Transportation Research Part B 160 (2022) 97–124

Contents lists available at ScienceDirect

Transportation Research Part B

journal homepage: www.elsevier.com/locate/trb

A framework for efficient dynamic routing under stochastically varying conditions[☆]

Nikki Levering[a,*], Marko Boon[b,c], Michel Mandjes[a,c,d], Rudesindo Núñez-Queija[a,e]

[a] Korteweg–de Vries Institute for Mathematics, University of Amsterdam, Amsterdam, The Netherlands
[b] Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
[c] Eurandom, Eindhoven University of Technology, Eindhoven, The Netherlands
[d] Amsterdam Business School, University of Amsterdam, Amsterdam, The Netherlands
[e] Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

ARTICLE INFO

Keywords:
Routing
Real time
Shortest path
Semi-Markov decision processes
Dynamic programming

ABSTRACT

Despite measures to reduce congestion, occurrences of both recurrent and non-recurrent congestion cause large delays in road networks with important economic implications. Educated use of Intelligent Transportation Systems (ITS) can significantly reduce travel times. We focus on a dynamic stochastic shortest path problem: our objective is to minimize the expected travel time of a vehicle, assuming the vehicle may adapt the chosen route while driving. We introduce a new stochastic process that incorporates ITS information to model the uncertainties affecting congestion in road networks. A Markov-modulated background process tracks traffic events that affect the speed of travelers. The resulting continuous-time routing model allows for correlation between velocities on the arcs and incorporates both recurrent and non-recurrent congestion. Obtaining the optimal routing policy in the resulting semi-Markov decision process using dynamic programming is computationally intractable for realistic network sizes. To overcome this, we present the EDSGER* algorithm, a Dijkstra-like shortest path algorithm that can be used dynamically with real-time response. We develop additional speed-up techniques that reduce the size of the network model. We quantify the performance of the algorithms by providing numerical examples that use road network detector data for The Netherlands.

1. Introduction

To reduce congestion in road networks, a wide variety of measures can be thought of. Perhaps the most basic measure is to increase the network's capacity — the Dutch government for example increased the total road length from 130.446 km in 2001 to 141.361 km in 2020 (CBS, 2020). Alternatively, one may attempt to better exploit the existing resources, examples being the use of reversible lanes (Wolshon and Lambert, 2006) and the transit lane experiments in various US states (Boriboonsomsin and Barth, 2008; Chang et al., 2008). Despite such measures, recurrent congestion (i.e., congestion during peak hour) and non-recurrent congestion (i.e., congestion due to incidents) remain a major concern. Consequences include substantial delay in travel times, increased economic costs, and negative environmental effects.

So as to reduce the effects of recurrent congestion, historical data is used to incorporate periodically occurring events (rush hours, weekly patterns, etc.) into routing algorithms. Regarding non-recurrent congestion, an important role is played by Intelligent

☆ This research project is partly funded by the NWO Gravitation project, Netherlands Networks, grant number 024.002.003.

* Corresponding author.
E-mail address: n.a.c.levering@uva.nl (N. Levering).

https://doi.org/10.1016/j.trb.2022.04.001
Received 21 May 2021; Received in revised form 21 November 2021; Accepted 1 April 2022
Available online 6 May 2022

Traffic Systems (ITS) capable of providing travelers with real-time information. The availability of data on virtually all thinkable network characteristics, combined with instant communication to individual travelers, offers the potential for routing with minimal delay and minimal congestion. A complication, however, lies in the unpredictability of traffic surges and incidents, and in addition in the computational effort needed to process all available information in real time. Our work provides an approach that handles both these challenges: providing fast and close-to-optimal routing, using a probabilistic framework that is well suited for modeling uncertainties that affect congestion.

Routing framework

In this paper we study routing by casting it as a dynamic stochastic shortest path problem. Our objective is to minimize the expected travel time of a vehicle pertaining to a given origin–destination pair, assuming that the vehicle is allowed to adapt the chosen route while driving. A key feature of our setup is that we model the travel-time dynamics using suitably chosen stochastic processes. Specifically, for each road of the network, we let the velocity of a vehicle on that road be determined by the state of a Markovian background process, describing incidents, weather conditions, etc.

Routing decisions in the proposed dynamic model can be expressed in terms of a semi-Markov decision process (SMDP) (Boucherie and van Dijk, 2017; Puterman, 1994; Sutton and Barto, 2018). The state of the SMDP consists of the location of the vehicle, which, in a practical context, can be traced by GPS, and the state of the background process, being provided by an ITS. The route can be adapted at each intersection along the traveled path. Based on the state of the SMDP at an intersection, a decision is taken that informs the driver which arc to travel next. Clearly, the use of a Markovian background process facilitates the modeling of non-recurrent, random events. Perhaps surprisingly, however, we will argue that it also allows us to incorporate the deterministic patterns corresponding to recurrent, (near-)deterministic events. Importantly, our framework allows for correlation between the travel times on the edges and is therefore well capable of modeling the spillback effect: an incident at an arc causes a drop in speed levels at upstream arcs.

In the context of our model, an optimal routing policy can be evaluated by the use of dynamic programming (DP), as it can be shown that such a policy satisfies the Bellman optimality equations. DP does however suffer from the curse of dimensionality, making it prohibitively slow for networks of practical relevance. To cope with this, we have developed the near-optimal EDSGER algorithm, as well as its more efficient variant EDSGER*, which are Dijkstra-like shortest path algorithms, but with the additional feature that the route can be adapted along the way, based on the current state of the background process. We furthermore present speed-up techniques that greatly reduce the size of the underlying network model and its corresponding state-space. These techniques can therefore be used as preprocessing steps to make the routing algorithms substantially more efficient.

To make our approach operational, the parameters pertaining to the background process must be estimated. Deterministic patterns and corresponding transitions in the background process can be identified from historical data. In addition, we need to estimate the frequency and consequences of non-recurrent events, such as the frequency of incidents or the drop in speed level under bad weather conditions. Examples of institutions that collect data on traffic flows and traffic speeds are the National Data Warehouse for Traffic Information (NDW) in The Netherlands, the Mobilitäts Daten Marktplatz (MDM) in Germany; in addition various US states have such an infrastructure (see for example MITS, the Michigan Intelligent Transportation Systems).

Literature overview

Routing algorithms have extensively been examined in literature. In a deterministic graph with positive arc costs satisfying the FIFO property, the shortest path can be obtained by Dijkstra's algorithm, developed by Edsger W. Dijkstra in the late 1950s (Dijkstra, 1959; Kaufman and Smith, 1993). Throughout the years many variations and speed-up techniques for Dijkstra's algorithm have been presented, examples of which are the A*-algorithm and the bidirectional Dijkstra algorithm (Fu et al., 2006; Lerner et al., 2009). Bellman (1958) and Ford Jr. (1956) constructed a shortest path algorithm that allows for both positive and negative arc costs.

The search for optimal routes can complicate considerably as soon as randomness is introduced. The stochastic shortest path problem is an extension of the deterministic shortest path problem in which uncertainty in travel times is taken into account. Several variations of this problem exist, each with its own objective. Examples of two well-studied objectives are the minimization of the expected travel-time and the maximization of the on-time arrival probability. In the present article we focus on the first objective; for studies on the latter we refer to Fan et al. (2005), Frank (1969), Nie and Wu (2009) and Pedersen et al. (2020) and references therein. In case of a minimum expected travel-time objective, Dijkstra's algorithm can still be applied if there is neither correlation (incidents on specific arcs do not influence travel times on other arcs) nor time-dependency (the travel times do not depend on the hour of the day) (Loui, 1983; Mirchandani and Soroush, 1986; Murthy and Sarkar, 1996). Recurrent events are well-described by time-dependent velocities (think of predictable, recurring events such as rush hours), but Hall (1986) showed that classic shortest path algorithms like Dijkstra's algorithm fail in stochastic time-dependent networks. Hall additionally argued that it is suboptimal to solely consider static routing and that it is advantageous to allow travelers to adapt the chosen route while traveling. These findings resulted in studies on optimal adaptive routing algorithms in stochastic time-dependent networks. Examples of such studies are Fu and Rilett (1998), Miller-Hooks (2001), Miller-Hooks and Mahmassani (2000). These algorithms do, however, not take ITS information into account and are therefore limited in their capability to incorporate non-recurrent congestion.

Algorithms that do take ITS information into account can be divided into two categories. The first category consists of algorithms that assume that ITS provides information on all realized travel times in the network. Without attempting to provide an exhaustive overview, we refer in this context to the algorithms presented in Bander and White III (2002), Cheung (1998), Davies and Lingras (2003), Gao and Chabini (2006), Gao and Huang (2012), Polychronopoulos and Tsitsiklis (1996), Provan (2003) and Waller and Ziliaskopoulos (2002). The second category consists of algorithms that assume that ITS provides information on the state of a background process in the network. Psaraftis and Tsitsiklis (1993) were among the first to work in this setting, presenting a framework in which the travel time on an arc depends on an environmental variable becoming known once a vehicle arrives at the attached node. Their setup was extended by Kim et al. (2005a), who work with a Markov process, the state of which state directly determines the travel time. By ITS, the state of a Markov process of every arc is known while traveling through the network, so that we can phrase the optimal routing problem in terms of a Markov decision process (MDP). The MDP framework is further explored in Güner et al. (2012), Sever et al. (2013) and Thomas and White III (2007), under the assumption of independence of the Markov processes on the arcs. Güner et al. (2012) and Sever et al. (2013) furthermore assume that the travel time on a link is known once the intersection at the head of the link is reached, whereas in reality the conditions on an arc may still change while traveling on that arc. Therefore, realized travel times can only take values from a previously chosen discrete collection of values. This is also the case in Thomas and White III (2007), who assume the link costs to follow a known discrete distribution. In the MDP framework an optimal policy can be characterized by the Bellman optimality equations and can therefore be evaluated by performing a DP recursion (Bellman, 1957a,b; Bertsekas, 1976, 2005; Puterman, 1994; Ross, 1983; Sutton and Barto, 2018). DP suffering from the curse of dimensionality, it may lead to excessive computational costs. Possible solutions are dimensionality reduction, as described in e.g. Sýkora (2008) and references therein, and approximative procedures from Reinforcement Learning (RL) (Bertsekas, 2012; Powell, 2007; Sutton and Barto, 2018). In the context of routing, dimensionality reduction is employed in Güner et al. (2012) and Kim et al. (2005b), while RL techniques are studied in Mao and Shen (2018) and Sever et al. (2013).

Contributions

This paper has two main contributions. First, we present the Markovian background process and corresponding SMDP framework in detail. To our knowledge, this is the first study that uses a continuous-time Markovian background process to develop an adaptive routing algorithm. Although the approaches of Ferris and Ruszczyński (2000) and Kharoufeh and Gautam (2004) also work with continuous-time Markovian processes, there are important differences with our proposal. In the setup of Ferris and Ruszczyński (2000), the travel times are directly modeled relying on a continuous-time Markov process. In Kharoufeh and Gautam (2004), a continuous-time Markov background process is considered to model the velocities, but it only describes the dynamics on a single arc and does not consider routing. With our continuous-time Markovian background process, applying to the network as a whole, we overcome the drawback of the discrete MDP framework that we mentioned above, as the continuity of the background process implies continuity in arrival times (i.e., travel times are a continuous function of the departure epoch). Moreover, our framework guarantees the FIFO-property (i.e., the arrival epoch is an increasing function of the departure epoch). In addition, we explicitly show how to exploit the fact that our setup allows for correlation and is capable of incorporating both recurrent and non-recurrent congestion.

As a second contribution, after having presented a way to evaluate the optimal policy, which suffers from the curse of dimensionality, we propose efficient approximating alternatives. The evaluation, through DP, of the optimal policy being prohibitively slow due to dimensionality issues, we propose the highly efficient EDSGER* algorithm. Our numerical examples demonstrate that our approach is well capable of incorporating both recurrent and non-recurrent congestion. Notably, it clearly outperforms deterministic Dijkstra-type algorithms in which the per-arc travel times are replaced by their expected values. Comparison of the EDSGER* algorithm to DP shows that EDSGER* is orders of magnitude faster. Indeed, the computational costs of DP grow exponentially with the network size, while EDSGER* provides essentially real-time response, performing just slightly below optimal. The considered examples all concern specific parts of the Dutch road network, with instances that are based on the data provided by NDW to make sure they are representative for real traffic scenarios.

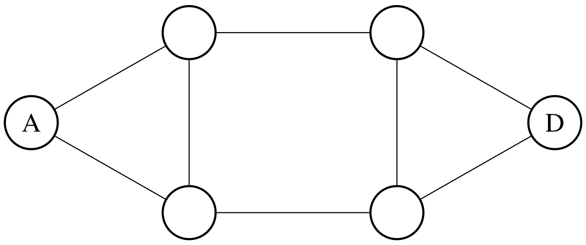
The organization of the paper is now as follows. Section 2 contains a motivational example. The example demonstrates the background process and the performance of the routing algorithms DP and EDSGER* in this low-complexity setting. The general structure of the background process and SMDP framework are presented in Section 3. Section 4 considers routing in this framework, describing the optimal routing algorithm, heuristic algorithms and speed-up techniques. Numerical examples to show the usefulness of the model and to assess the efficiency of the routing algorithms are presented in Section 5. Section 6 contains concluding remarks.

2. Motivational example

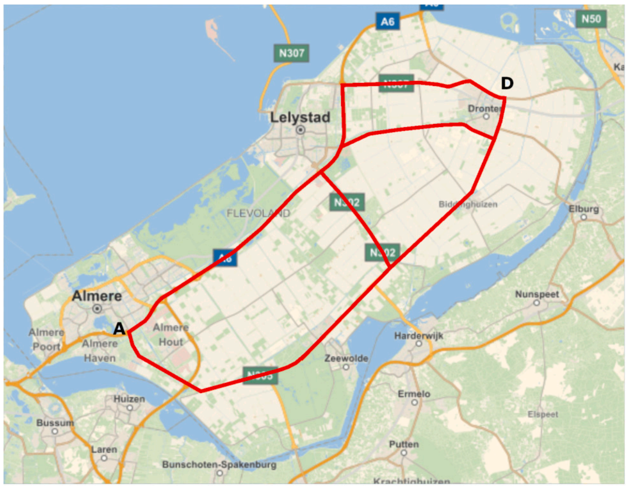
This section provides a motivational example that illustrates the proposed framework and the various routing algorithms developed for it. For transparency we consider a relatively small network, and describe the structure of a typical background process corresponding to this network. We briefly describe the expected travel time and the run-time resulting from our routing algorithm as well as competing algorithms, so as to provide an indication of the gain that can be achieved by our proposed routing scheme.

Consider the road network of Fig. 1(b), depicting the roads that can be used to travel from Almere (A) to Dronten (D), two cities in the Netherlands. As in the rest of this paper, the goal is to minimize the expected travel time. Fig. 1(a) shows the corresponding routing graph in which each node represents an intersection in Fig. 1(b), and each link represents the road between two of these intersections. The Dutch government has imposed maximum velocities on the 16 roads (8 bidirectional arcs, that is) in the network of Fig. 1. However, due to e.g. bad weather conditions and traffic incidents vehicles may not always be able to drive at these maximum velocities.

We model the variability in velocities by a Markovian background process. This background process records the events that affect these speeds. In this example, the only events discussed are incidents and rain showers, but it can be extended to include various sorts of other events in an obvious manner. To model traffic accidents we let $\{X_i(t), t \geqslant 0\}$ be a Markov process that denotes whether arc i is blocked by an incident at time t. Specifically, we choose to set $X_i(t) = 1$ if the arc is not blocked, and 2 otherwise. Furthermore, we let $\{Y(t), t \geqslant 0\}$ be a Markov process that describes whether it rains at time t or not: $Y(t) = 1$ indicates it is dry whereas $Y(t) = 2$ indicates it rains. The Markovian background process can then be written as the vector $B(t) := (Y(t), X_1(t), \dots, X_{16}(t))$, having a state



(a) Graphical situation.



(b) Real situation.

Fig. 1. Road network Almere-Dronten.

space of dimension 2^{17}. The state of $B(t)$ describes the velocities on each of the arcs: we let $v_i(s)$ be the velocity on arc i when $B(t) = s$.

In the sequel we impose the natural assumption that the processes $X_1(t), \dots, X_{16}(t)$ are independent, which effectively means that the occurrence of an incident on a specific link has no impact on the occurrence of an incident on other links. This assumption does not imply that there is no correlation between travel times on the arcs, as the velocity on an arc depends on the state of $B(t)$.

