Bicriterion shortest hyperpaths in random time-dependent networks

LARS RELUND NIELSEN[*] KIM ALLAN ANDERSEN
Department of Operations Research
University of Aarhus
Ny Munkegade, building 530
DK-8000 Aarhus C
Denmark

DANIELE PRETOLANI[*]
Dipartimento di Matematica e Informatica
Università di Camerino
Via Madonna delle Carceri
I-62032 Camerino (MC)
Italy

Abstract

In relevant application areas, such as transportation and telecommunications, there has recently been a growing focus on *dynamic networks*, where arc lengths are represented by time-dependent discrete random variables. In such networks, an optimal routing policy does not necessarily correspond to a path, but rather to an *adaptive strategy*. Finding an optimal strategy reduces to a *shortest hyperpath* problem that can be solved quite efficiently.

Bicriterion shortest path problems have been extensively studied for many years. Recently, extensions to dynamic networks have been investigated. However, no attempt has been made to study *bicriterion strategies*. This is the aim of this paper.

Here we model bicriterion strategy problems in terms of *bicriterion shortest hyperpaths*. For several problems arising in this context, optimal solutions can be found quite efficiently. Moreover, the general problem of listing *efficient strategies* can be successfully dealt with by means of heuristic methods. A computational experience is reported, where we consider several variants of the above problems. Finally, the relevant features of the bicriterion hyperpath model are discussed and compared to the classical bicriterion path approach.

Keywords: *random time-dependent networks, bicriterion shortest path, directed hypergraphs, shortest hyperpath.*

1 Introduction

One of the most classical problems encountered in the analysis of networks is the shortest path problem. Traditionally the shortest path problem was a single objective problem with

[*]Corresponding authors (e-mail: daniele.pretolani@unicam.it;relund@imf.au.dk)

the objective being minimizing total distance or travel time. Nevertheless, due to the multiobjective nature of many transportation and routing problems, a single objective function is not sufficient to completely characterize most real-life problems. In a road network for instance, two parameters, time and cost, can be assigned to each arc. Clearly, often the fastest path may be too costly or the cheapest path may be too long. Therefore the decision maker must choose a solution among the paths, where it is not possible to find a different path such that time or cost is improved without getting a worse cost or time, respectively (efficient path). The problem is called the bicriterion shortest path problem (*bi-SP*) and has generated wide interest in multicriterion linear integer programming, see e.g. [8, 9]. Garey and Johnson [11] showed that bi-SP is $\mathcal{NP}$-hard since there can be exponentially many different efficient paths.

Several solution methods has been developed to solve bi-SP. They can be partitioned into two main categories, namely *path/tree* approaches and *node labelling* (label setting/label correcting) methods. For node labelling methods see [3, 25]. The path/tree methods can be further partitioned into "two-phases" methods and "pure K-shortest path" methods. In Martins [4] the K-shortest path method was used and the problem was solved by first finding an upper bound on one criteria and then using a K-shortest path procedure to find all efficient solutions below that upper bound. The method seems to be slow, since there are too many paths to search [18]. In Mote [18] a two-phases approach was considered. First phase found the *extreme nondominated* points using an LP-relaxation and second phase searched for more nondominated points using a label correcting approach. More recently, interactive approaches which find only a part of the nondominated solutions have been studied [7, 6]. Here the two-phases method is used where the first phase finds the extreme nondominated points by solving shortest path problems and the second phase finds more nondominated points by using a K-shortest path procedure. For a recent overview of solution methods for bi-SP we refer to [24].

In relevant application areas, such as transportation and telecommunications, several other extensions of the shortest path problem have been considered. Hall [12] introduced the problem of finding the minimum expected travel time (*MET*) through a dynamic network where arc lengths are represented by time-dependent random variables. He pointed out that the best route through the network is not necessarily an origin-destination path, but rather a strategy that assigns optimal successors to a node as a function of time. Note the MET problem can be seen as a stochastic multistage recourse model (see [2]). A decision is taken each time we leave a node and after each stage the travel time for the path traveled so far is known. Pretolani [23] showed that finding the optimal MET strategy reduces to solving a shortest hyperpath problem on a time expanded directed hypergraph when discrete random variables are assumed. For directed hypergraphs, shortest hyperpaths have been well examined and fast algorithms exist, see among others [10, 14, 15, 19].

Now, consider the minimum expected travel time path problem (*METP*) in random time-dependent networks which consists in finding a path that minimizes expected travel time. Hall [12] showed that METP can not be solved using standard shortest path methods and later METP has been proven to be $\mathcal{NP}$-hard even for (non-random) time-dependent networks [23]. Nonetheless, there have been a few attempts to solve the METP problem on random time-dependent networks [22]. Furthermore, recently a bicriterion version of the METP problem has been considered where *efficient paths* are searched. Here the first criteria is MET and the second is expected cost [16, 17]. Since METP is $\mathcal{NP}$-hard, only an approximation of the true set of efficient paths is found.

To the authors' knowledge, no one has yet tried to find efficient strategies instead of efficient paths. Since a strategy corresponds to a hyperpath, this amounts to search efficient hyperpaths.

In this paper we model bicriterion strategies in terms of bicriterion shortest hyperpaths. For several problems arising in this context optimal solutions can be found quite efficiently. Furthermore, the general problem of listing all efficient strategies can be successfully dealt with by means of heuristic methods. A two-phases approach is used where first phase finds efficient strategies on the boundary of the solution space by using a NISE-like procedure [5]. In the second phase we find efficient strategies by searching inside the solution space, by means of a newly developed K-shortest hyperpath procedure (see [21]).

The paper is organized as follows. Directed hypergraphs and random time-dependent networks are introduced in Section 2. The bicriterion shortest hyperpath problem is described in Section 3 and different procedures are developed. In Section 4 computational results are reported. 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 $\mathcal{V} = (v_1, ..., v_n)$ is the set of nodes, and $\mathcal{E} = (e_1, ..., e_m)$ 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 set of *tail* nodes and $h(e) \in \mathcal{V} \setminus T(e)$ denotes the *head* node. Note that a hyperarc has exactly one node in the head, and possibly several nodes in the tail. A more general class of hypergraphs, where hyperarcs can have several nodes in the head, was introduced by Gallo *et al.* [10]. The class of hypergraphs considered here were denoted as *B-graphs* in [10].

The *cardinality* of a hyperarc e is the number of nodes it contains, i.e. $|e| = |T(e)| + 1$. We call e an *arc* if $|e| = 2$. The *size* of $\mathcal{H}$ is the sum of the cardinalities of its hyperarcs:

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

We denote by

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

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 $\mathcal{H}$ is a sequence:

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

where, for $i = 1, \ldots, q$, $v_i \in T(e_i)$ and $v_{i+1} = h(e_i)$. A node v is *connected* to node u if a path P_{uv} 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

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

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

Notice that, in a valid ordering any node $u_j \in T(e)$ precedes node $h(e)$. The next theorem has been proven by Gallo *et al.* [10], and generalizes a well-known property of acyclic directed graphs:

Theorem 1 *$\mathcal{H}$ is acyclic if and only if a valid ordering of the nodes in $\mathcal{H}$ is possible.*

Note that a valid ordering in an acyclic hypergraph is in general not unique, which is also the case for acyclic directed graphs. An $O(\mathit{size}(\mathcal{H}))$ algorithm finding a valid ordering is given in [10].

2.1 Hyperpaths and hypertrees

Consider a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. A *hyperpath* π_{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. $s, t \in \mathcal{V}_\pi = \bigcup_{e \in \mathcal{E}_\pi} \big(T(e) \cup \{h(e)\}\big)$
3. $u \in \mathcal{V}_\pi \setminus \{s\} \Rightarrow u$ is connected to s in $\mathcal{H}_\pi$.

We say that node t is *hyperconnected* to s in $\mathcal{H}$ if there exists in $\mathcal{H}$ a hyperpath π_{st}. Note condition 3 and the minimality imply that, for each $u \in \mathcal{V}_\pi \setminus \{s\}$, there exists a unique hyperarc $e \in \mathcal{E}_\pi$, such that $h(e) = u$; hyperarc e is the *predecessor* of u in π_{st}. Conversely, condition 3 can be replaced by:

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

where $\mathcal{N} = \mathcal{V}_\pi \setminus \{s\}$. The definition of hyperpath can be extended to *hypertrees.*

Definition 2 *A directed hypertree with root node s is an acyclic hypergraph $\mathcal{T}_s = (\{s\} \cup \mathcal{N}, \mathcal{E}_\mathcal{T})$ with $s \notin \mathcal{N}$ satisfying condition 4.*

It is not difficult to show that a directed hypertree $\mathcal{T}_s = (\{s\} \cup \mathcal{N}, \mathcal{E}_\mathcal{T})$ contains a unique s-u hyperpath for each node $u \in \mathcal{N}$. Moreover, $\mathcal{T}_s$ can be described by a *predecessor function* $p : \mathcal{N} \to \mathcal{E}$; for each $u \in \mathcal{N}$, $p(u)$ is the unique hyperarc in $\mathcal{T}_s$ which has node u as the head. An emphasized hypertree in an acyclic hypergraph is shown in Figure 3 (see Section 2.3).

2.2 Shortest and K-shortest hyperpaths

Consider a hypergraph $\mathcal{H}$ where each hyperarc e is assigned a non-negative real weight $w(e)$. Given a hyperpath π_{st} in $\mathcal{H}$, a *weighting function* W_π assigns a weight $W_\pi(u)$ to each node u in π_{st}. The weight of hyperpath π_{st} is $W_\pi(t)$. In particular, we consider *additive weighting functions*, that can be defined by the recursive equations:

$$W_\pi\,(u) = \begin{cases} w(p(u)) + F(p(u)) & u \in \mathcal{V}_\pi \setminus \{s\} \\ 0 & u = s \end{cases} \tag{1}$$

where $F(e)$ is a nondecreasing function of the weights of the nodes in $T(e)$. Several weighting functions have been introduced in the literature (see e.g. [1, 10, 13]); here, we consider two of them, namely the *distance* and the *value*.

The *distance* function is obtained by defining $F(e)$ as follows:

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

and the *value* function is obtained as follows:

$$F(e) = \sum_{v \in T(e)} a_e(v)\, W_\pi(v)$$

where $a_e(v)$ is a nonnegative multiplier defined for each hyperarc e and node $v \in T(e)$. In this paper, we shall concentrate on a particular case of the value function, namely the *mean*, that arises when for each hyperarc the multipliers sum up to one:

$$\sum_{v \in T(e)} a_e(v) = 1, \quad \forall\, e \in \mathcal{E}.$$

The distance (value, mean) of a hyperpath π_{st} is the weight of π_{st} with respect to the distance (value, mean) weighting function. Trivially, for each hyperpath the mean is a lower bound on the distance.

Note that the value of an *s*-*t* hyperpath π defined by the predecessor function p can be written as:

$$W(\pi_{st}) = \sum_{u \in \mathcal{V}_\pi \setminus \{s\}} f_\pi(u) w(p(u)) \tag{2}$$

where f_π is defined by the following recursive equations:

$$f_\pi(u) = \begin{cases} 1 & u = t \\ \sum_{e \in FS(u)} a_e(u) f_\pi(h(e)) & u \in \mathcal{V}_\pi \setminus \{s, t\} \end{cases} \tag{3}$$

Intuitively, $f_\pi(u)$ is the "contribution" of the node weight $W_\pi(u)$ to the hyperpath weight $W(\pi_{st})$, and can be computed by processing the nodes backwards according to a valid ordering V for π. More precisely, the following proposition holds.

Proposition 1 Given any valid ordering V of the nodes in the hyperpath π, we have that $f_\pi(u)$ does not depend on the values of f_π for nodes that precede u in V.

