# Ranking paths in stochastic time-dependent networks

Lars Relund Nielsen* and Kim Allan Andersen
CORAL, Department of Economics and Business, Aarhus University, Denmark

Daniele Pretolani
Department of Sciences and Methods for Engineering , University of Modena and Reggio Emilia, Italy

**Abstract:** In this paper we address optimal routing problems in networks where travel times 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. Nevertheless, in some particular cases an origin-destination path must be chosen *a priori*, since time-adaptive choices are not allowed. Unfortunately, finding the a priori shortest path is an NP-hard problem.
In this paper, we propose a solution method for the a priori shortest path problem, and we show that it can be easily extended to the ranking of the first $K$ shortest paths. Our method exploits the solution of the time-adaptive routing problem as a relaxation of the a priori problem. Computational results are presented showing that, under realistic distributions of travel times and costs, our solution methods are effective and robust.

*Keywords: shortest paths; ranking; stochastic time-dependent networks; routing.*

## 1 Introduction

Classical optimization models for routing commodities, vehicles, passengers etc. in a transportation network assume that link travel times are deterministically known and do not evolve over time. In real cases this assumption is often unrealistic, indeed, several different ways of representing uncertainty and/or variability have been proposed in the literature. In this paper we consider *stochastic time-dependent networks* (*STD networks*) where link travel times are represented by random variables with probability distributions varying as a function of departure times. We distinguish between *discrete* and *continuous* STD networks, according to the representation of time and the nature of the random variables. Routing models based on STD networks have been often adopted in application areas such as hazardous material transportation [4], advanced vehicle navigation and traveler information systems [2, 7] and transit passenger path choice [9].

Optimal routing in STD networks was first addressed by Hall [8], who considered the minimization of *expected travel time* for a given origin/destination pair and starting time. Hall pointed out two different ways of formulating the problem. If a route must be specified before travel begins, and no deviations are permitted, a simple (i.e., loopless) path must be selected; this is referred to as *a priori route choice*. However, the shortest 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*. The adaptive problem turns out to be computationally easy; in a discrete STD-network an optimal strategy can be found in linear time in the size of the network description

*Corresponding author (lars@relund.dk)

[11, 19]. By contrast, a priori route choice has been shown to be strongly NP-hard also for discrete and deterministic time-dependent networks [18, 19].

A solution approach for a priori routing, valid for both discrete and continuous STD networks, was proposed by Hall [8]; his method enumerates loopless paths in a suitably defined deterministic and static version of the STD network. As observed in [12], Hall's method has relevant potential drawbacks, since it may process all existing paths before termination. For discrete STD networks, Miller-Hooks and Mahmassani [12] proposed a *labeling* algorithm that finds a priori optimal paths, to a given destination, from all the other nodes and for all possible leaving time. In fact, they allow the paths to be looping, which is a major depart from Hall's original model. This method has similar drawbacks to Hall's one (see [12, Proposition 4]) but turns out to be quite effective for sparse random networks. For continuous STD networks, Fu and Rilett [6] proposed a heuristic approach based on a technique for approximating the expected length of a path; based on similar techniques, an efficient algorithm was devised by Fu [5], and later extended to the multicriterion case by Chang *et al.* [4].

Despite being less flexible and computationally harder, a priori route choice may represent the only alternative in several situations. This is the case, for example, if the traveller does not have access, or is not willing (think of a daily commuter) to react to information made available during the travel. Another relevant example is the transportation of highly sensitive substances, where it is necessary to commit in advance to a specific path, that must be preapproved and monitored; Chang et al. [4] address a priori routing in this context. Moreover, on the strategic level the difference between the minimum cost under a priori and time-adaptive routing may provide an indication of the value of using on-line information. This indication may support the decision of investing in on-board navigation systems and road infrastructures, see e.g. [2, 3, 7] for some related issues. A comparison between a priori and adaptive routing goes beyond the scope of this work; the interested reader may find some results in [12, 13].

In many cases where a priori routing is mandatory, finding a single shortest path may not be satisfactory, and it becomes relevant to determine a set of alternative optimal or nearly-optimal paths. This allows e.g. to select the shortest path satisfying some additional constraints not captured by the network model. In other cases we may be interested in selecting a set of *spatially dissimilar* paths, rather than a single one; this typically happens in hazardous materials transportation, in order to equally distribute the risk among the population. A common approach (see e.g. [1]) consists in selecting dissimilar paths out of a (large) set of previously generated "attractive" paths. In the situations above, one needs effective methods for *ranking* a priori paths, that is, the STD counterpart of the classical *K shortest (loopless) paths* in directed graphs [20]. However, to the best of our knowledge, ranking of a priori paths in STD networks has never been addressed in the literature, with the exception of Nielsen *et al.* [17] where a ranking procedure was embedded in a *two phase method* for bicriterion a priori routing.

In this paper we consider a priori route choice in discrete STD networks, for a single origin, a single destination, and a given departure time. Our goals are to devise solution methods for the shortest and $K$ shortest path problems, and to evaluate their effectiveness and robustness against a set of challenging instances. In the following we describe in detail the aim and contribution of our work. First, it is worthwhile to add a couple of remarks on the model considered here. Since we address a priori route choice, we ignore all kind of on-line information, including arrival times at intermediate nodes, which implies that waiting is forbidden. Furthermore, according to the original model proposed by Hall, we assume that travel times are *independent* random variables. Models of STD networks with *correlated* travel times have been proposed, see e.g. [10] for a priori route choice; these models are much more computationally demanding than the classical one.

We devise a *best-first* branch and bound method, where subproblems correspond to subnetworks

of the STD network. Due to the best-first policy this method generates paths in non-decreasing order or cost, and thus deals quite naturally with the ranking of a priori paths. A relevant methodological contribution of our approach is that we solve time-adaptive problems as a *relaxation* of a priori problems. The fact that the former problem is a relaxation of the latter has already been pointed out [12], but has never been exploited for algorithmic purposes. Within our method, the use of the adaptive relaxation allows to skip (or at least delay) the processing of unpromising subproblems, thus avoiding the pitfalls of the previously proposed methods. We also devise a version of our method solving time-adaptive subproblems via *reoptimization*, see Nielsen, Pretolani, and Andersen [16]. This version turns out to be consistently faster.

To the best of our knowledge, our work is the first one addressing the ranking of a priori paths in STD networks, both from a methodological and a computational point of view. It is instructive to point out that previous approaches for the shortest a priori path do not seem suitable for ranking purposes. This may shed light on the motivations and the relevance of our methodological contribution. The method proposed by Hall, despite some similarities to our one, does not necessarily generate paths in nondecreasing order of travel time. Indeed, it generates paths in nondecreasing order of *length*, which is a (quite loose) lower bound on the expected travel time. As a consequence, the path generation stops only when the length of the last generated path meets the $K$-th best solution found so far, provided this happens before enumerating all the existing paths. Similar and possibly worse drawbacks affect labeling methods, in particular *label correcting* ones. In fact, we do not know how a labeling method may be extended to ranking in STD networks, at least without the addition of suitable bounds or domination rules. Note that the above drawbacks are avoided in our method, due to the best-first policy and the use of the time-adaptive relaxation.

When considering finding the shortest a priori path (i.e., the case $K = 1$) we do not make strong claims on the merits of our approach, since our setting differs substantially from those of previous proposals. Indeed, labeling methods for the discrete case are conceived for a much more general version of the problem; on the other hand, solution methods based on the continuous model are inherently approximate, even if they may offer a better trade-off between computational cost and solution quality. Based on these premises, it seems apparent that a direct computational comparison to previous algorithmic proposals would be questionable, if not arbitrary. Therefore, in our computational analysis we concentrate on the validation of our ranking methodology, which is the main focus of this paper. In fact, an appraisal of the merits and drawbacks of the many existing approaches to a priori routing in STD networks would be an interesting contribution, but this goes far beyond the scope of this paper.

To assess the quality and robustness of our methods we set up a particularly challenging experimental setting. To this aim, we concentrate on networks with a *grid* topology, and we exclude the *final steady state* adopted e.g. in [10, 12]. The combined impact of these two choices on the difficulty of the instances is discussed in detail in Section 4. For grids of different size and shape, we consider several different models for the *link behaviour*, i.e., different width and shape of the fluctuations of link travel times and costs. The most important issue in our tests is that we address minimization of *costs*, in addition to minimization of travel times usually addressed in the literature. Instances involving costs instead of travel times turn out to be computationally much more demanding; a possible explanation of this behaviour is given in Section 4. Nevertheless, our methods turn out to be reasonably stable under many different scenarios.

We remark that the restriction to grid networks fits the aims of our analysis, and cannot be considered as a limitation. In particular, our benchmark instances simulate road networks with congestion effects, and thus can be considered as a realistic representation (and most likely, a "worst case" example) of "real world" transportation networks. Benchmarks derived from road networks have been

occasionally used in the literature on STD networks, but we believe that they would be redundant in our case. Besides, adapting the available network descriptions to our setting would be rather arbitrary, since there is no clearly established methodology for assessing the link behaviour.

The paper is organized as follows. The definitions of discrete STD networks and of the related routing problems 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 we describe our test instances, and report computational results for finding the shortest and $K$ shortest a priori paths. Finally, we summarize original contributions and directions for further research in Section 5. Appendix A provides an example illustrating several concepts introduced throughout the paper.

## 2 Stochastic time-dependent networks

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 $\delta$, 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 network with node set $N$ and arc set $A$. We will refer to $G$ as the *topological network*. As usual, $\mathrm{FS}(u) = \{(u,v) \in A\}$ 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.,

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

Throughout the paper we assume $L(o) = \{0\}$, that is, it is only possible to leave the origin at time zero. 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 $\theta_{uvt}(t_i)$. We assume that travel times are positive, that is, $t_i > t$ for each $t_i \in I(u,v,t)$. We denote by

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

the total number of possible arrival times over all arcs and possible departure times. The value $\kappa$ can be considered as the space required to describe the STD network, that is, the size of the input to our problem. Note that $\kappa$ grows with the number of arcs, the length of the time horizon, and the size of the support of the random travel time variables.