We let the transition rate matrix for $X_i(t)$, for $i = 1, \dots, 16$, be written as

$$Q_i = \begin{bmatrix} -\alpha_i & \alpha_i \\ \beta_i & -\beta_i \end{bmatrix},$$

with α_i the incident rate and β_i the clearance rate of an incident at arc i. The structure of this transition rate matrix, in which the rows sum to zero, arises from the theory on continuous-time Markov chains, which is described in detail in the works of Norris (1997) and Ibe (2013). The $2^{16} \times 2^{16}$-dimensional transition rate matrix of the vector process $(X_1(t), \dots, X_{16}(t))$ is now given by $Q_0 := Q_{16} \oplus Q_{15} \oplus \cdots \oplus Q_1$; here the operation '$\oplus$' denotes the Kronecker sum (Pease III, 1965), which, given '$\otimes$' denotes the direct product and $I_{\dim(C)}$ denotes the identity matrix with the same dimensions as the matrix C, is defined as

$$A \oplus B = A \otimes I_{\dim(B)} + I_{\dim(A)} \otimes B, \tag{1}$$

for two square matrices A and B. For example, the Kronecker sum for the two matrices Q_2 and Q_1 is given by

$$Q_2 \oplus Q_1 = \begin{bmatrix} -\alpha_1 - \alpha_2 & \alpha_1 & \alpha_2 & 0 \\ \beta_1 & -\beta_1 - \alpha_2 & 0 & \alpha_2 \\ \beta_2 & 0 & -\alpha_1 - \beta_2 & \alpha_1 \\ 0 & \beta_2 & \beta_1 & -\beta_1 - \beta_2 \end{bmatrix},$$

which can be interpreted as the transition rate matrix of $(X_1(t), X_2(t))$ that lives on the state space $(1,1)$, $(2,1)$, $(1,2)$, $(2,2)$. Regarding $Y(t)$, we denote the transition rate from 1 to 2 as λ and from 2 to 1 as μ. This means that $1/\lambda$ represents the mean time between showers, and $1/\mu$ the mean shower duration. Combining the above, the Q-matrix for the full vector $B(t)$ can be written as

$$Q = \begin{bmatrix} Q_0 - \lambda I & \lambda I \\ \mu I & Q_0 - \mu I \end{bmatrix}.$$

Table 1
Results example Almere-Dronten.

	Average expected travel time	Weighted average expected travel time	Run-time (s)
DS	0.510	0.433	$1.35 \cdot 10^{-3}$
EDSGER*	0.443	0.410	$1.13 \cdot 10^{-2}$
DP	0.442	0.410	$4.26 \cdot 10^{-1}$

Remark 1. For this example we have assumed that $X_i(t)$ is independent of $Y(t)$, which informally means that rain does not affect the occurrence of incidents. This may sound unnatural, but, importantly, our model can be adapted in a straightforward manner to make sure that weather conditions do affect the rate of incidents. The general setup will be introduced in full detail in the next section. $\diamondsuit$

In the model introduced above the travel time on any arc is completely determined by the proposed velocity dynamics. This means that for any given arc, if the state of the background process when leaving the origin node is given, the expected travel time to the destination node can be computed (jointly with the state of the background process when arriving at the destination node). This in principle allows the computation of the optimal routing policy. Below we will argue that this optimal policy can be found relying on dynamic programming (DP) methods. DP-based methods, however, are typically prohibitively computationally demanding in case the underlying state space is large (the *curse of dimensionality*). Focusing on our specific routing objective, it is clear that such computational issues will arise, since, as we saw above, the number of background states is large, even in a small network.

The EDSGER* algorithm, that will be presented in Section 4, succeeds in overcoming the high computation complexity of DP-based methods. The idea behind EDSGER* is inspired by the A*-algorithm, a speed-up version of Dijkstra's algorithm. We assess EDSGER* on two criteria, namely:

- *distance-to-optimality*, i.e., the difference between the objective function value and the optimal value.
- *efficiency*, i.e., the run-time of the algorithm.

To provide an impression of the achievable performance of EDSGER*, we conducted an experiment on the network presented in Fig. 1. We define the 17-dimensional background process $B(t)$ as above. We chose $\alpha_i = 0.1\ \mathrm{h}^{-1}, \beta_i = 2\ \mathrm{h}^{-1}$ for $i = 1, \dots, 16$ and $\lambda = \mu = 0.25\,\mathrm{h}^{-1}$, meaning that the expected time between two incidents is 10 hours and the expected clearance time of an incident 30 minutes, whereas both the expected duration of a rain shower and the expected duration of a dry period equal 4 hours. These parameter values were chosen merely for illustrative purposes; in all later experiments we base ourselves on historical data. In case that it does not rain and there is no incident on the arc we let the vehicle speed be 120 km/h if there is no incident on the directly adjacent arcs, and 100 km/h otherwise. Regarding the case that it does not rain and there is an incident on the arc, we let the vehicle speed be 50 km/h if there is no incident on the directly adjacent arcs, and 20 km/h otherwise. In case it does rain the velocities on this arc, in the four situations discussed above, are 100 km/h, 80 km/h, 20 km/h and 10 km/h, respectively.

The results pertaining to this example are shown in Table 1. The performance of the EDSGER* algorithm is compared with a deterministic Dijkstra-type algorithm that assumes that a vehicle can always drive at the maximum speed level and thus does not take any stochasticity into account ('DS', being an abbreviation of 'Deterministic Static'). We in addition implemented a competitive DP algorithm for determining the policy that minimizes the expected travel time ('DP').

- The first column shows the expected travel time from A to D under the different policies, averaged (evenly) over all possible initial background states.
- The second column provides a weighted average of the expectations corresponding to all possible initial background states; for a given initial background state, the weight is chosen equal to its limiting probability.
- The last column contains the run-time of the three algorithms.

A first conclusion is that the objective function achieved by EDSGER* is close to its minimal value (as provided by DP). Secondly, even in this small network, the run-time of EDSGER* is significantly lower than the run-time of the competitive DP algorithm. Numerical experiments later in this paper will show that the run-time of the competitive DP algorithm grows exponentially with the network size, whereas the run-time of the EDSGER* algorithm remains essentially real-time (i.e., being in the sub-second scale). We in addition conclude that ignoring the stochasticity, as is done by DS, leads to a fast but far from optimal algorithm; note in particular that the corresponding objective function is substantially higher than its minimal value (as provided by DP).

3. Markovian velocity model

After the motivating example of the previous section, we now formally introduce the general mathematical framework for the Markov model that describes the attainable velocities of the vehicles, and point out how in the context of this model routing can be expressed in terms of an SMDP. As before, we consider the objective of minimizing the expected travel time between a given origin–destination (OD) pair in the road network. Let $G = (N, A)$ be a graph representing the road network, with N and A the set of nodes in G, and the set of directed arcs in G, respectively. The set N represents the intersections in the road network, whereas the

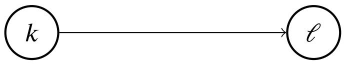


Fig. 2. Single arc network.

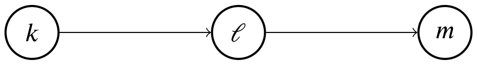


Fig. 3. Two-arc network.

set A represents the roads that connect these intersections, implying $k\ell \in A$ only if there is a (direct) road in the network between the intersections that are labeled by k and ℓ. Let $d_{k\ell}$ be the length of arc $k\ell$.

A realistic feature of our setup is that when this arc is traveled by a vehicle, the speed of this vehicle is not necessarily constant. Indeed, it may vary between finitely many values, related to e.g. the occurrence of incidents and weather conditions. To deal with these changing velocities, we introduce a Markovian background process on the arcs A, of which the background process discussed in Section 2 is a special case. As we will argue below, the use of this stochastic process will allow us to incorporate random effects (corresponding to non-recurrent congestion, that is) as well as (near-)deterministic patterns in the attainable speeds on the arcs (corresponding to recurrent congestion). We assume that the state of the background process and the corresponding traffic velocities are available while traveling. Indeed, travelers are able to adapt the chosen route while driving, based on the information available. In the sequel we will phrase this dynamic stochastic shortest path problem with information on velocity levels in terms of a finite semi-Markov decision model.

Background process

Before presenting the full Markov-modulated environmental process, for expositional reasons we will first show examples of background processes on small networks that fit into our framework and gradually extend this setup. First consider the single-arc network in Fig. 2. Define $\{X_{k\ell}(t), t \geqslant 0\}$ as the continuous-time Markovian background process on this arc such that

$$X_{k\ell}(t) = \begin{cases} 1 & \text{no incident on arc } k\ell \text{ at time } t \\ 2 & \text{otherwise,} \end{cases}$$

similar to the modeling of incidents in the motivational example of the preceding section. The process $X_{k\ell}(t)$ provides information on possible events on $k\ell$ and determines the velocity of a vehicle traveling on $k\ell$: the speed at time t equals $v_{k\ell}(i)$ if $X_{k\ell}(t) = i$, for $i = 1, 2$. For technical reasons it is throughout assumed that all these velocities are strictly positive, but this is, in practical terms, not a restriction as they are allowed to have arbitrarily small values. The speed at which vehicles can travel on this arc is now completely described through the background process $X_{k\ell}(t)$ and the corresponding $v_{k\ell}(i)$.

In the above example there are two possible speeds, but note that in practice one could work with more than two levels; one could e.g. think of the period between the clearance of an incident and the time the free-flow speed can be attained again. These dynamics can be incorporated by allowing the state space of the process $X_{k\ell}(t)$ to consist of more than two states. In general we let this state space be denoted by $\mathcal{S}_{k\ell} = \{1, \dots, n_{k\ell}\}$. When $X_{k\ell}(t) = s \in \mathcal{S}_{k\ell}$, the speed at which vehicles are moving on arc $k\ell$ is $v_{k\ell}(s)$. We use the notation $Q_{k\ell}$ to denote the corresponding transition rate matrix of dimension $n_{k\ell} \times n_{k\ell}$.

We now extend the network with an arc ℓm (Fig. 3), and let $B(t) = (X_{k\ell}(t), X_{\ell m}(t))$ be the Markovian background process on this two-arc network. Here $\{X_{k\ell}(t), t \geqslant 0\}$ and $\{X_{\ell m}(t), t \geqslant 0\}$ are independent Markovian background processes on the arcs $k\ell$ and ℓm, respectively. As in the single-arc case we denote the state space of $X_{k\ell}(t)$ by $\mathcal{S}_{k\ell} = \{1, \dots, n_{k\ell}\}$ and the transition rate matrix as $Q_{k\ell}$. We can equivalently define $\mathcal{S}_{\ell m}$ and $Q_{\ell m}$ for the process $X_{\ell m}(t)$. With $X_{k\ell}(t)$ and $X_{\ell m}(t)$ being independent, the transition rate matrix Q of $B(t)$ is of the form

$$Q = Q_{\ell m} \oplus Q_{k\ell}, \tag{2}$$

where '$\oplus$' is the Kronecker sum that was defined in (1). If $B(t)$ is in state $s \in \mathcal{I} := \mathcal{S}_{k\ell} \times \mathcal{S}_{\ell m}$, we let the speed at which vehicles are moving be $v_{k\ell}(s)$ on arc $k\ell$ and $v_{\ell m}(s)$ on arc ℓm. Importantly, this means that the velocity on each of the two arcs is allowed to depend on both $X_{k\ell}(t)$ and $X_{\ell m}(t)$. Hence, this way of modeling defines an implicit dependence between the velocities on the two arcs.

The presented setup can easily be extended by adding more arcs to the network. Consider the network $G = (N, A)$ and label the directed edges in A by $k_1\ell_1, k_2\ell_2, \dots, k_n\ell_n$ with $n := |A|$.[1] The Markovian background process can now be written as $B(t) = (X_{k_1\ell_1}(t), \dots, X_{k_n\ell_n}(t))$. Here $\{X_{k_j\ell_j}(t), t \geqslant 0\}$, with $j \in \{1, \dots, n\}$, is a continuous-time Markov process with state space $\mathcal{S}_{k_j\ell_j} = \{1, \dots, n_{k_j\ell_j}\}$ and transition rate matrix $Q_{k_j\ell_j}$, such that $X_{k_j\ell_j}(t)$ is the congestion level at arc $k_j\ell_j \in A$ at time t. Assuming that $X_{k_i\ell_i}(t), X_{k_j\ell_j}(t)$ evolve independently for $i \neq j$, the transition rate matrix Q of $B(t)$ is of the form

$$Q = Q_{k_n\ell_n} \oplus Q_{k_{n-1}\ell_{n-1}} \oplus \cdots \oplus Q_{k_2\ell_2} \oplus Q_{k_1\ell_1}.$$

[1] Observe the difference between n, denoting the number of arcs in the network, and $n_{k\ell}$, denoting the number of states in $X_{k\ell}(t)$.

Dependence between the arcs can again be realized by letting the speed on each of the arcs depend on the state of the full vector $B(t)$: when $B(t)$ is in state $s \in \mathcal{I} = \mathcal{S}_{k_1\ell_1} \times \ldots \times \mathcal{S}_{k_n\ell_n}$ the speed at which vehicles are moving on arc $k_j\ell_j$ is $v_{k_j\ell_j}(s)$. The dynamics of the vehicles moving on the network are now completely defined through the background process $B(t)$ and the corresponding velocity levels. For instance, if $B(t') = s$, then the travel time on edge $k_i\ell_i$ for traveling a distance $d \in [0, d_{k_i\ell_i}]$ when leaving node k_i at time t' is distributed as $\tau^s_{k_i\ell_i}(d)$, with

$$\tau^s_{k_i\ell_i}(d) := \min\left\{ t \geqslant 0 : \int_0^t v_{k_i\ell_i}(B(u))\,\mathrm{d}u \geqslant d \;\middle|\; B(0) = s \right\},$$

where we use the fact that the travel distance d can be computed by an integral over the travel speed. For transparency of notation we abbreviate

$$\tau^s_{k_i\ell_i}(d_{k_i\ell_i}) \equiv \tau^s_{k_i\ell_i}$$

to denote the travel time for arc $k_i\ell_i$ when leaving at time t' from node k_i.

There is global correlation in the above setup, due to the fact that the speed that a vehicle on a given arc can drive at depends on the full background process $B(t)$, containing information on the congestion levels of *all* arcs. Previous research has shown that correlation due to non-recurrent congestion is primarily local; the congestion level of an arc mainly affects the attainable velocities at upstream arcs that are within a certain distance of the incident arc (Guo et al., 2019; Priambodo et al., 2020). We model this property through the following assumption.

Assumption 1 (*Local-r-correlation*)**.** Denote with $A^r_{k\ell}$ the set of arcs that are at most r arcs away from arc $k\ell$ (including $k\ell$ itself). We assume the velocity on arc $k\ell$ to depend only on the congestion levels of the arcs at most r arcs away: $v_{k\ell}(s) = v_{k\ell}(s')$ if $s_i = s'_i$ for all $i \in A^r_{k\ell}$.

Remark 2. Note that, if the dependence structure is not the same for all arcs, this asymmetry can still be captured by Assumption 1. More specifically, let, for every $k\ell \in A$, $A_{k\ell}$ be the set of arcs whose congestion level affects the velocity on $k\ell$. Observe that, with $r_{k\ell}$ denoting the longest distance (in terms of number of arcs) from an element of $A_{k\ell}$ to $k\ell$, there is local-r-correlation for $r := \max\{r_{k\ell} \mid k\ell \in A\}$. Evidently, this local-$r$-correlation assumption is unnecessarily strong, and can be weakened to capture the dependence structure per arc. All presented results can be obtained under the weakened assumption of arc-specific local correlation as well. However, for notational convenience, we will consistently use the (stronger) local-r-correlation assumption throughout the paper. $\diamondsuit$

Before continuing to the SMDP routing framework induced by our proposed model, the next remark discusses a possible extension of our model, which was already noted in the motivational example.

