WORKING PAPER L-2004-05

WP

Lars Relund Nielsen, Daniele Pretolani & Kim Allan Andersen

K shortest paths in stochastic time-dependent networks

K shortest paths in stochastic time-dependent networks

Lars Relund Nielsen[*]
Biometry Research Unit
Danish Institute of Agricultural Sciences
Research Centre Foulum, P.O. Box 50
DK-8830 Tjele
Denmark

Kim Allan Andersen
Department of Accounting, Finance and Logistics
Aarhus School of Business
Fuglesangs Allé 4
DK-8210 Aarhus V
Denmark

Daniele Pretolani
Dipartimento di Matematica e Informatica
Università di Camerino
Via Madonna delle Carceag
I-62032 Camerino (MC)
Italy

November 18, 2004

Abstract

A substantial amount of research has been devoted to the shortest path problem in networks where travel times are *stochastic* or (deterministic and) time-dependent. More recently, a growing interest has been attracted by networks that are both stochastic and time-dependent. In these networks, the best route choice is not necessarily a path, but rather a *time-adaptive strategy* that assigns successors to nodes as a function of time. In some particular cases, the shortest origin-destination path must nevertheless be chosen *a priori*, since time-adaptive choices are not allowed. Unfortunately, finding the a priori shortest path is $\mathcal{NP}$-hard, while the best time-adaptive strategy can be found in polynomial time.

In this paper, we propose a solution method for the a priori shortest path problem, and we show that it can be easily adapted to the ranking of the first K shortest paths. Moreover, we present a computational comparison of time-adaptive and a priori route choices, pointing out the effect of travel time and cost distributions. The reported results show that, under realistic distributions, our solution methods are effective.

Keywords: shortest paths; K shortest paths; stochastic time-dependent networks; routing; directed hypergraphs.

1 Introduction

Travel time between an origin and a destination is often the primary objective when routing data, commodities, vehicles etc. in a network. The problem of finding a minimal travel time path, if travel time is deterministic and time-independent, has been the subject of extensive

[*]Corresponding author, e-mail: lars@relund.dk

research for many years. However, a transportation network in which travel times between locations are deterministic and time-independent is often unrealistic.

We describe a network as *time-dependent* if the travel times on the arcs are functions of departure time, and as *stochastic* if the travel time is represented by probability distributions rather than simple scalars. The terms *time-varying* and *random* are often used with similar meaning.

Several papers address stochastic shortest path problems in stochastic time-independent networks; different optimality criteria have been considered. The problem of finding a path which maximizes the *expected utility* was first considered by Loui [10] and generated wide interest, see e.g. Murthy and Sarkar [16] and Eiger, Mirchandani, and Soroush [5]. Stochastic problems with *recourse*, where a traveller can switch to a different path upon arrival at an intermediate node, have been studied, see e.g. Psaraftis and Tsitsiklis [24] and Provan [23]. See Birge and Louveaux [2] for a general overview of stochastic recourse models.

Shortest path problems on non-stochastic time-dependent networks have been studied as early as 1966 by Cooke and Halsey [4]. Since then, various models and problems have been addressed, taking into account different aspects such as discrete vs. continuous representation of time, travel costs, waiting, etc. In particular, *discrete* time-dependent networks, where travel times are integers, have been the subject of extensive research; quite efficient algorithms have been designed, see e.g. Chabini [3].

It is evident that both the stochastic and the time-dependent aspects should be taken into account in a realistic transportation network model. Therefore, stochastic time-dependent networks (*STD networks*) provide a more powerful modelling tool. These networks were first addressed by Hall [7] who considered the problem of minimizing the expected origin-destination travel time when leaving the origin at a specific time. He pointed out two different ways of formulating the problem.

If a route must be specified before travel begins, and no deviations are permitted, a *loopless* path must be selected, i.e., each node except the destination must be assigned a unique *successor* arc. This is referred to as *a priori route choice* and corresponds to the case in which the traveller does not have access to, or cannot make decisions based on, information made available during the travel.

Routes with lower expected travel time may be obtained by allowing the traveller to make decisions based on the actual arrival times at intermediate nodes. The best time-adaptive route is not necessarily a path but rather a *time-adaptive strategy* that assigns optimal successors to a node as a function of leaving time. This is referred to as *time-adaptive route choice* and can be considered as a multistage recourse problem (Birge and Louveaux [2]) where decisions are based on realizations of arc travel times.

Note that a path may also be considered a strategy, namely a strategy assigning to each node the same successor for all possible leaving times. In the following we use the term *strategy* for routes under both time-adaptive and a priori route choices, moreover, we use the term *path-strategy* to denote a strategy defining a loopless path, that is, a feasible route for the a priori case.

Hall [7] pointed out some intrinsic complications of a priori route choice, arising from the combination of both stochastic and time-dependent features. He presented the simple example reproduced in Figure 1, where route A has deterministic travel time, route B has stochastic travel time and route C has time-dependent travel time. Suppose a traveller leaves node 1 at 2 o'clock. Although route A has the lowest expected travel time to node 2, path BC has a lower expected travel time overall than path AC. In this example, a subpath of

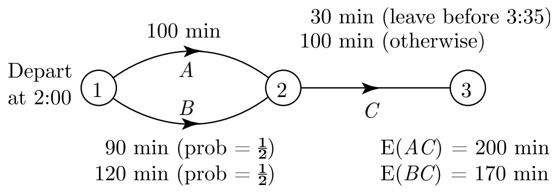

30 min (leave before 3:35)
100 min (otherwise)
100 min
Depart at 2:00
A
B
C
1
2
3
90 min (prob = $\frac{1}{2}$)
120 min (prob = $\frac{1}{2}$)
E(AC) = 200 min
E(BC) = 170 min

Figure 1: Hall's example.

the optimal path is not optimal. This violates the well-known *principle of optimality* that holds for the deterministic shortest path problem and also for stochastic time-independent networks, as pointed out by Loui [10].

Based on the above observations, Hall concluded that standard shortest path methods were inadequate for the a priori route choice problem and proposed an approach combining branch and bound and K shortest paths techniques. The worst-case complexity of the algorithm is non-polynomial, but no complexity results have been provided in Hall [7]. Later, Pretolani [22] showed that the problem of finding a minimal expected travel time path under a priori route choice is $\mathcal{NP}$-hard, also for discrete, deterministic time-dependent networks.

Hall [7] also proposed a solution approach for the time-adaptive case. His method deals with general distribution functions and was devised for networks with limited size. Discrete STD networks were considered by Pretolani [22] who showed that the best time-adaptive strategy can be found in linear time in the size of the network description. This result applies to several criteria other than expected travel time. In particular, Pretolani [22] presented a *directed hypergraph* model for discrete STD networks and showed that the *best strategy* (the strategy minimizing the criterion) can be found by solving a *shortest hyperpath* problem.

Directed hypergraphs are an extension of directed graphs, successfully used in many different research areas; see Gallo, Longo, Pallottino, and Nguyen [6] for an introduction and Ausiello, Franciosa, and Frigioni [1] for a recent overview. A different algorithm for finding minimum expected time strategies was proposed by Miller-Hooks [11] by adapting a label-correcting procedure previously used to obtain a lower bound on the a priori case (Miller-Hooks and Mahmassani [14]).

As shown above, time-adaptive route choice is more flexible; moreover, finding the shortest path (i.e. best path-strategy) is harder than finding the best strategy. In some practical situations, it may nevertheless be necessary to adopt a priori route choice because time-adaptive routing decisions are not possible. This may be the case, for example, in the transportation of highly sensitive substances, for which the travelled path must be preapproved. A priori route choice may be more suitable also in those situations where the traveller (e.g. a daily commuter) is not willing to react to new information during the travel. A priori route choice has been addressed in Miller-Hooks and Mahmassani [12], where paths (possibly containing loops) with *least possible* travel time were searched. Miller-Hooks and Mahmassani [14] also considered the general problem of finding the minimum expected travel time path to a given destination from each node and for each possible leaving time. Bicriterion problems were addressed too, in particular, with applications to hazardous material transportation (Miller-Hooks and Mahmassani [13]).

In many practical situations, finding the best route in a transportation network is not sufficient, and we are interested in ranking the best K solutions, e.g. in order to provide alternative optimal or nearly-optimal solutions, or to search for a sub-optimal solution satisfying some additional constraint. To our knowledge, only limited attention has been paid to the problem of finding the K best strategies in an STD network. This problem is an extension of the classical K *shortest paths* problem, i.e. ranking loopless paths between two nodes in non-decreasing order of length. This problem can be solved by the well-known algorithm proposed by Yen [25], and later discussed by Lawler [9] in a more general framework. Several heuristic improvements to Yen's algorithm have been proposed, see for instance Hershberger, Maxel, and Suri [8] for a recent proposal.

Under time-adaptive route choice, the best strategy corresponds to a shortest hyperpath (Pretolani [22]), thus finding the K best strategies corresponds to finding the K *shortest hyperpaths*. This problem was addressed in Nielsen, Andersen, and Pretolani [19], where efficient procedures were devised by extending Yen's algorithm for loopless paths. The computational complexity for *acyclic hypergraphs* has been improved in Nielsen, Pretolani, and Andersen [21] using reoptimization techniques. These procedures for ranking strategies have been used by Nielsen, Andersen, and Pretolani [18] to solve bicriterion problems under time-adaptive route choice.