Adaptive routing in the STD network is described by a *(time-adaptive) strategy*, that is, a function $s : (N \setminus \{d\} \times H) \to A$ that assigns to each node $u \neq d$ and time $t \in L(u)$ a successor arc $s(u,t) = (u,v)$ such that $t \in L(u,v)$. According to $s$, a traveller leaving a node $u$ at time $t \in L(u)$ travels along arc $s(u,t)$. From now on, we concentrate on travellers leaving the origin node $o$ at time zero.

**Definition 1** A *route* is a pair $R = (D_R, s_R)$ where $s_R$ is the restriction of some strategy $s$ over the domain $D_R$, and $D_R \subseteq (N \setminus \{d\} \times H)$ which is recursively defined as follows:

1. $(o,0) \in D_R$;
2. if $(u,t) \in D_R$, $s(u,t) = (u,v)$ and $v \neq d$ then $(v,t') \in D_R$ for each $t' \in I(u,v,t)$.

A route $R$ provides a *complete* and *minimal* set of routing choices for a traveller that leaves the origin at time zero and moves towards the destination. Indeed, $D_R$ contains a pair $(u,t)$ if and only if the traveller has a non-zero probability of leaving node $u$ at time $t$. Note that we do not allow *waiting* at intermediate nodes: a traveller arriving at node $u \neq d$ at time $t$ leaves $u$ at time $t$, along arc $s_R(u,t)$. A route allows the traveller to arrive at node $d$ within time $t_{\max}$ for every possible realization of the travel times. In other words, a route $R$ exists if and only if the STD network allows to travel from $o$ to $d$. We refer the reader to [19] for a formal discussion of these properties.

**Definition 2** Given a route $R = (D_R, s_R)$, let $G_R = (V_R, A_R)$ denote the subgraph of $G$ defined by $R$, where

$$V_R = \{u \in V : \exists (u,t) \in D_R\} \cup \{d\}, \;\; A_R = \cup_{(u,t) \in D_R} \{s_R(u,t)\}. \tag{1}$$

The network $G_R$ contains the arcs which may be used with a positive probability when following the route $R$. Note that $G_R$ may contain several $o$-$d$ paths, and is not necessarily acyclic.

**Definition 3** A route $R$ is a *path-route* if $G_R$ is a directed and loopless $o$-$d$ path.

Let $P = (o = u_1, u_2, \ldots, u_l, u_{l+1} = d)$ be a loopless $o$-$d$ path in $G$. A traveller following path $P$ adopts a *time-independent* routing choice, that is, travels along arc $(u_i, u_{i+1})$ regardless of the leaving time from node $u_i$. We say that $P$ is *feasible* if a traveller following $P$, and leaving $o$ at time zero, arrives at $d$ within time $t_{\max}$ for every possible realization of the travel times. In other words, a path $P$ is feasible if a traveller following $P$ cannot arrive at an intermediate node $u_i$ at time $t \notin L(u_i, u_{i+1})$.

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

**Proof** By definition, a path-route defines a unique feasible path. To prove the converse, consider a feasible path $P = (o = u_1, u_2, \ldots, u_l, u_{l+1} = d)$ in $G$. Let $D_P$ denote the set of pairs $(u,t)$ such that a traveller following $P$ has a non-zero probability of leaving $u$ at time $t$. The set $D_P$ can be defined recursively as follows:

1. $(o,0) \in D_P$;
2. if $(u_i,t) \in D_P$ and $u_i \neq d$ then $(u_{i+1},t') \in D_P$ for each $t' \in I(u_i, u_{i+1}, t)$.

Indeed, feasibility of $P$ implies that $t \in L(u_i, u_{i+1})$ for each pair $(u_i,t) \in D_P$. Furthermore, define the function $s_P : D_P \to A$ such that $s_P(u_i,t) = (u_i, u_{i+1})$ for each pair $(u_i,t) \in D_P$. Clearly, $R = (D_P, s_P)$ is a path-route. ■

Several optimality criteria for routing in STD networks have been considered in the literature. Let us consider the minimization of expected cost. Costs can be introduced in our STD model by letting $c(u,v,t), t \in L(u,v)$ denote the expected travel cost of leaving node $u$ at time $t$ along arc $(u,v)$. Moreover, we let $g_d(t)$ be a penalty cost of arriving at node $d$ at time $t$. The expected cost of a route

$R = (D_R, s_R)$ can be defined by means of recursive equations, associating a value $E^R(u,t)$ to each pair $(u,t) \in D_R$. In particular, if $s_R(u,t) = (u,v)$ we have:

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

where $E^R(d,t) = g_d(t)$ for each $t \in H$. The value $E^R(u,t)$ is the expected cost incurred when leaving node $u$ at time $t$, following $R$ towards $d$. The expected cost of $R$ is therefore $E^R(o,0)$, which we denote $c(R)$. Note that the minimization of expected costs includes as particular cases the minimization of the *expected travel time* and the maximization of the *reliability*, that is, the probability of arriving at $d$ within a given time. In particular, the former case can be formulated as the minimization of the expected cost by setting $g_d(t) = t$ for each $t \in H$ and $c(u,v,t) = 0$ for each $(u,v)$ and $t \in L(u,v)$. Other optimality criteria may be dealt with by our solution methods, but we only consider expected travel time and cost in our computational experience.

In light of Theorem 1, the *shortest a priori path* problem (SAP) addressed in this paper can be formulated as finding the path-route $R$ with minimum expected cost $c(R)$. This problem is well known to be NP-hard [18]. The ranking version of SAP, that is the *K-shortest a priori path* problem (K-SAP) consists in generating the first $K$ path-routes in nondecreasing order of expected cost.

Under time-adaptive route choice, the optimal routing problem can be formulated as finding the route $R$ (not necessarily a path-route) with minimum $c(R)$. This problem can be solved in linear time; more precisely, it takes $O(\kappa)$ time to find the optimal route $R$ or show that no feasible route exists. In Appendix A we present an example showing how the optimal route $R$ can be computed, both for the case of costs and of travel times.

Since path-routes are a subset of routes, the time-adaptive problem is a relaxation of SAP, referred to as the *time-adaptive relaxation* in the following. Note in particular that the cost of a path-route $R$ is the same in SAP and in the time-adaptive relaxation, a fact that will be exploited in our solution methods.

# 3 Finding and ranking path-routes in STD networks

In this section we first present our algorithm for SAP, and then describe the extension to K-SAP. Furthermore, we devise a faster variant that exploits reoptimization techniques.

In their overall structure our algorithms bear some resemblance with the classical *Yen's method* [20] for ranking loopless paths in directed graphs. There are, however, significant technical differences, as can be expected since, in STD networks, even finding the shortest path is NP-hard. In fact, our approach consists in adapting path-ranking methods in directed graphs and combining them with the solution of the time-adaptive relaxation.

## 3.1 An enumeration algorithm for SAP

Our solution algorithm for SAP is essentially a *branch and bound* method, where each sub-problem corresponds to a sub-network of the original STD network, defined by a subgraph of the topological network $G$. In particular, we adopt a *best-first* enumeration strategy, that is, we select subproblems to be processed in nondecreasing order according to a lower bound on their optimal solution value. The lower bound is obtained by solving a time-adaptive relaxation for each generated sub-problem.

Consider a generic subproblem $S$, defined by the subgraph $G_S$, and let $R$ denote the optimal route returned by the time-adaptive relaxation of $S$. Clearly, if $R$ is a path-route then it is also the optimal

solution to $S$. Suppose otherwise that $S$ is selected but $R$ is not a path-route; in this case we need to apply a branching operation. The goal of our branching rule is to partition the set of loopless *o-d* paths in $G_S$; to this aim, we adapt the branching rule defined by Yen [20] for ranking paths in directed graphs. Our partition technique is based on a *branching path* $p_R$, contained in the graph $G_R$ defined by $R$, that starts from the origin $o$ but is not necessarily an *o-d* path. From now on, let us denote by $\mathrm{FS}_R(u)$ the forward star of a node $u$ in the graph $G_R$.

**Definition 4** Given a route $R$, a *branching path* $p_R = (o = u_1, \ldots, u_l, u_{l+1})$ in $G_R$ is a path satisfying

1. $\mathrm{FS}_R(u_i) = \{(u_i, u_{i+1})\}, \forall i = 1, \ldots, l-1$.
2. $|\mathrm{FS}_R(u_l)| \geq 2$.

Note that $G_R$ always contains a branching path with $l \geq 2$, since we assume $L(o) = \{0\}$ and therefore $(o,0)$ is the only pair $(o,t)$ contained in $D_R$, that is, $\mathrm{FS}_R(o) = \{u_1, u_2\}$. Note also that we may have $u_{l+1} = d$. By definition, $G_R$ contains a unique path from $o$ to $u_l$, while several arcs may be used as the last arc in $p_R$; we do not make any assumption about the way this arc is chosen. Let $\Pi_S$ denote the set of *o-d* paths in $G_S$.

**Definition 5** Given a branching path $p_R = (o = u_1, ..., u_{l+1})$ we partition the set $\Pi_S$ into the subsets $\Pi_S^{(i)} \subset \Pi_S$, $1 \leq i \leq l+1$, as follows:

1. for $1 \leq i \leq l$, paths in $\Pi_S^{(i)}$ contain the subpath $p_R^{(i)} = (u_1, ..., u_i)$ of $p_R$ but do not contain arc $(u_i, u_{i+1})$;
2. paths in $\Pi_S^{(l+1)}$ contain the branching path $p_R$.

Note that one or more of the sets $\Pi_S^{(i)}$ in Definition 5 may be empty. The partition of $\Pi_S$ implicitly defines a family of subgraphs of $G_S$.

**Definition 6** Given a branching path $p_R = (o = u_1, ..., u_{l+1})$ the subgraph $G_S^{(i)}$, $i = 1, ..., l+1$, is obtained from $G_S$ as follows:

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

The following result is rather intuitive; a formal proof may be given by adapting the proof of correctness for Yen's algorithm.

**Lemma 1** Given a branching path $p_R = (o = u_1, ..., u_{l+1})$, for each $1 \leq i \leq l+1$ we have that $\Pi_S^{(i)}$ is the set of loopless *o-d* paths in subgraph $G_S^{(i)}$.