Remark 3. Suppose there is a Markovian process $Y(t)$ that prompts global correlation in the network, e.g. weather conditions that affect the velocities on *all* arcs in the network. This can be incorporated into our framework by expanding the process $B(t)$ with the process $Y(t)$. We let $B(t) = (Y(t), X_{k_1\ell_1}(t), \ldots, X_{k_n\ell_n}(t))$ be such that, given the state of $Y(t)$, the future evolution of $X_{k_i\ell_i}(t)$ and $X_{k_j\ell_j}(t)$ are independent for $i \neq j$. The transition rate matrix for the case $Y(t) = s$ is then given by

$$Q^s = Q^s_{k_n\ell_n} \oplus Q^s_{k_{n-1}\ell_{n-1}} \oplus \cdots \oplus Q^s_{k_2\ell_2} \oplus Q^s_{k_1\ell_1}, \tag{3}$$

with $Q^s_{k_j\ell_j}$ the transition rate matrix for the congestion levels at arc $k_j\ell_j$ given $Y(t) = s$. In this way our model allows for global dependence when information on processes that affect velocities in the whole network is available. In the remainder of this paper we will mainly focus on the case without global correlation, but, by using transition matrices of the type (3), the results of this paper can be extended to the case of a known global dependence process. $\diamondsuit$

Remark 4. By the use of phase-type distributions the proposed Markov model for congestion is not limited to exponentially distributed times between states of congestion (Asmussen, 2003). Importantly, phase-type distributions can model random quantities that are less variable than the exponential distribution (e.g. by using Erlang distributions) as well as random quantities that are more variable than the exponential distribution (e.g. by using hyperexponential distributions). In particular, for highly predictable events (think of recurring events, such as rush hours) Erlang distributions with a high number of phases, thus leading to a low variance, are well suited. $\diamondsuit$

Semi-Markov Decision Process

The objective of this subsection is to phrase our optimization problem in terms of an SMDP. Above we introduced our stochastic process modeling the velocity dynamics. Our setup is somewhat in the spirit of the one used in the work of Psaraftis and Tsitsiklis (1993), Kim et al. (2005a,b) and Sever et al. (2013), who use an MDP to capture the stochasticity in travel times. Importantly, as opposed to such earlier MDP-based approaches, we impose a *continuous-time*, rather than discrete-time, stochastic process on the arc speeds. Although at the expense of additional computational complexity, this has several advantages:

- When a vehicle travels an arc, changing conditions on this arc are taken into account. In the conventional MDP framework it is assumed that if a vehicle is at an intersection, travel times on the attached arcs are known and choices are based on these known travel times. Conditions on these attached arcs may however change while driving, affecting the arrival time.
- Travel times are a continuous function of the departure epoch. Therefore the consistency- or FIFO-property (Wellman et al., 1995), is naturally satisfied (see Appendix A for the proof):

Proposition 1. *Let $k\ell \in A$ and denote with $\tau^t_{k\ell}(d)$ the travel time on arc $k\ell$ for traveling a distance $d \in [0, d_{k\ell}]$ when starting at $t \geqslant 0$. Then $t_1 \leqslant t_2$ implies that $t_1 + \tau^{t_1}_{k\ell}(d) \leqslant t_2 + \tau^{t_2}_{k\ell}(d)$.*

Routing in our framework can be analyzed using a finite semi-Markov decision process (SMDP). The state of the SMDP consists of the location of the vehicle combined with the state of the background process. In a practical context, the first could be tracked by GPS, whereas the latter can be provided by an ITS. Decision epochs are the arrival times at the intersections, in the sense that at these epochs one has the opportunity to adapt the route. Denote at decision-epoch i the state of the SMDP as $(K_i, B(t_i))$, with $K_i \in N$ the label of the current intersection and $B(t_i)$ the state of the background process at the current time t_i. The following (reasonable) assumptions are made:

- Waiting at a node is not allowed. Note that this assumption does not affect an optimal policy, as the FIFO property implies that waiting is never advantageous.
- Information on the congestion levels on all arcs is available at all times.
- There is at least one path connecting every node with the destination node. If this assumption is not fulfilled, we have a non-connected graph for which we can only construct routing policies for OD-pairs within connected components.

At decision epoch i, the goal is to choose a neighbor k_{i+1} of k_i such that the expected travel time from k_i to the destination $k^\star$ is minimized. A policy matrix assigns a successor node for each node and each current state of the background process (i.e., each pair (k, s) with $k \in N, s \in \mathcal{I}$). Given destination $k^\star$, the expected travel time under a policy π and initial state $(K_0, B(t_0))$ is given by

$$J^\pi(K_0, B(t_0)) = \mathbb{E}\left[\sum_{i=0}^{I^\star-1} \tau^{B(t_i)}_{K_i K_{i+1}} \,\middle|\, K_{i+1} = \pi(K_i, B(t_i)), \quad i = 1, \dots, I^\star - 1\right], \tag{4}$$

in which $I^\star$ denotes the number of decision epochs until the destination $k^\star$ is reached, t_i the time of decision epoch i, and $\pi(K_i, B(t_i))$ the action under policy π given the pair $(K_i, B(t_i))$. The optimal policy is now a policy $\pi^\star$ such that

$$\pi^\star = \arg\min_{\pi\in\Pi} J^\pi(K_0, B(t_0)),$$

with Π denoting the set of admissible policies (i.e., policies for which the probability of reaching the destination is one for all initial states). For every non-admissible policy π set $J^\pi(k, s) := \infty$ for all $(k, s) \in N \times \mathcal{I}$.

We conclude this section by deriving expressions for both the expected travel time and the transition probabilities pertaining to a single arc. The resulting quantities are building blocks of the routing policies that will be discussed in the next section. To this end, define, for $d \in [0, d_{k\ell}]$ and $s, s' \in \mathcal{I}$, the expected travel time on $k\ell$:

$$\phi_s(d \mid k, \ell) := \mathbb{E}[\tau^s_{k\ell}(d)], \qquad \Phi(d \mid k, \ell) := (\phi_s(d \mid k, \ell))_{s\in\mathcal{I}}$$

In addition, we will work with the Laplace–Stieltjes transform of the travel time on edge $k\ell$ intersected with the event that upon completion the background state is s':

$$\psi_{ss'}(d \mid \gamma, k, \ell) := \mathbb{E}[e^{-\gamma\tau^s_{k\ell}(d)} 1\{B(\tau^s_{k\ell}(d)) = s'\}], \qquad \Psi(d \mid \gamma, k, \ell) := (\psi_{ss'}(d \mid \gamma, k, \ell))_{(s,s')\in\mathcal{I}\times\mathcal{I}},$$

with $\gamma \in \mathbb{R}_{\geqslant 0}$. Here $1\{E\}$ is the indicator function of an event E, i.e., a random variable that has value 1 if E is true and 0 otherwise; $\mathbb{E}[X\, 1\{E\}]$ is the expectation of a random variable X, but only in the event that E applies. The objects introduced above can be evaluated by conditioning on a possible jump of the background process in a small time-interval. As shown in the following theorem, it leads to a system of linear differential equations, whose solution can be given in terms of matrix exponentials. A proof is given in Appendix B.

Theorem 1. *Given a graph $G = (N, A)$ with a pair of nodes $k, \ell \in N$, $\gamma \in \mathbb{R}_{\geqslant 0}$ and a distance $d \geqslant 0$, it holds that*

$$V_{k\ell}\,\Phi'(d \mid k, \ell) = \mathbf{1} + Q\,\Phi(d \mid k, \ell),$$
$$V_{k\ell}\,\Psi'(d \mid \gamma, k, \ell) = (Q - \gamma I)\Psi(d \mid \gamma, k, \ell),$$

with $V_{k\ell} := \mathrm{diag}\,\{(v_{k\ell}(s))_{s\in\mathcal{I}}\}$ and $\mathbf{1}$ a $|\mathcal{I}|$-dimensional column vector of ones.

A solution for this system of linear differential equations can be written as

$$\Phi(d \mid k, \ell) = \begin{bmatrix} I & \mathbf{0} \end{bmatrix} \exp\left\{\begin{bmatrix} dV^{-1}_{k\ell}Q & dV^{-1}_{k\ell}\mathbf{1} \\ \mathbf{0}^\top & 0 \end{bmatrix}\right\} \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \tag{5}$$

$$\Psi(d \mid \gamma, k, \ell) = \exp\{d\, V_{k\ell}^{-1}(Q - \gamma I)\} \tag{6}$$

with $\mathbf{0}$ *an* $|\mathcal{I}|$*-dimensional column vector of zeros.*

Due to our requirement that all the $v_{k\ell}(s)$ are positive, the matrix $V_{k\ell}^{-1}$ is well-defined. Applying the above result, the expected travel time on an arc can be directly computed using the expression for $\Phi(d \mid k, \ell)$. An expression for the transition probabilities can be found by setting γ equal to 0 in (6):

Corollary 1. *For a pair of nodes* $k, \ell \in N$ *and pair of background states* s, s',

$$\mathbb{P}(B(\tau_{k\ell}^{s}) = s') = \left[\exp\left(d_{k\ell} V_{k\ell}^{-1} Q\right)\right]_{s,s'} \tag{7}$$

Remark 5. The upper left $|\mathcal{I}|\times|\mathcal{I}|$-block of the matrix exponential in (5) is the matrix exponential of $dV_{k\ell}^{-1}Q$. Thus to compute both the expectation $\mathbb{E}[\tau_{k\ell}^{s}]$ and the transition probabilities $\mathbb{P}(B(\tau_{k\ell}^{s}) = s')$, computation of the matrix exponential given in (5) suffices. $\Diamond$

4. Dynamic routing algorithms

Recall that our objective is to minimize the expected travel time for a given OD-pair in a road network in which a vehicle may experience changing velocities and in which one knows the current state of the driving background process $B(t)$. After having provided background on the optimal policy resulting from DP (Section 4.1), we present two Dijkstra-like shortest path algorithms that aim to output a (near-)optimal routing policy. These algorithms are dynamic shortest path algorithms, in that at every new intersection a shortest path algorithm is run again. The analysis of Section 4.2 reveals that the first of these two algorithms, which we have called EDSGER, suffers from the curse of dimensionality. The second presented algorithm, EDSGER*, as introduced in Section 4.3, overcomes this drawback, and offers real-time response (i.e., being in the sub-second scale). Useful implementation details, for these two algorithms as well as for the DP-based approach, will be provided in Section 4.4. We conclude with Section 4.5, which presents speed-up techniques that reduce the state space and/or network size, to be used to further decrease the computational costs.

4.1. Optimal policy by dynamic programming

In this subsection we present an algorithm that outputs an optimal routing policy. As we will see in our numerical examples, this method is prohibitively slow; in this paper we mainly use it as a benchmark for our routing algorithms EDSGER and EDSGER*. We will argue below that optimal policies can be characterized as solutions of a set of Bellman optimality equations (Bellman, 1957a,b; Bertsekas, 1976, 2005; Puterman, 1994; Ross, 1983; Sutton and Barto, 2018). DP methods, competitive algorithms that solve these optimality equations, have high computational costs, so that they are not particularly suitable for real-time routing purposes. Details on the implementation of the competitive DP algorithm are provided in Section 4.4.

We proceed by arguing that algorithms outputting an optimal routing policy suffer from the curse of dimensionality. In case of an MDP, Güner et al. (2012), Kim et al. (2005a,b) and Sever et al. (2013) have characterized an optimal policy to satisfy the Bellman optimality equations. Importantly, this property carries over to our framework, as can be seen as follows. First note that we have a destination node $k^\star$ which is cost-free and absorbing, i.e., $k^\star = \pi(k^\star, s)$ and $J^\pi(k^\star, s) = 0$ for all $s \in \mathcal{I}$. Furthermore, denoting $k_1 := \pi(k_0, B(t_0))$ as the next node under an admissible policy π given the state $(k_0, B(t_0))$ with $k_0 \in N \setminus k^\star$ and $t_0 \in \mathbb{R}$, (4) implies

$$J^\pi(k_0, B(t_0)) = \mathbb{E}\left[\tau_{k_0k_1}^{B(t_0)}\right] + \sum_{s'\in\mathcal{I}} \mathbb{P}\left(B(\tau_{k_0k_1}^{B(t_0)}) = s'\right) J^\pi(k_1, s').$$

Thus, any admissible policy satisfies a set of Bellman equations. Denote with $\mathrm{NB}(k)$ the set of neighbors of a node $k \in N$, i.e., $\ell \in \mathrm{NB}(k)$ only if $k\ell \in A$ (note that $k \notin \mathrm{NB}(k)$ as it is not allowed to wait at a node). An optimal policy $\pi^\star$ can then be given through the Bellman optimality equations:

$$\pi^\star(k_0, B(t_0)) = \operatorname*{arg\,min}_{k_1\in\mathrm{NB}(k_0)} \left\{ \mathbb{E}\left[\tau_{k_0k_1}^{B(t_0)}\right] + \sum_{s'\in\mathcal{I}} \mathbb{P}\left(B(\tau_{k_0k_1}^{B(t_0)}) = s'\right) J^{\pi^\star}(k_1, s') \right\}. \tag{8}$$

The expected travel times and transition probabilities in (8) can be computed relying on the expressions derived in the previous section (in particular Eqs. (5) and (7)).

Algorithms that solve the Bellman optimality equations have received substantial attention in the literature. Examples of frequently used solution methods are the dynamic programming (DP) methods (e.g. value iteration and policy iteration) and linear programming. It should be realized that the size of our state space may explode: even in a simple setup in which every link has only two possible background states, a network with $|A|$ links leads to a state space of size $2^{|A|}$. We will therefore employ value iteration (VI) to solve the optimality equations (Sutton and Barto, 2018). For large state spaces the VI algorithm does however suffer from high complexity:

- in every iteration of the algorithm the right hand side of (8) is computed for every possible $B(t_0) \in \mathcal{I}$, so that the complexity of the value iteration algorithm contains a term $|\mathcal{I}|^2$;
- computation of the expectations and the transition probabilities, which requires the evaluation of a matrix exponential as in (7), will be intractable when the transition rate matrix Q is large;

- a large policy matrix will lead to high memory costs.

DP suffers from the curse of dimensionality; expanding the network leads to an exponential increase in the size of the state space and corresponding exponential increase in computational costs.

Based on the above, it can be concluded that VI has limited potential in practical settings. We will therefore introduce two dynamically-used Dijkstra-like shortest path algorithms, EDSGER and EDSGER*; here 'dynamically-used' means that the shortest path algorithm is run at any decision epoch along the way. EDSGER provides near-optimal solutions, but, similar to VI, suffers from the curse of dimensionality. EDSGER* is an adaptation of EDSGER that uses the local-r-correlation assumption to overcome the curse of dimensionality.

4.2. EDSGER *algorithm*

The EDSGER algorithm (where EDSGER is an abbreviation of 'Expected Delay in Stochastic Graphs with Efficient Routing') can be seen as a dynamically-used stochastic version of the A*-algorithm. More concretely, the method works as follows:

- At every decision epoch (every arrival at a node, that is) we use a stochastic shortest path algorithm to identify a path with lowest expected cost, from the current node to the destination, provided that one is *not* allowed to change the route along the way. The resulting policy $\pi_{\rm E}$ is to travel to the next node along this path. Thus, given that a vehicle is at node $k_0 \in N$ at time t_0, EDSGER uses a shortest path algorithm with input $(k_0, B(t_0))$ to output a path from k_0 to $k^\star$. Then, denoting k_1 for the first node in this path, we have that $\pi_{\rm E}(k_0, B(t_0)) = k_1$.
- This procedure is then 'dynamically-used' in the sense that it is repeated when arriving at k_1, so as to identify $k_2 := \pi_{\rm E}(k_1, B(t_1))$. Given the fact that the state of the background process may have changed while traveling from k_0 to k_1, this next node may differ from the one that was selected at k_0. The procedure thus exploits the information currently available.
- One proceeds along these lines until the destination node $k^\star$ has been reached.

A first important remark is that this procedure is not necessarily minimizing the expected delay, i.e., the resulting route does not always coincide with the one generated by the DP-based algorithm discussed earlier. We note that one reason for this potential loss is the fact that EDSGER uses a stochastic A*-like shortest path algorithm. In a network with *stochastic* travel times the notion that a subpath of a shortest path is a shortest path, a property on which Dijkstra and A* are built, does not generally hold, so that the output is not guaranteed to be the shortest path in expectation. An example (in our context) of this finding, which is often attributed to Hall (1986), is given in the following example.