In this paper we concentrate on a priori route choice. In particular, we consider the problem of finding the *a priori shortest path* (i.e., the best path-strategy) between two given nodes, and for a specific leaving time. We propose a solution algorithm and show that it can be easily adapted to the ranking of the K best path-strategies. Two variants of the algorithm are devised, and the extension to STD networks where *waiting* is allowed is discussed. The effectiveness and robustness of our algorithms are computationally evaluated on different classes of benchmarks. Finally, we report computational results for the K best time-adaptive strategy problem, with the aim of analyzing and comparing the strategies obtained under both a priori and time-adaptive route choices.

The paper is organized as follows. The definition of discrete STD networks and the hypergraph model are given in Section 2. In Section 3 we provide our algorithms for the a priori shortest and K shortest path problems. In Section 4 computational results for finding the K best strategies under both time-adaptive and a priori route choices are given. Finally, we summarize original contributions and topics for further research in Section 5.

2 Stochastic time-dependent networks

In this section we describe discrete stochastic time-dependent networks, and we introduce the terminology to be used throughout the paper. We also present a class of directed hypergraphs used to model STD networks. The hypergraph model is explained by means of an example, which is later used to illustrate some technical passages.

2.1 Basic definitions

We consider discrete STD networks where departure times are integer, and travel times are independent integer-valued discrete random variables with time-dependent density functions. We assume that departure and arrival times belong to a finite *time horizon*, i.e. a set $H = \{0, 1, ..., t_{\max}\}$ of integer values. In practice, we assume that the relevant time period is discretized into time intervals of length δ, i.e., the time horizon H corresponds to the set of time instances $0, \delta, 2\delta, ..., t_{\max}\delta$.

Let $G = (N, A)$ be a directed graph with node set N and arc set A. We will refer to G as the *topological network*. As usual, $FS(u) = \{(u, v) \in A\}$ denotes the forward star of node u. Let $o \in N$ and $d \in N$ denote the *origin* and *destination* node in G, respectively. For each arc $(u, v) \in A$ let $L(u, v) \subset H$ be the set of possible leaving times from node u along arc (u, v). Moreover, let $L(u)$, $u \neq d$ denote the set of possible leaving times from node u, i.e.,

$$L(u) = \bigcup_{(u,v) \in FS(u)} L(u, v)$$

and let $L(d)$ denote the set of possible arrival times at node d. For each arc $(u, v) \in A$ and $t \in L(u, v)$, let $X(u, v, t)$ denote the arrival time at node v when leaving node u at time t along arc (u, v). The arrival time $X(u, v, t)$ is a discrete random variable with density

$$\Pr(X(u, v, t) = t_i) = \theta_{uvt}(t_i), \quad t_i \in I(u, v, t)$$

where

$$I(u, v, t) = \left\{t_1, ..., t_{\kappa(u,v,t)}\right\}$$

denotes the set of $\kappa(u, v, t)$ possible arrival times at node v when leaving node u at time t along arc (u, v). That is, for each $t_i \in I(u, v, t)$ the probability of arriving at node v at time t_i when leaving node u at time t is $\theta_{uvt}(t_i)$. Denote by

$$\kappa = \sum_{(u,v) \in A,\ t \in L(u,v)} \kappa(u, v, t)$$

the total number of possible travel times. The value κ can be considered as the size of the STD network.

We assume that travel times are positive and that the traveller cannot get stuck at an intermediate node v. Hence, if it is possible to arrive at node v at time t_i, then it is also possible to leave node v at time t_i. Note that a traveller cannot wait at intermediate nodes; the case where waiting is allowed will be considered in Section 3.

Definition 1 A *strategy* is a function S with domain

$$Dm(S) \subseteq \{(u, t) : u \in N \setminus \{d\},\ t \in L(u)\}$$

assigning to each pair $(u, t) \in Dm(S)$ a successor arc $(u, v) \in FS(u)$. Furthermore, strategy S must satisfy the following conditions:

1. If $(u, t) \in Dm(S)$ and $S(u, t) = (u, v)$ then $t \in L(u, v)$.
2. If $(u, t) \in Dm(S)$ and $S(u, t) = (u, v)$, $v \neq d$, then $(v, t') \in Dm(S), \forall t' \in I(u, v, t)$.

Strategy S provides routing choices for travelling from all nodes and leaving times in the domain $Dm(S)$ towards the destination d. Therefore, a traveller leaving node u at time t travels along arc $S(u, t)$. Note that a strategy S must provide a routing choice for each possible arrival time at an intermediate node, as required by Condition 2 above. Moreover, given S, we denote by:

$$Dd(S) = \left\{(d, t) :\ \exists (u, t') \in Dm(S),\ S(u, t') = (u, d),\ t \in I(u, d, t')\right\}$$

the set of pairs (d,t) where t is a possible arrival time at d when following strategy S. Note that Definition 1 extends the definition given in [22] where a strategy had domain

$$\mathcal{D} = \{(u,t) : u \in N \setminus \{d\}\,,\ t \in L(u)\}\,. \tag{1}$$

Our assumption that a traveller cannot get stuck at intermediate nodes implies that any traveller who leaves node i at time t arrives at the destination d within time $t_{\max}$. This can be formally stated requiring that a pair (i,t) belongs to $\mathcal{D}$ if and only if it belongs to the domain of some strategy.

In this paper we consider strategies providing route choices when leaving a specific node i at a specific time t towards the destination d. This leads to the definition of an (i,t) strategy.

Definition 2 An (i,t) strategy is a *minimal* strategy S such that $(i,t) \in Dm(S)$. Here, minimality means that there does not exist another (i,t) strategy with domain strictly contained in $Dm(S)$.

In particular, we are interested in $(o,0)$ strategies, defining the route travelled when leaving the origin node o at time zero. In the following, unless otherwise specified, a strategy S refers to a $(o,0)$ strategy.

A strategy is a *path-strategy* if the successor arcs do not depend on time; in other words, a path-strategy must satisfy

$$S(u,t) = S\left(u,t'\right), \quad \forall\,(u,t)\,,\left(u,t'\right) \in Dm\,(S)\,. \tag{2}$$

Clearly, a path-strategy S defines a unique, loopless o-d path in G. Indeed, S defines a unique successor arc (u,v) for each u in G such that $(u,t) \in Dm(S)$ for some t. Assume that for each arc (o,v) in G it is possible to leave o at time zero travelling along (o,v), i.e., $0 \in L(o,v)$. With this assumption, each loopless o-d path

$$P = (o = u_1, u_2, \ldots, u_l, u_{l+1} = d)$$

in G defines exactly one path-strategy S. In particular, we have

$$Dm(S) = \bigcup_{i=1}^{l} D^{(i)}, \qquad Dd(S) = D^{(l+1)} \tag{3}$$

where $D^{(1)} = \{(o,0)\}$ and, for $1 < i \le l+1$:

$$D^{(i)} = \big\{(u_i,t) :\ t \in I(u_{i-1},u_i,t'),\ (u_{i-1},t') \in D^{(i-1)}\big\}. \tag{4}$$

Clearly, we have $S(u_i,t) = (u_i,u_{i+1})$ for each $(u_i,t) \in D^{(i)}$, $1 \le i \le l$. Based on the above observations, we can state the following theorem.

Theorem 1 *There is a one-to-one correspondence between o-d paths in G and path-strategies in the STD network.*

2.2 Optimality criteria

Several definitions of the *weight* of a strategy can be given, in fact, several optimality criteria have been considered in the literature. The most frequently used criterion for finding the best strategy is the minimization of the expected travel time, introduced by Hall [7]. In this case, the weight of a strategy corresponds to the expected arrival time at the destination when leaving the origin at time zero. If travel costs are considered, strategies can be ranked according to their expected cost. Moreover, instead of considering expectations, worst cases may be of concern; i.e., our criterion becomes the minimization of *maximum possible* travel time or cost. Other criteria, such as the *minimum possible* travel time, have been considered in the literature [12].

The results reported in this paper apply to each of the criteria mentioned above. In our computational experience we shall concentrate on expected costs. Costs can be introduced in our STD model by letting $c\,(u,v,t)$, $t \in L\,(u,v)$ denote the travel cost of leaving node u at time t along arc (u,v). Note that we assume that c is deterministic, although time-dependent. Moreover, let $g_d(t)$ be a penalty cost of arriving at node d at time t. The expected cost of a strategy S can be defined by means of recursive equations, associating a value to each pair (u,t) in $Dm(S)$. In particular, if $S(u,t) = (u,v)$, we have:

$$E^S(u,t) = c(u,v,t) + \sum_{t' \in I(u,v,t)} \theta_{uvt}(t') E^S(v,t')$$

where $E^S(d,t) = g_d(t)$ for each $t \in H$. Here, $E^S(u,t)$ represents the expected cost (including penalty costs) incurred when leaving node u at time t following strategy S towards d. The expected cost of S is therefore $E^S(o,0)$. The other criteria cited above can be given a formal definition using similar recursive equations, see Pretolani [22]. Recall that for all the above criteria, finding the best path-strategy is an $\mathcal{NP}$-hard problem, whereas the best strategy can be found in $O(\kappa)$ time, see Pretolani [22]. Moreover, recall that path-strategies are a subset of strategies and, therefore, the weight of the best strategy provides a (quite efficiently computable) lower bound on the weight of the best path-strategy.

