which is a weak lower bound on infinity. Subproblem  $(\mathcal{H}^{21}, 12, \text{nil})$  is inserted in  $L$  and, if selected later, it will be discarded in Step 3.

### 3.3. Acyclic hypergraphs

The  $K$  shortest path problem in acyclic graphs is computationally much easier, since algorithms allowing loops in paths can be used in this case. Up to a certain extent, this situation extends to acyclic hypergraphs. Here we shall devise a specialized procedure where only one shortest hypertree computation is needed.

Observe that, according to Theorem 6, the lower bound  $\bar{W}(t)$  gives the actual shortest hyperpath weight in an acyclic hypergraph. Indeed, the predecessor function  $\bar{p}$  defines a hypertree, as follows from Definition 2. Moreover, in order to compute the lower bounds in the branching step, it is not necessary to find a shortest hypertree in each selected subproblem. This can be explained as follows.

Consider an acyclic hypergraph  $\mathcal{H}$ , and let  $V$  be a valid ordering of the nodes. Clearly,  $V$  induces a valid ordering for the nodes in any  $s$ - $t$  hyperpath; we assume that  $V$  is used in the branching steps. Assume now that a sub-hypergraph  $\mathcal{H}'$  is obtained from  $\mathcal{H}$  by deleting hyperarc  $p(u_i)$  and possibly by fixing the hyperarcs in an end-tree  $\eta^i$ . In order to obtain the lower bound  $\bar{W}(t)$  in  $\mathcal{H}'$ , we need to know the minimum weights for the nodes that precede  $u_i$  in  $V$ . For these nodes, the shortest weights (as well as the corresponding shortest hyperpaths) are the same as in  $\mathcal{H}$ . Therefore, the value  $\bar{W}(t)$  and the shortest hyperpath in each subhypergraph can be computed using the shortest hypertree and hyperpath weights for  $\mathcal{H}$ .

For acyclic hypergraphs, we devise a specialized version of Procedure *Yen*, referred to as *AYen*, where we do not apply procedure *SBT* in Step 3(a). Instead, we compute the lower bound  $\bar{W}(t)$ , which provides us with the shortest hyperpath  $\tilde{\pi}^i$  used in Step 3(b) to create a new subproblem. As discussed above, the computation of the lower bound takes  $O(\text{size}(\mathcal{H}))$  time. We thus have the following theorem.

**Theorem 8.** *Procedure AYen finds the  $K$  shortest hyperpaths in  $O(\text{size}(\mathcal{H})Kn)$  time.*

Note that we can find the shortest hypertree in  $\tilde{\mathcal{H}}^i$  in time  $O(\text{size}(\mathcal{H}))$  by applying procedure *SBT<sub>acylic</sub>*, instead of computing the lower bound. However, as we shall see in the next section, the computation of the lower bound is much faster.

Table 1  
Randomly generated test problems

| Class | 1    | 2      | 3      | 4      | 5      | 6    | 7      | 8      | 9      | 10     |
|-------|------|--------|--------|--------|--------|------|--------|--------|--------|--------|
| Nodes | 100  | 300    | 500    | 800    | 1000   | 1000 | 3000   | 5000   | 8000   | 10 000 |
| Arcs  | 400  | 1200   | 2000   | 3200   | 4000   | 2000 | 6000   | 10 000 | 16 000 | 20 000 |
| Harc  | 5000 | 15 000 | 25 000 | 40 000 | 50 000 | 4000 | 12 000 | 20 000 | 32 000 | 40 000 |

#### 4. Computational results

In this section we test the procedures described in Section 3. The procedures have been implemented in C++ and run on a 700 MHz PIII computer with 512MB RAM using a Linux operating system. The programs have been compiled using the GNU C++ compiler (version 2.96) with optimize option -O.