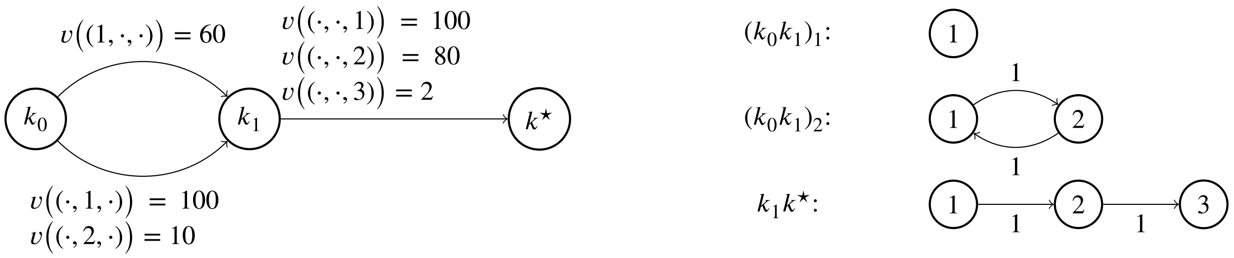
Example. A vehicle wants to travel from k_0 to $k^\star$ in the network of Fig. 4, in which each arc is 60 km long. A state of the background process $B(t)$ is a 3-tuple consisting of the state of $(k_0k_1)_1$ (the upper link between k_0 and k_1), $(k_0k_1)_2$ (the lower link between k_0 and k_1) and $k_1k^\star$, in that order. A closer look at the velocities reveals that the speed on link $(k_0k_1)_1$ is always 60 km/h, whereas the speeds on links $(k_0k_1)_2$ and $k_1k^\star$ can take two and three values, respectively. For instance, the speed on link $(k_0k_1)_2$ is 100 km/h if this link is in state 1 and 10 km/h if this link is in state 2 (irrespective of the states of the other links). We consider the situation that the initial state of the background process is set $B(t_0) = (1,1,1)$. Using Theorem 1 we find the costs of using the upper and lower link:

$$\mathbb{E}[\tau^{B(t_0)}_{(k_0k_1)_1}] = 1\,\text{h}, \qquad \mathbb{E}[\tau^{B(t_0)}_{(k_0k_1)_2}] = 1.02\,\text{h}.$$

EDSGER would therefore advise to travel via $(k_0k_1)_1$. Comparison of the expected travel times of the two paths between k_0 and $k^\star$ does however reveal that this is not optimal, as these are 12.21 h and 11.58 h for traveling via $(k_0k_1)_1$ and $(k_0k_1)_2$ respectively. Thus even though $(k_0k_1)_1$ is the optimal path to k_1, it is not in the expected shortest path to $k^\star$. A reason for this phenomenon lies in the fact that EDSGER only takes the expectation into account, while variability plays a role here as well, as this variability directly relates to different conditions on future links. To see this, first note that traveling to k_1 via the upper link will always take 1 h. Upon arrival in k_1 there is a high probability that the background process of $k_1k^\star$ has transitioned to a state with reduced speed. If link $(k_0k_1)_2$ spends most time in state 1, traveling to k_1 via the lower link can be done within 1 h. In that case there is still a high probability that link $k_1k^\star$ can (partly) be traveled at speed 100. Traveling via $(k_0k_1)_2$ therefore yields a lower expected travel time than traveling via $(k_0k_1)_1$, despite the fact that we have $\mathbb{E}[\tau^{B(t_0)}_{(k_0k_1)_1}] < \mathbb{E}[\tau^{B(t_0)}_{(k_0k_1)_2}]$. $\diamondsuit$

A second reason for the potential loss in optimality lies in the fact that EDSGER does not exploit its dynamical use in its fullest extent: due to the fact that in the DP approach one *knows* that one is allowed to change the path along the way, it potentially leads to lower cost than the dynamically used version of EDSGER. This effect is highlighted in the example provided below.

Example. The objective is to minimize the expected travel time from k_0 to $k^\star$ in the network of Fig. 5 with $B(t_0) = (1,1,1,1)$. Every arc has two states and a vehicle can drive 100 km/h on an arc if the state of the arc is 1 and 80 km/h otherwise. The transitions from state 1 to state 2 have rate 0.1 and the transitions from state 2 to state 1 have rate 1 for all arcs. We first note that the expected travel times on all paths are equal, such that EDSGER cannot distinguish between these paths. However, EDSGER does not take into account that, in case k_1 is chosen as next node, updated information about the speeds on the two optional paths from k_1 to $k^\star$ can be used to pick the optimal path given this updated information. VI does use this information and as such picks the route via k_1, yielding a 37 s improvement in expected travel time. Note that, for a travel distance of 100 km, this is only a very small improvement. $\diamondsuit$



$v((1,\cdot,\cdot)) = 60$
$v((\cdot,\cdot,1)) = 100$
$v((\cdot,\cdot,2)) = 80$
$v((\cdot,\cdot,3)) = 2$
k_0
k_1
$k^\star$
$v((\cdot,1,\cdot)) = 100$
$v((\cdot,2,\cdot)) = 10$
$(k_0k_1)_1$:
$(k_0k_1)_2$:
$k_1k^\star$:
1
2
3

Fig. 4. Example in which EDSGER is not optimal, as optimal subpath is not in optimal path.

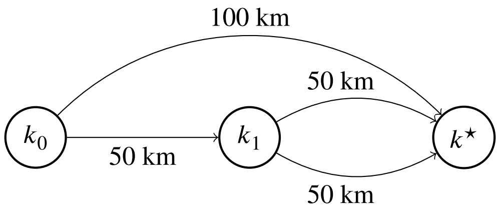


Fig. 5. Example in which EDSGER is not optimal, as it does not fully exploit the possibility to change the chosen route.

Whereas the above instance shows that in principle we can construct cases in which DP leads to lower expected delay than EDSGER, the experiments of Section 5 show that the difference is typically tiny (if there is a difference at all). Importantly, the modest loss of optimality that EDSGER experiences is compensated by a potentially significant improvement in terms of the computational cost: the experiments show that the computational effort of EDSGER can be substantially lower than that of the VI algorithm. This is because, in contrast to VI, EDSGER does not necessarily compute the expectation and transition probabilities on all arcs in the network.

We now provide a detailed description of the shortest path algorithm that is used within EDSGER. As discussed above, EDSGER uses this stochastic shortest path algorithm to output a path from the current node to the destination. The first node in this path defines the next action, i.e., the policy of EDSGER is to travel to this node.

The shortest path algorithm is a stochastic version of the A*-algorithm. It assigns a label to every node in the graph and updates these labels iteratively. The initial labels are $D_{k_0} = 0$ for the source k_0 and $D_k = \infty$ for $k \in N \setminus \{k_0\}$. A set $\mathcal{V}$ is used to store nodes with labels that are no longer altered by subsequent steps of the algorithm, and is initialized by $\mathcal{V} := \emptyset$. In every iteration of the algorithm a new 'current node' c is chosen according to

$$c := \arg\min_{k\in N\setminus\mathcal{V}}\{D_k + \text{LB}_k\}. \tag{9}$$

Here LB_k is a lower bound on the travel time from node k to destination $k^\star$. Since the maximum speeds and the distances of all arcs are known, lower bounds on the arc travel times can be computed in an elementary way. Applying Dijkstra's algorithm on a graph with these lower bounds yields LB_k for $k \in N$.

With the current node c chosen according to (9), the earlier derived expressions (5) and (7) are used to compute

$$d_k := D_c + \sum_{s\in\mathcal{I}} p_c^s\,\mathbb{E}[\tau_{ck}^s] = D_c + \boldsymbol{p}_c^\top \boldsymbol{\Phi}_{ck} \tag{10}$$

$$p_k^{s'} := \sum_{s\in\mathcal{I}} p_c^s\,\mathbb{P}(B(\tau_{ck}^s) = s') \tag{11}$$

for all $k \in \text{NB}(c)$ and $s' \in \mathcal{I}$. In the expressions (10) and (11) we let p_k^s denote the probability that upon arrival in $k \in N$ the background state is $s \in \mathcal{I}$, and $\boldsymbol{p}_k = (p_k^s)_{s\in\mathcal{I}}$ is the vector of these probabilities. In addition, $\boldsymbol{\Phi}_{ck}$ is the $|\mathcal{I}|$-dimensional vector with entries $\mathbb{E}[\tau_{ck}^s] = \phi_s(d_{ck} \mid c, k)$ for $s \in \mathcal{I}$. Note that, as D_c can be seen as an estimate of the expected travel time from k_0 to c, d_k is an estimate for the expected travel time from k_0 to k. The labels of the neighbors $k \in \text{NB}(c) \setminus \mathcal{V}$ are now updated, i.e., if $d_k < D_k$ we set $D_k = d_k$ and store the vector $\boldsymbol{p}_k$ for this node (replacing a previous stored value if present). We also store the proposed path to k, which is the stored path to c with k itself appended. After performing this updating step, c is added to $\mathcal{V}$ and a new current node c is chosen, again according to (9).

The described procedure is now repeated until the destination node $k^\star$ is set as the current node c. The path the algorithm outputs is the stored path for $k^\star$. In the implementation of the algorithm a heap H can be used to reduce the complexity of the algorithm, similar to the use of the heap in Dijkstra's algorithm (Goldberg and Tarjan, 1996). Pseudocode is provided in Algorithm 1.

Despite the fact that, in contrast to the VI algorithm, EDSGER does not necessarily compute the expectation and transition probabilities for all arcs in the network, the algorithm still suffers from the curse of dimensionality. In this respect, note that the cost of evaluating the matrix exponential in (5) will contribute substantially to the cost of EDSGER. Moreover, in case of a large state space, the dimension of Q will make this evaluation intractable. To overcome this drawback, we will introduce the EDSGER*

```
Result: path from $k_0$ to $k^\star$ given $B(0) = s$
Initialization: $D_{k_0} = 0, D_k = \infty$ for $k \neq k_0$, heap $H = \{(D_{k_0} + \text{LB}_{k_0}(\text{key}), p_{k_0}, k_0, \{k_0\}\ (\text{path})\}, \mathcal{V} = \emptyset$;
while $H$ nonempty do
    1. Extract tuple $(D_k + \text{LB}_k, p_k, k, \text{path})$ with minimum first entry (key) from $H$;
    2. If $k = k^\star$ quit and return path. Else if $k \in \mathcal{V}$ go back to step 1. Else continue;
    3. for neighbor $k'$ in $\text{NB}(k) \setminus \mathcal{V}$ do
        a. Compute $d_{k'}$ and $p_{k'}$ with (5) and (7);
        b. If $d_{k'} < D_{k'}$ set $D_{k'} = d_{k'}$ and insert $(D_{k'} + \text{LB}_{k'}, p_{k'}, k', \text{path} + \{k'\})$ in $H$;
    end
    4. Add $k$ to $\mathcal{V}$
end
```

Algorithm 1: Outline shortest path algorithm in EDSGER

algorithm. EDSGER* can be regarded as an improved version of EDSGER, as it exploits the local-correlation structure of the background process to reduce the computational costs of EDSGER drastically.

4.3. EDSGER* *algorithm*

Both the VI algorithm and the EDSGER algorithm do not use the assumption of local-r-correlation (Assumption 1), which states that the velocity on arc $k\ell \in A$ is only dependent on the congestion levels of the arcs at most r arcs away. The main idea behind the EDSGER* algorithm is to exploit this assumption, so as to overcome the curse of dimensionality. Numerical examples in the next section will show that EDSGER* is indeed highly efficient and offers real-time response, while still providing near-optimal results.

Similar to EDSGER, at any decision epoch EDSGER* uses a shortest path algorithm that aims to output a minimal path from the current node to the destination. The next arc in this path is traveled and upon arrival in a new node the shortest path algorithm is called again. We denote the resulting routing policy by $\pi_{\text{E}^\star}$. The shortest path algorithm used in EDSGER* (Algorithm 2 below) is very similar to the algorithm used in EDSGER, but EDSGER* exploits Assumption 1, in combination with a specific approximation, to drastically speed up the computation of $d_{k'}$ and $p_{k'}$.

The main idea behind the EDSGER*-algorithm is that, under the assumption of local-r-correlation, the velocity levels on an arc are only dependent on the state of the background processes on the arcs in $A^r_{k\ell}$. This implies that the expectation of the travel time can be computed using just these processes. As above, we write $A = \{k_1\ell_1, \dots, k_n\ell_n\}$. Denote with $Q^r_{k\ell}$ the transition rate matrix of the process $B^r_{k\ell}(t) = (X_{k_i\ell_i}(t), k_i\ell_i \in A^r_{k\ell})$ with state space $\mathcal{I}^r_{k\ell}$.

The dynamics on arc $k\ell$ can be completely described by the process $B^r_{k\ell}(t)$, as the velocity levels on arc $k\ell$ depend only on the state of this process. Thus, the velocity on arc $k\ell$ at time t, defined as $v_{k\ell}((X_{k_i\ell_i}(t))_{k_i\ell_i\in A})$, only depends on the values of entries $k_i\ell_i \in A^r_{k\ell}$. We can therefore write the velocity on arc $k\ell$ at time t as $v_{k\ell}((X_{k_i\ell_i}(t))_{k_i\ell_i\in A^r_{k\ell}})$. Denote the truncation of $s = (s_{k_i\ell_i})_{k_i\ell_i\in A} \in \mathcal{I}$ to the entries corresponding to links in $A^r_{k\ell}$ by $s^r_{k\ell}$, i.e., $s^r_{k\ell} = (s_{k_i\ell_i})_{k_i\ell_i\in A^r_{k\ell}}$. Then we write $v_{k\ell}(s) = v_{k\ell}(s^r_{k\ell})$ for the velocity level on arc $k\ell$ whenever $s \in \mathcal{I}$.

Theorem 2. *Denoting $\Phi^r(d \mid k, \ell) = (\mathbb{E}[\tau^s_{k\ell}(d)])_{s^r_{k\ell}\in\mathcal{I}^r_{k\ell}}$, it holds that*

$$\Phi^r(d \mid k, \ell) = \begin{bmatrix} I & \mathbf{0} \end{bmatrix} \exp\left\{ \begin{bmatrix} d\,(V^r_{k\ell})^{-1} Q^r_{k\ell} & d\,(V^r_{k\ell})^{-1}\mathbf{1} \\ \mathbf{0}^\top & 0 \end{bmatrix} \right\} \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}. \tag{12}$$

Here $V^r_{k\ell}$ is a diagonal matrix with entries $v_{k\ell}(s^r_{k\ell})$ for $s^r_{k\ell} \in \mathcal{I}^r_{k\ell}$.

Proof. The claim follows from Theorem 1, local-r-correlation (Assumption 1), and the independence of the Markov processes on the arcs. □

Particularly when r is relatively small, this way of computing the expected per-edge travel times yields significant computational savings, due to the fact that the dimension of $Q^r_{k\ell}$ no longer grows exponentially with the number of arcs. This does however not directly yield a solution to the curse of dimensionality, as the computation of the transition probabilities still involves the matrix Q. We therefore propose to use the following (typically highly accurate) approximation for the transition probabilities. Recall that the transition probabilities of a Markov process $B(t)$ with transition rate matrix A after a time $t \in \mathbb{R}$ can be expressed in terms of a matrix exponential:

$$\mathbb{P}(B(t) = s \mid B(0) = s') = [e^{tA}]_{s',s}.$$

Note furthermore that we have a good estimate, D_c, for the travel time from k_0 to $c \in N$. Based on these two observations we approximate p^s_c, the probability that $B^r_{k\ell}(t) = s$ upon arrival in c given that $B(0) = s'$, by

$$[e^{D_c Q^r_{ck}}]_{(s')^r_{k\ell},s}.$$

This results in the following expression for d_k:

$$d_k := D_c + \sum_{s\in\mathcal{I}^r_{ck}} [e^{Q^r_{ck}D_c}]_{(s')^r_{k\ell},s}\,\mathbb{E}\big[\tau^{s^r_{ck}}_{ck}\big] = D_c + [e^{Q^r_{ck}D_c}]_{(s')^r_{k\ell}}\Phi^r(d_{ck} \mid c,k); \tag{13}$$

cf. (10). The corresponding pseudocode is given in Algorithm 2.

```
Result: path from $k_0$ to $k^\star$ given $B(0) = s$
Initialization: $D_{k_0} = 0, D_k = \infty$ for $k \neq k_0$, heap $H = \{D_{k_0} + \text{LB}_{k_0}$ (key), $k_0, \{k_0\}$ (path)$\}, \mathcal{V} = \emptyset$;
while $H$ nonempty do
    1. Extract tuple $(D_k + \text{LB}_k, k, \text{path})$ with minimum key from $H$;
    2. If $k = k^\star$ quit and return path. Else if $k \in \mathcal{V}$ go back to step 1. Else continue;
    3. for neighbor $k'$ in $\text{NB}(k) \setminus \mathcal{V}$ do
        a. Compute $d_{k'}$ with (13);
        b. If $d_{k'} < D_{k'}$ set $D_{k'} = d_{k'}$ and insert $(D_{k'} + \text{LB}_{k'}, k', \text{path} + \{k'\})$ in $H$;
    end
    4. Add $k$ to $\mathcal{V}$
end
```

Algorithm 2: Outline shortest path algorithm in EDSGER*

4.4. Implementation details

We now discuss several implementation details for the VI-, EDSGER-, and EDSGER*-algorithms. We will in particular show how to rewrite specific computations in a form that allows the application of efficient numerical functions. Moreover, we will recommend the use of efficient data structures. We start by discussing the use of these techniques for the VI algorithm and continue with the EDSGER-, and the EDSGER*-algorithms.