2.3 A directed hypergraph model for STD networks

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 one or more nodes in the tail. 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.

In particular, we here consider *acyclic* hypergraphs, where there exists a *valid ordering* $V = (v_1, v_2, \ldots, v_n)$ of the nodes such that, for any $e \in \mathcal{E}$, each node $v_j \in T(e)$ precedes node $h(e)$ in V. The class of directed hypergraphs used here was denoted acyclic B-graphs in Gallo et al. [6] which considered the general class of directed hypergraphs. However, we here use the term "hypergraph" to denote the subclass appropriate in this context.

As shown in Pretolani [22] a *time-expanded hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ can be used to model an STD network. We illustrate the model by means of the following example.

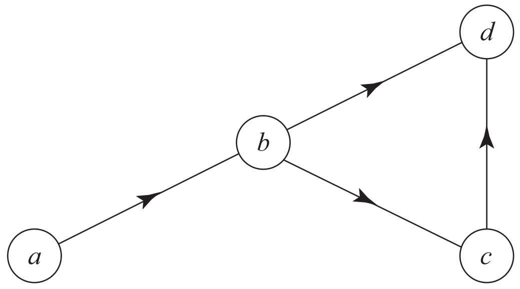


Figure 2: The topological network G.

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

Table 1: Input parameters.

Example 1 Consider the topological network $G=(N,A)$ in Figure 2, where a is the origin node and d is the destination node. For each arc in G, the possible departure and arrival times are listed in Table 1. Here a pair $((u,v),t)$ corresponds to a possible leaving time t from node u along arc (u,v). For the sake of simplicity, we assume that $X\left(u,v,t\right)$ has a uniform density, i.e., for each $t' \in I(u,v,t)$, we have $\theta_{uvt}(t') = 1/|I(i,j,t)|$. 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.

The time-expanded hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{E})$ is shown in Figure 3; numbers and dotted lines will be explained below. The set $\mathcal{V}$ contains one node u^t for each pair (u,t), $t \in L(u)$ and an origin node s. For each $(u,v)\in A$ and $t \in L(u,v)$, we introduce a hyperarc

$$e_{uv}(t) = \big(\{v^{t_i}:\ t_i \in I\left(u,v,t\right)\}, u^t\big).$$

Moreover, a dummy arc $e_d\left(t\right) = \big(\left\{s\right\}, d^t\big)$ is defined for each $t \in L(d)$. ■

It is obvious that $\mathcal{H}$ is an acyclic hypergraph: a valid ordering can be found by ranking the nodes in *decreasing* order of time. Furthermore, the size of $\mathcal{H}$ is $O(\kappa)$ and $\mathcal{H}$ can be built in $O\left(\kappa\right)$ time. Given any strategy S, let us define the sets

$$\mathcal{V}_S = \big\{u^t : (u,t) \in Dm(S) \cup Dd(S)\big\} \cup \big\{s\big\}$$

and

$$\mathcal{E}_S = \big\{e_{uv}(t):\ (u,t)\in Dm(S),\ S(u,t)=(u,v)\big\} \cup \big\{e_d(t):\ (d,t) \in Dd(S)\big\}.$$

Note that $\mathcal{V}_S$ contains a node u^t for each pair (u,t) corresponding to either leaving from an intermediate node or arriving at the destination. Moreover, for each node $u^t \in \mathcal{V}_S$, a single hyperarc in $\mathcal{V}_S$ with head u^t exists, more precisely, a dummy arc $e_d(t)$ if $u=d$, and a hyperarc $e_{uv}(t)$ if $u \neq d$. Let us denote by $r = o^0$ the node in $\mathcal{H}$ corresponding to the pair $(o,0)$. It has been shown in Pretolani [22] that $\pi_S = (\mathcal{V}_S, \mathcal{E}_S)$ is a *hyperpath* from node s to node r (s-r hyperpath) in $\mathcal{H}$. More precisely, the following property holds:

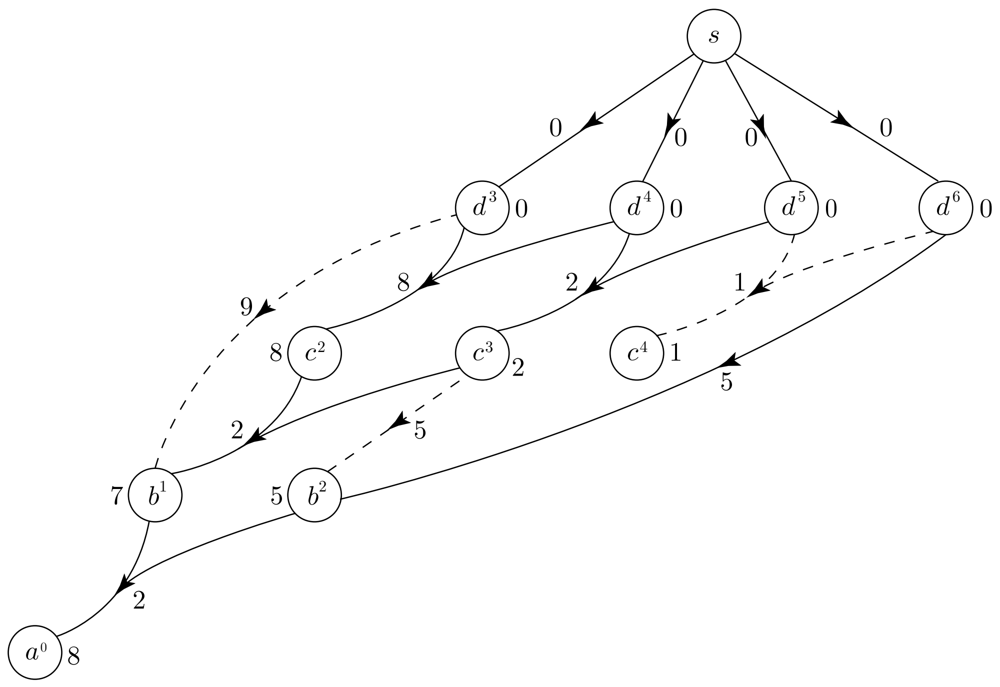


Figure 3: The time-expanded hypergraph $\mathcal{H}$.

Property 1 *There is a one-to-one correspondence between* $(o, 0)$ *strategies and* s-r *hyperpaths in* $\mathcal{H}$.

Pretolani [22] showed that the value of a $(o, 0)$ strategy under each one of the optimality criteria in Section 2.2 corresponds to the weight of the corresponding s-r hyperpath in $\mathcal{H}$ for a suitable definition of hyperpath weight, given in terms of *additive weighting functions*, see Gallo et al. [6]. Therefore, the best strategy can be found by finding the minimum weight s-r hyperpath, i.e., by solving a *shortest hyperpath* problem in $\mathcal{H}$. Quite efficient procedures for finding shortest hyperpaths are defined in Gallo et al. [6]; for acyclic hypergraphs, the computational complexity is linear in the size of the hypergraph. Thus, under time-adaptive route choice, the best strategy can be found in $O(\kappa)$ time. Clearly, this result does not extend to the a priori case in which a path-strategy is required.

Example 1 (continued) Hyperarcs in solid lines in Figure 3 represent the s-r hyperpath π_S corresponding to the best $(o, 0)$ strategy S for the expected cost criterion. Close to each hyperarc $e_{uv}(t)$, we report cost $c(u, v, t)$; penalty costs are zero and reported close to dummy arcs $e_d(t)$. The number close to each node u^t is the expected travel cost $E_S(u, t)$; thus, the minimum expected cost is 8. We have $S(c, t) = (c, d)$ for each time $t = 2, 3$, thus, S defines a unique successor for node c. However, the successor of b is (b, c) at time 2 and (b, d) at time 3, thus S is not a path-strategy.

Note that S defines the time-adaptive route to travel when leaving node a at time zero, but it does not define a successor for all possible nodes and leaving times, e.g. for node c at time 4. A non-$(o, 0)$ strategy S' defining the route to travel for all possible nodes and leaving times (i.e., with domain $Dm(S') = \mathcal{D}$) is obtained by adding the pair $(c, 4)$ to $Dm(S)$ and

```
procedure $Hall(G, o, d)$
    $k := 1$; $lb = 0$; $ub = \infty$;
    while $(lb < ub)$ do
        $P := FindKPath(G, o, d, k)$; $lb := L(P)$;
        if $(ET(P) < ub)$ then
            $ub := ET(P)$; $P^* := P$;
        end if
        $k := k + 1$;
    end while
    RETURN $P^*$;
end procedure
```

Figure 4: Hall's algorithm for finding the best a priori path.

defining $S'(c, 4) = (c, d)$.

Note also that it is not possible to arrive at node c at time 4. Thus, the pair $(c, 4)$ might be eliminated from $\mathcal{V}$, i.e., time 4 might be eliminated from $L(c)$ an $L(c, d)$. However, pair $(c, 4)$ shall be used later, in relation to waiting.

■

3 Finding the best and K best path-strategies

In this section we consider the problem of finding the shortest simple *o*-*d* path in G, which in light of Theorem 1 is equivalent to finding the best path-strategy. Since our solution approach is based on ranking strategies in nondecreasing order of weight, it is rather straightforward to adapt our method to the ranking of the K best path-strategies. In fact, we shall directly provide a description of the procedure for the K best path-strategies case. Specialized versions and extensions are discussed later. We start the section by discussing some technical issues related to previous work on the subject that provided some guidelines for our approach.