Example 1 A hyperpath π is shown in Figure 1; the weight $w(e)$ is given close to each hyperarc e. We consider the *mean* weighting function, where multipliers are defined as: $a_e(u) = 1/2$ if $|T(e)| = 2$ and $a_e(u) = 1$ otherwise. The weight $W_\pi(u)$ is reported inside each node u; the number close to u is $f_\pi(u)$. ■

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* $\mathcal{T}_s$ which provides minimum weight hyperpaths to all hyperconnected nodes. The shortest hyperpath problem has been shown to be polynomially solvable provided that the hypergraph does not contain *decreasing cycles*. In this situation, quite efficient procedures for finding the shortest hypertree exist, see [10] for a general

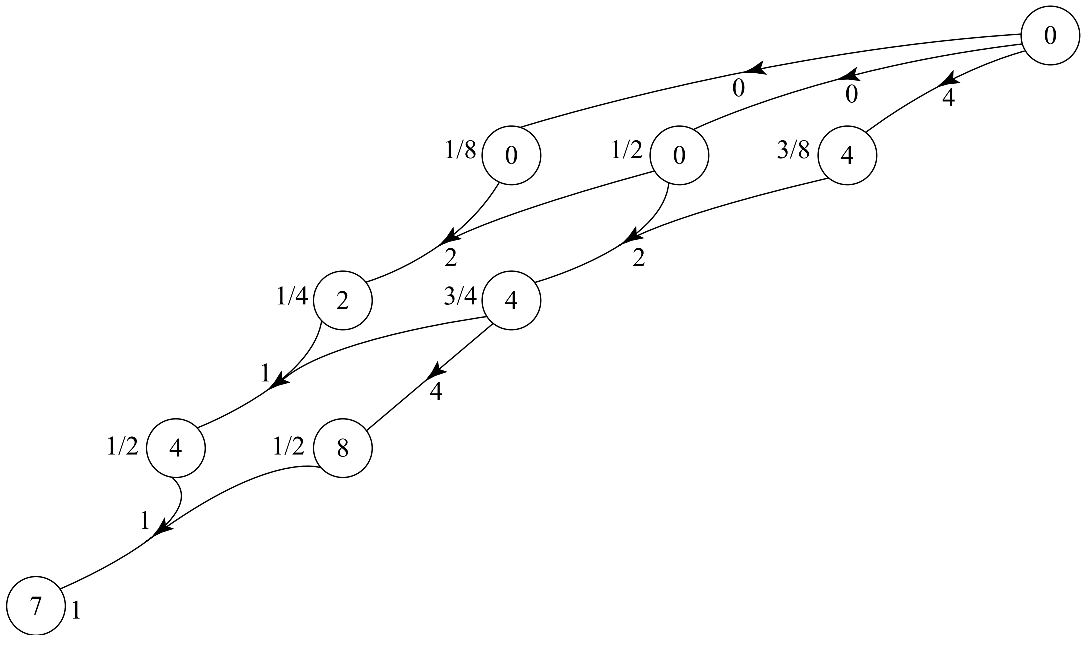

Figure 1: The hyperpath π.

overview. In this paper we shall concentrate on acyclic hypergraphs, that clearly contain no decreasing cycles. For this particular case, a simple and fast shortest hyperpath procedure exists (procedure *SFT-Acyclic*, see [10]) whose computational complexity is $O(size(\mathcal{H}))$.

The *K-shortest hyperpaths problem* consists in ranking the first K s-t hyperpaths in non-decreasing order of weight, with respect to a given weighting function, and for a given pair of nodes s and t. In a more general version, that we shall consider here, all the hyperpaths with weight up to a given upper bound must be ranked. This problem can be considered as a hypergraph extension of the classical K-shortest paths problem in graphs.

Efficient algorithms for K-shortest hyperpaths were developed in [21]. These algorithms are based on an implicit enumeration method, where the set of solutions is partitioned into smaller sets by recursively applying a *branching* step. Given the hypergraph $\mathcal{H}$, denote by Π the set of hyperpaths from s to t. Assume that a shortest hyperpath π is known with valid ordering

$$V = (s = u_1, u_2, \ldots, u_{q+1} = t)$$

In the branching step, the set $\Pi \setminus \{\pi\}$ is partitioned into q subsets Π^i, $1 \leq i \leq q$ as follows:

- s-t hyperpaths in Π^q do not contain hyperarc $p(u_{q+1})$, that is $p(t)$;
- for $1 \leq i < q$, s-t hyperpaths in Π^i contain hyperarcs $p(u_j)$, $i+1 < j \leq q+1$, and do not contain hyperarc $p(u_{i+1})$.

In this case, finding a shortest hyperpath $\pi^i \in \Pi^i$ reduces to solving a shortest hypertree 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 $\mathcal{H}^i$ is obtained from $\mathcal{H}$ by *branching on node* u_{i+1}. As a consequence, each set Π^i can be represented by the corresponding hypergraph $\mathcal{H}^i$. A *branching operation on* π 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 $\Pi \setminus \{\pi\}$.

The algorithms developed in [21] maintain an ordered list L of *subproblems*; each subproblem p_r corresponds to a particular subhypergraph $\mathcal{H}_r$. At each step, a subproblem p_r is selected from L, and the shortest *s*-*t* hyperpath π_r in $\mathcal{H}_r$ (if any) is stored. Then, branching is applied to π_r, adding to L the returned hypergraphs. The various algorithms differ in the way subproblems are ranked in L.

In the basic version (procedure *Yen*) each subproblem p_r is ranked according to the weight of the shortest hyperpath π_r in $\mathcal{H}_r$. In a faster version (procedure *LBYen*) problems are ranked according to a lower bound on the weight of the shortest hyperpath. In this way, hyperpaths are not necessarily stored in the right order; however, the following property holds:

Property 1 When a subproblem with lower bound *lb* is selected from L, all the hyperpaths with weight less than *lb* have been stored.

Thus both procedures *Yen* and *LBYen* can be used to find all hyperpaths with weight up to a given upper bound. Furthermore, it can be proved that the lower bound used in procedure *LBYen* gives the actual shortest hyperpath length on acyclic hypergraphs. Therefore, for this particular case, a specialized and quite efficient version of procedure *LBYen* can be devised, denoted as *AYen*.

Procedure *LBYen* can also be adapted to the case where we do not have a lower bound on the hyperpath weight, but just an *estimate*. In this case, Property 1 does no longer hold, i.e. procedure *LBYen* becomes a heuristic method for generating "good" hyperpaths. Clearly, the quality of the method depends on the tightness of the estimate. An application of this approach, based on a particular estimate function, will be discussed in the following sections.

2.3 Random time-dependent networks

In a *random time dependent network* (*RTDN*), often referred to as *dynamic network*, the travel time through an arc is a *random variable* of which the distribution depends on the departure time. In particular, we concentrate on *discrete* RTDNs, where both departure and travel times are integers in a finite interval.

Hall [12] introduced the problem (denoted as *MET* here) of finding the *minimum expected travel time* through a dynamic network. He pointed out that the best route in a dynamic network does not necessarily correspond to an origin-destination path, but rather to a *strategy*, that assigns optimal successors to a node as a function of time. Hall also proposed a solution approach to finding an optimal strategy, but he did not provide an actual algorithm. Moreover, he observed that the proposed approach was supposed to be effective only for networks of limited size.

As shown in [23], directed hypergraphs can be used to model discrete dynamic networks; the minimum expected travel time problem then reduces to solving a suitable shortest hyperpath problem. We illustrate the hypergraph model by means of the following example.

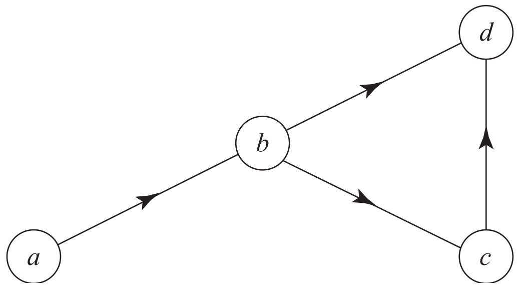

Figure 2: The topological network G.

$(i,j),t$	$(a,b),0$	$(b,c),1$	$(b,c),2$	$(b,d),1$	$(b,d),2$	$(b,d),4$	$(c,d),2$	$(c,d),3$
$I(i,j,t)$	$\{1,2\}$	$\{2,3\}$	$\{3\}$	$\{3\}$	$\{5,6\}$	$\{6,7\}$	$\{3,4\}$	$\{4,5\}$

Table 1: Input parameters.

Example 2 Consider the *topological network* $G = (N, A)$ in Figure 2, where a is the origin node and d is the destination node. Recall that travel times along arcs in G are discrete, integer valued random variables. For each arc in G, the possible departure and arrival times are listed in Table 1. Here a pair $((i,j),t)$ corresponds to a possible leaving time t from node i along arc (i,j), while $I(i,j,t)$ denotes the corresponding set of possible arrival times at node

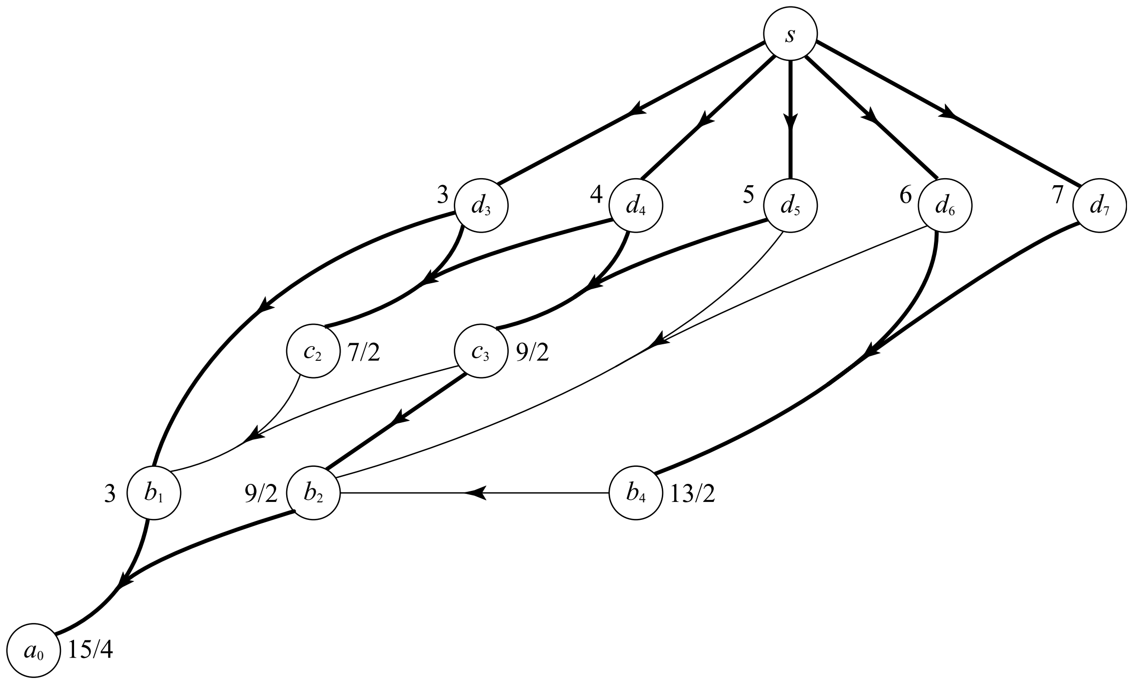

Figure 3: The time expanded hypergraph $\mathcal{H}$.

j. For each $t' \in I(i,j,t)$ we denote by $p_{ijt}(t')$ the corresponding probability. For the sake of simplicity, we assume that each random variable has a uniform distribution, i.e. for each $t' \in I(i,j,t)$ we have $p_{ijt}(t') = 1/|I(i,j,t)|$. For example, if we leave node c at time 2 along arc (c,d) we arrive at node d at time 3 or 4 with the same probability 1/2.

Observe that it is not possible to arrive at node b (from node a) at time 4, however, it is possible to leave node b (towards node d) at time 4. Here we assume that a passenger arriving at node b at time 2 can *wait in node* b until time 4, and then proceed along arc (b,d); we also assume that waiting is not allowed elsewhere.