∘ *The value iteration algorithm.* As discussed in Section 4.1, this DP algorithm solves the Bellman optimality equations and outputs a routing policy that minimizes our objective function. It is an iterative procedure that assigns values $\tilde{J}^{\pi^\star}_0(k,s)$ to all $(k,s) \in N \times \mathcal{I}$ and updates these values such that they converge to $J^{\pi^\star}$, the set of optimal values. Concretely, the algorithm sets $\tilde{J}^{\pi^\star}_0(k^\star,s) = 0$ and $\tilde{J}^{\pi^\star}_0(k,s) = \infty$ for $k \in N, k \neq k^\star, s \in \mathcal{I}$ and iteratively updates these values according to the scheme

$$\tilde{J}^{\pi^\star}_i(k,s) = \min_{\ell\in\text{NB}(k)} \left\{ \mathbb{E}[\tau^s_{k\ell}] + \sum_{s'\in\mathcal{I}} \mathbb{P}(B(\tau^s_{k\ell}) = s')\tilde{J}^{\pi^\star}_{i-1}(\ell,s') \right\}, \tag{14}$$

cf. the scheme (8). Convergence of the iterative procedure has been widely studied in the literature; we refer to e.g. Bertsekas and Tsitsiklis (1991) and Bonet (2007) and references therein. Specifically, it has been proven that $\tilde{J}^{\pi^\star}_i(k,s)$ converges to $J^{\pi^\star}(k,s)$ for all $k \in N, s \in \mathcal{I}$.

The expectations and transition probabilities in the updating step (14) can be computed by applying Theorem 1 and Corollary 1. As was already noted in Remark 5, the computation of the $(|\mathcal{I}|+1)\times(|\mathcal{I}|+1)$-dimensional matrix exponential

$$\exp\left\{ \begin{bmatrix} d_{k\ell}V^{-1}_{k\ell}Q & d_{k\ell}V^{-1}_{k\ell}\mathbf{1} \\ \mathbf{0}^\top & 0 \end{bmatrix} \right\} \tag{15}$$

suffices. Note that the upper left $|\mathcal{I}|\times|\mathcal{I}|$- and upper right $|\mathcal{I}|\times 1$-block correspond to $\Big(\mathbb{P}(B(\tau^s_{k\ell}) = s')\Big)_{(s,s')\in\mathcal{I}\times\mathcal{I}}$ and $\Big(\mathbb{E}[\tau^s_{k\ell}]\Big)_{s\in\mathcal{I}}$, respectively. This means that, by writing i_s for the index of s in $\mathcal{I}$, $\boldsymbol{p}^s_{k\ell} = (\mathbb{P}(B(\tau^s_{k\ell}) = s'))_{s'\in\mathcal{I}}$ and $\tilde{\boldsymbol{J}}^{\pi^\star}_{i-1}(\ell) = (\tilde{J}^{\pi^\star}_{i-1}(\ell,s'))_{s'\in\mathcal{I}}$, the iterative step (14) can be written as

$$\tilde{J}^{\pi^\star}_i(k,s) = \min_{\ell\in\text{NB}(k)} \left\{ \begin{bmatrix} (\boldsymbol{p}^s_{k\ell})^\top & \mathbb{E}[\tau^s_{k\ell}] \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{J}}^{\pi^\star}_{i-1}(\ell) \\ 1 \end{bmatrix} \right\} = \min_{\ell\in\text{NB}(k)} \left\{ \exp\left\{ \begin{bmatrix} d_{k\ell}V^{-1}_{k\ell}Q & d_{k\ell}V^{-1}_{k\ell}\mathbf{1} \\ \mathbf{0}^\top & 0 \end{bmatrix} \right\}_{i_s} \begin{bmatrix} \tilde{\boldsymbol{J}}^{\pi^\star}_{i-1}(\ell) \\ 1 \end{bmatrix} \right\}. \tag{16}$$

Importantly, it now suffices to compute the *product of* the matrix exponential (15) and the vector $(\tilde{\boldsymbol{J}}^{\pi^\star}_{i-1}(\ell), 1)^\top$ to evaluate the objective function in (16). For such a product, also known as the *action* of a matrix exponential, most programming software include compiled functions, examples of which are MatrixExp[ ] in Mathematica and scipy.linalg.expm( ) in Python; these functions are typically considerably faster than first computing the matrix exponential and the vector, and subsequently their product.

Besides the evaluation of the matrix exponential, the computational costs of DP are strongly affected by the data structure used for the Q-matrix. Implementing the Q-matrix in a sparse way significantly decreases the costs of constructing and storing this matrix. Sparsity additionally reduces the costs of updating the values $\tilde{J}^{\pi^\star}_i(k,s)$, since compiled matrix exponential functions in programming software are generally faster in case the input is a sparse matrix.

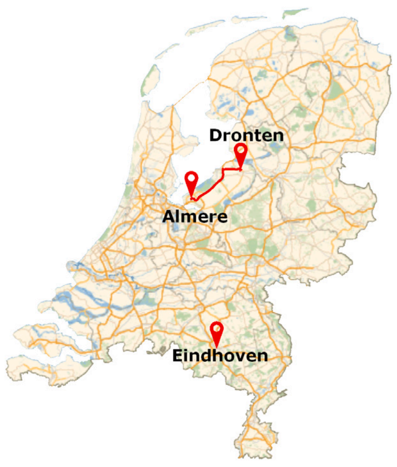


Fig. 6. Distance Almere-Dronten to Eindhoven.

∘ *The* EDSGER- *and* EDSGER⋆*-algorithms.* Also in the implementation of EDSGER and EDSGER⋆, a significant speed-up can be achieved by (i) working with the action of the matrix exponential and (ii) exploiting the sparsity of the Q-matrix. Here we will focus on the former speed-up, i.e., the one due to the use of the action of the matrix exponential, as the motivation for using a sparse Q-matrix is identical to the one in the value iteration case.

Note that EDSGER uses a matrix exponential in the computations of d_k and $\boldsymbol{p}_k$ (see (10) and (11)). If we let $\boldsymbol{p}_{ck}$ be the matrix $(\mathbb{P}(B(\tau^s_{ck}) = s'))_{(s,s')\in\mathcal{I}\times\mathcal{I}}$, we can derive the following equivalence:

$$\begin{bmatrix}\boldsymbol{p}_k^\top & d_k - D_c\end{bmatrix} = \boldsymbol{p}_c^\top \begin{bmatrix}\boldsymbol{p}_{ck} & \boldsymbol{\Phi}_{ck}\end{bmatrix} = \begin{bmatrix}\boldsymbol{p}_c^\top & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{p}_{ck} & \boldsymbol{\Phi}_{ck}\\ \boldsymbol{0}^\top & 1\end{bmatrix} = \begin{bmatrix}\boldsymbol{p}_c^\top & 0\end{bmatrix}\exp\left\{\begin{bmatrix}d_{ck}V_{ck}^{-1}Q & d_{ck}V_{ck}^{-1}\boldsymbol{1}\\ \boldsymbol{0}^\top & 0\end{bmatrix}\right\}.$$

Recall that for a vector $\boldsymbol{p}$ and square matrix A for which $\boldsymbol{p}e^A$ exists, we have that $(\boldsymbol{p}e^A)^\top = (e^A)^\top \boldsymbol{p}^\top = e^{A^\top}\boldsymbol{p}^\top$, yielding

$$\begin{bmatrix}\boldsymbol{p}_k\\ d_k\end{bmatrix} = \begin{bmatrix}\boldsymbol{0}\\ D_c\end{bmatrix} + \exp\left\{\begin{bmatrix}d_{ck}Q^\top V_{ck}^{-1} & \boldsymbol{0}\\ d_{ck}\boldsymbol{1}^\top V_{ck}^{-1} & 0\end{bmatrix}\right\}\begin{bmatrix}\boldsymbol{p}_c\\ 0\end{bmatrix}.$$

The same procedure can be followed to compute (13) in the implementation of EDSGER⋆.

4.5. Decreasing state space

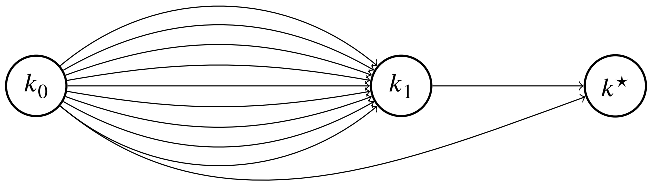
EDSGER⋆ yields significant computational savings compared to DP (using value iteration) and the EDSGER algorithm. However, further substantial reductions of the computational costs are possible. This is achieved by performing preprocessing steps to reduce the network size and the state space of the background process. These are general speed-up techniques and can also be performed in the context of value iteration or EDSGER.

4.5.1. Decreasing network size

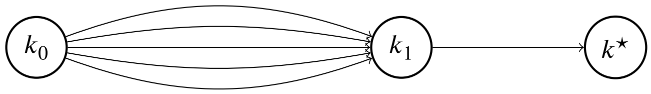
Three specific ideas will be discussed in greater detail. The first one uses Yen's algorithm (Yen, 1970, 1971) to reduce the size of the network, by deleting arcs that cannot be on shortest paths. The other proposed ideas relate to decreasing the state space of the background process: the first idea considers hitting probabilities and excludes background states for which the hitting probability is below a given threshold, whereas the second uses historical data to exclude background states which are (extremely) rare events.

The first proposed speed-up technique decreases the network size by deleting certain nodes and arcs in the network. The technique is motivated by the fact that not all arcs in the network will be considered by travelers. The network in Fig. 6 depicts the shortest route (in distance) from Almere to Dronten, in the Dutch road system. In case there is congestion on this route a traveler might wish to take a different route. One can argue, however, that the conditions in the network will never be such that the traveler wishes to use an arc around distant cities, such as, in our example, Eindhoven. As a consequence, the roads around the city of Eindhoven can be omitted when considering the roads to travel from Almere to Dronten.

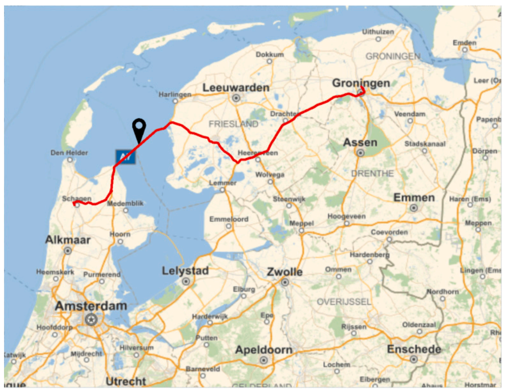
We now present a procedure, Yen's algorithm, that algorithmically determines the arcs that can be excluded. The algorithm is first used to derive the m shortest paths from source k_0 to destination $k^\star$ in a deterministic network, in which a vehicle can drive at maximum velocity levels. Denote with $G' = (N', A')$ the network that solely consists of the nodes and arcs in these m shortest paths. Then we propose to reduce the network size by only considering G' in the routing problem. Note that the value of m should not be too large, as this will not yield computational savings, but also not too small, as this might eliminate arcs that are on the optimal route. In the numerical experiments of Section 5, the final example considers routing for different values of m, and shows that, in a typical setting, a value between 1 and 5 yields good results.



(a) Before Yen's algorithm.



(b) After Yen's algorithm.



(c) Real-world example.

Fig. 7. Yen's algorithm deletes important alternative route.

Güner et al. (2012) propose to reduce the network G to a network similar to G', and compute a routing policy for this reduced network. We however propose an additional step in the construction of a reduced network, in which we add edges to G' which offer alternative routes in case these do not exist in G'. An example is provided in Fig. 7. Applying Yen's m-shortest path algorithm with $m = 5$ in the graph of Fig. 7(a) yields Fig. 7(b), which shows that arc $k_1k^\star$ is present in every of the 5 shortest paths outputted by Yen's algorithm. This is undesirable, since this arc can be congested during the period of travel, meaning a traveler might prefer arc $k_0k^\star$. After an application of Yen's algorithm, $k_0k^\star$ is however no longer in the considered network and therefore not in the routing policy. A real-world example is shown in Fig. 7(c), where a vehicle wishes to travel from Groningen to Schagen. Application of Yen's m-shortest path algorithm may lead to m paths that all contain the Afsluitdijk dam (black marker). A few times a year the Afsluitdijk is closed for a few hours due to a major accident and in this case there are no alternative routes in G'.

To avoid a situation as described above, in which there is no alternative to a heavily congested arc, a second step is introduced, in which arcs that offer these alternative routes are added to G'. To this end, it is first checked which arcs are in all of the m shortest paths. Yen's algorithm is then repeatedly used to find l shortest paths in the network in which one of these arcs is deleted each time. The sets of l shortest paths are added to G' and this network is used to construct a routing policy. Observe that, since these l shortest paths are included to guarantee alternatives in case of congestion, and such alternatives do already arise for small l, choosing l equal to 1 or 2 typically suffices.

To analyze the dynamics on $k\ell \in A'$, we only need information on the state of the process $B^r_{k\ell}(t)$. Decreasing the network size as described above can therefore yield a significant reduction in the size of the state space as well: for $k\ell \in A \setminus (\cup_{i\in A'} A^r_i)$ the corresponding Markov processes $X_{k\ell}(t)$ give no information on the dynamics on G' and can therefore be omitted in any further analysis.

4.5.2. Use of bounds on hitting probabilities

One technique for reducing the state space of the background process is based on hitting probabilities. The idea is to only include, for a small number ϵ, background states $s \in \mathcal{I}$ for which $\mathbb{P}(B(t) = s) > \epsilon$ for some $t \leqslant T$, with $T > 0$ denoting the time of arrival at destination $k^\star$. Since this time horizon is evidently not known in advance, one could work with a $M > 0$ that serves as a crude upper bound on T. Given an initial background state $s \in \mathcal{I}$ an upper bound for the hitting probabilities is then given by

$$\mathbb{P}(\exists t \in [0, M] : B(t) = s' \mid B(0) = s) \leqslant \prod_{i=1}^{|A|} \mathbb{P}(\exists t \in [0, M] : X_{k_i\ell_i}(t) = s'_{k_i\ell_i} \mid X_{k_i\ell_i}(0) = s_{k_i\ell_i}) \tag{17}$$

In case $\mathcal{S}_{k_i\ell_i} = \{1, 2\}$ for some $i \in \{1, \dots, |A|\}$ with transition rates λ_i, μ_i, we have

$$f_{12}(M) := \mathbb{P}(\exists t \in [0, M] : X_{k_i\ell_i}(t) = 2 \mid X_{k_i\ell_i}(0) = 1) \leqslant \mathbb{P}(Y_{\lambda_i}(t) \leqslant M) = 1 - e^{-\lambda_i M},$$

where $Y_{\lambda_i}(t) \sim \text{Exp}(\lambda_i)$. We equivalently have that $f_{21}(M) \leqslant 1 - e^{-\mu_i M}$, and by definition $f_{11}(M) = f_{22}(M) = 1$.

Whenever $\mathcal{S}_{k_i\ell_i} = \{1, \dots, n_{k_i\ell_i}\}$ with $n_{k_i\ell_i} > 2$, we can rely on standard results on sums of independent exponentially distributed random variables (Bibinger, 2013) in order to find an upper bound on the probabilities in (17). An example is given in Appendix C, in which it is shown how to use these results in case the Markov process on an arc is a birth–death process. Now, if (an upper bound of) the derived bound from (17) is smaller than some predefined $\epsilon > 0$, we do not consider the corresponding background state $s' \in \mathcal{I}$ in our further analysis.

4.5.3. Use of historical data

A technique to further reduce the background state space is directly based on historical data of the process $B(t)$ on G. When this data is available, it can be analyzed to determine which states $s \in \mathcal{I}$ are most common to occur in reality and which states have never occurred during the time period during which the historical data was recorded. For example, in a network with 50 arcs in which each arc has two states (congested and uncongested), the situation in which *all* arcs are congested can theoretically occur, but will not be observed in real road networks, as will be confirmed by the historical data. Hence a strategy could be to eliminate all states $s \in \mathcal{I}$ that have never been attained by the process $B(t)$, so as to reduce the state space of this background process.