In light of Lemma 1, in our branching operation we select a path $p_R$ and we create $l+1$ subproblems $S^{(1)}, \ldots, S^{(l+1)}$, where each $S^{(i)}$ corresponds to the STD sub-network defined by subgraph $G_S^{(i)}$. For each subproblem $S^{(i)}$ we solve the time-adaptive relaxation, and we distinguish two possible cases. If the relaxation yields an optimal route $R^{(i)}$ then we assign to $S^{(i)}$ the lower bound $c(R^{(i)})$, and we store $S^{(i)}$ for later processing. Otherwise, we discard $S^{(i)}$ since the corresponding STD sub-network does not contain any feasible route. An example of application of our branching rule is worked out in details in Appendix A. Some relevant properties are worth pointing out.

**Step 1** *(initialization)* let $Q = \{(LB(G), G)\}$;

**Step 2** *(selection)* if $Q = \emptyset$ then STOP (no feasible path); otherwise, remove from $Q$ a pair $(lb_S, G_S)$ with minimum $lb_S$; let $R = \text{OptRoute}(G_S)$;

**Step 3** *(branching)* if $R$ is a path-route go to Step 5; otherwise, choose a branching path $p_R$ and let $F = F(G_S, p_R)$;

**Step 4** *(bounding)* for each subgraph $G_S^{(i)} \in F$ such that $\text{MinCost}(G_S^{(i)}) < +\infty$ insert into $Q$ the pair $(\text{MinCost}(G_S^{(i)}), G_S^{(i)})$; go to Step 2;

**Step 5** *(termination)* OUTPUT $R$ and STOP.

Figure 1: The branch and bound algorithm for SAP.

**Property 1** Given a branching path $p_R = (o = u_1, ..., u_{l+1})$, the following statements hold true:

1. none of the subgraphs $G_S^{(i)}$, $1 \le i \le l+1$, contains all the arcs in $G_R$, and thus, all the arcs in $G_S$;
2. if $\Pi_S^{(i)} = \emptyset$ then the subproblem $S^{(i)}$ is discarded;
3. none of the subproblems $S^{(i)}$, $1 \le i \le l+1$, admits $R$ as a feasible route.

A summary of our branch and bound method is given in Figure 1. We use a priority queue $Q$ to store subproblems to be processed; actually $Q$ stores pairs $(lb', G')$, where $lb'$ is the lower bound assigned to the STD subnetwork defined by subgraph $G'$. We denote by $F(G_S, p_R)$ the family of subgraphs in Definition 6. We also introduce two functions, namely $\text{MinCost}(G')$ and $\text{OptRoute}(G')$, that solve a time-adaptive relaxation in the STD network defined by graph $G'$. Function $\text{MinCost}(G')$ returns the minimum expected route cost, or $+\infty$ if no feasible route exists; function $\text{OptRoute}(G')$ returns the optimum route, assuming that one exists, that is $\text{MinCost}(G') < +\infty$. Note that the algorithm terminates as soon as a path-route $R = \text{OptRoute}(G')$ is found in Step 2. For the sake of simplicity, we assume $\text{MinCost}(G) < +\infty$, that is, that the original STD network contains at least one feasible adaptive route.

**Theorem 2** The algorithm given in Figure 1 correctly solves the SAP problem in a finite number of iterations.

**Proof** Finiteness follows from Property 1. Correctness follows from Lemma 1 and from the termination rule in Step 3; indeed, if $R = \text{OptRoute}(G_S)$ is a path-route we have $lb_S = c(R)$, thus no other path-route may have a cost less than $c(R)$. ■

The worst case complexity is exponential, since it may be necessary to enumerate all the subgraphs of $G$. Note, however, that the algorithm takes linear time $O(\kappa)$ for each subproblem inserted into $Q$. We claim (omitting proof) that a smart implementation takes time $O(|\Pi|\kappa)$, where $\Pi$ is the set of loopless *o-d* paths in $G$. This implies a polynomial complexity in the (rather unlikely) case where $\Pi$ is polynomial in the input size $\kappa$. Concerning the space complexity a (small) constant amount of information needs to be stored for each subproblem inserted into $Q$. The interested reader may consult [13] for a description of the data structures and the implementation details.

### 3.2 An enumerative method for K-SAP

As discussed above, our algorithm for SAP has several similarities with Yen's method for ranking paths in directed graphs. Thus the extension to K-SAP is rather straightforward: instead of stopping the enumeration as soon as a path-route is found, we apply Yen's branching rule to the corresponding *o-d* path, and continue until $K$ path-routes are obtained.

Let us briefly discuss the meaning of Yen's branching rule when applied in our context. Consider a subproblem $S$ defined by graph $G_S$, assume that $R = \text{OptRoute}(G_S)$ is a path-route, and let $G_R$ correspond to the loopless path $p = (o = u_1, ..., u_{q+1} = d)$. We partition the set $\Pi_S \setminus \{p\}$ into $q$ subsets $\Pi^{(i)}$, $1 \leq i \leq q$. Accordingly, we generate $q$ proper subgraphs $G_S^{(i)}$ of $G_S$, for $1 \leq i \leq q$, such that $\Pi^{(i)}$ is the set of *o-d* paths in $G_S^{(i)}$. For completeness, recall that we obtain the subgraph $G_S^{(i)}$ from $G_S$ by removing the arc $(u_i, u_{i+1})$ and by fixing arc $(u_j, u_{j+1})$ for each $j = 1, ..., i-1$. We then create a subproblem $S^{(i)}$ for each $G_S^{(i)}$, and we proceed as in the SAP algorithm.

Let us denote by $F(G_S, R)$ the family of subgraphs defined by Yen's rule for a subproblem $S$ with shortest path-route $R$. The algorithm for K-SAP is obtained from the one in Figure 1 replacing Step 5 with the following step, where we assume that the counter $k$ is initialized to zero at the outset:

**Step 5'** *(Yen's rule)* OUTPUT $R$; let $k = k+1$; if $k = K$ then STOP, otherwise let $F = F(G_S, R)$ and go to Step 4.

The correctness of the K-SAP algorithm then follows from the correctness of Yen's method and from Theorem 2.

### 3.3 A faster method based on reoptimization

A clear drawback of our solution methods for SAP and K-SAP is that, in Step 4, the time-adaptive relaxation $\text{MinCost}(G_S^{(i)})$ is solved for each generated subproblem $S^{(i)}$, including discarded ones. This is quite expensive in terms of computation times. A possible alternative is to compute a lower bound $\text{LB}(G_S^{(i)})$ on the minimal route cost, instead of $\text{MinCost}(G_S^{(i)})$. In this way, the time-adaptive relaxation is solved only when (and if) the subproblem is selected from $Q$ in Step 2. As a consequence, however, subproblems are not necessarily selected in the desired order. Moreover, a subproblem $S^{(i)}$ such that $\text{MinCost}(G_S^{(i)}) = +\infty$ may fail to be discarded, and enter $Q$ with a finite lower bound $\text{LB}(G_S^{(i)})$. Therefore, a more complex treatment of the selection phase is necessary.

Consider the pair $(lb_S, G_S)$ removed from $Q$ in Step 2, and let $lb_{min} = \min\{l : (l, g) \in Q\}$ denote the minimum lower bound among the pairs remaining in $Q$; we have $lb_{min} = +\infty$ if $Q$ is empty. Solving the time-adaptive relaxation, three mutually exclusive cases may arise.

1. $\text{MinCost}(G_S) = +\infty$: in this case, we discard $(lb_S, G_S)$ and repeat Step 2;
2. $lb_{min} < \text{MinCost}(G_S) < +\infty$: in this case, we discard $(lb_S, G_S)$, but we insert into $Q$ the pair $(\text{MinCost}(G_S), G_S)$, and repeat Step 2;
3. $\text{MinCost}(G_S) < +\infty$ and $\text{MinCost}(G_S) \leq lb_{min}$: we let $R = \text{OptRoute}(G_S)$ and go to Step 3.

Note that in case 2 the subproblem is reinserted in $Q$ with the lower bound $\text{MinCost}(G_S)$; thus a subproblem cannot be reinserted more than once. Moreover, in case 3 the selection of $(lb_S, G_S)$ is correct even if we have $\text{LB}(G_S) < \text{MinCost}(G_S)$. Since the new algorithm only differs slightly from the algorithm in Figure 1, we omit a pseudo-code description here.

A fast and tight lower bound on the expected cost of the optimal route can be computed by exploiting the *reoptimization techniques* proposed by Nielsen, Andersen, and Pretolani [15], Nielsen et al. [16], in the context of *K shortest hyperpaths* procedures. The results in [15, 16] are technically rather involved, and are not described here. Reoptimization techniques cannot be deployed in full strength in our context, since they are devised for a different branching rule, and for searching in a different solution space. Nevertheless, they turn out to be quite useful in practice, as we shall see in the next section.

# 4 Computational results

In this section we report the results of the computational experience with our methods for a priori paths. In our tests we only consider the ranking version of the algorithm; however, we also address problem SAP, since we report statistics about the generation of the first (and thus shortest) path. Two optimization criteria are tested, namely, the minimization of expected travel time (referred to as *MET*) and expected cost (*MEC*).

We address the versions of the ranking algorithm with and without reoptimization, respectively denoted by *K-SAPreopt* and *K-SAP*. Both versions have been implemented in C++ and tested on a 2.9 GHz Intel Core i7 laptop with 16 GB RAM using a Windows 7 operating system (Enterprise 64-bit SP1). The programs have been compiled with the GNU C++ compiler (mingw32-g++) with optimize option -O2.

## 4.1 Test classes

In all our tests, the underlying topological network $G$ is assumed to be a grid of *base* $b$ and *height* $h$, with origin $o$ in the bottom-right corner node and destination $d$ in the upper-left corner node. The choice of grids aims at obtaining challenging benchmarks. Indeed, in a $b \times h$ grid the *topological length* (i.e., number of arcs) of an *o-d* path is at least $b+h-2$; moreover, the number of these topologically shorter paths grows exponentially with $b$ and $h$. A first consequence is that the solution space to be explored is quite dense: even recognizing that the $K$ shortest paths are likely to be topologically short, the number of candidate paths remains large and grows quickly with the grid dimensions. More important, in an STD network, longer paths imply more intermediate nodes and arrival times spread in a wider interval, and this in turns imply more opportunities for an optimal adaptive route to divert from an a priori path; as we shall see, the adaptive behaviour of routes makes SAP and K-SAP harder. A further effect is due to the absence of a final steady state, which requires every feasible route to terminate at $d$ within the time horizon $H = \{0, 1, ..., t_{\max}\}$. In order to grant feasibility of the routes of interest for K-SAP, $t_{\max}$ cannot be chosen arbitrarily, but must increase with the grid dimensions. In particular, in our instances $t_{\max}$ grows with the topological length of paths, i.e., roughly linearly in $b+h$. Now suppose that both $b$ and $h$ are increased by a multiplicative factor $f$, leaving the average value $\kappa(u,v,t)$ unchanged; since the number of arcs is linear in $b \times h$, the input size $\kappa$ increases by a factor roughly $f^3$.