Given G and Table 1, a *time expanded hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ can be defined as follows. Introduce a node i_t for each possible departure or arrival time t from/at node i. For each pair $((i,j),t)$ create a hyperarc $e_{ij}(t) = \big(\{j_h : h \in I((i,j),t)\}, i_t\big)$. Moreover, introduce an arc $\big(\{b_4\}, b_2\big)$ to represent waiting at node b from time 2 to time 4. Finally, a dummy node s and dummy arcs $e_s(t) = \big(\{s\}, d_t\big)$ are created. The time expanded hypergraph $\mathcal{H}$ is shown in Figure 3; numbers close to nodes will be explained later. Note that the orientation of the hyperarcs is opposite to the orientation of arcs in G, and that solid lines define a hypertree $\mathcal{T}_s$ in $\mathcal{H}$. ■

As shown in [23], each strategy in G is represented by a hypertree $\mathcal{T}_s$ in $\mathcal{H}$; the predecessor $p(i_t)$ in $\mathcal{T}_s$ provides the successor (an arc in G) for node i at time t. For example, the hypertree in Figure 3 states that if we leave node b at time 1, then we travel along arc (b,d), while if we leave node b at time 2 we travel along arc (b,c).

Let us assign the following weights and multipliers to the hyperarcs in $\mathcal{H}$:

- For each arc $e = \big(\{u\}, v\big)$ in $\mathcal{H}$, let $a_e(u) = 1$;
- For each remaining hyperarc $e = e_{ij}(t)$, let $a_e(u) = p_{ijt}(t')$ for each node $u = (j_{t'} \in T(e))$;
- Assign a weight t to each arc $e_s(t)$;
- Assign weight zero to the remaining hyperarcs.

Consider a strategy represented by hypertree $\mathcal{T}_s$. It can be proved [23] that the expected arrival time at the destination for a traveller leaving node i at time t is given by the *mean* $W(i_t)$ of the unique hyperpath from s to i_t in $\mathcal{T}_s$. Clearly, given the pair (i,t), arrival time and travel time coincide, up to the additive constant t. Therefore, MET reduces to finding optimal expected arrival times, i.e. to a minimum mean hyperpath problem in $\mathcal{H}$.

The hypertree $\mathcal{T}_s$ in Figure 3 represents the strategy that minimizes expected arrival times for each node and departure time. Expected arrival times are given, close to each node.

Note that the time expanded hypergraph is acyclic: a valid ordering is obtained by ranking nodes in reverse order of time. In addition, the size of $\mathcal{H}$ is proportional to the size of the input data. Since the minimum mean hyperpath problem in acyclic hypergraphs can be solved in linear time, we can state the following proposition:

Proposition 2 The MET problem in discrete dynamic networks can be solved in linear time.

Optimal strategies under different objectives can be found by using suitable weights, multipliers, and weighting functions. In this paper, we shall consider generic *expected cost* criteria. Moreover, we shall consider *min-max* problems, where the goal is to minimize the maximum possible cost or arrival time. These problems reduce to solving a minimum distance hyperpath problem [23]. Additional features, such as time windows, can be easily introduced in the model, but shall not be considered here.

Consider now a *path strategy*, that assigns to each node $i \in G$ the same successor arc (i, j) for each possible departure time from i. It can be shown that a path strategy defines an origin-destination path in G. The *minimum expected travel time path* problem (*METP*) consists in finding the path strategy that minimizes expected travel times. Equivalently, METP can be defined as the problem of finding the origin-destination path in G that minimizes the expected travel time.

Hall [12] observed that METP cannot be solved by standard shortest path methods, and proposed a *dual* enumerative solution method. However, he provided no computational complexity results. Later, METP has been proved [23] to be NP-hard also for standard (i.e. non-random) time-dependent networks. In light of the above results, we stress on the fact that finding an optimal *path* in a discrete dynamic network is in general a difficult problem, while finding an optimal *strategy* (i.e. a hyperpath) is easy. This observation motivated the extension of the hyperpath model to bicriterion problems, that we explore in the next section.

3 Bicriterion shortest hyperpaths

In this section we consider the bicriterion shortest hyperpath problem (bi-SBT) which is the extension to directed hypergraphs of the bicriterion shortest path problem (bi-SP). It is evident that bi-SBT has a much richer structure than bi-SP due to the possible choices of weighting functions. Furthermore, combinations of different weighting functions are now possible. Nevertheless, the standard terminology usually adopted in the formal treatment of bi-SP immediately extends to bi-SBT. Therefore, here we introduce the terminology for bi-SBT directly, and consider bi-SP as a particular case. After a formal statement of the problem, we introduce the *two-phases* method for bi-SP. Then, we discuss the *parametric weight* shortest hyperpath problem, which is then used to devise a two-phases method for bi-SBT.

We focus on the application to random time-dependent networks (see Section 2.3) where the two criteria may correspond to time as well as cost, and the purpose may be to minimize the expected as well as the maximum possible time or cost. This can be accomplished by suitably choosing the mean and distance functions for the two objectives. Note that cost and time can be treated in a uniform way. Here we concentrate on the *mean/mean* case, where two mean functions with the same multipliers (corresponding to probabilities) are considered. The resulting problem is denoted by bi-SBT$_{m/m}$. Two other choices of weighting functions for the two criteria are considered, namely the *distance/distance* case bi-SBT$_{d/d}$ and the *mean/distance* case bi-SBT$_{m/d}$.

Given a hypergraph $\mathcal{H}$, assume that each hyperarc e is assigned two real weights $w_1(e)$ and $w_2(e)$. Furthermore, let $W_i(\pi)$, $i = 1, 2$ denote the weight of hyperpath π using weights $w_i(e)$. The *bicriterion shortest hyperpath problem* (bi-SBT) can now be informally stated as follows:

$$\min_{\pi \in \Pi} \{(W_1(\pi), W_2(\pi))\} \tag{4}$$

where Π is the set of possible s-t hyperpaths. Here, minimization is intended in terms of *Pareto-optimality*, that is, finding hyperpaths where the two weights are minimal in the sense that we cannot improve one weight without worsening the other. In order to formally define problem (4), we need a few definitions. We follow the terminology of [24].

Definition 3 A hyperpath $\pi \in \Pi$ is *efficient* if and only if

$$\nexists \textit{path } \tilde{\pi} \in \Pi : W_1(\tilde{\pi}) \leq W_1(\pi) \text{ and } W_2(\tilde{\pi}) \leq W_2(\pi)$$

with at least one strict inequality; otherwise π is *inefficient.*

Efficient hyperpaths are defined in the decision space Π, and their counterpart are points in the *criterion space*:

$$\mathcal{W} = \left\{ W(\pi) \in \mathbb{R}^2 \mid \pi \in \Pi \right\}$$

where $W(\pi) \in \mathbb{R}^2$ is the vector with components $W_1(\pi)$ and $W_2(\pi)$.

Definition 4 A point $W(\pi) \in \mathcal{W}$ is a *nondominated* criterion point if and only if π is an efficient hyperpath. Otherwise $W(\pi)$ is a *dominated* criterion point.

Let us define

$$\begin{aligned} \Pi_{\text{Eff}} &= \{\pi \in \Pi \mid \pi \text{ is efficient}\} \\ \mathcal{W}_{\text{Eff}} &= \left\{ W(\pi) \in \mathbb{R}^2 \mid \pi \in \Pi_{\text{Eff}} \right\} \end{aligned}$$

Now we are in a position to explain what "solving" problem (4) means. It means finding the set of efficient hyperpaths Π_{Eff}, or equivalently, the set of nondominated criterion points $\mathcal{W}_{\text{Eff}}$.

The criterion points can be partitioned into two kinds, namely *supported* and *unsupported.* The supported ones can be further subdivided into *extreme* and *nonextreme.* To this aim, let us define the following set

$$\mathcal{W}^{\geq} = \text{conv}\left(\mathcal{W}_{\text{Eff}}\right) \oplus \left\{ \mathbf{w} \in \mathbb{R}^2 \mid \mathbf{w} \geq 0 \right\};$$

where $\oplus$ as usual denotes direct sum, and conv($\mathcal{W}$) denotes the convex hull of $\mathcal{W}$.

Definition 5 $W(\pi) \in \mathcal{W}_{\text{Eff}}$ is a *supported* nondominated criterion point if $W(\pi)$ is on the boundary of $\mathcal{W}^{\geq}$. Otherwise $W(\pi)$ is *unsupported.*

Definition 6 A supported point $W(\pi)$ is a *extreme* if $W(\pi)$ is an extreme point of $\mathcal{W}^{\geq}$. Otherwise $W(\pi)$ is *nonextreme.*

Notice that unsupported nondominated points (in fact, all vectors in $\mathcal{W}$) are dominated by a convex combination of extreme supported points [26]. It is well known that a set of nondominated points $\Phi = \left\{W^1, W^2, \ldots, W^k\right\} \subseteq \mathbb{R}^2$ can be ordered such that:

$$W_1^1 < W_1^2 < \ldots < W_1^k, \qquad W_2^1 > W_2^2 > \ldots > W_2^k$$

We call Φ an *ordered nondominated set.* In the following, we use the term *frontier* to denote the ordered nondominated set of extreme supported points in $\mathcal{W}$.

We shall also need the concepts of ε-domination and ε-approximation. The definitions below follow the terminology given in [27].

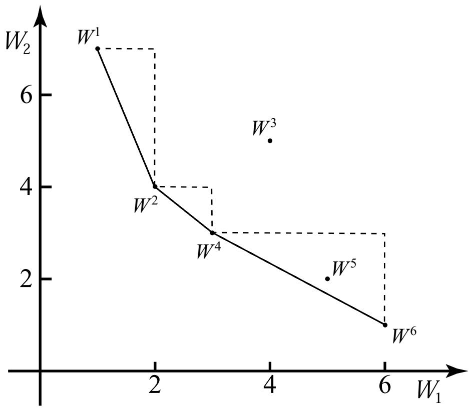

Figure 4: The criterion space.

Definition 7 A point (W_1, W_2) *ε-dominates* point $\left(\hat{W}_1, \hat{W}_2\right)$ if

$$\hat{W}_1 \geq (1-\varepsilon)\, W_1, \qquad \hat{W}_2 \geq (1-\varepsilon)\, W_2$$

Definition 8 A nondominated set Φ_1 is an ε-approximation of another nondominated set Φ_2 if for each point $\hat{W} \in \Phi_2$ there exists $W \in \Phi_1$ such that W ε-dominates $\hat{W}$.

Example 2 (continued) Assume that some hyperarcs in Figure 3 are assigned two weights as shown in the following table; the remaining hyperarcs are assigned zero weights.

$e_{ab}\,(0)$	$e_{bc}\,(1)$	$e_{bd}\,(1)$	$e_{bc}\,(2)$	$e_{bd}\,(2)$	$e_{bd}\,(4)$	$e_{cd}\,(2)$	$e_{cd}\,(3)$	$e_s\,(5)$
$(1,1)$	$(0,1)$	$(6,0)$	$(0,4)$	$(2,0)$	$(4,0)$	$(0,2)$	$(0,2)$	$(0,4)$

In this particular case there are 6 hyperpaths from node s to node a_0 depending on the choice of predecessor in node b_1 and b_2. In criterion space the corresponding vectors $W = (W_1, W_2)$ are given by: $W^1 = (1,7)$, $W^2 = (2,4)$, $W^3 = (4,5)$, $W^4 = (3,3)$, $W^5 = (5,2)$ and $W^6 = (6,1)$. These points are illustrated in Figure 4; the solid lines represent the frontier, that contains the four extreme supported nondominated points W^1, W^2, W^4 and W^6. Note that the frontier defines three triangles, shown with dashed lines, where it may be possible to find unsupported nondominated points such as W^5. Points which do not lay inside the triangles such as W^3 are dominated. ■

The *two-phases approach* has been widely developed in the bi-SP literature, see e.g. [6, 7, 18]. As the name suggests, the method splits the search for nondominated points into two phases. In phase one the frontier is determined; this defines the triangles in which further nondominated points may be found. Phase two proceeds to search the triangles one at the time.