The three techniques discussed in this section can be used as preprocessing steps in dynamic routing. In case of EDSGER and EDSGER*, they can be applied prior to every call of the shortest path algorithm. An outline of the resulting procedure is given in Algorithm 3.

```
Result: Travel policy from $k_0$ to $k^\star$
initialization: network $G = (N, A)$, $B(0) = s$, Node = $k_0$;
Step 1: Preliminary node deletion;
    a. Derive $m$ shortest paths on $G$ to form $G'$;
    b. Identify arcs $\{k_1\ell_1, \dots, k_n\ell_n\}$ that are on all shortest paths;
    c. Derive $l$ shortest paths on $G \setminus k_i\ell_i$ for $i = 1, \dots, n$ and add to $G'$;
    d. Reduce state space background process by looking at deleted arcs;
Step 2: Preliminary background state deletion;
    a. Delete states with hitting probability constraint;
    b. Delete states by historical data analysis;
Step 3: Use shortest path algorithm;
    a. Determine path from $k_0$ to $k^\star$;
    b. Travel to outputted next node. Stop if this is $k^\star$. Otherwise return to Step 1 with state upon arrival as initial state;
```
Algorithm 3: Outline Dynamic Routing with Preprocessing steps

5. Numerical experiments

This section presents a series of numerical experiments that demonstrate that the EDSGER* algorithm performs near-optimally with high efficiency. That is, corresponding to the earlier introduced notions of *distance-to-optimality* and *efficiency*, EDSGER* outputs a value close to the minimally achievable value while essentially being real-time. To substantiate this claim, we have considered a broad range of traffic scenarios and networks of various dimensions.

More specifically, we first consider a small network and show that value iteration (VI), EDSGER, and EDSGER* outperform three deterministic algorithms (in terms of distance-to-optimality), in case of a simple as well as a more sophisticated background process. Second, we increase the size of the network to show that the run-time of VI and EDSGER grows exponentially in the network size, whereas the run-time of EDSGER* is substantially less affected. Importantly, EDSGER* still yields close-to-optimal results in these larger networks. Last, we consider routing in a network of realistic size and show that EDSGER* is still highly efficient and nearly optimal.

The first two experiments consider routing on the highway system around Amsterdam. They compare the distance-to-optimality of VI, EDSGER and EDSGER* to three deterministic algorithms. The three deterministic algorithms, used as benchmarks, all employ the A*-algorithm. The first deterministic algorithm, as before referred to as 'Deterministic Static' (abbreviated to DS), applies the A*-algorithm on the network with *maximum* velocities. The second deterministic algorithm, which we will call 'Stationary Static' (abbreviated to SS), employs the stationary expected travel times as link weights. Importantly, as the maximum speeds as well as the expected travel times are fixed, both algorithms execute the A*-algorithm once to determine the complete travel path, which is followed until the destination is reached. The third benchmark algorithm, called 'Deterministic Dynamic' (abbreviated to DD), does use the available information on the velocities in the network. Every intersection the algorithm calls the A*-algorithm on a network with the *current* velocities to determine the next arc to travel.

Experiment 1 uses a Markovian background process in which every arc has just two states: congested or uncongested. It is shown that VI, EDSGER and EDSGER* outperform the deterministic algorithms in terms of distance-to-optimality in this simple setup.

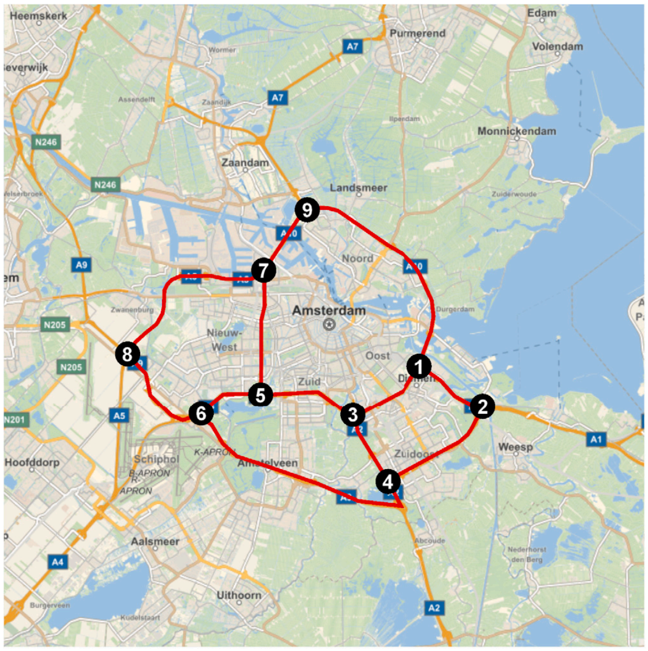


Fig. 8. Amsterdam highway system.

For a transparent comparison of the algorithms, potential preprocessing steps (Section 4.5) are omitted. Experiment 2 considers a more detailed background process in which the Markov process of each arc can attain three possible states (so as to more accurately model the speeds the vehicles can drive at). The background process in addition includes a global event with Erlang distributed holding time (which could represent a reasonably predictable change in the global circumstances, e.g. rush hour; recall Remark 4). Graphical and numerical summaries show that the difference in distance-to-optimality between our algorithms and the deterministic algorithms is even more significant than in Experiment 1. Comparing efficiency shows that the deterministic algorithms and EDSGER* can be executed in real-time, contrary to VI and EDSGER, whose computational costs suffer from the size of the state space.

In Experiment 3 we evaluate the computational costs of the various routing algorithms as function of the network size. We do so by working with an elementary network model, that is then extended with additional arcs to assess its impact on the algorithms' speeds. In addition, we quantify the effect of the dimension of the background process. The final experiment, Experiment 4, considers routing on the entire Dutch highway network, and highlights the influence of various model parameters on the performance of the algorithms. Moreover, the experiment studies the effect of the speed-up techniques introduced in Section 4.5, which turn out to yield a significant contribution to the run-time reduction in this large network. For the experiments we implemented the networks and routing algorithms in Wolfram Mathematica 12.0 on an Intel® Core™ i7-8665U 1.90 GHz computer.

Experiment 1. To get a first impression of the performance of the various algorithms, we start by considering a relatively small network. Our findings reveal that in this network, using a background process in which every link has just two states, both EDSGER and EDSGER* efficiently yield close-to-optimal results. Consider the network of the Amsterdam highway system (Fig. 8), with 9 intersections and 24 links between these intersections (i.e., 12 bidirectional arcs). We pick the following framework:

- The background process of every arc contains just two states, uncongested (corresponding to state 1) and congested (corresponding to state 2). Transition rates are tuned with NDW data, by identifying incidents on a large Dutch highway segment using drops in the vehicle speed. The average duration of these incidents is set as the reciprocal of the clearance rate, and the average time between these incidents as the reciprocal of the incident rate.
- The state of an arc only affects the speeds on the directly attached arcs, i.e., there is local-1-correlation.
- There is no process Y that induces global correlation.
- In case there is no incident on the arc we let the vehicle speed be 100 km/h if there is no incident on the directly adjacent arcs, and 80 km/h otherwise. In case there is an incident on the arc we let the vehicle speed be 40 km/h if there is no incident on the directly adjacent arcs, and 20 km/h otherwise.
- In the network there is a maximum of three incidents simultaneously, to bound the size of the state space and guarantee tractability of VI.

Consider a traveler interested in minimizing the total expected travel time from node 1 to node 8. We compare the DP policy with the routing policies under EDSGER, EDSGER* and the three deterministic algorithms DS, SS and DD. In line with what was stated

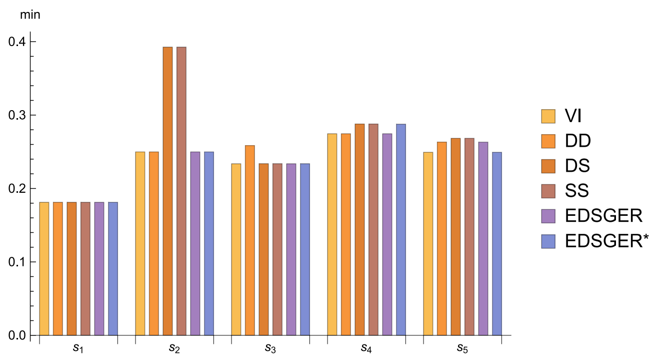


Fig. 9. Expected travel time when traveling from node 1 to node 8 (Fig. 8) under the six policies and in five different initial states. State s_1 corresponds to a 'fully uncongested' network upon departure. In state s_2 there are incidents on links $(1,3),(3,5)$ and $(5,6)$. The links with incidents in states s_3, s_4 and s_5 are $\{(7,5),(6,8),(7,9)\}$, $\{(3,5),(6,8),(8,6)\}$ and $\{(5,6),(6,5),(8,6)\}$ respectively.

Table 2
Costs algorithms.

	Average	1 Inc.	2 Inc.	3 Inc.	WA	Run-time (s)
Value iteration	12.75	11.60	12.26	12.83	11.16	2.032
EDSGER	12.76	11.60	12.27	12.84	11.16	0.094
EDSGER*	12.76	11.61	12.27	12.84	11.16	0.023
DD	12.83	11.64	12.33	12.91	11.18	0.003
SS	13.08	11.61	12.41	13.19	11.17	<0.001
DS	13.08	11.61	12.41	13.19	11.17	<0.001

above, we do not apply the preprocessing steps of Section 4.5, to make the comparison as fair as possible. The DP policy is derived from the VI-algorithm, using the implementation guidelines as described in Section 4.4. Implementation of the deterministic policies follows from a standard implementation of the A*-algorithm. The policies of EDSGER and EDSGER* can be found by storing the output of their shortest path algorithms for every initial state $(k_0, s) \in N \times \mathcal{I}$. The example below, intended to demonstrate the principles underlying EDSGER in a concrete setting, shows that the policy of EDSGER in node 1, with as background state that every arc except for the arc from node 2 to node 1 is uncongested, is to travel to node 3.

Example. We use EDSGER to route from node 1 to node 8 in Fig. 8, when upon leaving the only congested arc is the arc from node 2 to node 1. As a first step, Dijkstra is used on a network with maximum speeds, to determine the lower bounds LB_k of the travel time from node $k \in N$ to node 8, which in this case are given by

$$\text{LB}_1 = 0.18,\ \text{LB}_2 = 0.23,\ \text{LB}_3 = 0.13,\ \text{LB}_4 = 0.16,\ \text{LB}_5 = 0.09,\ \text{LB}_6 = 0.05,\ \text{LB}_7 = 0.09,\ \text{LB}_8 = 0,\ \text{LB}_9 = 0.13$$

Now the shortest path algorithm within EDSGER is used to determine the next node to travel to. Node 1 is set as current node and we initialize $D_1 = 0, D_2 = \cdots = D_9 = \infty$. Then $\exp(d_{1k'} V^{-1} Q)$ is computed to determine $d_{k'}$ and $\boldsymbol{p}_{k'}$ in (10) and (11) for $k' = 2, 3, 9$, the neighbors of 1, to give:

$$d_2 = 0.19,\ d_3 = 0.04,\ d_9 = 0.10.$$

Since $d_{k'} < D_{k'}$ for $k' = 2, 3, 9$ we set $D_{k'} = d_{k'}$ and store $(D_{k'} + \text{LB}_{k'}, p_{k'}, k', \{1, k'\})$ for all neighbors k' of 1. The updated labels now yield:

$D_1 = 0$	$D_2 = 0.19$	$D_3 = 0.04$	$D_4 = \infty$	$D_5 = \infty$	$D_6 = \infty$	$D_7 = \infty$	$D_8 = \infty$	$D_9 = 0.10$

The new current node is then $\arg\min_{k' \in N \setminus \{1\}} \{D_{k'} + \text{LB}_{k'}\} = 3$. Computing $d_{k'}$ and $p_{k'}$ for $k' = 4, 5$ gives

$$d_4 = 0.07,\ d_5 = 0.08.$$

We therefore store $(d_{k'} + \text{LB}_{k'}, p_{k'}, k', \{1, 3, k'\})$ for $k' = 4, 5$ and update the labels:

$D_1 = 0$	$D_2 = 0.19$	$D_3 = 0.04$	$D_4 = 0.07$	$D_5 = 0.08$	$D_6 = \infty$	$D_7 = \infty$	$D_8 = \infty$	$D_9 = 0.10$

The next current node is set as $\arg\min_{k' \in N \setminus \{1,3\}} \{D_{k'} + \text{LB}_{k'}\} = 5$. This procedure is repeated until the current node is set as 8. The paths stored for node 8 is $(1, 3, 5, 6, 8)$, indicating the first node to travel to is node 3. $\Diamond$

Fig. 9 shows the expected travel time (in minutes) of the derived policies in five initial background states. The first set of five bars corresponds to a state of complete non-congestion, i.e., all of the arcs are uncongested upon departure. The expected travel times under the different policies are in this case similar. Note that this is not surprising, as the influence of unpredictable events is minimal: the distance between node 1 and node 8 is small and therefore the probability of occurrence of an incident on the shortest path as well. The second set of bars shows the largest difference in expected travel time between VI and both SS and DS. This difference in expected travel time is significant, the expectation under these deterministic policies being more than 1.5 as large as the expectation under VI. The third, fourth and fifth set of bars show the largest difference between the expected travel time under VI and the expected travel time under DD, EDSGER* and EDSGER respectively. Note that these differences are small, and their policies are thus close-to-optimal.

The fact that DD, EDSGER and EDSGER* yield close-to-optimal results in the framework of this experiment can also be seen in Table 2:

- The first column shows the expected travel time from node 1 to node 8 under the different policies, averaged (evenly) over all possible initial background states.
- The second column shows the expected travel time from node 1 to node 8, averaged (evenly) over all initial background states in which one incident has occurred. Columns three and four show the same for respectively two and three incidents.
- The fifth column provides a weighted average (WA) of the expectations corresponding to all possible initial background states; for a given initial background state, the weight equals its limiting probability.
- The last column contains the run-time of the algorithms (in sec.).

The average and weighted averages of EDSGER and EDSGER* are close to the results of VI. DD performs relatively well, whereas the values under SS and DS are noticeably suboptimal. Comparison in run-time shows that the computational costs of VI are an order larger than those of the other algorithms. In Experiment 3 we will investigate the efficiency of the algorithms in greater detail, and show that the difference in computational costs becomes more substantial when increasing the maximum number of incidents in the network.

We conclude that in this simple framework, in which every arc can only have two states, EDSGER and EDSGER* yield nearly optimal results, while being roughly one order of magnitude faster than VI. It was also shown that EDSGER and EDSGER* perform better than the three deterministic algorithms SS, DS and DD. Especially in case there are incidents in the network, the difference in distance-to-optimality is significant. As the occurrence of incidents is relatively rare, the distance-to-optimality of the four algorithms is similar if there are no incidents in the network upon departure. The next experiment shows that the difference-to-optimality typically grows when adding more detail to the model.

Experiment 2. In this experiment we consider a more involved background process, and show that EDSGER and EDSGER* still yield nearly optimal results. The two algorithms outperform the deterministic algorithms (in terms of distance-to-optimality), and are at the same time considerably faster than VI. The example, moreover, demonstrates the comprehensiveness of our model, by illustrating the possibility of (i) adding recurrent events and (ii) working with more velocities per arc.

We again consider the network of the Amsterdam highway system (Fig. 8), but extend the background process of Experiment 1. Concretely, we have three speeds per arc (rather than two), and include a global event. We include the extra speed level and global event in the following way:

- Every arc has three states, uncongested (state 1), congested (state 2) and recovery (state 3). A link can transition from an uncongested to a congested state, but not vice versa: from a congested state a link must first enter the recovery state before it returns to the uncongested state. The recovery state represents the time between the clearance of an incident and the time the traffic conditions return to the free-flow speed. The incident rate has been determined as in Experiment 1. The total clearance rate of Experiment 1 has now been split such that, on average, 90% of the incident is spent in the congested state, and 10% in the recovery state.
- The state of an arc only affects the speeds on the attached arcs, i.e., there is local-1-correlation.
- There is a recurrent event that induces global correlation, to be interpreted as a rush hour. This event affects the speeds on the roads on the inner circle of the highway system, i.e., all arcs between nodes 1, 3, 5, 7 and 9. We will denote these arcs as category I arcs, whereas we will refer to the other arcs as category II. As a rush hour is a recurring event the time till its onset has a relatively low variance. That is the reason why we chose to not model this time by an exponential distribution but rather by an Erlang distribution (as pointed out in Remark 4). In our experiments we took four phases (see Fig. 10). Only in the last phase, which we identify as the start of the rush hour, the speeds on the arcs in category I are affected.
- Speed levels for the different scenarios can be found in Table 3.
- In the network there is again a maximum of three non-uncongested links simultaneously, to bound the size of the state space and guarantee tractability of VI.