In our implementation, the branching tree representing the list  $L$  (see Fig. 2) is implemented as a dynamic binary tree; all the information related to the subproblems is associated with nodes in the tree, while a heap of tree node pointers is used for the selection phase. In fact, at the end of the procedure the branching tree provides a representation of the  $K$  shortest hyperpaths. Anyway, we do not perform any actual “output” operation on the generated hyperpaths.

Ten classes of randomly generated hypergraphs were considered, as shown in Table 1. The generated hypergraphs can be divided into two groups. Hypergraphs in class 1–5 have fewer nodes and are dense: the number of arcs is  $4n$  and the number of “true hyperarcs” is  $50n$ . This gives an average number of 54 hyperarcs in the backward star of a node. Hypergraphs in class 5–10 have more nodes and are sparse, the number of arcs is  $2n$  and the number of “true hyperarcs” is  $4n$  resulting in an average number of 6 hyperarcs in the backward star of a node.

For all classes, the size of each true hyperarc is randomly generated with a uniform distribution in the interval  $[3,5]$ . The weights for each arc is between 500 and 1000, and for each hyperarc it is between 1 and 100. This choice has been made to favor hyperpaths with many true hyperarcs; nevertheless, the percentage of arcs in the generated hyperpaths tends to be relevant, in particular for sparse hypergraphs.

##### 4.1. Non-acyclic hypergraphs

Our main goal here is to evaluate and compare the behavior of procedures *Yen* and *LBZen*; we consider the sum and distance weighting function, and we generate  $K = 500$  hyperpaths. Five non-acyclic hypergraphs were generated for each class in Table 1; results are reported in Table 2, where each row contains the average results over the five generated hypergraphs. Column “Cardinality” contains the average size of the generated hyperpaths. Column “BTsize” gives the total number of subproblems in the branching tree for procedure *LBZen*. The branching tree for procedure *Yen* is approximately the same and is not reported. In column “Reinsert” we give the number of subproblems reinserted in the list  $L$  by procedure *LBZen*. Finally, columns “CPU-LB” and “CPU-Yen” contain the CPU time, reported in seconds, for procedures *LBZen* and *Yen*, respectively.

Table 2  
Sum and distance weighting functions,  $K = 500$

| Class | Nodes  | Arcs   | H arcs | Sum function |       |         |          |        |         | Distance function |       |         |          |        |         |
|-------|--------|--------|--------|--------------|-------|---------|----------|--------|---------|-------------------|-------|---------|----------|--------|---------|
|       |        |        |        | Cardinality  | Arc % | BT size | Reinsert | CPU-LB | CPU-Yen | Cardinality       | Arc % | BT size | Reinsert | CPU-LB | CPU-Yen |
| 1     | 100    | 400    | 5000   | 12.0         | 44.4  | 4314    | 12       | 1.7    | 15.1    | 43.5              | 20.9  | 10009   | 60       | 2.2    | 31.3    |
| 2     | 300    | 1200   | 15 000 | 12.9         | 50.4  | 4794    | 3        | 8.0    | 78.3    | 55.7              | 34.6  | 13 147  | 7        | 9.4    | 220.0   |
| 3     | 500    | 2000   | 25 000 | 13.1         | 52.4  | 5026    | 6        | 14.5   | 147.6   | 66.4              | 39.8  | 14 010  | 44       | 17.8   | 427.0   |
| 4     | 800    | 3200   | 40 000 | 12.4         | 54.2  | 4811    | 3        | 24.4   | 234.0   | 62.9              | 46.2  | 11 668  | 26       | 29.0   | 606.5   |
| 5     | 1000   | 4000   | 50 000 | 11.9         | 54.4  | 4557    | 3        | 31.4   | 284.2   | 86.2              | 45.4  | 16 590  | 11       | 36.4   | 1113.4  |
| 6     | 1000   | 2000   | 4000   | 11.2         | 79.8  | 3469    | 4        | 2.4    | 14.6    | 52.9              | 70.4  | 7820    | 6        | 2.8    | 35.8    |
| 7     | 3000   | 6000   | 12 000 | 10.1         | 78.7  | 3467    | 2        | 11.1   | 68.1    | 55.4              | 72.3  | 8974    | 4        | 12.5   | 189.4   |
| 8     | 5000   | 10 000 | 20 000 | 9.6          | 78.7  | 3408    | 0        | 20.4   | 121.9   | 47.3              | 71.3  | 7972    | 0        | 22.6   | 301.8   |
| 9     | 8000   | 16 000 | 32 000 | 9.0          | 77.7  | 3282    | 0        | 34.9   | 196.9   | 41.7              | 72.7  | 6899    | 0        | 38.7   | 445.3   |
| 10    | 10 000 | 20 000 | 40 000 | 8.5          | 78.1  | 3200    | 1        | 45.0   | 243.7   | 39.1              | 71.2  | 6832    | 3        | 50.0   | 563.4   |