All tests are performed on STD networks generated with the *TEGP* (Time-Expanded Generator with Peaks) generator [14]. TEGP includes several features inspired by typical aspects of road networks, in particular for the simulation of congestion effects. We consider *cyclic time periods* (cycles) of 144 time instances (e.g. 12 minutes with a time step $\delta$ of 5 seconds) where each cycle contain one or more *peaks*. Each peak consists of three parts: a *transient* part of length $t_{\texttt{trans}}$ where the mean travel time (congestion) increases, a *pure peak* part of length $t_{\texttt{pure}}$ where it stays the same, and a tran-

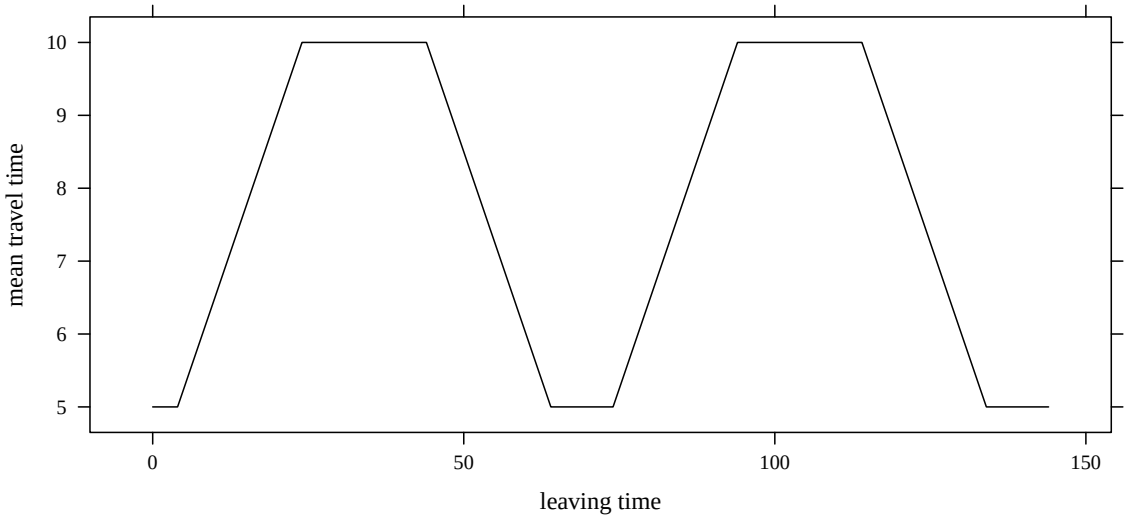



Figure 2: Peak effect on mean travel time and cost.

sient part of length $t_{\texttt{trans}}$ where it decreases again. This feature gives travel time distributions with higher mean and higher standard deviation in peaks. The pattern of the *mean travel time* $\mu(u,v,t)$ on a grid arc $(u,v)$, when two peaks are considered, is shown in Figure 2. The mean travel time $\mu(u,v,t)$ equals the *off-peak mean travel time* $\mu_{uv}$ until it reaches the transient part of the first peak, where it increases up to $(1+\psi)\mu_{uv}$, where $\psi$ denotes the *peak increase factor*. Next the mean travel time stays the same during the pure peak part and then decreases again to $\mu_{uv}$ where it stays the same until the second peak is reached.

The off-peak mean travel time $\mu_{uv}$ is generated randomly, for each grid arc $(u,v)$, in the interval $[2,6]$. We somehow take for granted that a wider interval of values for $\mu_{uv}$ results in more difficult instances, and we do not explore this possibility in our test. Given $\mu(u,v,t)$, possible travel times are the integers in the interval $[\lfloor 0.75\mu(u,v,t)\rfloor, \lceil 1.25\mu(u,v,t)\rceil]$; probabilities are calculated using a binomial distribution. In most cases, we shall assume a peak increase factor of $\psi = 1$ (100%), i.e., the mean travel time can increase up to 12, and the maximum travel time can be at most 15.

Under the MEC criterion *peak dependent costs* are generated. That is, the travel cost $c(u,v,t)$ follows the same pattern as the mean travel time in Figure 2. Off-peak costs are generated in the interval $[1,1000]$; this gives travel costs in the interval $[0,2000]$ during peaks. Note that off-peak costs and travel times are generated independently, and are not correlated to each other. The penalty costs $g_d(t)$ are set to zero for each time $t$.

In our instances the time horizon length $t_{\max}$ is computed as an upper bound on the expected travel time along a topologically short *o-d* path; the computation takes into account all the features described above, see [14] for details. As long as MET is considered, the resulting $t_{\max}$ is large enough to grant feasibility of every route of interest for K-SAP. This is no longer guaranteed if MEC is considered: since travel times and costs are not correlated, a cheap path may need a long travel time. Nevertheless, we decided to keep the same $t_{\max}$ for both MET and MEC in our tests, to avoid an unreasonable growth of the problem size. Note that a time horizon of the desired length $t_{\max}$ is obtained considering a sequence of successive cyclic periods, where possibly the last one is truncated to a time instant

smaller than 144.

In the following, we use the term *test class* to define a particular setting of the TEGP input parameters; for each class, different STD networks (i.e. different instances of each class) can be generated by choosing different seeds. In our tests, ten instances were generated for each class.

## 4.2 Aims and statistics

The aim of our computational experience is twofold. On the one hand, we try to evaluate the performances of our procedures, pointing out the relevance of reoptimization techniques. On the other hand, we try to explain the behaviour of our algorithms on the basis of the optimization criterion considered, the structure of the instances and of the solution sets. The reported statistics can be divided into different groups. The first group considers the performance of the procedures; the abbreviations used in the tables are given in parentheses.

*CPU time* ($\texttt{cpu}_K$): CPU time in seconds for finding the $K$ shortest paths. Does not include input/output time.

*First CPU time* ($\texttt{cpu}_1$): CPU time in seconds for finding the shortest path.

*Number of iterations* ($\texttt{ite}_K$): The number of subproblems selected from the candidate set $Q$ before finding the $K$'th shortest path, that is, the overall number of iterations of the ranking algorithm.

*Number of iterations* ($\texttt{ite}_1$): The number of iterations before finding the shortest path.

*Average number of subproblems created* ($\texttt{ave}_{\texttt{BT}}$): Average number of subproblems inserted into the candidate set $Q$ when branch, i.e. the average number of nodes added to the branching tree.

*Number of reinsertions* (`reins`): For *K-SAPreopt*, the number of pairs reinserted into the candidate set $Q$ after computing the actual shortest route (see Section 3.3). Reported in percent of the number of iterations $ite_K$.

Note that the total number of subproblems inserted in the branching tree is given by $\texttt{ite}_K \cdot \texttt{ave}_{\texttt{BT}}$. The second group of statistics is related to the cost (travel time for MET and travel cost for MEC) of the solutions.

*Relative increase in expected cost* ($\texttt{inc}_{1-K}$): The cost increase between the shortest and the $K$'th shortest path. Reported in percent.

*Relative increase route to shortest path* ($\texttt{inc}_{R-P}$): The cost increase between the optimal time-adaptive route and the shortest path. Reported in percent.

The last group contains one single statistic, related to the structure of the optimal route $R$ returned by the time-adaptive relaxation.

*Average number of successor arcs* ($\texttt{ave}_{\texttt{FS}}$): The average number of arcs in the forward star of the nodes in $G_R$, omitting the destination $d$.

| | | | | *K-SAP* | | | | | | *K-SAPreopt* | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Class | Size | $\|H\|$ | $\kappa$ | $\texttt{inc}_{1-K}$ | $\texttt{inc}_{R-P}$ | $\texttt{ite}_1$ | $\texttt{cpu}_1$ | $\texttt{ite}_K$ | $\texttt{cpu}_K$ | $\texttt{ite}_1$ | $\texttt{cpu}_1$ | $\texttt{ite}_K$ | $\texttt{cpu}_K$ | reins | $\texttt{ave}_{\texttt{FS}}$ | $\texttt{ave}_{\texttt{BT}}$ |
| 1 | $10\times10$ | 171 | 161608 | 5.86 | 0.00 | 4 | 0.02 | 122 | 0.62 | 3 | 0.00 | 123 | 0.15 | 1.22 | 1.05 | 7 |
| 2 | $20\times20$ | 327 | 1161842 | 2.13 | 0.01 | 2 | 0.28 | 154 | 10.66 | 2 | 0.04 | 154 | 1.94 | 0.00 | 1.03 | 10 |
| 3 | $30\times30$ | 482 | 3807538 | 1.18 | 0.01 | 5 | 1.73 | 181 | 54.92 | 5 | 0.23 | 181 | 7.40 | 0.00 | 1.03 | 14 |
| 4 | $40\times40$ | 638 | 8943761 | 0.50 | 0.01 | 16 | 10.86 | 228 | 196.90 | 16 | 1.64 | 228 | 22.68 | 0.00 | 1.03 | 17 |
| 5 | $6\times18$ | 202 | 166037 | 7.50 | 0.01 | 2 | 0.02 | 132 | 0.87 | 2 | 0.00 | 133 | 0.20 | 1.13 | 1.05 | 8 |
| 6 | $12\times36$ | 389 | 1416937 | 1.49 | 0.00 | 4 | 0.53 | 156 | 17.45 | 4 | 0.07 | 156 | 2.44 | 0.00 | 1.03 | 14 |
| 7 | $18\times54$ | 576 | 4680921 | 0.72 | 0.02 | 6 | 3.38 | 178 | 78.71 | 6 | 0.36 | 178 | 10.07 | 0.00 | 1.02 | 16 |
| 8 | $24\times72$ | 762 | 11027383 | 0.33 | 0.03 | 17 | 18.32 | 235 | 253.24 | 17 | 2.28 | 235 | 31.06 | 0.00 | 1.02 | 14 |

Table 1: Results for finding the $K = 100$ shortest paths under the MET criterion (2 peaks/cycle, $t_{\texttt{trans}} = t_{\texttt{pure}} = 20$ and $\psi = 100\%$).