Fig. 11 shows the expected travel time (in minutes) of the algorithms in five initial background states. The first set of six bars again corresponds to a state of complete non-congestion. The expected travel times under the different policies are in this case close. The second, third and fourth set of bars shows the largest difference in expected travel time between VI and DS, DD and SS, respectively. This difference in expected travel time is significant, especially for DD. The fifth set of bars shows the largest difference between the expected travel time under VI and the expected travel time under both EDSGER* and EDSGER. Note that this difference is small, and hence the policies EDSGER and EDSGER* are close to optimal.

Table 3
Speed levels on the arcs in km/h. The rows specify the category and state of the arc, the columns specify if there is rush hour or not, and distinguish between the numbers of attached arcs that is not in state 1.

		Non-rush		Rush	
		0	⩾1	0	⩾1
Cat. I	1	100	80	70	60
	2	40	20	20	10
	3	70	50	50	40
Cat. II	1	100	80	100	80
	2	40	20	40	20
	3	70	50	70	50

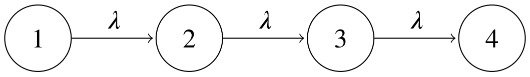

$$\begin{tikzcd} 1 \arrow[r, "\lambda"] & 2 \arrow[r, "\lambda"] & 3 \arrow[r, "\lambda"] & 4 \end{tikzcd}$$

Fig. 10. The four Erlang phases of Rush hour.

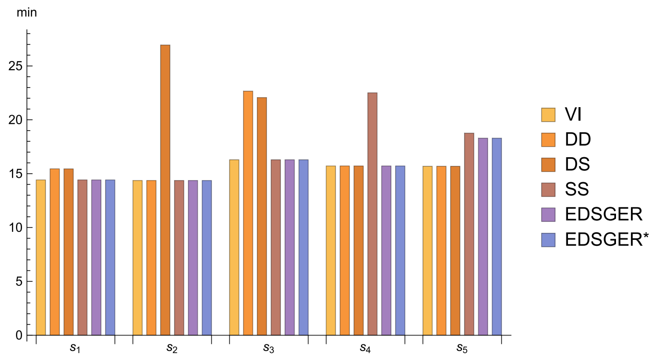


Fig. 11. Expected travel time when traveling from node 2 to node 8 (Fig. 8) under the six policies and in five different initial states. State s_1 corresponds to a 'fully uncongested' network upon departure. In state s_3 there are incidents on links $(2,1)_3, (3,5)_2$ and $(6,8)_2$, with subscripts 2 and 3 denoting the congested and recovery phase of the incident. The links with incidents in states s_2, s_4 and s_5 are $\{(1,3)_2, (3,5)_2, (5,6)_2\}$, $\{(2,4)_2, (4,6)_2, (6,4)_2\}$ and $\{(4,3)_2, (4,6)_3, (6,4)_3\}$ respectively.

Table 4
Average (A) and weighted average (WA) expected travel time of the algorithms in the four different initial phases of the rush hour. Run-time (in sec.) reported as well.

	Phase 1		Phase 2		Phase 3		Phase 4		Run-time (s)
	A	WA	A	WA	A	WA	A	WA	
Value iteration	15.55	14.56	15.70	14.59	15.75	14.60	15.77	14.61	41.513
EDSGER	15.68	14.59	15.73	14.59	15.76	14.61	15.78	14.61	13.154
EDSGER⋆	15.70	14.60	15.74	14.60	15.77	14.61	15.78	14.61	0.154
DD	16.20	15.53	16.76	16.33	17.20	17.00	15.87	14.61	0.003
SS	15.86	14.61	15.86	14.61	15.86	14.61	15.86	14.61	<0.001
DS	17.64	15.73	18.56	16.59	19.27	17.30	19.57	17.62	<0.001

The low distance-to-optimality of EDSGER and EDSGER⋆ can also be observed from Table 4. The table shows the average and weighted average expected travel time (in minutes) under the different policies and in the four different initial Erlang phases of the event rush hour. Similar to Experiment 1, the weights are set equal to the limiting probabilities. The averages and weighted averages of EDSGER and EDSGER⋆ are very close to the results of VI. DD and SS also perform well, whereas the values obtained by DS are noticeably suboptimal. In the case of SS, we notice that the expected travel times are hardly affected by the initial rush-hour phase. Naturally, this can be explained by the fact that SS works with the stationary rush hour setting, with the resulting policy primarily containing nodes whose speed are not reduced by the rush hour. The last column of Table 4 shows the run-time (in seconds) of the different algorithms. Note that the computational costs of VI and EDSGER are substantially larger than those of EDSGER⋆ and the three deterministic algorithms.

Thus, summarizing the results of the first two experiments, EDSGER⋆ has the best performance in terms of distance-to-optimality and run-time. More precisely, EDSGER⋆ outperforms the deterministic algorithms in terms of distance-to-optimality, and outperforms

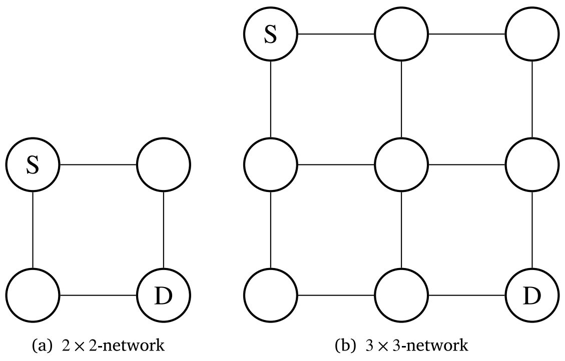


Fig. 12. Examples of networks of size $n \times n$. Here 'S' is the source, and 'D' the destination.

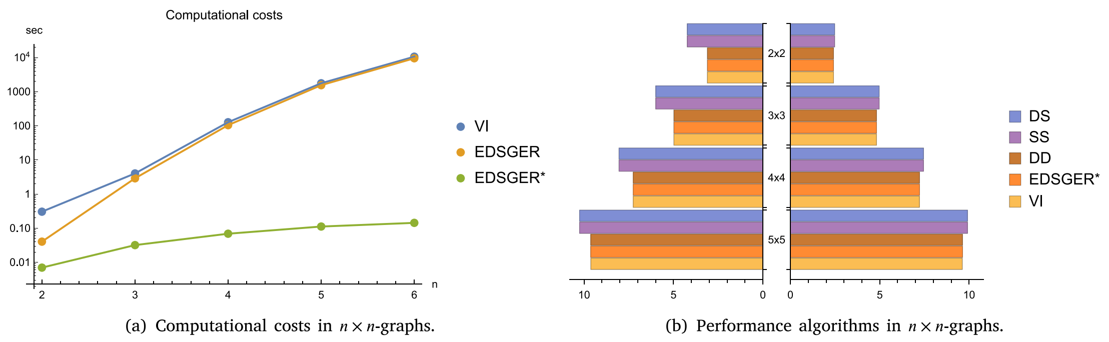


Fig. 13. Effect of increasing the network size.

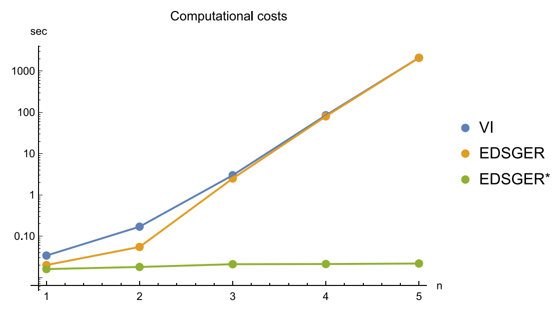
VI and EDSGER in terms of efficiency. The next example will show that the savings in run-time will become even more pronounced when increasing the network size and/or the maximum number of incidents.

Experiment 3. We consider elementary networks to assess the sensitivity of the run-time as a function of the network size, and in addition as a function of the maximum number of incidents. In the previous examples, in which the network size and maximum number of incidents were small, we already noted that the computational costs of both VI and EDSGER are considerably higher than those of EDSGER⋆. To directly demonstrate the computational savings when using EDSGER⋆, we use networks of the type depicted in Figs. 12(a) and 12(b). These networks show a 2 × 2- and 3 × 3-structure, but extending the model in the obvious way produces an $n \times n$-structure for any $n \geqslant 2$. We increase the value of n, so as to evaluate the impact on the computational costs for VI, EDSGER, and EDSGER⋆. For simplicity, the conditions on the network are set identical to those of Experiment 1. This includes the condition that there can only be three incidents in the network simultaneously. However, below we also assess how increasing this bound on the number of incidents affects the run-time of the algorithms. Note that, besides this assumption on a maximum number of incidents, we do not include the preprocessing steps of Section 4.5, so as to facilitate a fair comparison between the different routing algorithms.

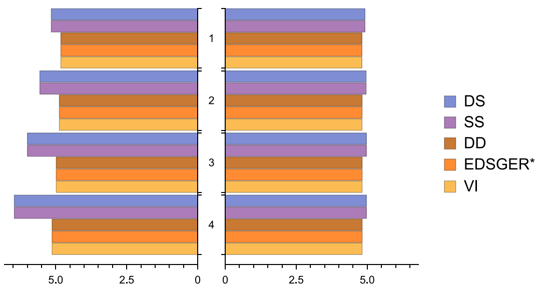
Fig. 13(a) shows that, contrary to the run-time of EDSGER⋆, the run-times of both VI and EDSGER grow exponentially with the network size (note the logarithmic scale). This exponential increase can be explained by the fact that the size of the background process, and thus the size of Q, grows exponentially with the size of the network. Since EDSGER⋆ only uses the part of the state space that corresponds to the states on the directly attached links, its computational costs are hardly affected by the network size. We conclude that in larger networks VI and EDSGER become intractable, whereas EDSGER⋆ still offers real-time response.

Importantly, the substantial reduction of the run-time when using EDSGER⋆ (instead of VI, that is) does not correspond to a significant increase of the objective function: Fig. 13(b) shows that EDSGER⋆ yields close to optimal results. The figure shows the average and weighted average expected travel time (in minutes), with weights again chosen as the stationary probabilities, for different network sizes. Observe that DD also yields close-to-optimal results in this framework. This can, however, be explained by the fact that the considered instance is relatively simple; adding more detail to the framework, as we did in Experiment 2, again leads to a more pronounced suboptimality of this algorithm.

We observe the same behavior (in terms of computational costs and the value of the objective function) when, instead of the network size, the bound on the maximum number of incidents is increased (see Fig. 14). That is, considering the 3 × 3-network displayed in Fig. 12(b), the run-time of VI and EDSGER grows exponentially with the maximum number of incidents. Moreover, the run-time of EDSGER⋆ is hardly affected by the increase of the maximum number of incidents, while still performing nearly optimally. The substantial difference in run-time is again due to the rapid growth of the dimension of Q as function of the number of incidents.



(a) Computational costs with max. of n incidents.



(b) Performance algorithms in case max. n incidents.

Fig. 14. Effect of increasing the number of incidents.

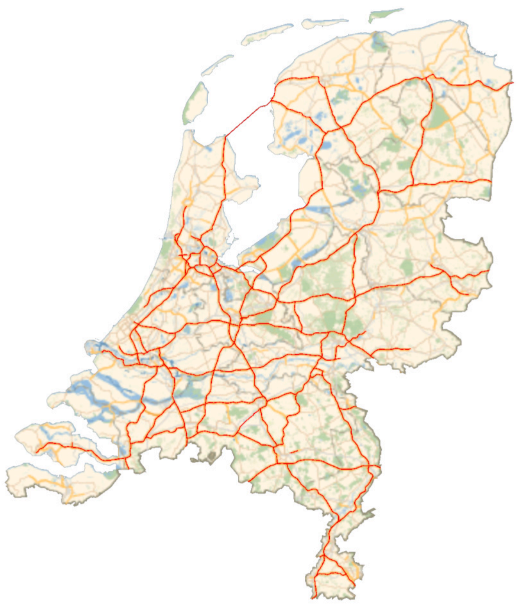


Fig. 15. Dutch highway system.

In contrast, the matrices $Q^{r}_{k\ell}$, as used by EDSGER*, involve significantly fewer arcs, and are therefore less affected by the number of incidents. Note that, in case of local-1-correlation, the computational costs of EDSGER* will e.g. no longer increase if the number of incidents exceeds the maximum degree of the graph.

From Figs. 13 and 14 we conclude that VI and EDSGER are inefficient, making these algorithms unsuitable for practical purposes. The results of EDSGER* are considerably more promising, as these showed that its run-time is hardly affected by the network size or maximum number of incidents, while performing close-to-optimal in terms of cost. This is why in our last experiment we investigate the tractability of EDSGER* in networks of realistic size. The experiment also considers the influence of the model parameters and demonstrates the use of the speed-up techniques introduced in Section 4.5.

Experiment 4. To confirm the feasibility of EDSGER* in large networks, the final experiment considers routing in a network of realistic size: the Dutch highway network (Fig. 15), with 93 intersections and 262 directed roads between these intersections. Besides confirming the feasibility of EDSGER*, the experiment has the following three objectives:

- Show that EDSGER* still yields accurate results in terms of expected travel time, supporting our conclusions of the experiments above. We will omit VI and EDSGER in our analysis, as these algorithms are intractable in large networks, as demonstrated in Experiment 3. Instead, we use simulations to compare the travel time under EDSGER*, DD, SS and DS with the travel time under an optimal path;
- Assess the effect of the parameters on the distance-to-optimality of EDSGER*. Concretely, we compare the results of EDSGER* with the results of DD, SS and DS for different parameter values;
- Demonstrate how the speed-up techniques described in Section 4.5 can be used in this network, and how the different techniques affect the distance-to-optimality and the performance of the algorithms.

Table 5
Average percentage loss in travel time (U%,W%) and run-time (t, in sec.) of the algorithms; background process as in Experiment 1.

	Short			Medium			Long		
	U%	W%	t	U%	W%	t	U%	W%	t
EDSGER⋆	6.5	0.4	0.09	5.8	0.4	0.14	3.4	0.3	0.22
DD	7.3	0.4	0.01	5.9	0.8	0.02	4.9	0.7	0.03
SS	25.3	1.5	<0.01	5.5	0.6	<0.01	3.0	0.4	<0.01
DS	25.7	1.3	<0.01	5.1	0.5	<0.01	3.0	0.4	<0.01

Table 6
Average percentage loss in travel time (W%) and run-time (t, in sec.) of the algorithms; background process as in Experiment 2.

	Short	
	W%	t
EDSGER⋆	2.0	0.60
DD	8.5	0.01
SS	2.6	<0.01
DS	11.8	<0.01

Table 7
Average percentage loss for EDSGER⋆ (left), DD (middle) and DS (right) and different values of the incident rate α and clearance rate β.

EDSGER⋆				DD				DS			
α	β			α	β			α	β		
	10^{-4}	1	100		10^{-4}	1	100		10^{-4}	1	100
10^{-4}	0.00	0.00	0.00	10^{-4}	0.00	0.00	0.00	10^{-4}	39.83	0.03	0.00
1	0.00	10.75	0.05	1	0.00	12.58	1.30	1	0.00	38.52	0.05
100	0.00	0.22	1.50	100	0.00	1.46	29.28	100	0.00	0.22	1.50

To show the travel time under EDSGER⋆ — still executed without the preprocessing steps of Section 4.5 — is indeed close-to-optimal, we simulate the travel time for three OD-pairs. The considered OD-pairs, denoted by 'Short', 'Medium' and 'Long', differ notably in length, with distances 22.5, 98.5 and 191.7 km respectively. For simplicity we again use the same background process as in Experiment 1. We simulate realizations of this background process and determine for every realization the travel time of EDSGER⋆, as well as the travel times of DD, SS and DS. These travel times are compared with the optimal travel time, i.e., the minimal achievable travel time under the given realization of the background process. To measure the difference in travel time, we compute the *average percentage loss in travel time*:

$$\text{Average}\left(\frac{\text{travel time algorithm} - \text{optimal travel time}}{\text{optimal travel time}} \cdot 100\%\right).$$

Note that a high value of this measure can also arise for an algorithm that is optimal in terms of expected travel time, e.g. VI. This is due to the difference between (i) the notion of being (close-to-)optimal in expectation and (ii) the notion of being optimal for a given realization. Using this measure we can therefore only compare algorithms, not assess individual values.