As a preliminary remark, note that the sum weighting function gives hyperpaths of smaller size, and containing fewer arcs, with respect to the distance function. This in turn has an impact on the branching tree size, which is expected to be roughly proportional to the hyperpath cardinality. This behavior is not surprising, since the sum weighting function can grow quite rapidly if the hyperpath contains many large hyperarcs.

For what concerns our main goal here, procedure *LBZen* clearly outperforms procedure *Zen*. Indeed, the results confirm that CPU time is roughly proportional to the number of times the SBT procedure is used. Procedure *Zen* solves a shortest hyperpath problem for each subproblem, while *LBZen* does so only for selected subproblems, i.e.  $K$  times plus the number of reinsertions. It must be remarked that the actual number of reinsertions is quite low (12% of  $K$  in the worst case), which implies that the lower bound is mostly tight.

Observe that the branching tree size is roughly constant within each group of results, i.e. for a fixed weighting function and density. However, CPU times increase with  $n$ , due to the fact that the SBT procedures are applied to larger hypergraphs. In order to investigate this behavior more deeply, we plotted the CPU time against the number of nodes  $n$  for each generated hypergraph. The results for the distance function on dense hypergraphs are reported in Fig. 4. The figure shows a linear dependence in  $n$ , even though procedure *Zen* tends to be less stable for higher  $n$ . On the contrary, procedure *LBZen* is not only faster but seemingly more stable. We omit the plots for the other groups of results, since they show a similar linear dependence and an even more stable behavior.

We also tried to evaluate the computational effort required by each generated hyperpath. To this aim, we recorded the elapsed CPU time for every ten generated hyperpaths. Fig. 5 refers to a large dense hypergraph (class 5) using the distance weighting function. A linear dependence is obvious. As before, the results obtained for other combinations of hypergraph and weighting function show a similar dependence.

#### 4.1.1. Acyclic hypergraphs

The set of experiments described above for the non-acyclic case were replicated for the acyclic case. In particular, five acyclic hypergraphs were generated for each class in Table 1. Besides sum and distance, the mean weighting function was also tested, with multipliers given by  $a_e(u) = 1/|T(e)|$ ,

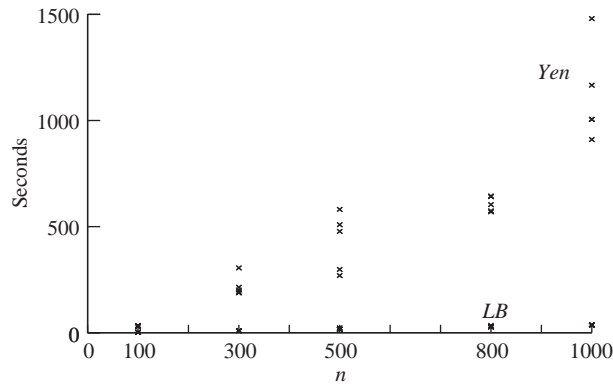


Fig. 4. CPU times for dense hypergraphs of increasing size using the distance weighting function.