Hall [7] proposed an enumerative method for finding the minimum expected travel time path. His method consists in generating *o*-*d* paths in nondecreasing order of *length*, where the length of a path is the sum of the minimum possible travel times on its arcs; clearly, the length provides a lower bound on the expected travel time. For each generated path, the actual expected travel time is computed, and the best solution found so far is kept, providing an upper bound on the optimal solution. Enumeration stops as soon as the current best solution is found to be optimal, i.e., when the length of the next generated path is greater than or equal to the upper bound. Hall's algorithm is described in Figure 4; here, function $FindKPath(G, o, d, k)$ provides the k'th shortest *o*-*d* path, while $L(P)$ and $ET(P)$ are the length and the expected travel time of path P, respectively.

The enumeration of paths in Hall's algorithm can be performed by means of K shortest paths procedures. However, the path length defined by Hall may provide a quite poor lower bound on the expected travel time. Therefore, the algorithm may generate a huge number of paths, and possibly, may generate all of them without meeting the stopping criterion. This is clearly a major drawback of Hall's approach. A better bound can perhaps be obtained by generating paths according to their *least possible travel time*, see [12]. No actual implementation of Hall's method has been reported in the literature so far.

Later, a different enumeration approach was suggested in Pretolani [22]. This consists in enumerating strategies in nondecreasing order of weight until the first (and thus best) path-strategy is generated. However, this approach also has a clear drawback, due to the fact that path-strategies are likely to be a small subset of strategies. Thus, a huge number of non-path strategies may be generated before finding a path-strategy. In the next section we report and discuss computational results that support this hypothesis.

The method we propose in this paper aims at combining the most promising aspects of the previous approaches. On te one hand, our method generates paths according to a lower bound, as proposed by Hall. On the other hand, it uses strategies as a lower bound on path-strategies, which is likely to provide a much tighter lower bound. In fact, our method takes advantage of the techniques devised for the K shortest hyperpaths problem, and adapts them to the enumeration of paths by introducing a specialized branching rule.

3.1 A new enumeration schema

Consider an STD network with topological network G, and let S be the best $(o, 0)$ strategy. In principle, S might be a path-strategy and thus the best solution to our problem. This situation can be verified quite easily. Indeed, if S corresponds to a path $P = (u_1, u_2, \dots, u_{l+1} = d)$ in G, we can generate iteratively the subsets $D^{(i)}$ defined in (4): given $D^{(1)} = \{(o, 0)\}$, at each iteration we obtain $D^{(i+1)}$ by processing the pairs in $D^{(i)}$. Clearly, all the generated subsets must satisfy equation (2), more precisely, each $D^{(i)}$ must contain pairs (u, t) with $u = u_i$.

In principle, the above iterative method can also be applied to a non path-strategy. In this way, we will eventually generate a set $D^{(q+1)}$ that does not contain pairs corresponding to the same node in G. As a result S defines a subpath P_S from o to node u_q in G. We provide a formal definition of P_S.

Definition 3 A non path-strategy S defines a o-u_q subpath $P_S = (o = u_1, u_2, , ..., u_q)$, $u_q \neq d$, satisfying

$$D^{(i)} \subseteq \{(u_i, t) : \ t \in H\}, 1 \leq i \leq q$$

where the sets $D^{(1)}$, $\dots$,$D^{(q)}$ are defined as in (4). Furthermore, $D^{(q)}$ contains at least two pairs (u_q, t) and (u_q, t') such that $S(u_q, t) \neq S(u_q, t')$.

Note that we always have $q \geq 2$ in Definition 3, as $D^{(1)}$ contains the single pair $(o, 0)$. A procedure that finds the subpath P_S, denoted as *findSubP*, has been described in [17].

Let $\mathcal{S}$ and $\mathcal{S}_P \subseteq \mathcal{S}$ denote the sets of strategies and path-strategies, respectively; moreover, let $\mathcal{P}$ denote the set of loopless o-d paths in G. Our method for finding the best path-strategy follows the classical approach adopted for ranking the K shortest loopless path in a directed graph. We thus recursively partition the set $\mathcal{S}_P$, or equivalently the set $\mathcal{P}$, into smaller subsets. In our case, however, we adopt a particular partition technique, which is based on subpaths rather than paths. The idea is to partition $\mathcal{P}$ by using the subpath P_S, defined by the best strategy S (see Definition 3). To this aim, we apply to P_S the branching operation defined by Yen [25], with a special treatment of node u_q.

Definition 4 Given non path-strategy S defining subpath P_S let

$$P_B = P_S \cup a = (o = u_1, ..., u_q, u_{q+1})$$

denote the *branching path* obtained from P_S by adding an arc $a = (u_q, u_{q+1})$ used by S; i.e., $S(u_q, t) = a$ for some $(u_q, t) \in D^{(q)}$.

As follows from Definition 3, several arcs may be used to obtain P_B in Definition 4; here, we do not make any assumptions about the way a is chosen.

Branching Operation 1 *Given subpath $P_B = (o = u_1, ..., u_{q+1})$, the set $\mathcal{P}$ can be partitioned into disjoint subsets $\mathcal{P}^i$, $1 \leq i \leq q+1$ as follows:*

1. *For $1 \leq i \leq q$, paths in $\mathcal{P}^i$ contain path $P_B^i = (u_1, ..., u_i)$ but do not contain arc (u_i, u_{i+1});*
2. *Paths in $\mathcal{P}^{q+1}$ contain path P_B.*

It is quite obvious that the sets $\mathcal{P}^i$ in Branching Operation 1 are disjoint and define a partition of $\mathcal{P}$. Note, if $u_{q+1} = d$, then $\mathcal{P}^{q+1}$ contains a single path, namely P_B. Clearly, Branching Operation 1 only considers the partition of $\mathcal{P}$. However, it can be recursively applied to the subsets $\mathcal{P}^i$. The partition of $\mathcal{P}$ induces an obvious partition of $\mathcal{S}_P$ into subsets $\mathcal{S}_P^1, \ldots, \mathcal{S}_P^{q+1}$; moreover, it implicitly defines a family of subgraphs of G.

Definition 5 Given P_B, let subgraph G^i, $i = 1, ..., q+1$ be obtained from G as follows

1. For each node u_j, $j = 1, ..., i-1$, remove each arc in $FS(u_j)$ except (u_j, u_{j+1}), i.e., *fix* arc (u_j, u_{j+1});
2. If $i \neq q+1$, remove arc (u_i, u_{i+1}).

Finally, for each subgraph G^i, we can build a corresponding time-expanded hypergraph $\mathcal{H}^i$.

Theorem 2 *Given the sets $\mathcal{P}^i$, $1 \leq i \leq q+1$, the following statements are equivalent:*

1. *$P \in \mathcal{P}^i$.*
2. *P is an o-d path in G^i.*
3. *There is a unique path-strategy $S^i \in \mathcal{S}_P^i$ corresponding to path P in G^i.*
4. *There is a unique s-r hyperpath π in $\mathcal{H}^i$, corresponding to path-strategy $S^i \in \mathcal{S}_P^i$.*

Proof The fact that 1. and 2. are equivalent is obvious; a formal proof may be given by adapting the proof of correctness for Yen's algorithm. The equivalence of the other two statements follows from the one-to-one correspondence between paths and path-strategies (see Theorem 1) and between strategies and hyperpaths (see Proposition 1). ■

Using Theorem 2, each subset $\mathcal{P}^i$, or equivalently $\mathcal{S}_P^i$, can be represented by its corresponding subgraph G^i. In other words, each subgraph G^i defines an STD network where the set of strategies is $\mathcal{S}^i$ and the set of path-strategies is given by the subset $\mathcal{S}_P^i \subseteq \mathcal{S}^i$. We, thus, obtain the following useful result.

Corollary 1 *The weight of the best strategy in $\mathcal{S}^i$ is a lower bound on the weight of the best path-strategy in $\mathcal{S}_P^i$.*

Recall that the best strategy in $\mathcal{S}^i$ can be found in $O(\kappa)$ time by solving a shortest hyperpath problem in $\mathcal{H}^i$.

Consider again the best strategy S in the STD defined by G, and suppose we branch on S, applying Branching Operation 1. It is easy to see that none of the graphs G^i, $1 \leq i \leq q+1$, contains all the arcs that are used as successors by S. Thus, $S \notin \mathcal{S}^i$ for any $1 \leq i \leq q+1$. As a consequence, if we apply the branching rule recursively, we shall never branch on S again.

We can now describe our method more formally. The algorithm maintains a set of pairs $(\tilde{lb}, \tilde{G})$, representing the current partition of $\mathcal{P}$. Graph $\tilde{G}$ represents a subset $\tilde{\mathcal{S}}_P \subseteq \mathcal{S}_P$, or equivalently, a subset $\tilde{\mathcal{P}} \subseteq \mathcal{P}$; $\tilde{lb}$ is the weight of the best strategy $\tilde{S}$ in the STD network defined by $\tilde{G}$. At each iteration, the pair $(\tilde{lb}, \tilde{G})$ with minimum $\tilde{lb}$ is removed from the candidate set, and the best strategy $\tilde{S}$ is considered. If $\tilde{S}$ is a path-strategy, then it is the best one among the remaining path-strategies. If the K shortest path-strategy has been found, the algorithm terminates; otherwise it continues with the next iteration. If $\tilde{S}$ is not a path-strategy, Branching Operation 1 is applied to subpath P_B of $\tilde{S}$ (see Definition 4), and the resulting pairs are added to the candidate set.