Recall that $R$ is a path-route if and only if $G_R$ defines an *o-d* loopless path, that is, $\texttt{ave}_{\texttt{FS}} = 1$. Therefore, the lower the value of $\texttt{ave}_{\texttt{FS}} \geq 1$, the more "path-like" – i.e., the less "adaptive" – is the route $R$. Since our algorithms are based on the time-adaptive relaxation, their performance is likely to depend on how adaptive the optimal routes found are. If there are many routes $R$ where $G_R$ does not resemble a path we may have to branch many times before a path is found. On the contrary if the routes are very "path-like", i.e. $G_R$ is close to a path, the size of the branching tree will be smaller and ranking paths will be faster.

### 4.3 Results

In Table 1 we report the results of our algorithms for the MET criterion, and for grid graphs of increasing size. We consider square grids and rectangular grids with a ratio $h/b = 3$. Note that rectangular grids have slightly more nodes and sligtly longer topologically shorter paths than the corresponding square grids. We assume two peaks for each cyclic period, with $t_{\texttt{trans}} = t_{\texttt{pure}} = 20$, and with a peak increase factor $\psi = 100\%$. The values $\texttt{ave}_{\texttt{FS}}$ and $\texttt{ave}_{\texttt{BT}}$ are reported only once, since the difference in the values computed for *K-SAP* and *K-SAPreopt* is negligible.

Observe that square and rectangular grids give quite similar results, the latter being slightly more difficult than the former, as expected due to the higher number of nodes and minimal path length $b+h-2$. Algorithm *K-SAPreopt* turns out to be much faster than *K-SAP*; CPU times are reduced up to 88% (class 4) and on average 84%. This is clearly due to the fact that the reoptimization technique is very effective for these classes; indeed, the reoptimization bound is very tight, since there are no reinsertions except for the smallest grid dimensions. In addition, the average number of insertions per iteration ($\texttt{ave}_{\texttt{BT}}$) is relatively high, which implies that many shortest route computations are saved using reoptimization. Note that the reduction of CPU times obtained by *K-SAPreopt* is essentially the same (up to small fluctuations) for SAP and K-SAP, and for square and rectangular grids. In light of the above results, we shall only report the results of algorithm *K-SAPreopt* in the forthcoming tables.

We remark that both our algorithms show a stable behaviour for increasing grid dimensions. In particular, both the number of iterations ($\texttt{ite}_K$) and the number of insertions ($\texttt{ave}_{\texttt{BT}}$) grow *less than linearly* with the problem size $\kappa$. Recall that algorithm *K-SAP* computes an optimal route

| class | Size | $\|H\|$ | $\kappa$ | Peak | $\texttt{ite}_1$ | $\texttt{cpu}_1$ | $\texttt{ite}_K$ | $\texttt{cpu}_K$ | $\texttt{ave}_{\texttt{FS}}$ | $\texttt{ave}_{\texttt{BT}}$ | $\texttt{inc}_{1-K}$ | $\texttt{inc}_{R-P}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | $25\times25$ | 404 | 2092203 | {1:40,40;100} | 5 | 0.13 | 180 | 4.20 | 1.03 | 9.43 | 1.81 | 0.00 |
| 10 | $25\times25$ | 404 | 2202345 | {2:20,20;100} | 5 | 0.13 | 188 | 4.48 | 1.04 | 9.41 | 1.25 | 0.02 |
| 11 | $25\times25$ | 404 | 2234998 | {4:10,10;100} | 5 | 0.14 | 200 | 4.70 | 1.04 | 8.84 | 1.52 | 0.10 |
| 12 | $25\times25$ | 404 | 2127711 | {8:5,5;100} | 54 | 1.24 | 500 | 11.33 | 1.08 | 5.93 | 1.48 | 0.73 |
| 13 | $25\times25$ | 260 | 1030336 | {2:20,20;0} | 1 | 0.02 | 100 | 1.42 | 1.00 | 14.62 | 1.30 | 0.00 |
| 14 | $25\times25$ | 332 | 1596607 | {2:20,20;50} | 5 | 0.10 | 182 | 3.38 | 1.03 | 9.23 | 1.50 | 0.01 |
| 15 | $25\times25$ | 404 | 2202345 | {2:20,20;100} | 5 | 0.12 | 188 | 4.39 | 1.04 | 9.41 | 1.25 | 0.02 |
| 16 | $25\times25$ | 549 | 4097464 | {2:20,20;200} | 6 | 0.25 | 197 | 7.16 | 1.04 | 8.85 | 1.89 | 0.04 |
| 17 | $25\times25$ | 459 | 2631558 | {2:5,50;100} | 14 | 0.37 | 297 | 7.87 | 1.05 | 7.02 | 1.20 | 0.15 |
| 18 | $25\times25$ | 441 | 2563420 | {2:10,40;100} | 8 | 0.21 | 203 | 5.29 | 1.04 | 9.36 | 0.92 | 0.06 |
| 19 | $25\times25$ | 404 | 2202345 | {2:20,20;100} | 5 | 0.13 | 188 | 4.38 | 1.04 | 9.41 | 1.25 | 0.02 |
| 20 | $25\times25$ | 368 | 1966910 | {2:30,0;100} | 7 | 0.16 | 182 | 3.86 | 1.03 | 9.25 | 1.71 | 0.02 |

Table 2: Results for finding the $K = 100$ shortest paths under the MET criterion: varying peak effects. Column "Peak" describes peaks in the format {peaks/cycle: $t_{\texttt{trans}}, t_{\texttt{pure}}; \psi$}.

for each created subproblem, thus its execution time is expected to grow linearly in $\texttt{ite}_K \cdot \texttt{ave}_{\texttt{BT}} \cdot\kappa$; for *K-SAPreopt* the growth factor decreases to $\texttt{ite}_K \cdot\kappa$. Overall, the CPU times ($\texttt{cpu}_K$) grow more than linearly, but less than quadratically in the problem size $\kappa$; a rough estimation could be $\texttt{ite}_K \approx O(\kappa \cdot \log(\kappa))$. Similar observations can be made if we consider SAP (i.e., $\texttt{ite}_1$ and $\texttt{cpu}_1$) instead of K-SAP. Note that the solution space explored by our algorithms becomes more and more dense for increasing grid dimensions, as shown by the the decreasing values of $\texttt{inc}_{1-K}$ (from 7.5% to 0.33% for rectangular grids) that denote an increasing number of paths with expected travel time close to the shortest path.

The good behaviour of our methods can be related to the low degree of adaptivity of routes. Indeed, the value $\texttt{ave}_{\texttt{FS}}$ is always quite close to 1, and is lower for higher grid dimensions; therefore, the optimal routes found are in general rather path-like. This is confirmed if we consider the increase in travel time between the best route and the shortest path, which is close to zero (at most 0.03%) in all cases.

The results in Table 1 show that, at least for the settings adopted in Classes 1–8, paths are competitive with adaptive routes. However, different congestion effects (simulated by the peak feature of TEGP) may lead to a different behaviour. Some results for different peak settings are given in Table 2 (class 9-20) for a fixed grid size of $25 \times 25$. The column "Peak" provides a complete description of the peak setting, i.e., the number of peaks per cycle, the transient length $t_{\texttt{trans}}$, the pure peak length $t_{\texttt{pure}}$, and the increase factor $\psi$, reported in percent.

In the first block (classes 9–12) we consider the number of peaks per cycle, increasing from one to eight; clearly, the peak length decreases proportionally. Note that $t_{\max}$ does not change, but the number of complete peaks arising throughout the time horizon $H$ increases from 2 (class 9) to 22 (class 12). Increasing the number of peaks makes travel time fluctuate at a higher frequence. As can be expected, this leads to a more adaptive behaviour, as shown by the value $\texttt{ave}_{\texttt{FS}}$ and also by the branching path length $\texttt{ave}_{\texttt{BT}}$. Interestingly, the performance of our algorithms does not seem to be

| class | Size | $\|H\|$ | $\kappa$ | $\texttt{ite}_1$ | $\texttt{cpu}_1$ | $\texttt{ite}_K$ | $\texttt{cpu}_K$ | $\texttt{ave}_{\texttt{FS}}$ | $\texttt{ave}_{\texttt{BT}}$ | $\texttt{inc}_{1-K}$ | $\texttt{inc}_{R-P}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 21 | $10\times10$ | 171 | 161825 | 3 | 0.00 | 178 | 0.20 | 1.07 | 5.58 | 6.45 | 0.12 |
| 22 | $20\times20$ | 327 | 1216683 | 10 | 0.12 | 383 | 4.67 | 1.07 | 5.35 | 2.36 | 0.10 |
| 23 | $30\times30$ | 482 | 4039118 | 37 | 1.53 | 649 | 26.39 | 1.09 | 5.79 | 1.11 | 0.17 |
| 24 | $40\times40$ | 638 | 9492826 | 145 | 14.80 | 2257 | 227.75 | 1.12 | 4.44 | 0.57 | 0.35 |
| 25 | $6\times18$ | 202 | 197175 | 3 | 0.01 | 194 | 0.31 | 1.06 | 5.28 | 6.77 | 0.04 |
| 26 | $12\times36$ | 389 | 1618354 | 10 | 0.18 | 475 | 8.10 | 1.07 | 5.91 | 1.80 | 0.14 |
| 27 | $18\times54$ | 576 | 5418905 | 49 | 2.89 | 672 | 39.52 | 1.08 | 7.01 | 0.85 | 0.22 |
| 28 | $24\times72$ | 762 | 12836186 | 88 | 11.84 | 1209 | 160.78 | 1.09 | 5.27 | 0.62 | 0.38 |

Table 3: Results for finding the $K = 100$ shortest paths under the MET criterion; no peak-effect on vertical grid arcs.

significantly affected, except for class 12; also in this case, however, $\texttt{ite}_K$ and $\texttt{cpu}_K$ increase less than a factor 3. On the other hand, finding the shortest path becomes much harder (a factor around 10) for class 12; this can be related to the fact that paths are less competitive w.r.t. adaptive routes, as shown by $\texttt{inc}_{R-P}$.