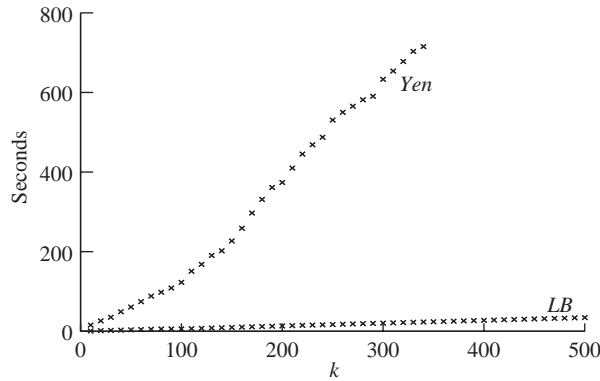


Fig. 5. CPU time per hyperpath (dense hypergraph—distance weighting function).

$u \in T(e)$ . Most of the observations made for general hypergraphs apply to acyclic hypergraphs too. Also in this case, procedure *LB**Yen* outperforms *Yen*; recall that no reinsertions are performed by *LB**Yen* in this case. More important, *AYen* is about five times faster than *LB**Yen* on average. This behaviour was expected, since *AYen* does not solve shortest hyperpath problems on the generated sub-hypergraphs. Like for general hypergraphs, we made plots similar to the ones in Figs. 4 and 5. They revealed a similar behavior.

### 5. Conclusions

In this paper, we introduced and investigated the  $K$  shortest hyperpath problem in directed hypergraphs. Even though several hyperpath models have been proposed in the literature, this problem has not yet been considered. Several areas where  $K$  shortest hyperpaths algorithms have potential applications was given. The main contributions of this paper can be summarized as follows.

First, we pointed out the lack of symmetry between the graph and hypergraph case, which prevents the “forward branching” approach from being used. Second, we extended to hypergraphs the method of Yen. Next, we proposed an algorithmic improvement that turned out to be quite effective in computational experiments. We also pointed out that acyclic hypergraphs are an easier case, as happens for graphs, and we devised a quite fast specialized procedure.

The results in this paper provide a starting point for further research, in particular for what concerns routing problems in random time dependent networks. In this context, procedure *AYen* has proven to be an effective tool for the implementation of a two-phase method for the bicriterion shortest hyperpath problem [14]. Moreover, suitable extensions of the methods described in this paper can be devised for the *a priori* version of the optimal routing problem [10,11], which is NP-hard [12]. This application is currently under investigation, and will be the subject of a forthcoming paper.

### Appendix. The subtree constrained hyperpath problem

Assume that we are given a weighted hypergraph  $\mathcal{H}$ , a hypertree  $\mathcal{T}$  rooted at  $s$  in  $\mathcal{H}$ , and a node  $t$  of  $\mathcal{H}$  not in  $\mathcal{T}$ . The *Subtree Constrained Hyperpath problem* (SCH) consists in finding a shortest  $s$ - $t$  hyperpath containing  $\mathcal{T}$  as a subtree. We show that this problem is NP-hard, also if  $\mathcal{T}$  only contains arcs in  $FS(s)$ . We consider the *distance* function here, a simpler construction can be given for the *value* weighting function. We provide a reduction from the *Set Covering* problem (SC), which is well-known to be strongly NP-hard. An instance of (SC) is defined by a family  $\mathcal{F} = \{F_1, F_2, \dots, F_n\}$  of subsets of  $\{1, 2, \dots, m\}$ , where each  $F_i$  has a cost  $c_i$ . The problem is to find a subset  $\mathcal{C} \subseteq \{1, 2, \dots, n\}$  with minimum cost  $c(\mathcal{C}) = \sum_{j \in \mathcal{C}} c_j$  and such that

$$\{1, 2, \dots, m\} = \bigcup_{j \in \mathcal{C}} F_j.$$

**Theorem 9.** *Problem (SCH) for the distance function is NP-hard in the strong sense.*

**Proof.** Given an instance of (SC), define an instance of (SCH) as follows. Let  $\mathcal{H}^C = (\mathcal{V}^C, \mathcal{E}^C)$  be a weighted hypergraph where