Our algorithm is described by procedure *K-BPS* in Figure 5. Clearly, the best path-strategy is obtained by setting $K = 1$. Procedure *K-BPS* uses the following subprocedures:

BSW(G): Returns the weight of the best strategy in the STD network defined by subgraph G.

BestStrategy(G): Returns the best strategy in the STD network defined by subgraph G.

BranchingSet(G, S): Returns the set of subgraphs of G obtained by applying Branching Operation 1 to subpath P_B of S (see Definition 4 and Definition 5).

delMin$()$: Removes from the candidate set and returns the pair (lb, G) with minimum value lb.

insert(lb, G): Adds to the candidate set the pair (lb, G).

In fact, in order to obtain a version of *K-BPS* that ranks the K best path-strategies, a further detail must be taken into account. Consider a pair $(\tilde{lb}, \tilde{G})$ and suppose that the best strategy $\tilde{S}$ is a path-strategy, thus defining the shortest path $P = (o = u_1, \ldots, u_{q+1})$ in the subset $\tilde{\mathcal{P}}$. It is clear that we cannot apply Branching Operation 1 here, since the branching path P_B is not defined for a path-strategy. However, we can still apply Branching Operation 1 on path P. In this case, set $\mathcal{P}^{q+1}$ contains the single path P, and we obtain a partition of $\tilde{\mathcal{P}} \setminus \{P\}$ by skipping $\mathcal{P}^{q+1}$.

In the following we assume that procedure *BranchingSet*(G, S) returns the set of subgraphs of G obtained by applying Branching Operation 1 to subpath P_B if S is not a path-strategy. If S is a path-strategy, *BranchingSet*(G, S) returns the subgraphs $G^1 \ldots G^q$ of G, obtained by applying Branching Operation 1 to the optimal path P.

As discussed above, by applying Branching Operation 1 we never branch on the same strategy twice. Clearly this is also true if we branch on P instead of P_B. This suffices to prove the following result:

Theorem 3 *Procedure K-BPS finds the K best path-strategies in a finite number of iterations.*

```
procedure $K\text{-}BPS(\mathcal{H}, G, s, t, K)$
    $w := BSW(G)$;
    if $(w = \infty)$ then STOP (there is no path-strategy);
    $insert(w, G)$; $k := 0$
    while $(k < K)$ do
        $(\tilde{lb}, \tilde{G}) := delMin()$;
        if $((\tilde{lb}, \tilde{G}) = null)$ then STOP (there are no more path-strategies);
        $\tilde{S} := BestStrategy(\tilde{G})$;
        if $(\tilde{S} \in \mathcal{S}_P)$ then $k := k + 1$, OUTPUT the $k$'th path-strategy $\tilde{S}$;
        if $(k < K)$ then
            $\Gamma := BranchingSet(\tilde{G}, \tilde{S})$;
            for $(G' \in \Gamma)$ do
                $lb' := BSW(G')$;
                if $(lb' < +\infty)$ then $insert(lb', G')$;
            end for
        end if
    end while
end procedure
```

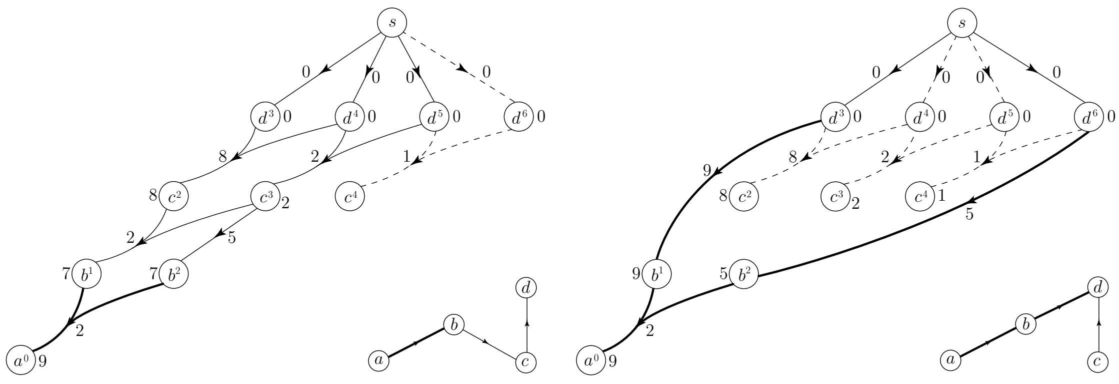
Figure 5: Finding the K best path-strategies.

The worst case complexity of *K-BPS* is exponential, as it may be necessary to enumerate an exponential number of strategies. The same holds true for $K = 1$. Note, however, that the procedure takes time $O(\kappa)$ for each subproblem inserted into the candidate set. Concerning the space complexity, it is possible to show that a (small) constant amount of information needs to be stored for each subproblem inserted into the candidate set. [17] may be consulted for a description of the data structures and the implementation details.

Example 1 (continued) Consider the best $(o, 0)$ strategy S shown in Figure 3; S is not a path-strategy and defines the subpath $P_S = (u_1 = a, u_2 = b)$. Assume that arc (b, d) is chosen in path $P_B = (u_1 = a, u_2 = b, u_3 = d)$; Branching Operation 1 defines three subgraphs G^1, G^2 and G^3. Here, G^1 does not contain arc (a, b) and, thus, does not contain any a-d paths; G^2 does not contain arc (b, d) and consists of a single a-d path (a, b, c, d); G^3 does not contain arc (b, c) and contains a single a-d path (a, b, d). Graphs G^2 and G^3 and the corresponding expanded hypergraphs $\mathcal{H}^2$ and $\mathcal{H}^3$ are shown in Figure 6(a) and Figure 6(b), respectively. Fixed arcs and corresponding hyperarcs are shown in bold lines; solid lines represent the best a-d strategy, which in both cases is a path-strategy with expected cost 9. ■

3.2 Improved procedures

We consider two possible enhancements of procedure *K-BPS*. The first is based on reoptimization techniques for shortest hyperpaths. The second is based on a different branching technique.



(a) G^2, $\mathcal{H}^2$ (b) G^3, $\mathcal{H}^3$

Figure 6: Subproblems defined by Branching Operation 1

3.2.1 A lower bound based on reoptimization

In procedure *K-BPS* each pair $(\tilde{lb}, \tilde{G})$ is ranked according to a lower bound on the weight of the best path-strategy, namely, the weight $\tilde{w}$ of the best strategy. In order to find $\tilde{w}$, a minimum weight hyperpath problem must be solved for each subgraph generated during the branching operation, which is quite expensive in terms of computation time. A possible alternative is to compute a fast lower bound on $\tilde{w}$. In this case, the computation of the minimum weight hyperpath will be performed only when (and if) the pair $(\tilde{lb}, \tilde{G})$ is selected from the candidate set.

Since pairs are selected according to a lower bound on the weight of the best strategy, strategies are not necessarily generated in non-decreasing order of weight. As a consequence, a slightly more complex treatment of the candidate set is necessary. Suppose we select a pair $(\tilde{lb}, \tilde{G})$, and let $\tilde{w}$ be the weight of the best strategy $\tilde{S}$. Two cases may arise:

1. $\tilde{w}$ is lower than or equal to the current minimum lb in the candidate set;
2. $\tilde{w}$ is greater than the current minimum lb in the candidate set.

In the first case, $\tilde{S}$ is indeed the best strategy not yet generated, thus we proceed as in procedure *K-BPS*. In the second case, this is not necessarily true, and we reinsert the pair $(\tilde{lb}, \tilde{G})$ into the candidate set with the lower bound $\tilde{lb}$ updated to $\tilde{w}$. Thus, we do not perform any branching nor "output" operation, but start a new iteration instead.

A lower bound on the minimum strategy weight $\tilde{w}$ can be found by exploiting *reoptimization techniques* for shortest hyperpaths. Such techniques have been proposed in Nielsen et al. [19], and further developed in Nielsen et al. [21], in the context of K shortest hyperpath procedures. The results in [19, 21] are technically rather involved, and are not reported here. In practice, these results have the following impact in our context: when Branching Operation 1 is applied, we can obtain a lower bound for each generated subproblem with an overall $O(\kappa)$ computational cost. A procedure based on reoptimization can be derived from *K-BPS*, and will be denoted as *K-BPSreopt*. Here we assume that procedure *BSW* computes a lower bound using reoptimization. Since *K-BPSreopt* only differs slightly from *K-BPS*, we omit a pseudo-code description here.

3.2.2 Multiple Branching

Another variant of procedure *K-BPS* can be obtained by adopting a different branching operation. In Branching Operation 1, *o-d* paths in subset $\mathcal{P}^i$ contain subpath P_B^i and do not contain arc (u_i, u_{i+1}), which is removed from G^i. We may further partition subset $\mathcal{P}^i$ by creating $|FS\,(u_i)| - 1$ subsets where we fix an arc $e \in FS\,(u_i)$, with $e \neq (u_i, u_{i+1})$. This gives rise to the following *multiple branching operation* defined in terms of the subpath P_S (see Definition 3).