Graphically, the triangle search resembles a "sweeping line" procedure, as illustrated in Figure 5. The line containing the two points W^i and W^{i+1} is moved upwards, and a point $W(\pi)$ is considered when the line intersects it. In principle, the line should span the whole triangle, up to the vertex marked ub_0 in Figure 5. However, when a new nondominated point

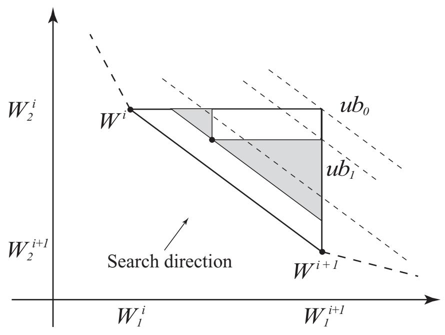

Figure 5: A triangle defined by W^i and W^{i+1}.

is found inside the triangle, the upper limit can be updated to ub_1, since the remaining part of the triangle cannot contain efficient points.

Computationally, the two-phases method requires to solve shortest hyperpath and K-shortest hyperpaths problems with respect to a *parametric weight*, that is a linear combination of the two criteria. These problems will be considered next. The two-phases method for bi-SBT, as well as some approximated variants, will be described in detail later.

3.1 The Parametric Weight Problem

Let $\gamma : (\Pi, \mathrm{I\!R}^+) \rightarrow \mathrm{I\!R}^+$ denote the *parametric weight* of a hyperpath π:

$$\gamma(\pi, \lambda) = W_1(\pi)\lambda + W_2(\pi).$$

Given $\lambda > 0$, the *parametric weight shortest hyperpath* problem, denoted by SBT(λ), consists in finding a hyperpath with minimum parametric weight. We shall denote by $\gamma(\lambda)$ and $\pi(\lambda)$ the minimum parametric weight and an optimal hyperpath (there may be many) for SBT(λ), respectively.

For a fixed λ, consider the hyperpaths with the same parametric weight δ; clearly, the corresponding points in the parametric space belong to the same straight line, defined by the equation:

$$\lambda W_1 + W_2 = \delta. \tag{5}$$

Therefore, solving SBT(λ) amounts to finding the minimum parametric weight δ such that the corresponding line (5) intersects $\mathcal{W}$. It follows immediately that for each λ a hyperpath with minimum parametric weight corresponds to a supported point. Note also that, for increasing δ, line (5) moves upwards. In other words, the triangle search described above corresponds to a K-shortest hyperpath procedure, where hyperpaths are ranked according to their parametric weight.

It is well known that, as long as directed graphs are considered, SBT(λ) reduces to solving a standard shortest path problem where each arc a is assigned a weight $w_\lambda(a) = w_1(a)\lambda + w_2$. Remarkably, this property can be extended to the bi-SBT$_{m/m}$ case. Let $\mathcal{H}_\lambda = (\mathcal{V}, \mathcal{E})$ denote the hypergraph in which the weight on each hyperarc $e \in \mathcal{E}$ is $w_\lambda(e) = w_1(e)\lambda + w_2(e)$. The following theorem holds.

Theorem 2 Consider problem SBT(λ) for the bi-SBT$_{m/m}$ case, and let $W_\lambda(\pi)$ denote the mean of a hyperpath π in $\mathcal{H}_\lambda$. For every $\lambda > 0$ and for every $\pi \in \Pi$ we have that $W_\lambda(\pi) = \gamma(\pi, \lambda)$.

Proof It suffices to write the mean $W_\lambda(\pi)$ of the s-t hyperpath π according to (2):

$$\begin{aligned} W_\lambda(\pi) &= \sum_{u\in\mathcal{V}_\pi\setminus\{s\}} f_\pi(u)w_\lambda(p(u)) \\ &= \sum_{u\in\mathcal{V}_\pi\setminus\{s\}} \big(f_\pi(u)w_1(p(u))\lambda\big) \;+\; \big(f_\pi(u)w_2(p(u))\big) \\ &= \lambda \sum_{u\in\mathcal{V}_\pi\setminus\{s\}} f_\pi(u)w_1(p(u)) + \sum_{u\in\mathcal{V}_\pi\setminus\{s\}} f_\pi(u)w_2(p(u)) \\ &= \gamma(\pi,\lambda). \end{aligned}$$

■

Unfortunately, a result similar to Theorem 2 does not hold for bi-SBT$_{d/d}$, as shown by the following example.

Example 2 (continued) Consider the hypergraph in Figure 3, and assume $\lambda = 1$. The minimum distance s-a_0 hyperpath π in $\mathcal{H}_\lambda$ has weight $W_\lambda(\pi) = 8$. However, if we consider the two distance functions separately, the minimum parametric weight in $\mathcal{H}$ is $\gamma(1) = 5 + 7 = 12$. ■

Note that in the above example we have $W_\lambda(\pi) \le \gamma(\lambda)$. This property holds true in general as shown in Theorem 3 below.

Theorem 3 Consider problem SBT(λ) for the bi-SBT$_{d/d}$ case, and let $W_\lambda(\pi)$ denote the distance of a hyperpath π in $\mathcal{H}_\lambda$. For every $\lambda > 0$ and for every $\pi \in \Pi$ we have that $W_\lambda(\pi) \le \gamma(\pi,\lambda)$.

Proof For each u in π, denote by $W_\lambda(u)$, $W_1(u)$ and $W_2(u)$ the distance of node u in π with respect to the weights w_λ, w_1 and w_2, respectively. Consider a valid ordering $V = (u_1, ..., u_p)$ for π. We shall prove by induction that for each u_i in V, $W_\lambda(u_i) \le W_1(u_i)\lambda + W_2(u_i)$. The property clearly holds for $u_1 = s$. Assume now that the property holds for each node preceding $u = u_i$ in V. Then

$$\begin{aligned} W_\lambda(u) &= \max_{v\in T(p(u))}\{W_\lambda(v)\} + w_\lambda(p(u)) \\ &= \max_{v\in T(p(u))}\{W_\lambda(v)\} + w_1(p(u))\lambda + w_2(p(u)) \\ &\le \big(\max_{v\in T(p(u))}\{W_1(v)\} + w_1(p(u))\big)\lambda \;+\; \big(\max_{v\in T(p(u))}\{W_2(v)\} + w_2(p(u))\big) \\ &= W_1(u)\lambda + W_2(u). \end{aligned}$$

Hence we have that $W_\lambda(\pi) \le \gamma(\pi,\lambda)$. ■

A similar result holds for the bi-SBT$_{m/d}$ case. The proof is not presented here, but it follows the same kind of reasoning as the proof of Theorem 3.

Theorem 4 Consider problem SBT(λ) for the bi-SBT$_{m/d}$ case, and let $W_\lambda(\pi)$ denote the mean of a hyperpath π in $\mathcal{H}_\lambda$. For every $\lambda > 0$ and for every $\pi \in \Pi$ we have that $W_\lambda(\pi) \le \gamma(\pi,\lambda)$.

Theorems 3 and 4 show that bi-SBT$_{d/d}$ and bi-SBT$_{m/d}$ are harder than the bi-SBT$_{m/m}$ case, since we cannot solve the SBT(λ) problem efficiently in these cases. Solving a shortest hyperpath problem on $\mathcal{H}_\lambda$ provides a lower bound $\pi(\lambda)$ and a feasible solution $\gamma(\lambda)$. In order to find good quality solutions for SBT(λ), we also developed a greedy heuristic procedure, derived from the Dijkstra-like shortest hyperpath procedure given in [10]. The heuristic incrementally builds a hypertree by adding a node (and its predecessor hyperarc) at each step. For each node u not yet in the hypertree a temporary label is maintained; this label corresponds to the parametric weight of a particular hyperpath from s to u in $\mathcal{H}_\lambda$. At each step, the node with minimum label is selected. Clearly, the above heuristic is not complete, since it considers only one locally optimal hyperpath for each node. Nevertheless, it often provides a solution quite close to a supported point.

In our computational experience, $W_\lambda(\pi)$ will be used as a lower bounding function for SBT(λ), while the hyperpath returned by the greedy heuristic will be used as an estimate (or upper bound) function.

3.2 First phase: Finding the frontier

Now consider the first phase of bi-SBT which consists in finding the frontier and the corresponding efficient hyperpaths. For bi-SP, this can be done in several ways, e.g. by parametric linear programming [18]; however, the most effective approach is based on a NISE like algorithm (see [5]). This method builds the frontier incrementally, adding one point at the time.

Let the ordered nondominated set $\Phi = \{W^1, W^2, \dots, W^k\}$ denote the frontier. At the beginning, the two points W^1 and W^k are determined. To this aim, we must be able to find a shortest hyperpath w.r.t. one criterion when the other one is fixed to its minimal weight. This can be done quite easily, see for example [23].

Consider now a pair of consecutive points $W^i = W(\pi^i)$ and $W^{i+1} = W(\pi^{i+1})$ in the subset of Φ determined so far. Compute the slope of the line containing them, that is the value λ such that $\gamma(\pi^i, \lambda) = \gamma(\pi^{i+1}, \lambda)$, and solve problem SBT($\lambda$). Similar to the bi-SP case, if $\gamma(\lambda) < \gamma(\pi^i, \lambda)$ then $\pi(\lambda)$ is efficient, and $W(\pi(\lambda))$ is a frontier point between W^i and W^{i+1} in the ordered set. In this case, the process is recursively repeated on the pairs $\left\{W^i, W(\pi(\lambda))\right\}$ and $\left\{W(\pi(\lambda)), W^{i+1}\right\}$. If otherwise $\gamma(\lambda) = \gamma(\pi^i, \lambda)$ then either $\pi(\lambda) \in \{\pi^i, \pi^{i+1}\}$ or $\pi(\lambda)$ is a nonextreme supported point.

As we shall see in Section 4, the set of extreme nondominated points can be very large, resulting in high CPU times. In some cases, we may be satisfied with an *ε-approximation* of the true frontier. Consider the four extreme nondominated points in Figure 6 found during the first phase. Any extreme nondominated points between W^+ and W^- must belong to the shaded area. In an ε-approximation of the frontier no new extreme points between W^+ and W^- have to be found if each point inside the shaded area is ε-dominated by either W^+ or W^-, i.e. if

$$\begin{array}{ll} (1-\varepsilon)\, W_1^- \lambda_1 + (1-\varepsilon)\, W_2^+ \leq \gamma\left(W^1, \lambda_1\right) & \text{or} \\ (1-\varepsilon)\, W_1^- \lambda_2 + (1-\varepsilon)\, W_2^+ \leq \gamma\left(W^2, \lambda_2\right) & \end{array} \qquad (6)$$

where λ_1 denotes the slope defined by W^1 and W^+ and λ_2 the slope defined by W^2 and W^-. We omit the proof of correctness of Condition (6) here.

Procedure *PhaseOne*, given below, finds an ε-approximation of the true frontier. The points W^+ and W^- are the current points used to define the slope λ and Φ is the current ordered set of extreme points. Given a point $W \in \Phi$, we let W_{next} denote the point following

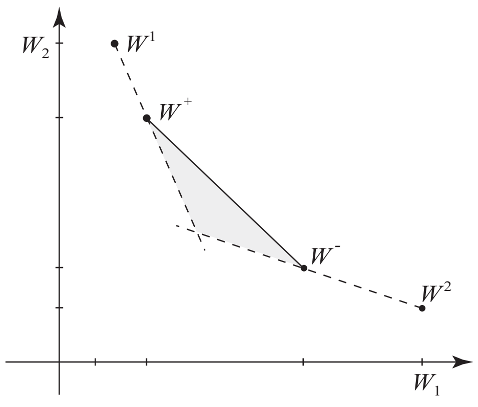

Figure 6: Using ε-dominance in the first phase (m/m case).

W in Φ. Clearly, the true frontier is obtained by omitting the control on Conditions (6) in Step 4.