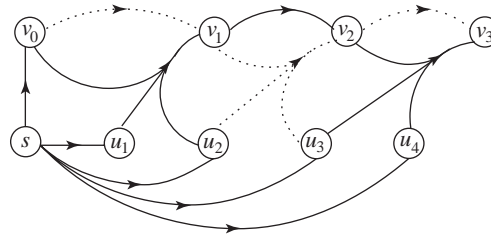
- $\mathcal{V}^C = \{s\} \cup \{u_i: 1 \leq i \leq m\} \cup \{v_j: 0 \leq j \leq n\}$ ;
- $\mathcal{E}^C = FS(s) \cup \{e_j^a: 1 \leq j \leq n\} \cup \{e_j^h: 1 \leq j \leq n\}$ ;

here  $FS(s)$  contains an arc from  $s$  to each node  $u_i$  and an arc  $(\{s\}, \{v_0\})$ ; moreover, for each  $1 \leq j \leq n$ :

- $e_j^a = (\{v_{j-1}\}, v_j)$ ;
- $e_j^h = (\{u_i: i \in F_j\} \cup \{v_{j-1}\}, v_j)$ .

The cost of each hyperarc  $e_j^h$  is  $c_j$ , arcs have zero costs. Finally, let the tree  $\mathcal{T}$  contain the arcs in  $FS(s)$ , and choose the destination node  $t = v_n$ .

Fig. 6 shows the hypergraph  $\mathcal{H}^C$  for an (SC) instance where  $m=4$ ,  $n=3$ ,  $F_1 = \{1, 2\}$ ,  $F_2 = \{2, 3\}$  and  $F_3 = \{3, 4\}$ . The optimal hyperpath is represented by solid lines.

Fig. 6. Hypergraph  $\mathcal{H}^C$ .

An  $s$ - $t$  hyperpath in  $\mathcal{H}^C$  is *feasible* if it contains  $\mathcal{T}$ . Observe that, in any feasible  $s$ - $t$  hyperpath  $\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$  it is  $\mathcal{V}_\pi = \mathcal{V}^C$ , and for each  $j > 0$ , the predecessor  $p(v_j)$  is either  $e_j^h$  or  $e_j^a$ . Therefore  $\pi$  is univocally defined by the set

$$\mathcal{C} = \{j: e_j^h \in \mathcal{E}_\pi\},$$

and the weight using the distance function of each node  $v_j$ ,  $j > 0$ , is given by the weight of  $v_{j-1}$  plus the cost of  $p(v_j)$ ; the weight of  $\pi$  is thus  $c(\mathcal{C}) = \sum_{j \in \mathcal{C}} c_j$ . Moreover, in a feasible  $\pi$ , each node  $u_i$  must belong to the tail of some hyperarc  $e_j^h$  with  $j \in \mathcal{C}$ . Therefore, hyperpath  $\pi$  is feasible for (SCH) if and only if the corresponding  $\mathcal{C}$  is a feasible solution for (SC). Since the cost of  $\pi$  is  $c(\mathcal{C})$ , we conclude that (SC) reduces to solving the above instance of (SCH), and the thesis follows.  $\square$

Consider again the forward branching approach defined in Section 3. It is easy to see that every instance of (SCH) can be interpreted as finding the shortest hyperpath  $\pi^i$  in a set  $\Pi^i$ . For example, consider the hypergraph  $\mathcal{H}^C$  in the proof of Theorem 9, and define a new hyperarc  $h = (\{v_0\} \cup \{u_i: 1 \leq i \leq m\}, t)$ . By adding hyperarc  $h$  to  $\mathcal{T}$  we obtain an  $s$ - $t$  hyperpath  $\pi_{st}$  with  $q = m + 2$  hyperarcs. When we apply forward branching to  $\pi_{st}$ , it turns out that finding the shortest hyperpath in  $\Pi^q$  corresponds to solving the instance of (SCH) defined on  $\mathcal{H}^C$ . This suffices to prove that finding the shortest hyperpath  $\pi^i$  in  $\Pi^i$  is NP-hard.