Branching Operation 2 *Given path $P_S = (o = u_1, ..., u_q)$ the set $\mathcal{P}$ can be partitioned into disjoint subsets as follows:*

1. *For $1 \leq i < q$ and $e \in FS\,(u_i) \setminus \{(u_i, u_{i+1})\}$, paths in $\mathcal{P}^{i,e}$ contain path P_S^i and arc e.*
2. *For $e \in FS\,(u_q)$, paths in $\mathcal{P}^{q,e}$ contain path P_S and arc e.*

Note that Branching Operation 2 applies also if strategy S defines an optimal path $P_S = (o = u_1, ..., u_q = d)$, provided that $FS(d)$ is empty. Similarly to Definition 5, the subset $\mathcal{P}^{i,e}$ corresponds to a subgraph $G^{i,e}$ in G, which in turn defines an expanded hypergraph contained in $\mathcal{H}$. A theorem similar to Theorem 2 can also be proven (see Nielsen [17]).

Observe that Branching Operation 2 usually creates more pairs than Branching Operation 1 and is, thus, not suitable if a shortest hyperpath problem is solved for each pair. However, Branching Operation 2 is likely to provide a tighter reoptimization lower bound, since smaller subsets of strategies are considered. If the size of the forward star of a node in G is small (as is often the case e.g. in a road network) and if the lower bound is fast, we may obtain a faster procedure. We denote as *K-BPS_MB* the variant of procedure *K-BPSreopt* where multiple branching is adopted.

3.3 Waiting allowed

In this section we will briefly discuss the case in which *waiting* at the nodes is allowed. The subject will not be discussed in great detail here, but a thorough treatment of the subject is available in [17].

The possibility of waiting at intermediate nodes has been often considered in transportation models. If waiting is allowed, a traveller arriving at node $u \neq d$ at time $t < t_{\max}$ can either proceed towards the destination, i.e., leave node u at time t, or *wait at u* until time $t' > t$, i.e., leave node u at time t'. In the underlying topological network, waiting at u can be represented by a self-loop arc (u, u), where travel times are deterministically equal to one. In terms of strategies, the successor of the pair (u, t) is the arc (u, u). Waiting can be easily included in the hypergraph model of the STD network, too. Here, waiting at node u from time t to time $t + 1$ is represented by the hyperarc $e_{uu}(t) = (\{u^{(t+1)}\}, u^t)$.

From a theoretical point of view, waiting at intermediate nodes should not be considered within an a priori route choice model, as it is an inherently time-adaptive behavior. Indeed, while travelling along a path, a traveller has to choose, at any given time, whether to wait or proceed to the next node in the path. Clearly, the decision cannot be "waiting" at each time, since a traveller cannot wait indefinitely at intermediate nodes. However, it may be interesting to consider waiting as a limited form of time-adaptive behavior, thus defining an intermediate model between a priori and time-adaptive route choices.

A key observation here is that an *o*-*d* path P in G defines several strategies, actually, an exponential number in the length of P. In one sense, all of these should be considered as path-strategies, distinguished from each other only by the use of waiting. Thus, the definition of path-strategy should be slightly modified: in a $(o, 0)$ path-strategy S, the successor of each pair $(u, t) \in Dm(S)$ is either a fixed arc (u, v) or the waiting arc (u, u).

As long as the shortest path problem is considered, the new definition can be adopted safely. Indeed, we are here interested in a single best path-strategy, and clearly this path-strategy is the one that exploits waiting best, among those corresponding to the same path. As a consequence, our solution method remains valid.

However, in the K shortest path problem ($K > 1$) we need to distinguish between two cases, namely whether the K path-strategies should correspond to different paths in G or not. In the former case, our algorithm is still valid, as it enumerates path-strategies corresponding to different paths. In the latter case, we may instead need to enumerate different path-strategies corresponding to the same path. In fact, in this case we deal with a time-adaptive K shortest path problem, and we need to use a different enumeration method. To this aim, the techniques developed within K shortest hyperpaths procedures can be adapted.

Both types of a priori K shortest path problems with waiting have been considered in [17], where solution methods and computational results are reported. Computational experience shows that as long as path-strategies must correspond to different paths, the introduction of waiting does not affect the behavior of the algorithms significantly. On the other hand, if strategies can correspond to the same path, the results are quite close to the ones obtained under time-adaptive route choice. For these reasons, we do not report computational results for the waiting case in the next section.

Example 1 (continued) Consider graph G^2 and hypergraph $\mathcal{H}^2$, shown in Figure 6(a), and suppose that waiting is allowed at node c. Two arcs $e_{cc}(c, 3) = (\{c^4\}, c^3)$ and $e_{cc}(c, 2) = (\{c^3\}, c^2)$ must be added to $\mathcal{H}^2$ in order to represent waiting. In this situation, both node c^2 and node c^3 appear in the head of two hyperarcs. Indeed, two choices are possible for the pairs $(c, 2)$ and $(c, 3)$, namely, waiting at c or proceeding towards d. Note that this gives four different path-strategies corresponding to the path (a, b, c, d), according to the above definition. If the cost of waiting is zero, we obtain $E^S(c, t) = 1$ for $t = 2, 3, 4$, and the weight of the best $(o, 0)$ strategy becomes 6.5. ■

4 Computational results

In this section we report on the computational experience for finding the K best strategies under both time-adaptive and a priori route choices. The algorithm for finding the K best strategies under time-adaptive route choice is given in Nielsen et al. [21]. 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 have been compiled with the GNU C++ compiler with optimize option -O.

4.1 The TEGP generator

All tests are performed on time-expanded hypergraphs generated with the TEGP generator used in Nielsen et al. [18] and Nielsen [17]. The generator includes several features inspired

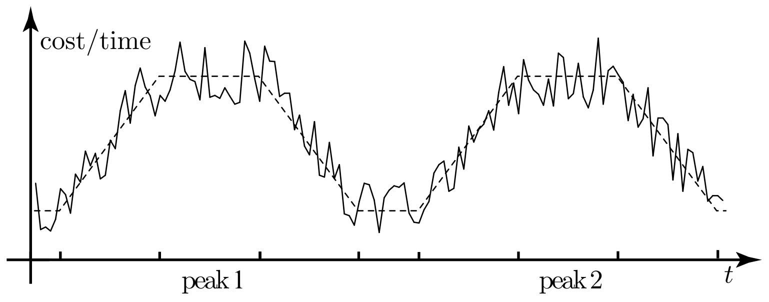


Figure 7: Peak effect and random perturbation.

by typical aspects of road networks (congestion effects, waiting, random perturbations etc.).

An underlying topological grid network G of *base* b and *height* h is assumed, and we search for optimal strategies from the bottom-right corner node (origin o) to the upper-left corner node (destination d). This choice is motivated by the fact that each origin-destination path has at least $b + h - 2$ arcs, and there is an exponential number of such paths in G.

The generator considers *cyclic* time periods (e.g. a day) and in each cyclic period there are some *peak periods* (e.g. rush hours). Each peak consists of three parts; a *transient* part where the mean travel time (traffic) increases, a *pure peak* part where it stays the same and a transient part where it decreases again. This feature gives travel time distribution with higher mean and higher standard deviation in peaks. The peak effect can be eliminated e.g. by setting the peak length to zero. The pattern of the mean travel time for a grid arc, when two peaks are considered, is shown in Figure 7 (dotted line).

Costs can be generated in two ways, namely using *peak dependent costs* or *random costs*. The generation of peak-dependent costs takes three components into account: the off-peak cost, the peak effect and a random perturbation. Hence, the cost has a component following the same pattern as the mean travel time, and a *random perturbation* introduces small variations, modelling factors not intercepted by the peak implementation, e.g. special information about the cost at exactly that leaving time. The pattern of the cost $c(u, v, t)$ for a grid arc (u, v) is shown in Figure 7 (solid line).

If the cost are random no off-peak component nor peak effect is considered, and the costs are generated randomly in a given interval. Note that with this setting, the travel costs of an arc at time t and $t + 1$ are not related. Clearly, in a road network, this model is not realistic.

If waiting is allowed, waiting costs are generated using an off-peak component and a random perturbation. For more details on the TEGP generator, see Nielsen [17, Section 3.5].

4.2 Hypergraph classes

In the following, we use the term *hypergraph class* to define a particular setting of the TEGP input parameters; for each class, different hypergraphs (i.e. different instances of the problem) can be generated by choosing different seeds. We considered six hypergraph classes, using three different grid sizes, (namely 5×10, 10×10 and 20×10) and two different cost structures. Five instances were generated for each class: Table 2 reports the average number of nodes n, the average number of hyperarcs m and the length of the time horizon H.

Classes 1-3 use peak dependent costs as an attempt to model “realistic” STD networks and show that our algorithms are expected to work well in practice. The mean travel times

	peak dependent costs			random costs		
Class	1	2	3	4	5	6
Grid size	5×10	10×10	20×10	5×10	10×10	20×10
n	2320	7573	21454	1497	3961	11856
m	7809	27278	79570	5056	14295	43991
H	118	156	237	75	101	155

Table 2: Hypergraph classes

and costs increase 100% in peaks, and the range of the random perturbation is set to 10% of the costs obtained including peak effects. These values of the parameters have been justified in Nielsen et al. [18]. Off-peak costs are generated in the interval $[1, 1000]$; due to the random perturbation, travel costs range in an interval $[0, 2200]$ during peaks.