In classes 13–16 we increase the factor $\psi$ from 0% to 200%, wih two peaks per cycle . Clearly, this gives a longer time horizon and therefore a greater problem size. Increasing $\psi$ has a lesser impact on the adaptivity of routes, but the behaviour of our methods remains quite similar to the one observed for classes 1–8. Note that class 13 actually does not have any peak effect since $\psi = 0$. In this case we have time-independent travel times and we do not find any optimal route with time-adaptive choices, thus obtaining $\texttt{ite}_K = K$.

Finally, consider class 17-20 where various peak shapes are tested, given a fixed total peak length, and again with two peaks per cycle. Here a shorter pure peak length (i.e., a longer transient peak length) leads to a less adaptive behaviour; this can be expected, since the pure peak period has a stronger impact on travel times. Due to this effect, and to the reduction of the input size, the performance of our algorithms improves for shorter pure peaks.

In Table 3 we simulate yet another setting of the peak feature, where the congestion effect (similar e.g. to the "step networks" in [10]) affects only a subset of the links. More precisely, classes 21–28 are equivalent to classes 1–8 except that the peak effect is applied only to "horizontal" grid arcs (leading east or west) while "vertical" arcs (leading north and south) have time-independent off-peak travel times. As a result, it may often be preferable to travel vertically to avoid a "long" horizontal arc, in particular during peaks. Therefore, a more adaptive behaviour can be expected, and this is confirmed by comparing the results in Table 3 to the ones in Table 1. In fact, the value $\texttt{ave}_{\texttt{FS}}$ increases only slightly (less than 10%) but the more adaptive behaviour is confirmed by the shorter length of the branching tree ($\texttt{ave}_{\texttt{BT}}$) and by the relative increase $\texttt{inc}_{R-P}$. As a consequence, *K-SAPreopt* obtains a worse performance, both in terms of iterations and CPU times; in the worst case (class 24) $\texttt{cpu}_K$ increases by an order of magnitude w.r.t. the corresponding class 4. Nevertheless, it must be remarked that CPU times grow more or less quadratically in the input size. Note that the impact on the performance is less relevant for rectangular grids where (since $h = 3b$) we mostly travel north (recall we travel from the bottom-right to the upper-left corner) and thus we need less links with congestion effects.

| class | Size | $\|H\|$ | $\kappa$ | $\texttt{ite}_1$ | $\texttt{cpu}_1$ | $\texttt{ite}_K$ | $\texttt{cpu}_K$ | $\texttt{ave}_{\texttt{FS}}$ | $\texttt{ave}_{\texttt{BT}}$ | $\texttt{inc}_{1-K}$ | $\texttt{inc}_{R-P}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 29 | $5\times 5$ | 93 | 22708 | 8 | 0.00 | 255 | 0.02 | 1.16 | 3.14 | 77.93 | 16.64 |
| 30 | $10\times 10$ | 171 | 161608 | 54 | 0.06 | 823 | 0.84 | 1.17 | 3.67 | 14.56 | 13.77 |
| 31 | $15\times 15$ | 249 | 486917 | 148 | 0.64 | 1596 | 7.01 | 1.15 | 4.07 | 7.31 | 8.76 |
| 32 | $20\times 20$ | 327 | 1161842 | 1187 | 13.48 | 9642 | 109.87 | 1.20 | 4.11 | 4.54 | 11.83 |
| 33 | $3\times 9$ | 109 | 23649 | 7 | 0.00 | 260 | 0.03 | 1.11 | 3.46 | 46.07 | 14.02 |
| 34 | $6\times 18$ | 202 | 166037 | 275 | 0.37 | 2435 | 3.18 | 1.11 | 4.09 | 12.24 | 14.34 |
| 35 | $9\times 27$ | 296 | 618618 | 1991 | 12.51 | 16499 | 106.36 | 1.22 | 3.89 | 5.41 | 9.73 |
| 36 | $12\times 36$ | 389 | 1416937 | 8973 | 130.08 | 71919 | 1048.35 | 1.21 | 4.01 | 3.81 | 9.59 |

Table 4: Results for finding the $K = 100$ shortest paths under the MEC criterion (2 peaks/cycle, $t_{\texttt{trans}} = t_{\texttt{pure}} = 20$ and $\psi = 100$).

Under the MEC criterion the problem turns out to be much harder than under MET. Results for increasing grid size are given in Table 4; we omitted larger grids due to the high CPU times and added grids of intermediate size. The peak settings are the same in Table 4 and Table 1; in particular, as long as travel times are considered, the STD networks in classes 30, 32, 34 and 36 are the same as in classes 1, 2, 5 and 6, respectively.

As clearly shown by the iteration counters, algorithm *K-SAPreopt* does not show the same stable behaviour as observed for MET. In particular, rectangular grids turn out to be much more challenging than square grids of comparable dimensions; recall that the difference was rather limited for MET. For example, CPU times for classes 32 and 36 differ by one order of magnitude; note that the minimum topological length is $b+h-2 = 38$ for class 32 and 46 for class 36. This confirms that the algorithmic performance is strongly affected even by small increase in the problem dimensions, which can be expected since SAP is a strongly NP-hard problem.

The higher difficulty of MEC can be related to the greater degree of adaptivity shown by the optimal routes, which is clearly shown by $\texttt{ave}_{\texttt{FS}}$, $\texttt{ave}_{\texttt{BT}}$, and $\texttt{inc}_{R-P}$. Also for MEC the solution space becomes denser for increasing grid size, but the values $\texttt{inc}_{1-K}$ are about twice the ones observed for MET: this fact, together with the higher values of $\texttt{inc}_{R-P}$, confirm that adaptive routes are more competitive than paths, opposed to what observed for MET.

The hardness of MEC can be explained considering the cost structure. Recall that MET is obtained by setting $c(u,v,t) = 0$ for each $(u,v) \in A$ and $t \in L(u,v)$, and thus MET can be considered as a quite particular case of MEC. Due to the cost structure, the peak effect is likely to affect the cost of a path more than it affects its travel time; in other words, the cost of a path (or subpath) may have wide fluctuations depending on the leaving time. This in turn may explain the higher adaptive behaviour observed for MEC.

Finally, results for different peak settings under MEC are given in Table 5; here the settings are the same as in Table 2, except that a smaller $(15\times 15)$ grid is used. The behaviour observed for MEC is essentially the same as pointed out for MET, but with a couple of significant differences. On one side, increasing the number of peaks (classes 37-40) has a rather unpredictable impact on the performance. On the other side, increasing the peak factor $\psi$ (classes 41-44) has a much more impressive impact compared to what observed for MET.

| class | Size | $\|H\|$ | $\kappa$ | Peak | $ite_1$ | $cpu_1$ | $ite_K$ | $cpu_K$ | $ave_{FS}$ | $ave_{BT}$ | $inc_{1-K}$ | $inc_{R-P}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 37 | $15 \times 15$ | 249 | 481982 | {1:40,40;100} | 29 | 0.14 | 422 | 2.02 | 1.08 | 4.53 | 6.94 | 4.14 |
| 38 | $15 \times 15$ | 249 | 486917 | {2:20,20;100} | 148 | 0.68 | 1596 | 7.44 | 1.15 | 4.07 | 7.31 | 8.76 |
| 39 | $15 \times 15$ | 249 | 483031 | {4:10,10;100} | 144 | 0.67 | 2274 | 10.79 | 1.23 | 3.62 | 8.26 | 8.12 |
| 40 | $15 \times 15$ | 249 | 468533 | {8:5,5;100} | 123 | 0.56 | 1883 | 8.56 | 1.28 | 3.39 | 8.14 | 5.69 |
| 41 | $15 \times 15$ | 160 | 226431 | {2:20,20;0} | 8 | 0.02 | 445 | 1.11 | 1.10 | 8.17 | 8.65 | 0.77 |
| 42 | $15 \times 15$ | 204 | 345646 | {2:20,20;50} | 63 | 0.22 | 1439 | 4.99 | 1.12 | 4.53 | 8.70 | 3.69 |
| 43 | $15 \times 15$ | 249 | 486917 | {2:20,20;100} | 148 | 0.67 | 1596 | 7.31 | 1.15 | 4.07 | 7.31 | 8.76 |
| 44 | $15 \times 15$ | 338 | 885470 | {2:20,20;200} | 1406 | 9.80 | 7891 | 55.24 | 1.19 | 3.55 | 7.37 | 18.79 |
| 45 | $15 \times 15$ | 282 | 542668 | {2:5,50;100} | 560 | 2.80 | 4813 | 24.06 | 1.19 | 3.67 | 7.67 | 9.35 |
| 46 | $15 \times 15$ | 271 | 520713 | {2:10,40;100} | 574 | 2.75 | 4491 | 21.55 | 1.16 | 3.89 | 7.06 | 10.83 |
| 47 | $15 \times 15$ | 249 | 486917 | {2:20,20;100} | 148 | 0.65 | 1596 | 7.13 | 1.15 | 4.07 | 7.31 | 8.76 |
| 48 | $15 \times 15$ | 227 | 436405 | {2:30,0;100} | 33 | 0.14 | 857 | 3.49 | 1.16 | 4.17 | 7.51 | 4.35 |

Table 5: Results for finding the $K = 100$ shortest paths under MEC under various peak effects.

# 5 Conclusions

In this paper we devised a solution method for the a priori shortest path problem in discrete STD networks, and extended this method to finding the $K$ shortest a priori paths. We also devised a faster version exploiting reoptimization techniques to compute fast lower bounds. We evaluated the effectiveness and robustness of our algorithms against a set of hard instances, and we pointed the impact of the problem structure on the performance of our algorithms. To the best of our knowledge, our paper is the first one addressing the $K$ shortest a priori path problem in STD networks.