## References

- [1] Hoffman W, Pavley R. A method for the solution of the N'th best path problem. *Journal of the Association for Computing Machinery* 1959;6:506–14.
- [2] Eppstein D. Finding the  $k$  shortest paths. *SIAM Journal on Computing* 1999;28(2):652–73.
- [3] Jiménez VM, Marzal A. Computing the  $k$  shortest paths: a new algorithm and an experimental comparison. *Lecture Notes in Computer Science* 1999;1668:15–29.
- [4] Yen YJ. Finding the  $k$  shortest loopless paths in a network. *Management Science* 1971;17(11):712–6.
- [5] Lawler EL. A procedure for computing the  $k$  best solutions to discrete optimization problems and its application to the shortest path. *Management Science* 1972;18(7):401–5.
- [6] Coutinho-Rodrigues JM, Climaco JCN, Current JR. An interactive bi-objective shortest path approach: searching for unsupported nondominated solutions. *Computers and Operations Research* 1999;26:789–98.
- [7] Handler GY, Zang I. A dual algorithm for the constrained shortest path problem. *Networks* 1980;10:293–310.
- [8] Gallo G, Longo G, Pallottino S, Nguyen S. Directed hypergraphs and applications. *Discrete Applied Mathematics* 1993;42:177–201.

- [9] Miller-Hooks ED. Adaptive least-expected time paths in stochastic, time-varying transportation and data networks. *Networks* 2000;37(1):35–52.
- [10] Miller-Hooks ED, Mahmassani HS. Optimal routing of hazardous materials in stochastic, time-varying transportation networks. *Transportation Research Record* 1998;1645:143–51.
- [11] Hall RW. The fastest path through a network with random time-dependent travel times. *Transportation Science* 1986;20(3):182–8.
- [12] Pretolani D. A directed hypergraph model for random time dependent shortest paths. *European Journal of Operational Research* 2000;123:315–24.
- [13] Cohen J. *Multiobjective programming and planning*. New York: Academic Press; 1978.
- [14] Nielsen LR, Andersen KA, Pretolani D. Bicriterion shortest hyperpaths in random time-dependent networks. *IMA Journal of Management Mathematics*, to appear.
- [15] Gallo G, Gentile C, Pretolani D, Rago G. Max Horn SAT and the minimum cut problem in directed hypergraphs. *Mathematical Programming* 1998;80:213–37.
- [16] Ausiello G, Italiano GF, Nanni U. Optimal traversal of directed hypergraphs. Technical report TR-92-073, International Computer Science Institute, Berkeley, CA, September 1992.
- [17] Gallo G, Scutellà MG. Minimum makespan assembly plans. Technical report 10, Dipartimento di Informatica, Università di Pisa, September 1998.
- [18] Nguyen S, Pallottino S. Equilibrium traffic assignment for large scale transit networks. *European Journal of Operational Research* 1988;37:176–86.
- [19] Thulasiraman K, Swamy MNS. *Graphs: theory and algorithms*. New York: Wiley-Interscience; 1992.
- [20] Gallo G, Pallottino S. Hypergraph models and algorithms for the assembly problem. Technical report 6, Dipartimento di Informatica, Università di Pisa, March 1992.
- [21] Martins EQV, Pascoal MMB. A new implementation of Yen's ranking loopless paths algorithm. Technical report, Centro de Informatica e Sistemas, 2000. Available at <http://www.mat.uc.pt/~eqvm/>.