Class 4-6 use random costs in the interval $[1, 2000]$ and no peak effect for the travel times. These classes aim at pointing out the impact of the random cost component, showing how the procedures work on difficult (unrealistic) instances.

In all classes, a cycle consists of 144 time instances. A cycle has two peaks, each one with a total length of 60 (each transient or pure peak period has a length of 20); the first peak starts at $t = 6$ and the second at time $t = 78$. For each arc in the underlying grid, the off-peak mean travel time μ ranges in the interval $[lb_t, ub_t] = [2, 6]$. Given μ, the possible travel times are the integers in the interval $[0.75\mu, 1.25\mu]$. For each grid size, the length of the time horizon H is an upper bound on the time required to travel along all *o-d* paths containing $(b + h)$ arcs. Note that H is much smaller for classes 4-6, where the peak effect on travel times is not present. As a consequence, the resulting hypergraph size is smaller.

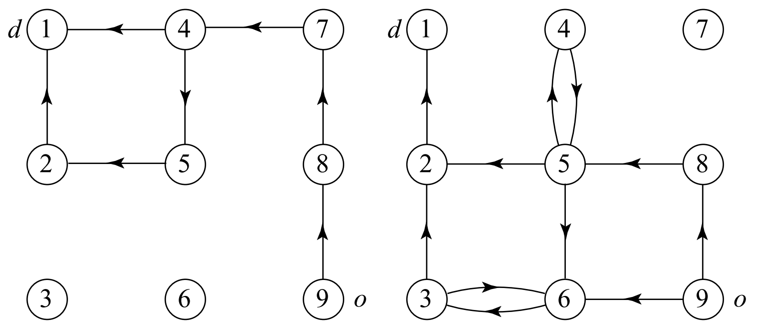
4.3 Aims and statistics

The aim of our computational experience is twofold. On the one hand, we try to evaluate the behavior of our algorithms on different classes of instances. On the other hand, we try to investigate the structure of the strategies and path-strategies obtained for those instances. This will allow us to compare time-adaptive and a priori route choice on the basis of both computing time requirements and expected structure of the resulting routes. The reported statistics can be divided into two groups. The first group considers the performance of the procedures, and is described below. The abbreviation´s used in the tables are given in parentheses.

CPU time (CPU_K): CPU time for finding the K best strategies (time-adaptive route choice) or path-strategies (a priori route choice). Does not include input/output time.

First CPU time (CPU_1): CPU time for finding the best strategy (time-adaptive route choice) or path-strategy (a priori route choice).

Number of iterations (ite_K): The number of strategies generated before finding the K best path-strategies under a priori route choice, i.e. the number of times the **while** loop in e.g. procedure *K-BPS* is executed.



(a) "path-like". (b) Not very "path-like".

Figure 8: A "path-like" and a not very "path-like" strategy.

Number of iterations (ite_1): The number of strategies generated before finding the best path-strategy under a priori route choice.

The second group of statistics is related to the structure and quality of the strategies used by the branching operation. Consider a strategy S with domain $Dm(S)$ on which the branching operation is applied. Denote by

$$A_S = \{a \in G : \ a = S(u,t), \ (u,t) \in Dm(s)\}$$

the *codomain* of a strategy S, i.e., the set of arcs in G that is used by S. The second group includes the following values.

Average domain size ($|Dm|$): The average cardinality of the domain of the strategies used by the branching operation.

Average codomain size ($|A_S|$): The average number of arcs used by the strategies.

Relative increase in weight (inc): The weight increase between the first and the Kth strategy (time-adaptive route choice) or path-strategy (a priori route choice). Reported in percent.

Relative increase strategy to path-strategy (inc_{S-PS}): The weight increase between the first strategy and the first path-strategy. Reported in percent.

Here we also examine how "path-like" the strategies are by considering how many paths the set of arcs A_S define. Recall that A_S defines an o-d path if and only if S is a path-strategy; otherwise, A_S defines a set of o-d paths, and can contain cycles. A path-like and a not very path-like strategy are shown in Figure 8 when considering a 3×3 grid network G.

In order to evaluate how path-like a strategy S is, we may consider the number of different successors of u in S, i.e., the number $\delta(u) = |\{(u,v) \in A_S\}|$. Since we consider an underlying grid graph G, $\delta(u)$ ranges between one and four. Given a strategy S, we compute the percentage of nodes with j successors, for $1 \leq j \leq 4$. Here we report these percentages, averaged on all the strategies used by the branching operation; note that this includes non-path strategies for the a priori case. We, thus, have the following statistics:

Average percentage of nodes with j successors ($\delta(u) = j$): the average percentage of nodes with j successors, $j = 1, 2, 3$ and 4.

Class	CPU_K	CPU_1	$\lvert Dm \rvert$	$\lvert A_S \rvert$	$\delta(u) = 1$	$\delta(u) = 2$	$\delta(u) = 3$	$\delta(u) = 4$	*inc*
1	9.46	0.01	340	26	65	24	9	2	0.00
2	24.79	0.03	895	54	60	29	10	1	0.00
3	70.17	0.07	1688	76	65	31	4	0	0.00
4	7.26	0.01	854	123	10	32	40	18	0.00
5	25.87	0.02	2162	253	6	24	41	29	0.00
6	75.79	0.05	7063	614	3	14	36	46	0.00

Table 3: Results for finding the $K = 1000$ best strategies.

4.4 Time-adaptive route choice

The results reported here have been obtained using the procedure in Nielsen et al. [21] that finds the K shortest hyperpaths in acyclic hypergraphs using reoptimization techniques in $O(\kappa K)$ time. Reoptimization techniques for K shortest hyperpaths have been shown to be quite effective (Nielsen et al. [19], Nielsen [17]), leading to procedures that can be order of magnitudes faster on the set of instances considered here.

Our main goal is to examine how the structure of the costs and travel times may effect the structure of the strategies. We only consider the minimum expected cost criterion, with no waiting allowed. Other criteria, as well as the introduction of waiting, lead to similar results, see Nielsen [17, Section 4.3] for details.

The results for finding the $K = 1000$ best strategies are reported in Table 3. Recall that for each hypergraph class, the measures are averaged over five runs using different seeds.

The most evident effect that can be pointed out is the impact of cost structure on strategies. In fact, peak-dependent classes generate strategies that are much more path-like than the ones generated by random classes. For example, the average number of successors increases from 1.39 in class 3 to 3.23 in class 6. As a consequence, strategies in random classes tend to have a much larger domain and codomain. Note also that the impact of random costs is more evident for larger classes. Using random costs, this situation can be explained recalling that the travelling costs for the same arc at different leaving times are uncorrelated. As a consequence, all the successor arcs are equally likely to provide the best solution, for a given node and leaving time.

A further evident observation concerns the relative increase in costs, which is negligible, namely, always less than one percent. In the context of K shortest hypergraph procedures, this situation has already been pointed out and discussed in Nielsen et al. [18]; a detailed formal treatment can be found in Nielsen et al. [21]. In terms of strategies, the following informal explanation can be given. Consider a strategy S and a pair $(u, t) \in Dm(S)$; it can be shown that an increase Δ in the expected cost $E^S(u, t)$ at (u, t) results in an increase Δf_{ut} in the expected cost $E^S(o, 0)$, where the multiplier f_{ut} is likely to be quite small for many pairs (u, t). Roughly speaking, changing the successor $S(u, t)$ will not affect the weight of S significantly. As a consequence, it is likely that a large number of strategies has almost similar expected costs. Similar observations apply to min/max criteria too (see Section 2.2).

Class	ite_K	CPU$_K$	ite_1	CPU$_1$	$\|Dm\|$	$\|A_S\|$	$\delta(u)=1$	$\delta(u)=2$	$\delta(u)=3$	$\delta(u)=4$	inc	$inc_{S\text{-}PS}$
12	356	8	13	0.3	233	22	82	14	4	1	49	5
13	454	37	11	1.0	411	31	82	14	4	0	17	7
14	2359	875	133	53.5	892	46	85	13	2	0	10	5
15	1427	25	133	2.5	203	46	50	25	20	5	18	43
16	9942	564	2722	157.9	475	90	39	21	27	12	8	54
17	203479	34227	89519	15127.5	1748	244	25	16	27	31	3	59

Table 4: Results for finding the $K = 100$ best path-strategies.

4.5 A priori route choice

Our main goal is to evaluate the behavior of our algorithms on different classes of instances and to compare the performance of procedure *K-BPS* to those of the two enhanced versions *K-BPSreopt* and *K-BPS_MB*. However, we start by pointing out the impact of cost structures on the generated strategies. Since the a priori case is expected to be much more difficult, we only generate 100 path-strategies here. The results of procedure *K-BPS* are reported in Table 4.

Observe that the generated strategies are much more path-like than in the time-adaptive case. The average number of successors increases from 1.17 for class 3 to 2.62 for class 6. As a consequence, the size of the domain and codomain is smaller too. Clearly, this is explained by the fact that our branching technique fixes subpaths, so that many nodes will have exactly one successor.