From a computational point of view, the reported results are quite encouraging. For the minimization of travel times (usually addressed in the literature on STD networks) our algorithms can be expected to be effective for instances arising from (reasonable approximations of) real networks, and rather robust when faced with larger or harder instances. We also addressed the minimization of travel costs, which turned out to be much harder than the problem involving travel times. Note however that we limited ourselves (due to space limitations) to the rather extreme case where travel costs are totally independent from travel times. Further analysis is required to evaluate intermediate situations, where costs and times may be partially correlated.

Observe that the implementation of our solution algorithms is rather straightforward: besides reoptimization, we did not exploit any sophisticated data structure or algorithmic technique. Clearly, further enhancements are possible. For example, faster and more effective reoptimization based lower bounds may be devised. Moreover, a smart heuristic rule may be adopted to select the branching path (see Section 3.1) giving priority to more promising sub-paths.

Finally, the extension of our approach to other variants or special cases of SAP and K-SAP (e.g. with correlated travel times) seems to provide a challenging direction for further research.

# References


[1] V. Akgün, E. Erkut, and R. Batta. On finding dissimilar paths. *European Journal of Operational Research*, 121:232–246, 2000. doi:10.1016/S0377-2217(99)00214-3 .

[2] M.K. Ardakani and L. Sun. Decremental algorithm for adaptive routing incorporating traveler information. *Computers & Operations Research*, 39(12):3012–3020, 2012. doi:10.1016/j.cor.2012.03.006 .

[3] S.D. Boyles and S.T. Waller. Optimal information location for adaptive routing. *Networks and Spatial Economics*, 11(2):233–254, June 2011. doi:10.1007/s11067-009-9108-9 .

[4] T.-S. Chang, L.K. Nozick, and M.A. Turnquist. Multiobjective path finding in stochastic dynamic networks, with application to routing hazardous materials shipments. *Transportation Science*, 39(3):383–399, 2005. doi:10.1287/trsc.1040.0094 .

[5] L. Fu. An adaptive routing algorithm for in-vehicle route guidance systems with real-time information. *Transportation Research Part B: Methodological*, 35:749–765, 2001. doi:10.1016/S0191-2615(00)00019-9 .

[6] L. Fu and L.R. Rilett. Expected shortest paths in dynamic and stochastic traffic networks. *Transportation Research Part B: Methodological*, 32(7):499–516, 1998. doi:10.1016/S0191-2615(98)00016-2 .

[7] S. Gao and H. Huang. Real-time traveler information for optimal adaptive routing in stochastic time-dependent networks. *Transportation Research Part C: Emerging Technologies*, 21(1):196–213, April 2012. doi:10.1016/j.trc.2011.09.007 .

[8] R.W. Hall. The fastest path through a network with random time-dependent travel times. *Transportation Science*, 20(3):182–188, 1986. doi:10.1287/trsc.20.3.182 .

[9] M.D. Hickman and D.H. Bernstein. Transit service and path choice models in stochastic and time-dependent networks. *Transportation Science*, 31(2):129–146, 1997. doi:10.1287/trsc.31.2.129 .

[10] H. Huang and S. Gao. Optimal paths in dynamic networks with dependent random link travel times. *Transportation Research Part B: Methodological*, 46:579–598, 2012. doi:10.1016/j.trb.2012.01.005 .

[11] E.D. Miller-Hooks. Adaptive Least-Expected time paths in stochastic, time-varying transportation and data networks. *Networks*, 37(1):35–52, 2001. doi:10.1002/1097-0037(200101)37:1<35::AID-NET4>3.0.CO;2-G .

[12] 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. doi:10.1287/trsc.34.2.198.12304 .

[13] L.R. Nielsen. *Route Choice in Stochastic Time-Dependent Networks*. PhD thesis, Department of Operations Research, University of Aarhus, 2004. URL `http://www.imf.au.dk/publs?id=499`.

[14] L.R. Nielsen. TEGP - time-expanded generator with peaks, January 2006. URL `http://www.research.relund.dk`.

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

[16] L.R. Nielsen, D. Pretolani, and K.A. Andersen. Finding the *K* shortest hyperpaths using reoptimization. *Operations Research Letters*, 34(2):155–164, 2006. doi:10.1016/j.orl.2005.04.008 .

[17] L.R. Nielsen, D. Pretolani, and K.A. Andersen. Bicriterion shortest paths in stochastic time-dependent networks. In V. Barichard, M. Ehrgott, X. Gandibleux, and V. T'Kindt, editors, *Multiobjective Programming and Goal Programming*, volume 618 of *Lecture Notes in Economics and Mathematical Systems*, pages 57–67. Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-540-85646-7_6 .

[18] A. Orda and R. Rom. Minimum weight paths in time-dependent networks. *Networks*, 21(3): 295–319, 1991. doi:10.1002/net.3230210304 .

[19] D. Pretolani. A directed hypergraph model for random time-dependent shortest paths. *European Journal of Operational Research*, 123(2):315–324, 2000. doi:10.1016/S0377-2217(99)00259-3 .

[20] J.Y. Yen. Finding the *K* shortest loopless paths in a network. *Management Science*, 17(11): 712–716, 1971. doi:10.1287/mnsc.17.11.712 .

# Appendix A

The main definitions introduced throughout the paper are illustrated here. To this aim, we show a very small example and we work out the details of our method, in particular, the computation of the optimal adaptive route and the application of our branching rule. We treat the minimization of expected cost first and in greater detail, and then the case of travel times more briefly. In order to represent STD networks graphically we adopt the *time expanded hypergraph* representation introduced in [19]. However, we use hypergraphs only as a graphic tool, without discussing related theoretical concepts.

Consider the topological network $G = (N,A)$ in Figure 3, where $a$ is the origin node and $d$ is the destination node. We assume that the time horizon is $H = [0,6]$. In Table 6 we list the possible departure times for each arc in $G$, together with the corresponding arrival times and travel costs. Here a pair $((u,v),t)$ corresponds to a possible leaving time $t$ from node $u$ along arc $(u,v)$, that is, $t \in L(u,v)$. Clearly, for the origin node $a$ we have $L(a) = \{0\}$. For the sake of simplicity, we assume that $X(u,v,t)$ 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 $1/2$. We denote by $D = \{3,4,5,6\}$ the set of possible arrival times at destination $d$. The penalty cost $g_d(t)$ is zero for each $t \in D$ and the input size of the problem is $\kappa = 13$.

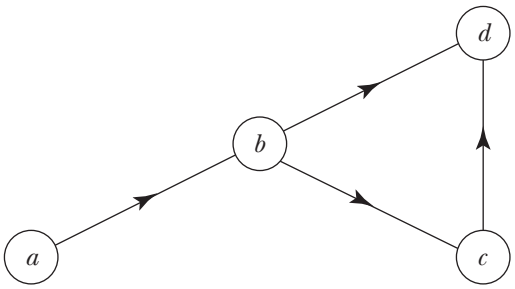



Figure 3: The topological network $G$.

| $(u,v),t$ | $(a,b),0$ | $(b,c),1$ | $(b,c),2$ | $(b,d),1$ | $(b,d),2$ | $(c,d),2$ | $(c,d),3$ | $(c,d),4$ |
|---|---|---|---|---|---|---|---|---|
| $I(u,v,t)$ | $\{1,2\}$ | $\{2,3\}$ | $\{3\}$ | $\{3\}$ | $\{6\}$ | $\{3,4\}$ | $\{4,5\}$ | $\{5,6\}$ |
| $c(u,v,t)$ | 2 | 2 | 5 | 9 | 5 | 8 | 2 | 1 |

Table 6: Arrival times and travel costs.

## Minimization of expected cost

The representation of the resulting STD network is shown in Figure 4. We introduce a node (circle) $u^t$ for each pair $(u,t)$ with $t \in L(u)$. For each $(u,v) \in A$ and $t \in L(u,v)$ we introduce a *hyperarc* $e_{uv}(t)$ that joins $u^t$ to the set of nodes $v^{t_i}$, with $t_i \in I(u,v,t)$. Note that the arrow in $e_{uv}(t)$ points towards $u^t$, which represents the departure from node $u$ at time $t$. The number close to each hyperarc $e_{uv}(t)$ is the travel cost $c(u,v,t)$. We also introduce a dummy source $s$, and dummy arcs from $s$ to each node $d^t$ with $t \in D$. The aim of these arcs is to carry the penalty costs, which are zero in our example.

Hyperarcs in solid lines in Figure 4 represent the minimum expected cost route $R = (D_R, s_R)$; that is, for each pair $(u,t) \in D_R$ the optimal successor is the arc $(u,v) = s_R(u,t)$ such that $e_{uv}(t)$

appears in solid lines. The number close to each node $u^t$ is the value $E^R(u,t)$ obtained from the successor $(u,v) = s_R(u,t)$, as shown in Section 2, where we additionally have $E^R(d,t) = g_d(t) = 0$ for each $t \in D$. Note that $D_R$ contains all the pairs $(u,t)$ with $u \neq d$ except $(c,4)$. This means that a traveller following $R$ cannot arrive at node $c$ at time 4. In fact, time 4 is not a possible arrival time at node $c$, regardless of the chosen route. The computation of the shortest route $R$ can be done as follows. First, the value $E^R(d,t) = g_d(t) = 0$ is assigned to each node $d^t$. Then the other nodes $u^t$ are processed in reverse order of time (i.e., right to left in Figure 4) breaking ties arbitrarily. Each node $u^t$ is assigned the minimum value $E^R(u,t)$ obtained from hyperarcs $e_{uv}(t)$ pointing at $u^t$. Take for example node $b^1$: the involved hyperarcs are $e_{bd}(1)$, yielding $E^R(b,1) = 0+9 = 9$; and $e_{bc}(1)$ yielding $E^R(b,1) = 2+(8+2)/2 = 7$. The latter gives the minimum value and thus appears in solid lines, denoting the optimal successor $s_R(b,1) = (b,c)$.