Procedure *PhaseOne*(ε)

```
Step 1 find the upper/left point $W^{ul}$ and the lower/right point $W^{lr}$;
       if $W^{ul} = W^{lr}$ then STOP (there is one nondominated point);
       otherwise set $\Phi = \{W^{ul}, W^{lr}\}$; $W^- = W^{ul}$;
Step 2 $W^+ = W^-$; if $W^+ = W^{lr}$ then STOP;
Step 3 $W^- := W^+_{next}$;
Step 4 if Conditions (6) are satisfied, go to Step 2; otherwise set
       $\lambda = \left|(W_2^- - W_2^+)/(W_1^- - W_1^+)\right|$;
       find the shortest hyperpath $\pi$ in $\mathcal{H}_\lambda$;
Step 5 if $W(\pi)$ is a new extreme point then set $\Phi := \Phi \cup \{W(\pi)\}$ and go to Step 3;
       otherwise go to Step 2.
```

Phase one requires to solve problem SBT(λ). Theorem 2 shows that this can be done efficiently for the bi-SPT$_{m/m}$ case. For the bi-SPT$_{d/d}$ and bi-SPT$_{m/d}$ cases, an *approximated* frontier can be computed by solving SBT(λ) approximately, as shown before. Note that an approximate frontier may contain only a subset of the true frontier points, and may also contain dominated points. Nevertheless, the approximate frontier defines a set of triangles that can be searched (approximately) in phase two.

Moreover, since SBT(λ) is solved approximately, it may happen that $W(\pi)$ dominates some points in the current frontier set Φ. In this case, it might be necessary to remove the dominated points, thus, a slightly more complex implementation of Step 5 is required to maintain Φ.

3.3 Phase two: Looking into the triangles

After the first phase an ordered set $\Phi = \{W^1, W^2, ..., W^k\}$ has been found (which might be an approximation of the true frontier); Φ gives rise to a set of $k-1$ triangles in which further nondominated points are searched in phase two. Note that each triangle is searched independently, thus some triangles may be ignored. If the *interactive* approach is adopted (see [7, 6]) a decision maker may discard some triangles *a priori*, based on external informations.

Let us consider the bi-SBT$_{m/m}$ case first, and let the current triangle be defined by W^q and W^{q+1} in Φ. Note since Theorem 2 holds W^q and W^{q+1} are extreme nondominated points of the true frontier. Furthermore, if there exists another extreme nondominated point between W^q and W^{q+1}, which is not in Φ, then it must be ε-dominated by W^q or W^{q+1} according to (6). Hence is the corresponding triangle also ε-dominated and not searched. In our computational experience, we denote the triangles searched "large" triangles (or "gaps").

Theorem 2 also assures that the true ranking of the parametric weights can be obtained. We can thus search the triangle using the K-shortest hyperpaths procedure *AYen* (see Section 2) until all hyperpaths which may correspond to nondominated points inside the triangle have been found. As shown before, the maximum parametric weight to be considered is $\lambda W_1^{q+1} + W_2^q$, obtained from the upper right vertex of the triangle. Recall that this upper bound can be decreased during the search, if new nondominated points are found. See [21] for further algorithmic details.

Clearly, the efficiency of phase two depends on how many points of $\mathcal{W}$ lay inside the triangle. Unfortunately, in the m/m case there may be a huge number of such points, and hence the search procedure may be unacceptably slow. This behaviour can be intuitively explained considering the vector f_π used in (2). By looking at the recursive equations (2), the reader can easily argue that $f_\pi(u)$ may be very small for some node $u \in \pi$. This is true, in particular, if hyperarc size is large and π contains long s-t paths. In this situation, a change in the predecessor of node u would have a negligible impact on the two weights of the hyperpath.

More formally, consider a predecessor function p defining a hypertree in the acyclic hypergraph $\mathcal{H}$, and let π be the s-t hyperpath contained in the hypertree. Obtain p' from p by changing the predecessor of a node u in π. It has been shown in [23, 21] that p' defines a hypertree, containing an s-t hyperpath $\pi' \neq \pi$. Considering equations (3) (we omit the details here) it can be shown that

$$W(\pi') = W(\pi) + f_\pi(u)\big(W_{\pi'}(u) - W_\pi(u)\big).$$

Roughly speaking, for a sufficiently small $f_\pi(u)$ *any* choice of $p'(u)$ would give almost the same weight of π and π'. Thus we can expect a huge number of hyperpaths with more or less the same weights.

In order to overcome this difficulty, it is necessary to reduce the number of hyperpaths generated by the K-shortest procedure. Consider the branching operation on hyperpath π in procedure *AYen*. Given a valid ordering V for the nodes, consider the subhypergraph $\mathcal{H}^i \in \mathcal{B}(\mathcal{H})$, where we delete the predecessor of node $u = u_{i+1}$ in π. Recall that the predecessor of the nodes following u in V are fixed, thus we have $f_{\tilde{\pi}}(u) = f_\pi(u)$ for each s-t hyperpath $\tilde{\pi}$ in $\mathcal{H}^i$, according to Proposition 1. For each criterion $c \in \{1, 2\}$, let $W_c(u)$ denote the weight of node u in hyperpath π, and let $m_c(u)$ ($m_c(t)$, respectively) denote the mean of the shortest s-u (s-t, respectively) hyperpath in $\mathcal{H}^i$. Note that $m_c(u)$ and $m_c(t)$ can be easily computed by inspecting $BS(u)$; see [21] for details.

Our goal here is to detect the situations where $\mathcal{H}^i$ can be discarded from the list L of subproblems under consideration. The following simple rule defines one such situation.

Rule 1 Suppose that $m_1(t) \geq W_1(\pi)$ and $m_2(t) \geq W_2(\pi)$. Then all hyperpaths of $\mathcal{H}^i$ are dominated by π.

Rule 1 considers the current branching hyperpath π. A stronger rule can be obtained by considering the whole set Φ of nondominated points currently found in the first and second phase. Moreover, we may adopt ε-dominance, rather than pure dominance. This is summarized in the following rule:

Rule 2 Suppose that for some $W(\bar{\pi}) \in \Phi$ it is $m_1(t) \geq (1-\varepsilon)W_1(\bar{\pi})$ **and** $m_2(t) \geq (1-\varepsilon)W_2(\bar{\pi})$. Then all hyperpaths of $\mathcal{H}^i$ will be ε-dominated.

While searching a triangle defined by W^q and W^{q+1} only points inside the triangle are of interest. The following rule also considers ε-dominance:

Rule 3 If $m_1(t) \geq W_1^{q+1}(1-\varepsilon)$ **or** $m_2(t) \geq W_2^q(1-\varepsilon)$, then all hyperpaths in $\mathcal{H}^i$ correspond to points either outside the triangle or ε-dominated by either W^q or W^{q+1}.

Rules 1-3 are *ε-safe*, since they guarantee ε-dominance, that is, the set of nondominated points found by applying the rules is an ε-approximation of the true set of nondominated points in the triangle. Unfortunately, in most cases these rules do not prune enough subproblems to speed up the triangle search significantly. Therefore, we must adopt an *approximated* triangle search procedure, referred to as *ε-search*.

The goal of ε-search is not to obtain ε-dominant solutions; instead we simply want to prevent the K-shortest hyperpath procedure from getting stuck due to the huge number of almost equivalent hyperpaths. The basic idea behind ε-search is quite simple: when branching on hyperpath π, we do not want to consider hyperpaths corresponding to points that are "too close" to $W(\pi)$. One way to prevent this is by skipping subproblem $\mathcal{H}^i$ when $f_\pi(u)$ is too small. Let us denote by ε_1 a lower bound on $f_\pi(u)$. We consequently have the following very simple rule:

Rule 4 If $f_\pi(u) \leq \varepsilon_1$ then discard subproblem $\mathcal{H}^i$.

In the context of random time-dependent networks, $f_\pi(a_t)$ denotes the probability of arriving at node a at time t, according to the strategy defined by π. The s-a_t hyperpath in π defines a *substrategy* for travelling from a to the destination, leaving at time t; this substrategy has a probability $f_\pi(a_t)$ of being used. From our previous observations, we can conclude that the hypergraph model cannot discriminate between substrategies that occur with low probability. From a decision-maker point of view, Rule 4 simply means that low-probability substrategies are not examined. Remark that this approach may be quite reasonable in an *on-line* setting, where a situation such as "leave node a at time t" would be considered only when - and if - the situation occurs.

Even if $f_\pi(u) > \varepsilon_1$, we can skip subproblem $\mathcal{H}^i$ if, for both criteria, the actual improvement that can be obtained by changing the predecessor of u is small. The maximal improvement for criterion c at node u is $W_c(u) - m_c(u)$. As discussed above, this gives an improvement $(W_c(u) - m_c(u))f_\pi(u)$ at node t. If this improvement is small for both criteria we skip subproblem $\mathcal{H}^i$. In the following rule, the improved weights at t for both criteria are compared to some previously found nondominated point. Here, ε_2 denotes a lower bound on the improvement.

Rule 5 Suppose that there exists hyperpath $\bar{\pi} \in \Phi$ satisfying $W_1(\pi) - (W_1(u) - m_1(u))f_\pi(u) \geq (1-\varepsilon_2)W_1(\bar{\pi})$ **and** $W_2(\pi) - (W_2(u) - m_2(u))f_\pi(u) \geq (1-\varepsilon_2)W_2(\bar{\pi})$. Then discard subproblem $\mathcal{H}^i$.

Note that the set of points determined by ε-search ε-dominates the true set of nondominated points for some ε, but we cannot determine how good the approximation is, i.e. the value of ε. However, an upper bound on ε can be found for each triangle examined. To this aim, it suffices to compute the minimum ε needed to ε-dominate all the points in the segment joining W^q to W^{q+1}. This value is referred to as ε_a in Section 4.

Now consider the bi-SBT$_{d/d}$ and bi-SBT$_{m/d}$ cases. Unfortunately Theorem 2 does not hold here, so we cannot apply procedure *AYen*. However, there are two possible alternatives. First, we may apply procedure *LBYen*, using the lower bound function for SBT(λ) discussed earlier. In light of Property 1, we would obtain a complete method, that is, all the hyperpaths with parametric weight below the upper bound would be obtained. A second alternative consists in using an estimate of problem SBT(λ) within procedure *LBYen*. As discussed earlier, this approach provides an approximation of the required set of hyperpaths.

In fact, both alternatives will be used within ε-search, and thus return an approximation. Note that ε-safe and ε-search rules can be adapted to the distance weighting function, except for Rule 4. The reason why ε-search is needed is that triangles may contain quite a lot of points, as for the bi-SBT$_{m/m}$ case. This can be explained intuitively as follows. Consider a node v and let $p(v) = e$ in a shortest hyperpath. Assume that $u \in T(e)$ is a maximum distance node, that is, $W(v) = W(u) + w(e)$. Now suppose that we branch on a node $u' \in T(e)$, $u' \neq u$: the change in the predecessor of u does not affect $W(v)$, unless the distance of u' becomes greater than $W(u)$. In other words, we may have a lot of hyperpaths with exactly the same distance.

Finally, recall that only an approximation of the frontier can be found in the first phase. Therefore new extreme nondominated points may be found during the second phase. If this is the case the current K-shortest procedure is stopped and restarted on the triangles defined by the new point. Other choices would be possible, but computational testing shows that this choice is acceptable in terms of computation time.

4 Computational Results

In this section we report the computational experience with the two-phases method described in Section 3. The procedures have been implemented in C++ and tested on a 1 GHz PIII computer with 1GB RAM using a Linux Red Hat operating system. The programs has been compiled with the GNU C++ compiler with optimize option -O.

We also implemented a particular generator of test hypergraphs, denoted as TEGP (*Time-Expanded Generator with Peaks*). This program includes several features inspired by typical aspects of road networks (congestion effects, waiting, random perturbations etc.). Note that TEGP like other generators only models a fraction of a real network. However, it provides alternative choices that may affect the behaviour of the algorithms.

The generator considers *cyclic* time periods. In each cyclic period there are q *peak periods* (e.g. rush hours). Each peak consists of three parts; a *transient* part p_1 where the traffic increases, a *pure peak* part p_2 where the traffic stays the same and a transient part p_3 where the traffic decreases again. The time horizon consists in one or more cyclic periods; peaks are placed at the same time in each cycle.