Again, random costs have a clear impact on how path-like the strategies are, although the effect is less than in the time-adaptive case. Furthermore, since strategies under random costs are in general not path-like, branching paths become short. As a result, the number of non-path strategies generated before the first path-strategy is found is high (ite_1 column), yielding high CPU times.

Interesting observations arise from the results of relative weight increase. Contrary to the time-adaptive case, there is a significant difference between the first and the K^{th} path-strategy. This is to be expected as path-strategies are less flexible than strategies. Note that the relative increase (*inc* column) is smaller for random costs and for larger grids. The effect of larger grid size is expected since the number of *o-d* paths is exponential in the grid dimensions, which for larger grids yields more paths with weight close to the optimum. The difference between peak-dependent and random costs can be explained as follows. In the peak-dependent case, the costs $c(u, v, t)$ are roughly proportional to the off-peak cost of arc (u, v); since off-peak costs on grid arcs can differ substantially , we can expect some *o-d* paths to be much cheaper than others. On the contrary, in the random case the costs $c(u, v, t)$ vary randomly in the whole cost interval; as a result, all the paths with the same number of arcs may be expected to be equivalent, i.e. to have expected cost close to each other.

As expected, the weight of the best path-strategy is greater (inc_{S-PS} column) than the weight of the best strategy. To some extent this confirms that path-strategies are a "small" subset of strategies. Note that the increase is much smaller for peak-dependent classes, where strategies are much more path like: actually, this is a consequence of the cost distribution of

Class	ite_K	$reins$	CPU$_K$	ite_1	CPU$_1$	ite_K	$reins$	CPU$_K$	ite_1	CPU$_1$
	K-BPSreopt					*K-BPS_MB*				
1	542	32	6	20	0.2	484	34	5	19	0.2
2	664	30	23	14	0.5	589	35	20	12	0.4
3	3732	36	375	194	20.1	3379	42	500	183	27.3
4	2297	38	24	205	2.2	1995	47	20	178	1.8
5	16336	39	529	4430	146.1	13896	50	424	3752	115.8
6	334272	39	27333	145971	11927.6	280451	51	27529	123048	12107.4

Table 5: Results for procedures *K-BPSreopt* and *K-BPS_MB* (100 path-strategies).

o-d paths, as discussed above. Indeed, for peak-dependent costs, the best strategy is likely to deviate only little from the shortest *o-d* path, and thus to be quite similar to the a priori shortest path.

Note that the increase is much smaller for peak-dependent classes, due to the fact that for peak-dependent costs, we find a path-strategy faster as the strategies are more path-like.

As discussed above, the random classes considered here do not seem to be a reasonable model for a realistic STD network. Instead, they are likely to provide very difficult instances for our algorithm. In fact, random classes are up to two order of magnitude more demanding, both in terms of computation time and number of iterations. Nevertheless, this behavior seems to provide some evidence of the robustness of our algorithms.

Procedures *K-BPSreopt* and *K-BPS_MB* were also tested on the same classes of instances. The results are reported in Table 5. The statistics related to the structure of the generated strategies are not reported here, since the results are similar to the ones discussed above. However, we report one further item of statistics, related to the behavior of procedures *K-BPSreopt* and *K-BPS_MB*.

Number of reinsertions (*reins*): The number of pairs reinserted into the candidate set after computing the actual best strategy (see Section 3.2.1). Reported in percent of the number of iterations ite_K.

Both procedures *K-BPSreopt* and *K-BPS_MB* turn out to be faster than *K-BPS*, about 20% on the hardest instances. Note that this is at the expense of the number of iterations, that increases up to 60% on the hardest instances. The number of reinsertions is relatively high, which may suggest that the reoptimization lower bound is not sufficiently tight, although tight enough to reduce the overall CPU time.

A comparison between *K-BPSreopt* and *K-BPS_MB* is not very significant, since the two procedures show similar results, the latter requiring less iterations and more reinsertions than the former. Note, however, that procedure *K-BPS_MB* has larger memory requirements due to its branching rule.

5 Conclusions

In this paper we devised a solution method for the a priori shortest path problem in stochastic time-dependent network and extended this method to include the finding of the a priori K

shortest paths. Two variants of the method were devised, introducing reoptimization based on lower bounding techniques and a different branching operation. The effectiveness and robustness of our algorithm were evaluated against a set of hard instances. Based on the algorithms proposed here and on an existing K shortest hyperpath algorithm, we compared the results obtained under time-adaptive and a priori route choices. Finally, we pointed out the impact of the problem data on the structure of strategies and path-strategies. The new contributions can be summarized as follows.

To our knowledge, this is the first attempt to solve the a priori shortest path, originally addressed by Hall who considered loopless paths between a single origin-destination pair and for a single leaving time. The method proposed by Miller-Hooks and Mahmassani [14] may possibly be adapted to the problem considered here. However, their approach is conceived for a much more general problem (multiple origin, multiple leaving time and looping paths) and, therefore, may not be suitable for finding the specific set of solutions that we are interested in. In addition, we here provide the first solution method for the a priori K shortest path problem. Finally, the computational analysis carried out here is original in at least two aspects:

1. The analysis of the structure of the best K routes generated under time-adaptive and a priori route choices;

2. The analysis of the impact of both travel times and costs on the structure of the generated strategies and path-strategies.

Previous work in this direction concentrated on travel times and best routes (Miller-Hooks [11]) and on theoretical aspects (Miller-Hooks and Mahmassani [15]).

From a computational point of view, the results reported in this paper are quite encouraging. Our algorithms may be expected to be effective for instances arising from (reasonable approximations of) real networks, and relatively robust when faced with hard instances. These results suggest a possible direction for further research, related to the bicriterion version of the a priori shortest path problem. In this context, it is possible to devise solution methods that take advantage of fast procedures for ranking shortest paths. Actually, these methods have been widely investigated in the literature in the case of deterministic networks. Along this line, the time-adaptive case in STD networks was treated in Nielsen et al. [18], and the a priori case is the subject of current research (Nielsen, Pretolani, and Andersen [20]). Note that the a priori problem has interesting applications already addressed in the literature (Miller-Hooks and Mahmassani [13]).

References

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

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

[3] I. Chabini. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. *Transportation Research Record*, 1645:170–175, 1998.

[4] K.L. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14:493–498, 1966.

[5] A. Eiger, P.B. Mirchandani, and H. Soroush. Path preferences and optimal paths in probabilistic networks. *Transportation Science*, 19(1):75–84, 1985.

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

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

[8] J.E. Hershberger, M. Maxel, and S. Suri. Finding the k shortest simple paths: A new algorithm and its implementation. In *Proc. 5th Worksh. Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2003.

[9] E.L. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path. *Management Science*, 18(7):401–405, March 1972.

[10] R.P. Loui. Optimal paths in graphs with stochastic or multidimensional weights. *Communications of the ACM*, 26:670–676, 1983.

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

[12] E.D. Miller-Hooks and H.S. Mahmassani. Least possible time paths in stochastic, time-varying networks. *Computers & Operations Research*, 25:1107–1125, 1998.

[13] 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.

[14] E.D. Miller-Hooks and H.S. Mahmassani. Least expected time paths in stochastic, time-varying transportation networks. *Transportation Science*, 34(2):198–215, 2000.

[15] E.D. Miller-Hooks and H.S. Mahmassani. Path comparisons for a priori and time-adaptive decisions in stochastic, time-varying networks. *European Journal of Operational Research*, 146(1):67–82, 2003.

[16] I. Murthy and S. Sarkar. A relaxation-based pruning technique for a class of stochastic shortest path problems. *Transportation Science*, 30(3):220–236, 1996.

[17] L.R. Nielsen. *Route Choice in Stochastic Time-Dependent Networks.* PhD thesis, Department of Operations Research, University of Aarhus, 2004. URL `http://www.research.relund.dk/`.

[18] L.R. Nielsen, K.A. Andersen, and D. Pretolani. Bicriterion shortest hyperpaths in random time-dependent networks. *IMA Journal of Management Mathematics*, 14(3):271–303, 2003.

[19] L.R. Nielsen, K.A. Andersen, and D. Pretolani. Finding the K shortest hyperpaths. *Computers & Operations Research*, 32(6):1477–1497, 2005. URL `http://dx.doi.org/doi:10.1016/j.cor.2003.11.014`.

[20] L.R. Nielsen, D. Pretolani, and K.A. Andersen. Bicriterion a priori route choice in stochastic time-dependent networks. URL `http://www.research.relund.dk/`. Submitted, 2004.

[21] L.R. Nielsen, D. Pretolani, and K.A. Andersen. Finding the K shortest hyperpaths using reoptimization. Technical Report WP-L-2004-04, Department of Accounting, Finance and Logistics, 2004. URL `http://www.asb.dk/departments/afl/research/lrg/workingpapers/`. Submitted.

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

[23] J.S. Provan. A polynomial-time algorithm to find shortest paths with recourse. *Networks*, 41(2):115–125, 2003.

[24] H.N. Psaraftis and J.N. Tsitsiklis. Dynamic shortest paths in acyclic networks with Markovian arc costs. *Operations Research*, 41(1):91–101, 1993.

[25] J.Y. Yen. Finding the K shortest loopless paths in a network. *Management Science*, 17 (11):712–716, 1971.

Handelshøjskolen i Århus

Aarhus
School of Business

ISBN 87-7882-006-5

Department of Accounting, Finance and Logistics
Faculty of Business Administration