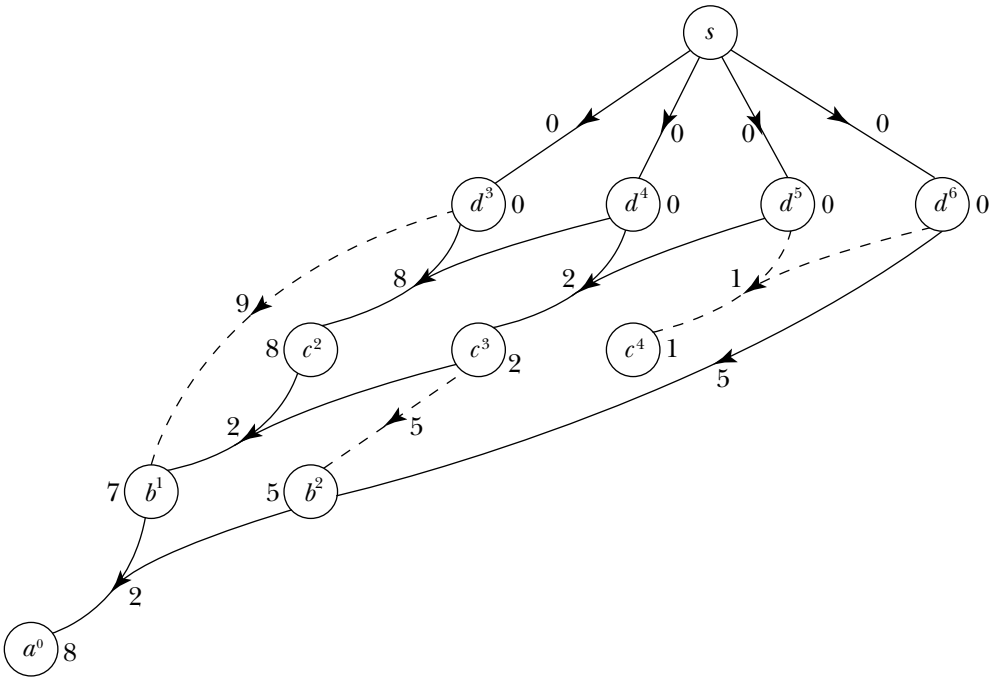
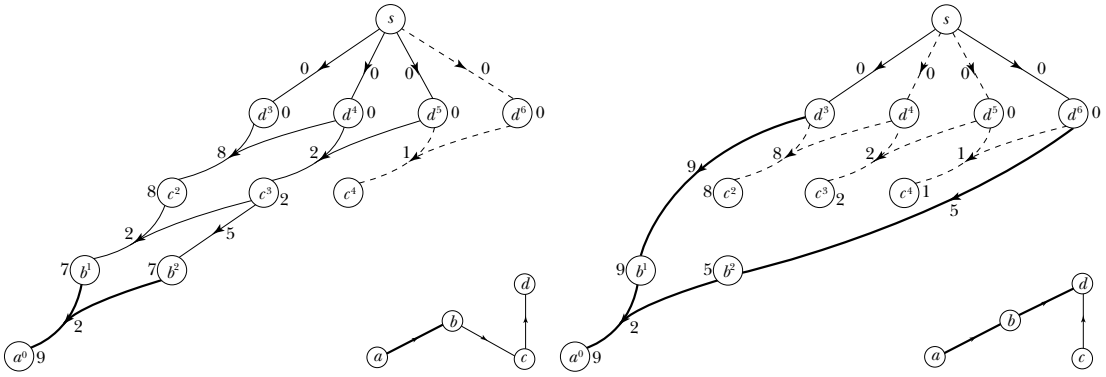



Figure 4: The STD network and the minimum cost route $R$.

Note that $R$ is not a path-route, since we have $s_R(b,1) = (b,c)$ and $s_R(b,2) = (b,d)$. In fact, the graph $G_R$ induced by $R$ coincides with the whole graph $G$. We have $FS(a) = \{(a,b)\}$ and $|FS(b)| = 2$, thus we have two possible options for the branching path $p_R$, namely, $p_R = (a,b,c)$ and $p_R = (a,b,d)$. Suppose the latter is chosen: according to Definition 6 we obtain three subgraphs, that we denote by $G^{(i)}$, $1 \leq i \leq 3$, since we have $G_S = G$.

The subgraph $G^{(1)}$ is obtained from $G$ by deleting the unique arc $(a,b)$ in $FS(a)$. Clearly, it is not possible to travel from $a$ to $d$ in $G^{(1)}$. Therefore, we have $\text{MinCost}(G^{(1)}) = +\infty$ in Step 4 of our algorithm, and the subproblem corresponding to $G^{(1)}$ is discarded. The other two subgraphs, with the corresponding STD networks, are shown in Figure 5a and Figure 5b. The fixed arcs and the corresponding hyperarcs are shown in bold lines; solid lines represent the shortest $a$-$d$ route in the subproblem. In both cases the shortest route is a path-route with expected cost 9. Therefore, the branching rule inserts into the candidate set $Q$ the two pairs $(9, G^{(2)})$ and $(9, G^{(3)})$. These pairs will be selected (in whatever order) and clearly will not generate further subproblems where $d$ is connected

to $a$.




(a) Subgraph $G^{(2)}$ (b) Subgraph $G^{(3)}$

Figure 5: Feasible subproblems obtained by branching on $R$.

## Minimization of expected travel time

The resulting STD network is represented in Figure 6. The structure is the same obtained for costs, but the values associated with arcs and hyperarcs change: hyperarcs $e_{uv}(t)$ carry a zero cost, while the arc from $s$ to $d^t$ carries the "penalty" $g_d(t) = t$. Solid lines represent the route $R$ yielding minimum expected travel time; note that $R$ is not a path-route and differs from the route minimizing cost. The computation of $R$ is performed as shown before. Taking again node $b^1$: hyperarc $e_{bd}(1)$ yields $E^R(b,1) = 3$, while hyperarc $e_{bc}(1)$ yields $E^R(b,1) = (3.5+4.5)/2 = 4$. In this case the former gives the minimum value, thus $s_R(b,1) = (b,d)$.

Also in this case the graph $G_R$ induced by $R$ coincides with the whole graph $G$. Assume that the branching path $p_R = (a,b,c)$ is chosen, i.e the branching operation generates the subgraphs $G^{(i)}$, $1 \leq i \leq 3$ where it is not possible to travel from $a$ to $d$ in $G^{(1)}$. The two subgraphs $G^{(2)}$ and $G^{(3)}$, with the corresponding STD networks and optimal routes, are shown in Figure 7a and Figure 7b. In this case, the branching rule inserts into $Q$ the two pairs $(4.5, G^{(2)})$ and $(4.25, G^{(3)})$, both corresponding to path-routes. Clearly, the last pair is selected first and adds no pair to $Q$, thus the latter pair is the next (and last) one selected.

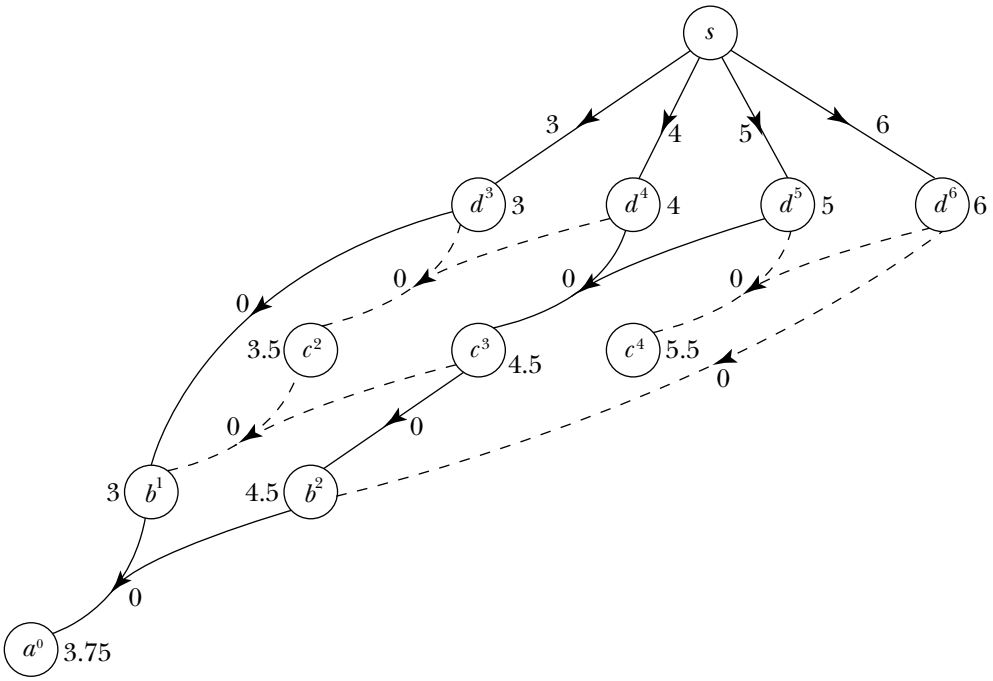
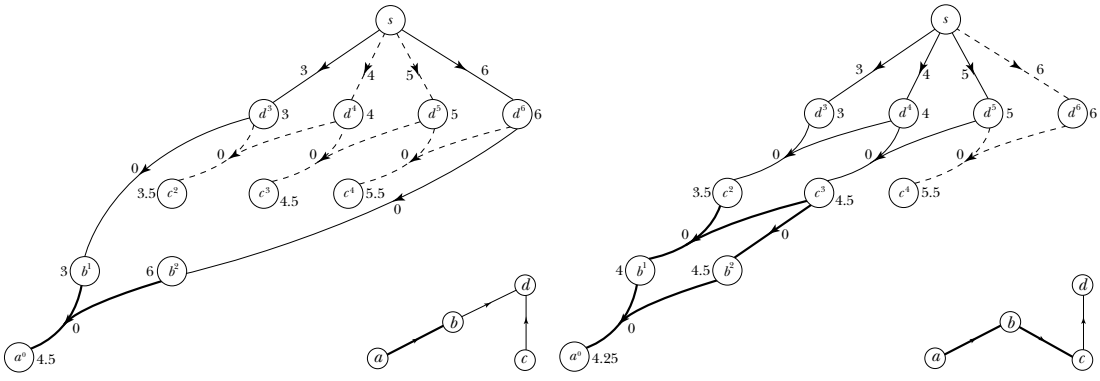



Figure 6: The STD network and the minimum travel time route $R$.




(a) Subgraph $G^{(2)}$

(b) Subgraph $G^{(3)}$

Figure 7: Feasible subproblems obtained by branching on $R$.