A underlying grid graph G of *base* b and *height* h is assumed, and we search optimal routes from the bottom-right corner node (*root* r) to the upper left corner node (*destination* d). This choice is motivated by the fact that each root-destination path has at least $b + h - 2$

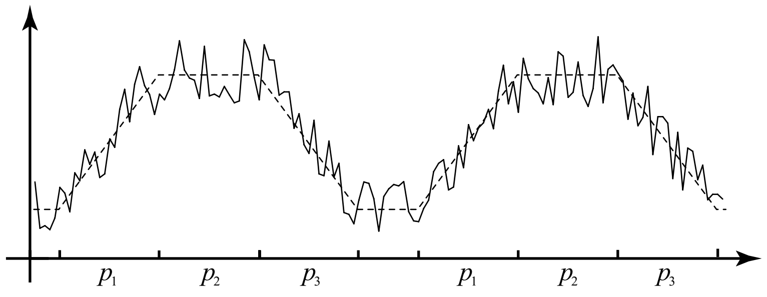

Figure 7: Peak effect and random perturbation.

arcs, and there are an exponential number of such paths in G. For each arc (i, j) in G a travel time $m_{ij} \in [lb_t, ub_t]$ and two weights $w_{ijk} \in [lb_w, ub_w]$, $k = 1, 2$, are generated. Here w_{ijk} and m_{ij} represent the weights and the average travel time out of the peaks. We refer to w_{ijk} as the *static costs* of arc (i, j). We assume that grid arcs have symmetric travel times and costs, i.e. $m_{ij} = m_{ji}$ and $w_{ijk} = w_{jik}$, $k = 1, 2$.

Let T denote the time horizon size, that is, the (finite) number of time instants in a cycle multiplied by the number of cycles. Each node v in G is now expanded to T nodes v_t. Furthermore, a dummy *origin* node s and dummy arcs $e_s(t) = (\{s\}, d_t)$ are created. Finally, each arc (i, j) in G is expanded to hyperarcs of the form $e_{ij}(t) = (\{j_h : h \in I(i, j, t)\}, i_t)$. Here i_t denote the leaving time from node i and $I(i, j, t)$ the set of possible arrival times. We assume that the travel time distribution for hyperarc $e_{ij}(t)$ is a rough approximation of the normal distribution with mean $\mu_{ij}(t)$ and standard deviation $\sigma_{ij}(t)$. The mean $\mu_{ij}(t)$ follows a pattern like the dotted line in Figure 7: at the beginning of a peak it increases from m_{ij} to $m_{ij}(1 + \eta)$, where η denote the *peak increase parameter*, then stays the same during the pure peak period, and then decrease to m_{ij} again. The same is the case for the standard deviation, which is defined by $\sigma_{ij}(t) = \rho\mu_{ij}(t)$ where ρ is the standard deviation mean ratio. Note that this setting gives higher mean travel time and higher dispersion in peaks. Waiting arcs $(\{i_{t+1}\}, i_t)$, $1 \leq t \leq T - 1$ for each node in G except the root and the destination can also be generated. We consider two *wait options*: no waiting allowed, i.e. no waiting arcs, and zero cost waiting arcs.

Before applying the two phase method all the nodes and hyperarcs that do not belong to s-t hyperpaths are deleted in a preprocessing step. This results in a hypergraph like the one in Figure 3, where some of the nodes v_t do not exist. Note that the parameters m_{ij}, η and ρ impact on the topological structure of the expanded hypergraph: larger values of these parameters yield larger hyperarcs, and yield smaller hypergraph after preprocessing.

The generation of the hyperarc weights takes into account three components: the static costs, the peak effect, and a random perturbation. The peak effect for the weights is similar to the one for travel time. For each hyperarc $e_{ij}(t)$ we define the costs $w_{ijk}(t) = w_{ijk}(1 + \eta)$, $k = 1, 2$.

The *random perturbation* introduces small variations in the hyperarc weights, due to other factors not intercepted by the peak implementation, e.g. special information about the cost at exactly that leaving time. For each hyperarc $e_{ij}(t)$ we generate a perturbation $\xi \in [-range_\xi, range_\xi]$, where $range_\xi$ is small percentage. Then, the weight $w_k(e_{ij}(t))$ of hyperarc $e_{ij}(t)$ becomes $w_{ijk}(t)(1 + \xi)$. Note that $w_k(e_{ij}(t))$ follows a pattern like the solid line

Class	1	2	3	4
Nodes	3342	2817	2314	1862
Arcs	102	90	81	76
HArcs	10976	9262	7612	6132
η (pct)	0	50	100	150

Table 2: Hypergraph classes for preliminary tests (grid size 5×8).

shown in Figure 7.

Many different *weight options* are possible with TEGP; Here, we report on three of them, denoted as *c/c - neg cor*, *c/c - no cor* and *t/c*.

In the first option, both weights represent a cost, and the two costs are assumed to be negatively correlated. This is a typical situation in haz-mat transportation, where travel cost and risk/exposure are conflicting. In this case, the static costs w_{ijk}, $k = 1, 2$ are generated so that if one belongs to the first half of the interval $[lb_w, ub_w]$ then the other belongs to the second half. Note that the arcs in $FS(s)$ are assigned zero weights.

The second option is similar to *c/c - neg cor*; however, here no correlation is assumed (*c/c - no cor*), i.e. both weights w_{ijk} are generated randomly in $[lb_w, ub_w]$.

Finally, the time/cost (*t/c*) case can be considered. Here the first criterion corresponds to time and the second to cost. Recall that in this case the first weight on each hyperarc is zero and on the dummy arcs $e_s(t)$ the first weight is t. The second weight behaves like in the *c/c* cases.

Note that weight options and *range*$_\xi$ do not affect the topological structure of the hypergraph. In the following a *class* of hypergraphs defines a set of hypergraphs with the same topological structure. Several combinations of classes, weight options and wait options will be reported. Our main focus will be on the *m/m* case.

Finally, remark the number of ways the options and input parameters of TEGP can be combined are tremendously high and a large number of hypergraphs was generated with TEGP and tested. Only a small fraction of the tests are reported here, since most of them lead to similar results.

4.1 The mean/mean case

The tests for this case can be split into two groups. First, some preliminary tests are carried out to point out the impact of some features of TEGP, and to justify the relevant parameter setting. Second, tests are carried out on different weight and waiting options. In all the tests the procedures use both *ε-safe* and *ε-search* rules, which are applied in the order the rules are mentioned in Section 3. All the problems generated have been preliminarily tested using safe rules only. Unfortunately, as pointed out in Section 3, this led to unacceptable CPU times.

The preliminary tests were carried out to examine the effect of peak increase changes and changes in the range of the random perturbation separately. Four hypergraph classes was used; the number of nodes, arcs, hyperarcs and the peak increase percentage η after preprocessing are reported in Table 2. For all classes an underlying grid size of 5×8 was used. The time horizon contains one cycle of 144 time instants, i.e. 12 hours divided in 5 minutes intervals. Each cycle has two peaks with a total length of 5 hours (each period p_1-p_3 last 1 hour and 40 minutes) and the first peak starts after half an hour ($t = 6$). The interval of

			ndom		ε_a		CPU	
Class	Φ	Gaps	Ave	Max	Ave	Max	Ave	Max
	t/c							
1	4	3	19	59	1.80	4.12	0.30	0.86
2	55	3	8	26	1.78	3.75	0.20	0.75
3	64	4	6	15	1.60	3.84	0.17	0.30
4	70	3	11	31	1.55	4.59	0.19	0.41
	c/c (neg cor)							
1	7	6	135	690	1.55	10.46	11.42	122.08
2	140	14	20	306	1.23	10.59	4.13	168.89
3	138	9	78	589	1.44	4.84	17.49	189.48
4	154	8	70	308	1.88	15.42	14.42	182.26

Table 3: Preliminary tests: change peak increase η.

possible off-peak mean travel times is $[lb_t, ub_t] = [4, 8]$, i.e. a mean travel time between 20 and 40 minutes. Furthermore, an off-peak cost interval $[lb_c, ub_c] = [1, 1000]$ is used. The deviation mean ratio is set to $\rho = 0.25$ in all classes. Classes differ in the peak increase percentage η. In class 1 η is set to zero, in class 2 it is set to 0.5, in class 3 it is set to 1 and in class four is set to 1.5. Finally no waiting arcs are allowed. Preliminary tests were run with ε-safe and ε-search parameters $\varepsilon = \varepsilon_1 = \varepsilon_2 = 0.01$.

First, the effect of a peak increase without a random perturbation is considered ($range_\xi = 0$). The four classes are considered separately. For each class two weight options were considered, namely *t/c* and *c/c (neg cor)*, and five hypergraphs were generated (with different seeds) for each weight option. In Table 3 we report the average frontier size (Φ) and number of gaps. Moreover, for each triangle searched by the second phase, we recorded the number of new nondominated points found (ndom), ε_a (reported in percent, see Section 3) and the CPU time in seconds. The average and maximum values (over the five hypergraphs) are reported in the Table.

Table 3 shows that the frontier size grows when the peak increase grows. On the other hand, a larger frontier tends to define smaller triangles, and this gives less nondominated points. In conclusion, it is relevant to model the peak effect in TEGP. Note also that option *c/c (neg cor)* is much harder than *t/c*, as shown by the values of ε_a.

Next we test the effect of changing the random perturbation without a peak effect. That is, we only consider class 1 where $\eta = 0$ and then change $range_\xi$. For each range and weight option five hypergraphs were generated and tested with the same rule parameters as before.

Results are shown in Table 4 where five possible range values (reported in percent) are considered. Like for peak increase, increasing the range make the number of extreme nondominated points higher. Consider the ε_a columns; here average and maximum ε_a fall when the range increases indicating that the triangles searched becomes smaller and nondominated points are found closer to the frontier. This is also reflected in the CPU columns where the CPU time falls. We can conclude that a larger random perturbation gives easier problems. Note that the same does not hold in the peak increase tests, hence it seems that the effect of the random perturbation is more relevant.

The preliminary tests show that both parameters η and $range_\xi$ must be chosen with

Class	$range_\xi$	Φ	Gaps	ndom Ave	ndom Max	ε_a Ave	ε_a Max	CPU Ave	CPU Max
		t/c							
1	0	4	3	19	59	1.80	4.12	0.31	0.85
1	5	36	3	5	9	1.78	4.06	0.18	0.70
1	10	57	3	5	9	1.78	4.01	0.21	0.54
1	20	85	3	5	21	1.82	3.90	0.25	0.79
1	50	157	3	10	37	1.59	3.68	0.62	3.32
		c/c (neg cor)							
1	0	7	6	135	690	1.55	10.46	11.07	113.71
1	5	93	13	20	364	1.23	10.27	2.56	85.99
1	10	118	13	15	230	1.21	9.90	2.08	54.80
1	20	171	11	14	78	1.21	8.51	1.25	31.80
1	50	360	11	12	104	1.10	2.73	1.02	24.48

Table 4: Preliminary tests: change random perturbation.

Class	1	2	3	4
Gridsize	5×8	5×8	10×10	10×10
Nodes	2254	2263	14877	14886
Arcs	80	2262	196	14885
Harcs	7383	7405	53220	53241
Waiting	no	yes	no	yes

Table 5: Hypergraph classes for the final tests.

caution. In the final tests the peak increase is set to $\eta = 1$, while the range of the random element to $range_\xi = 0.1$. A larger $range_\xi$ might results in too easy problems.

In the second group of tests, we consider four hypergraph classes with two different grid sizes and two wait options. The grid size, number of nodes etc. after preprocessing are shown in Table 5. The input parameters are set as discussed in the preliminary tests. For grid size 10×10 travelling from the root to the destination may take more than 144 time instances and hence two cycles are used. We consider all three weight options mentioned above and for each option we generate ten hypergraphs. In all tables below, average and maximal values over the ten hypergraphs are reported.

We consider the first and second phase separately and start by looking at the results for the first phase shown in Table 6. Column "W_{opt}" reports which weight option is used: *t/c* ($W_{opt} = 1$), *c/c - no cor* ($W_{opt} =$ 2) or *c/c - neg cor* ($W_{opt} =$ 3).

Columns "Φ" and "CPU" report the results when the exact set of extreme nondominated points are found, while the next two columns report results when an ε-approximation is found with $\varepsilon = 0.01$. The number of triangles searched by the second phase are reported in column "Gaps". Finally, x_I and y_I report the relative increase from the upper/left point W^{ul} to the lower/right point W^{lr} for the first and second criteria, defined as $(W_1^{lr} - W_1^{ul})/W_1^{ul}$ and $(W_2^{ul} - W_2^{lr})/W_2^{lr}$, respectively.

First, compare the exact results against the approximated ones. Here the number of

Class	W_{opt}	Φ	CPU	Φ_ε	CPU	Gaps	x_I	y_I
		Grid size 5×8						
1	1	75	2.10	20	0.32	2	38	125
1	2	76	2.21	25	0.42	5	78	86
1	3	169	4.89	60	1.03	10	170	416
2	1	183	6.03	32	0.58	3	80	153
2	2	78	2.44	22	0.43	5	63	83
2	3	137	4.36	43	0.89	10	147	349
		Grid size 10×10						
3	1	318	56.89	36	3.57	2	74	207
3	2	457	131.46	56	4.95	4	168	212
3	3	761	136.36	105	10.01	10	375	746
4	1	437	88.13	44	4.38	3	112	261
4	2	221	46.43	36	4.12	5	103	109
4	3	319	60.37	75	8.28	15	223	448

Table 6: First phase (final tests).

extreme nondominated points is significant lower for the approximation, resulting in large savings in CPU time. This implies that the set of "large" triangles can be determined at much lower cost by an approximate phase one. Anyway, it must be remarked that the number of "large" triangles (column "Gaps") is quite limited even if compared to the size of the approximated frontier. That is, the frontier contains many points close to each other. In this situation, a decision maker may be satisfied by the options offered by phase one, which would make phase two redundant.

Next, let us consider the exact frontier solution and compare the different waiting possibilities. For weight option t/c the number of extreme nondominated points raise when waiting arcs with zero costs are used. A possible explanation is that waiting may make the mean travel time a bit higher (first criteria) but may make the mean cost lower, introducing more nondominated points. Thus the upper/left point stays the same, but the lower/right point becomes larger. This is confirmed by the values x_I and y_I, that are larger when waiting is allowed. For the c/c weight options the situation is opposite: the number of extreme points falls when waiting is allowed. Indeed, waiting can make *both costs* become lower in some cases, thus some better solutions may arise. In graphical terms, the frontier moves towards the origin of the octant and "shrinks", resulting in smaller values x_I and y_I.

Consider now the second phase. We allowed at most 10,000 hyperpaths to be generated for each searched triangle, and we recorded the number of *unfinished triangles* (column U) where the K-shortest procedure terminated before reaching the upper bound on the parametric weight. For each hypergraph we recorded the average and maximum number of nondominated points found inside the triangles (ndom), the average and maximum CPU time for searching a triangle (CPU) and the average and maximum values of ε_a and ε_b. Here ε_b denotes the minimum value such that, for any two adjacent nondominated points of the triangle, at least one of the two ε_b-dominates the other.

Two values of the ε-search parameter ε_1 were used, namely 0.1 and 0.01. The results are reported in Table 7 for $\varepsilon_1 = 0.01$ and in Table 8 for $\varepsilon_1 = 0.1$. In both tables epsilons are

			ndom		CPU		ε_a		ε_b	
Class	W_{opt}	U	Ave	Max	Ave	Max	Ave	Max	Ave	Max
		Grid size 5×8								
1	1	0	14	63	2,67	21,03	1,83	3,65	1,81	4,93
1	2	0	13	50	0,48	2,33	1,51	3,28	1,50	2,48
1	3	0	16	171	0,87	25,05	1,08	2,75	1,41	3,77
2	1	1	22	80	28,79	104,30	1,59	4,80	1,87	5,95
2	2	2	39	184	34,80	95,86	1,72	5,59	2,29	7,60
2	3	2	83	705	28,65	110,19	1,61	9,79	2,36	15,44
		Grid size 10×10								
3	1	0	9	43	2,75	13,81	1,04	1,48	1,17	1,56
3	2	0	28	232	52,79	438,43	1,22	2,45	1,59	2,54
3	3	1	75	638	76,00	514,72	1,30	6,48	1,90	12,20
4	1	2	27	173	220,11	500,77	2,22	9,73	3,27	15,30
4	2	2	76	322	203,80	488,96	2,14	14,53	3,70	27,60
4	3	4	134	1150	176,35	834,07	1,31	6,85	2,03	9,83

Table 7: Second phase ($\varepsilon_1 = 0.01$).

reported in percent and the CPU time in seconds.

Consider Table 7 first, and observe that in general we find good results for most triangles. The average value of ε_a is between 1.04 and 2.22 in percent, moreover, ε_a does not seem to be affected by the hypergraph size. However, in few triangles poor values of ε_a are found. This does not necessarily mean that a poor approximation of the true set of nondominated points is found. Recall that ε_a is an upper bound on the value needed for the approximation to ε-dominate the exact solution; this upper bound might be poor in some cases. More important, ε_a is found by comparing the approximation to the frontier. High values of ε_a may be due to the fact that the true set of nondominated points lay deep inside the triangle.

If we compare the different weight options we see that the uncorrelated cases produce fewer nondominated points than the correlated one. This is a well known behaviour in the bi-SP case, see e.g. [25]. Not surprisingly, the *t/c* option seems to find even fewer points than *c/c (no cor)*, since for *t/c*, we use a zero first weight on each hyperarc. Clearly, the CPU time grows significantly for the larger hypergraphs, however, it is relatively stable for the three weighting options. On the contrary, CPU time is affected by the introduction of waiting, which makes the solution space much wider.

Now consider Table 8, where $\varepsilon_1 = 0.1$ has been used. Here Rule 4 is stronger, thus a worse approximation may be expected; this is actually the case when waiting is not allowed, indeed the average value of ε_a grows. However, the increase in ε_a is small and good savings in CPU time can be obtained. Moreover, for the zero cost waiting $\varepsilon_1 = 0.1$ yields better approximations, and fewer unfinished gaps. Note also that the number of nondominated points inside a triangle falls, but the "space" between adjacent points, i.e. ε_b, tends to decrease. We may argue that a higher value of ε_1 make the triangle search less selective but faster, and allows to search "deeper" in the triangles. The effect of increasing ε_1 is shown in Figure 8, where we consider one difficult gap, and we plot the results obtained with $\varepsilon_1 = 0.01$, $\varepsilon_1 = 0.1$ and $\varepsilon_1 = 0.2$. Here a lot of points close to the two vertices of the triangle are found when

			ndom		CPU		ε_a		ε_b	
Class	W_{opt}	U	Ave	Max	Ave	Max	Ave	Max	Ave	Max
					Grid size 5×8					
1	1	0	5	13	0.20	0.90	2.00	4.41	2.00	4.93
1	2	0	5	27	0.16	0.65	1.82	3.30	1.94	3.46
1	3	0	6	60	0.17	1.78	1.23	2.76	1.65	3.77
2	1	0	15	85	21.80	93.21	1.49	4.65	1.64	5.41
2	2	1	35	170	32.98	94.70	1.62	5.68	2.14	7.74
2	3	1	54	591	20.14	99.67	1.35	6.13	1.91	11.29
					Grid size 10×10					
3	1	0	5	23	1.93	11.00	1.08	1.65	1.22	1.47
3	2	0	10	88	2.75	24.10	1.35	2.54	1.71	3.25
3	3	0	10	141	6.85	340.05	1.13	2.00	1.55	3.05
4	1	1	23	147	233.95	499.77	2.13	9.73	3.14	15.30
4	2	2	61	219	185.27	491.26	2.03	14.57	3.50	27.66
4	3	2	52	350	93.17	502.56	1.19	5.36	1.79	7.61

Table 8: Second phase ($\varepsilon_1 = 0.1$).

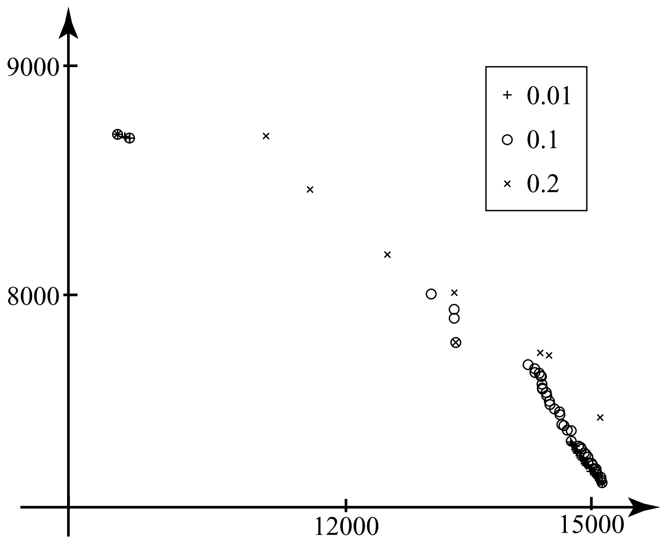

Figure 8: Changing ε_1 in a difficult triangle.

$\varepsilon_1 = 0.01$ is used, but the search stops before the whole triangle is searched resulting in large values of ε_a and ε_b. For increasing ε_1 less points are generated, but the triangle is searched deeper, and a better overall approximation is found. However, note that many points found with $\varepsilon_1 = 0.01$ dominate the points found with $\varepsilon_1 = 0.2$. Similar observations can be made for ε_2 too.

Finally, a few short remarks can be made about the "success rate" of Rules 1-5. All these rules proved to be useful in reducing the search space, except perhaps Rule 3, whose success rate was below 1%. Obviously, the success rate of the rules depend on the order in which

Class	Φ	CPU	ε_{Φ_s}
1	6	0.79	4.66
2	9	1.48	5.90
3	12	35.54	6.54
4	15	18.26	6.53

Table 9: Results frontier approximation (d/d case).

they are applied. For example, the success rate of Rule 5 which is tested after Rule 4 dropped from 23% for $\varepsilon_1 = 0.01$ to 3% for $\varepsilon_1 = 0.1$.

4.2 The distance/distance case

As pointed out in Section 3 this case is harder to solve, since we cannot solve SBT(λ) exactly. Here, we compare several approximated versions of phase two, based on different settings. For each generated hypergraph, the best solutions found with the different settings were merged into a nondominated set Φ_s, representing the best known solution of the hypergraph. Since the true frontier cannot be computed in this case, we used Φ_s as a touchstone for comparing the relative performance of the various settings.

Only one weight option, namely *c/c neg cor*, is considered here. Note that the *t/c* option is not relevant in this context. Indeed, the maximum distance with respect to the time criterion cannot be greater than the time horizon. Thus, efficient solutions can be found rather trivially by solving minimum cost hyperpath problems for different settings of the time horizon. This simpler approach can be much more effective in practice.

Let us first consider the first phase. Here an approximation is found by using both the estimate and the lower bound function. The number of frontier points (Φ) and the CPU time was recorded. Moreover, the ε needed for the approximated frontier to ε-dominate the frontier in Φ_s is reported. The results, shown in Table 9, show that the number of frontier points is much smaller compared to the m/m case and hence larger triangles have to be searched. Furthermore, only a rough approximation of Φ_s is obtained (ε_{Φ_s} between 4.66 and 6.54 percent). However, recall that in the second phase a triangle search is restarted if a new extreme nondominated point is found.

For what concerns the second phase, let us consider in detail the use of the estimate function. Clearly, this function does not rank hyperpaths exactly. This can be seen in Figure 9 where an example of the ranking of the parametric weight using the estimate function is shown. Suppose that the search is stopped as soon as the first weight over the upper limit *ub* is generated. In this case, we may miss some (possibly efficient) hyperpaths with weight below *ub*. This difficulty can be faced as follows. Split the sequence of generated hyperpaths into intervals of length l. The minimal weight w of the estimate function for each interval is then computed, and compared against *ub*. Thus, the search may stop after generating $k = l, 2l, 3l, \ldots$ hyperpaths ($k = 6l$ in the figure).

The estimate function was used with ε-safe and ε-search, with $\varepsilon_2 = 0.01$. Four different values of l were preliminary tested, namely, $l = 1, 10, 20, 50$; we only report $l = 1$ and $l = 10$ here, since greater values of l did not improve the solution significantly. We also used the lower bounding function with ε-safe and ε-search. In these cases, at most 10,000 hyperpaths were allowed for each triangle. Finally, we performed one test with the estimate function and

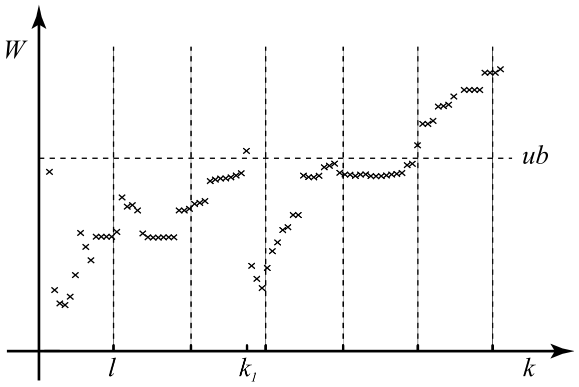

Figure 9: Ranking of hyperpaths when the ub function is used.

	ndom				CPU		ε_b		ε_{Φ_s}	
Class	Gaps	Ave	Max	New$_f$	Ave	Max	Ave	Max	Ave	Max
	Estimate function (ε-safe and ε-search. $l = 1$)									
1	6	2	8	1	0.19	0.63	5.84	22.43	2.04	10.95
2	10	4	26	1	18.20	111.05	2.73	18.66	1.10	7.92
3	12	3	12	1	3.51	21.60	3.73	25.30	1.23	8.59
4	17	9	94	1	459.99	2186.61	2.00	23.49	0.65	7.38
	Estimate function (ε-safe and ε-search. $l = 10$)									
1	6	2	8	1	0.54	1.76	5.57	22.43	1.39	7.99
2	10	4	26	1	12.20	108.09	2.68	18.66	1.04	7.92
3	12	3	12	2	6.38	27.87	3.66	25.30	1.15	6.85
4	17	9	94	2	309.08	2185.27	1.92	23.49	0.58	7.38
	lb function (ε-safe and ε-search)									
1	6	2	7	1	0.27	1.06	5.06	23.10	3.07	16.67
2	10	3	20	1	12.62	165.83	2.48	12.21	1.26	7.74
3	12	3	12	5	89.13	345.44	3.73	25.30	1.10	15.09
4	17	6	47	1	591.01	3987.53	2.46	24.33	1.01	11.91
	Estimate function (ε-safe $l = 50$)									
1	6	2	7	2	2.84	10.74	5.72	36.34	2.09	18.26
2	10	1	3	10	2.00	6.63	5.33	29.26	2.08	12.67
3	12	3	12	5	89.13	345.44	3.61	25.30	1.10	15.09
4	16	1	6	16	17.48	80.40	4.62	31.24	2.24	14.08

Table 10: Results second phase (d/d case).

ε-safe rules only; in this case, only 100 hyperpaths were allowed.

The results for the four settings are shown in Table 10, where "Gaps" denote the number of triangles defined by the frontier when the second phase stops, "New$_f$" the average number of new extreme points found, and "ε_{Φ_s}" the average and maximum value needed for the points to dominate each triangle of Φ_s.

First of all, observe that the quality of the approximations is relatively stable in the four

			ndom				CPU		ε_b	
Class	Φ	CPU	Gaps	Ave	Max	New$_f$	Ave	Max	Ave	Max
1	6	0.35	6	1	8	2	0.27	1.15	5.65	32.43
2	11	0.65	12	3	23	2	6.13	126.25	2.63	29.05
3	12	12.86	12	2	12	1	7.84	48.35	4.76	38.24
4	17	9.05	17	9	91	2	498.81	4592.02	1.82	24.80

Table 11: Results for the m/d case.

settings. As expected, the choice $l = 10$ gives better results than $l = 1$, without increasing the CPU times. Moreover, the estimate performs better than the lower bound; this may suggest that the lower bounding function is not tight enough, hence a lot of hyperpaths with parametric weight above ub are generated.

The most interesting results are obtained when the estimate function is used with ε-safe rules. In this case, the search procedure is expected to be more accurate, since less hyperpaths are skipped with respect to ε-search. This is actually the case, since more new frontier points are found using ε-safe rules. However, the procedure quickly begins to stall, i.e. the parametric weight does not increase. For this reason, we limited the number of generated hyperpaths to 100, which explains the low CPU times. Roughly speaking, we may argue that ε-safe rules search more accurately close to the frontier, but don't go deep into the triangles. Interestingly, the resulting approximation is comparable to the one obtained by ε-search. From the above observations, we may conclude that different settings return different set of points, and that the union of these sets (i.e. Φ_s) may be a good approximation of the true set of efficient points.

4.3 The mean/distance case

Finally we consider the m/d case. Here only weight option *c/c neg cor* is considered and tested using the estimate function with ε-safe, ε-search and $l = 10$ ($\varepsilon = \varepsilon_2 = 0.01$ and $\varepsilon_1 = 0.1$). The results, reported in Table 11, show that the number of frontier points and unsupported nondominated points is more or less the same as for the d/d case. However, the number of new frontier points found in phase two increases slightly, while the value ε_b tends to increase.

5 Conclusion

In this paper we considered bicriterion routing problems in random time dependent networks, in particular, we introduced and investigated the generation of efficient (Pareto-optimal) strategies. Bicriterion problems in dynamic networks have already been considered in the literature, however, the generation of efficient strategies has not yet been proposed. The choice of this approach was motivated by two simple observations. First, finding optimal strategies in dynamic networks is easy, while finding paths is difficult. Second, strategies are more general and thus more effective than paths.

We reduced the generation of efficient strategies to a bicriterion shortest hyperpath problem in acyclic directed hypergraphs. This problem was solved by a suitable extension of the classical two-phases approach for bicriterion shortest paths. Even though several hyperpath models have been proposed in the literature, the bi-SBT problem has not yet been considered. The main contributions of this paper can be summarized as follows.

First consider the problem, already treated in the literature, of minimizing expected costs or travel times, i.e. the m/m case. Here we showed that the parametric weight problem is easy, since the results from graph theory can be extended to hypergraphs. Therefore, finding the frontier in the first phase is easy. By contrast, no easy path problem exist in random time-dependent networks. From a decision makers' point of view finding the frontier might be enough in some cases, for example, in a transportation application (e.g. haz-mat) where routing is chosen on-line, based on updated information. Furthermore, by using a two phase method, the frontier can be used to guide the search for the second phase if interactive/on-line methods are used.

For the second phase approximation methods were needed to let the method converge, since the criteria space is so dense that not all nondominated points inside the triangles can be found. As pointed out this is a result of the fact that the model does not discriminate between sub-strategies with low probability. However, by introducing ε-safe and ε-search rules to prune the search, this situation can be dealt with. Computational tests showed that the rules are successful and good approximations can be found. Note that approximation methods was also necessary to find bicriterion shortest paths in random time-dependent networks (see e.g [16]). However, finding strategies instead of paths provide us with a more flexible/general model, since e.g. different weighting functions can be used. This flexibility comes at the expense of computational complexity if the d/d and m/d cases are considered.

For the d/d and m/d case we showed that the parametric weight problem is harder to solve. However, we have a lower bound and a heuristic estimate function, that can be used in the two-phases method. Theoretically, the lower bound can be used to obtain an exact method. In practice however the estimate function finds a better approximation.

Finally, note that, as in many multicriterion analysis frameworks, our goal is not the one of "solving" the problem, but rather the one of providing a reasonable set of alternatives. To this aim, our approach seems to be more general and flexible than the enumeration of paths.

Clearly, the two-phases method developed in this paper may be improved, in particular better lower bound and estimate functions may be found. Also, alternate techniques for finding good approximations of the efficient set could be investigated. Moreover, we believe that the results in this paper provide a stimulating starting point for further research. First, note that our model extends quite easily to the bicriterion shortest path problem in dynamic networks, and it would be possible to adapt our algorithms to the enumeration of path strategies. Then, a direct comparison of the strategy and path approaches might be tried. Next, *parametric shortest hyperpath* methods [20] might be used to find the frontier in the d/d or m/d cases. Finally, more effective branching strategies for the d/d case is under consideration, which may speed up the search in phase two. These topics will be the subject of forthcoming papers.

References

[1] G. Ausiello, P.G. Franciosa, and D. Frigioni. Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach. In *Lecture Notes in Computer Science 2202*, pages 312–328. Springer Verlag, 2001.

[2] J.R. Birge and F. Louveaux. *Introduction to stochastic programming.* Springer Series in Operations Research. Springer-Verlag, New York, 1997.

[3] J. Brambaugh-Smith and D. Shier. An empirical investigation af some bicriterion shortest path algorithms. *European Journal of Operational Research*, 43:216–224, 1989.

[4] J.C.N. Climaco and E.Q.V. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11:399–404, 1982.

[5] J. Cohen. *Multiobjective Programming and Planning.* Academic Press, New York, 1978.

[6] J.M. Coutinho-Rodrigues, J.C.N. Climaco, and J.R. Current. An interactive bi-objective shortest path approach: Searching for unsupported nondominated solutions. *Computers and Operations Research*, 26:789–798, 1999.

[7] J.R. Current, C.S. ReVelle, and J.L. Cohen. An interactive approach to identify the best compromise solution for two objective shortest path problems. *Computers and Operations Research*, 17(2):187–198, 1990.

[8] M. Ehrgott. *Multicriteria optimization*, volume 491 of *Lecture Notes in Economics and Mathematical Systems.* Springer-Verlag, Berlin, 2000.

[9] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22(4):425–460, 2000.

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

[11] M. Garey and D. Johnson. *Computers and Intractability. A Guide of the Theory of NP-Completeness.* W.H.Freeman, 1979.

[12] R.W. Hall. The fastest path through a network with random time-dependent travel times. *Transportation Science*, 20(3):182–188, 1986.

[13] R.G. Jeroslow, K. Martin, R.L. Rardin, and J. Wang. Gainfree Leontief substitution flow problems. *Mathematical Programming*, 57:375–414, 1992.

[14] R. De Leone and D. Pretolani. Auction algorithms for shortest hyperpath problems. *Siam J. Optim.*, 11(1):149–159, 2000.

[15] E.D. Miller-Hooks. Adaptive least-expected time paths in stochastic, time-varying transportation and data networks. *Networks*, 37(1):35–52, 2000.

[16] E.D. Miller-Hooks and H.S. Mahmassani. On the generation af nondominated paths in stochastic, time-varying networks. Technical report, University of Texas at Austin, 1998.

[17] E.D. Miller-Hooks and H.S. Mahmassani. Optimal routing of hazardous materials in stochastic, time-varying transportation networks. *Transportation Research Record*, 1645:143–151, 1998.

[18] J. Mote, I. Murthy, and D.L. Olson. A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research*, 53:81–92, 1991.

[19] S. Nguyen and D. Pretolani. Shortest hyperpath problems on oriented hypergraphs. Technical report, Centre de recherche sur les transports, September 1994.

[20] Lars Relund Nielsen. A bicriterion and parametric analysis of the shortest hyperpath problem. Progress report, University of Aarhus – Department of Operations Research, 2001. Available at http://home.imf.au.dk/relund/.

[21] L.R. Nielsen, K.A. Andersen, and D. Pretolani. Finding the K shortest hyperpaths: algorithms and applications. Technical Report WP-2002-2, Department of operations research, University of Aarhus, September 2002. Submitted, Available at http://home.imf.au.dk/relund/.

[22] G.H. Polychronopoulos and J.N. Tsitsiklis. Stochastic shortest path problems with recourse. *Networks*, 27:133–143, 1996.

[23] D. Pretolani. A directed hypergraph model for random time dependent shortest paths. *European Journal of Operational Research*, 123:315–324, June 2000.

[24] A.J.V. Skriver. A classification of bicriteria shortest path (BSP) algorithms. *Asia-Pacific Journal of Operational Research*, 17:199–212, September 2000.

[25] A.J.V. Skriver and K.A. Andersen. A label correcting approach for solving bicriterion shortest path problems. *Computers and Operations Research*, 27:507–524, sep 2000.

[26] R.E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application.* Wiley Interscience. Wiley, 1986.

[27] A. Warburton. Approximation of pareto optima in multiple-objective, shortest-path problems. *Operations Research*, 35(1):70–79, 1987.