Table 5 shows the results of the simulations. We first note that EDSGER⋆ is indeed feasible, as the displayed run-times t (in sec.) are sufficiently low. The columns denoted by U% and W% show the average percentage loss in travel time for a given OD-pair, with initial states chosen uniformly and weighed (with the corresponding limiting probabilities), respectively.

Table 5 shows that EDSGER⋆ is close-to-optimal in this large network. If the initial states are drawn according to the limiting probability (W%), the average percentage loss in travel time is below 1%. In case the initial state is chosen uniformly, the drawn initial states contain typically many congested links. Even in this more extreme setting, the average percentage loss in travel time is below 7%. Note that the results of DD are relatively close to the results of EDSGER⋆, which is not surprising, as we have chosen the same framework as in Experiment 1 (in terms of the structure of the background process). Table 6 illustrates that the difference in distance-to-optimality between EDSGER⋆ and DD becomes more significant if we add more detail to the background process (by using a setup as in Experiment 2). In Table 5 the results of DS and SS show the disadvantage of not taking into account any (updated) background information, mostly notable in case of the first OD-pair.

We can in addition assess the effect of the parameters on the distance-to-optimality of the algorithms. To this end, we again pick the setup of Experiment 1 and simulate, for different values of the incident rate α and clearance rate β, the travel time under EDSGER⋆, DD, SS and DS for the OD-pair 'Short'. Initial states are chosen according to their corresponding limiting probabilities. Table 7 shows the average percentage loss in travel time for EDSGER⋆, DD and DS, respectively. The output of SS is not explicitly shown, as the simulations revealed that the loss under SS is of the same order as DS, in agreement with Tables 2 and 5, which considered the Experiment 1 setting as well.

The tables reflect the non-stochastic nature of DS and the fact that DD does not take any information on the duration of incidents into account. A few observations we can do:

Table 8
Average percentage loss (%) and run-time (t, in sec.) for different OD-pairs and different values of m.

	$m = 1$						$m = 5$					
	Short		Medium		Long		Short		Medium		Long	
Nodes	8		18		35		9		18		35	
Arcs	9		22		42		12		23		43	
	%	t	%	t	%	t	%	t	%	t	%	t
EDSGER⋆	0.8	0.04	0.5	0.07	0.5	0.12	0.8	0.04	0.5	0.07	0.4	0.19
DD	0.8	0.01	0.9	0.01	0.6	0.04	0.8	0.01	0.9	0.01	0.5	0.10
DS	1.9	<0.01	0.6	<0.01	0.6	<0.01	1.9	<0.01	0.6	<0.01	0.5	<0.01
	$m = 10$						$m = 25$					
	Short		Medium		Long		Short		Medium		Long	
Nodes	10		19		37		16		22		40	
Arcs	15		25		48		27		35		55	
	%	t	%	t	%	t	%	t	%	t	%	t
EDSGER⋆	0.8	0.05	0.5	0.07	0.4	0.24	0.8	0.05	0.5	0.09	0.4	0.34
DD	0.8	0.01	0.9	0.01	0.5	0.14	0.8	0.01	0.9	0.02	0.5	0.25
DS	1.9	<0.01	0.6	<0.01	0.5	<0.01	1.9	<0.01	0.6	<0.01	0.5	<0.01

- EDSGER⋆ and DD perform extremely well in case the time between accidents is long (i.e., the incident rate is low). The probability of an accident occurring while traveling is in this case very low, thus any route circumventing existing incidents will suffice for a low travel time.
- A small clearance rate (i.e., it will take very long before an accident has been cleared) has a potentially high impact on the travel times of DS and SS. In case $\alpha = 1$ or $\alpha = 100$, the limiting probability of an arc being congested is high, resulting in a high number of congested arcs. Because there are so few uncongested arcs in the network, the optimal travel time is with high probability attained on the shortest path (in km), explaining the small average percentage loss of DS and SS; However, if $\alpha = 10^{-4}$, in the initial state approximately half of the arcs will be congested. DS and SS do not reroute in case of incidents, and thus have a high travel time in case an accident occurs on the chosen path.
- DD is clearly suboptimal in case the incident and clearance rate take higher values. This can be explained by the fact that DD will reroute in case an incident has occurred on a favorable path, whereas the high clearance rate will imply the incident duration to be short and thus the impact of the incident on the shortest path minimal.
- The highest value in the table of EDSGER⋆ arises in case $\alpha = \beta = 1$. Note that this loss is smaller than for DD and DS, implying EDSGER⋆ still performs better than the deterministic algorithms. The relatively high loss in travel time for EDSGER⋆ is, as argued above, due to the difference between the notion of being (close-to-)optimal in expectation and the notion of being optimal for a given realization. The fact that the highest loss arises for $\alpha = \beta = 1$ is not surprising, as, compared to the other settings, the travel time on a given path has the highest variance for $\alpha = \beta = 1$, and our methods only try to minimize the expected travel time. The loss under VI would likely be of the same order as that of EDSGER⋆. Computing the loss of both algorithms in the network of Experiment 1 with $\alpha = \beta = 1$ results e.g. in losses 7.00% (VI) and 7.27% (EDSGER⋆).

As stated above, the third objective of this experiment is to demonstrate the effect of the speed-up techniques described in Section 4.5 on the distance-to-optimality and the efficiency of the algorithms. The outcome of this experiment, in which we study the same routes as above (i.e., ‘Short’, ‘Medium’, ‘Long’), is shown in Table 8. Again, we omit the results of SS, as these are of the same order as DS. We use Yen’s m shortest path algorithm to decrease the network size, and distinguish in Table 8 between the different values of m, namely $m = 1, 5, 10, 25$. As argued in Section 4.5, this reduction of the size of the network directly leads to a smaller state space, as we do no longer track the states of the arcs not contained in the reduced network. The derived bound on the hitting probabilities in (17) gives that, for the OD-pair of ‘Short’, the probability of occurrence of four accidents while driving has a probability smaller than $\epsilon = 0.0005$. Background states that contain four additional incidents with respect to the initial state are therefore deleted. A similar argument can be used to reduce the size of the state space for the OD-pairs of ‘Medium’ and ‘Long’.

The table reveals that the number of nodes and arcs in the network increases with m and consequently the run-time t of the algorithms as well. A moderate value of m is therefore preferred. This is also justified by the distance-to-optimality of the algorithms for moderate values of m. That is, the table shows that for smaller values of m the algorithms still yields close-to-optimal results, with the average percentage loss below 2% in all cases. Note that the travel times for $m = 10$ and $m = 25$ are even similar to the case of $m = 5$. This again advocates the choice of a small m, e.g. between $m = 1$ and $m = 5$, as this reduces the run-time while the distance-to-optimality of the algorithms is not affected.

6. Concluding remarks

In this paper we developed a new mechanism for describing the evolution of the velocities in a road traffic network, capable of modeling both recurrent and non-recurrent congestion. For this flexible velocity model we developed a routing algorithm that aims at minimizing the expected travel time. Extensive experiments showed that it outperforms competing models, in that it provides near-optimal results but at the same time offers real-time response.

Regarding the velocity model, we advocate the use of a continuous-time Markovian background process to describe the speed driven by vehicles on the arcs in the network. We have developed various ways to deal with the potentially high dimension of this process. Future research could concern further operationalizing this model, with a specific focus on tuning the model's parameters using measurement data. Concretely, we have argued that the proposed background process is able to incorporate the influence of random events, as well as the influence of (nearly) deterministic patterns, on the vehicle speed. Insight into the occurrence of these events, and their effect on the velocities, can be provided relying on Intelligent Transportation Systems (ITS). However, it is not a priori evident how to map the information provided by a typical ITS on the parameters of our Markov model. A future study could explore such calibration issues.

Regarding the routing algorithms, we have evaluated these on the basis of (i) distance-to-optimality (i.e., minimizing the expected travel time between source and destination) and (ii) efficiency (i.e., run-time). Numerical experiments justify the advice to use EDSGER* as routing algorithm, as EDSGER* is close-to-optimal and has real-time run-time. In the implementation of EDSGER*, as well as the implementation of the other presented algorithms, the guidelines described in Section 4.4 were used. There may be opportunities to further speed-up the algorithm. For instance, parallel computing could potentially be used to speed up VI, EDSGER, and EDSGER*. Since the costs of computing the per-arc expected travel times are under EDSGER* significantly lower than under EDSGER and VI, also when applying parallel computing EDSGER* will outperform the competing algorithms.

Numerical experiments were conducted to show that EDSGER* efficiently yields close-to-optimal results under a broad range of realistic traffic scenarios. The model parameters have been calibrated from historical data on vehicle speeds and flows, collected by the National Data Warehouse for Traffic Information (NDW) in the Netherlands. Future research will focus on developing a more formal statistical procedure to estimate these parameters, performing a detailed analysis of the occurrence and consequences of incidents; cf. the earlier study by Snelder et al. (2013).

CRediT authorship contribution statement

Nikki Levering: Conceptualization, Methodology, Software, Investigation, Writing – original draft, Visualization. **Marko Boon:** Conceptualization, Software, Writing – review & editing. **Michel Mandjes:** Conceptualization, Methodology, Writing – review & editing. **Rudesindo Núñez-Queija:** Conceptualization, Methodology, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank dr. Maaike Snelder (TU Delft) for her helpful feedback on the manuscript as well as her suggestions on the analysis of the NDW data.

Appendix A. Proof of Proposition 1

Proposition 1. *Let $k\ell \in A$ and denote with $\tau^t_{k\ell}(d)$ the travel time on arc $k\ell$ for traveling a distance $d \in [0, d_{k\ell}]$ when leaving at $t \geqslant 0$. Then $t_1 \leqslant t_2$ implies that $t_1 + \tau^{t_1}_{k\ell}(d) \leqslant t_2 + \tau^{t_2}_{k\ell}(d)$.*

Proof. If $\int_{t_1}^{t_2} v_{k\ell}(B(v))\,\mathrm{d}v \geqslant d$ it follows that

$$t_1 + \tau^{t_1}_{k\ell}(d) = t_1 + \min\left\{ t \geqslant 0 : \int_{t_1}^{t_1+t} v_{k\ell}(B(v))\,\mathrm{d}v \geqslant d \right\} \leqslant t_1 + (t_2 - t_1) = t_2 \leqslant t_2 + \tau^{t_2}_{k\ell}(d).$$

If $\int_{t_1}^{t_2} v_{k\ell}(B(v))\,\mathrm{d}v < d$ it follows that

$$\begin{aligned} t_1 + \tau^{t_1}_{k\ell}(d) &\leqslant t_1 + \min\left\{ t \geqslant 0 : \int_{t_2}^{t_1+t} v_{k\ell}(B(v))\,\mathrm{d}v \geqslant d \right\} \leqslant t_1 + \min\left\{ t + t_2 - t_1 \geqslant 0 : \int_{t_2}^{t_2+t} v_{k\ell}(B(v))\,\mathrm{d}v \geqslant d \right\} \\ &= t_1 + (t_2 - t_1) + \tau^{t_2}_{k\ell}(d) = t_2 + \tau^{t_2}_{k\ell}(d). \end{aligned}$$

This completes the proof. □

Appendix B. Proof of Theorem 1

Theorem 1. *Given a graph $G = (N, A)$ with a pair of nodes $k, \ell \in N$, $\gamma \in \mathbb{R}_{\geqslant 0}$ and a distance $d \geqslant 0$, it holds that*

$$V_{k\ell}\, \Phi'(d \mid k, \ell) = \mathbf{1} + Q\, \Phi(d \mid k, \ell),$$
$$V_{k\ell}\, \Psi'(d \mid \gamma, k, \ell) = (Q - \gamma I)\, \Psi(d \mid \gamma, k, \ell),$$

with $V_{k\ell} := \operatorname{diag}\{(v_{k\ell}(s))_{s\in\mathcal{I}}\}$ and $\mathbf{1}$ a $|\mathcal{I}|$-dimensional column vector of ones. A solution for this system of linear differential equations can be written as

$$\Phi(d \mid k, \ell) = \begin{bmatrix} I & \mathbf{0} \end{bmatrix} \exp\left\{ \begin{bmatrix} dV_{k\ell}^{-1}Q & dV_{k\ell}^{-1}\mathbf{1} \\ \mathbf{0}^{\top} & 0 \end{bmatrix} \right\} \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix},$$
$$\Psi(d \mid \gamma, k, \ell) = \exp\{d\, V_{k\ell}^{-1}(Q - \gamma I)\},$$

with $\mathbf{0}$ an $|\mathcal{I}|$-dimensional column vector of zeros.

Proof. The proof uses a type of 'infinitesimal argumentation' that is frequently relied upon in the context of fluid storage systems. Let $d \in [0, d_{k\ell}]$, $s, s' \in \mathcal{I}$. Conditioning on a possible jump of the background process in $[0, \Delta]$, as $\Delta \downarrow 0$, recalling that scenarios with more than one jump have a probability that is $o(\Delta)$,

$$\begin{aligned} \phi_s(d \mid k, \ell) &= (1 + \Delta q_{ss})\mathbb{E}\big[\tau^s_{k\ell}(d) \mid B(t) = s \ \ \forall t \in [0, \Delta]\big] \\ &\quad + \sum_{s' \neq s} \Delta q_{ss'}\, \mathbb{E}\big[\tau^s_{k\ell}(d) \mid \exists t \in [0, \Delta] : B(u) = s \ \ \forall u \in [0, t), B(u) = s' \ \ \forall u \in [t, \Delta)\big] + o(\Delta) \\ &= \Delta + (1 + \Delta q_{ss})\phi_s(d - v_{k\ell}(s)\Delta \mid k, \ell) \\ &\quad + \sum_{s' \neq s} \frac{\Delta q_{ss'}}{1 - e^{-q_{ss'}\Delta}} \int_0^{\Delta} \phi_{s'}(d - v_{k\ell}(s)t - v_{k\ell}(s')(\Delta - t) \mid k, \ell)\, q_{ss'} e^{-q_{ss'}t}\, \mathrm{d}t + o(\Delta), \end{aligned}$$

where it is used that, for $s' \neq s$,

$$\begin{aligned} &\mathbb{E}\big[\tau^s_{k\ell}(d) \mid \exists t \in [0, \Delta] : B(u) = s \ \ \forall u \in [0, t), B(u) = s' \ \ \forall u \in [t, \Delta)\big] \\ &= \int_0^{\Delta} \mathbb{E}\big[\tau^s_{k\ell}(d) \mid B(u) = s \ \ \forall u \in [0, t), B(u) = s' \ \ \forall u \in [t, \Delta)\big] \frac{q_{ss'} e^{-q_{ss'}t}}{1 - e^{-q_{ss'}\Delta}}\, \mathrm{d}t \\ &= \frac{q_{ss'}}{1 - e^{-q_{ss'}\Delta}} \int_0^{\Delta} (\Delta + \phi_{s'}(d - v_{k\ell}(s)t - v_{k\ell}(s')(\Delta - t) \mid k, \ell)) e^{-q_{ss'}t}\, \mathrm{d}t. \end{aligned}$$

Define $f(t, \Delta) := \phi_{s'}(d - v_{k\ell}(s)t - v_{k\ell}(s')(\Delta - t) \mid k, \ell)e^{-q_{ss'}t}$. Now subtracting $\phi_s(d - v_{k\ell}(s)\Delta \mid k, \ell)$ from both sides, dividing by Δ and letting $\Delta \downarrow 0$ we obtain

$$\begin{aligned} v_{k\ell}(s)\phi_s'(d \mid k, \ell) &= 1 + q_{ss}\phi_s(d \mid k, \ell) + \sum_{s' \neq s} q_{ss'} \lim_{\Delta \downarrow 0} \frac{q_{ss'}}{1 - e^{-q_{ss'}\Delta}} \int_0^{\Delta} f(t, \Delta)\, \mathrm{d}t \\ &= 1 + q_{ss}\phi_s(d \mid k, \ell) + \sum_{s' \neq s} q_{ss'}\phi_{s'}(d \mid k, \ell), \end{aligned}$$

where the second step follows from L'Hopital's rule in combination with Leibniz' integral rule. We have thus obtained the desired system of linear differential equations:

$$v_{k\ell}(s)\phi_s'(d \mid k, \ell) = 1 + \sum_{s' \in \mathcal{I}} q_{ss'}\phi_{s'}(d \mid k, \ell).$$

The same steps can be performed to derive the second system of linear differential equations. Again conditioning on a possible jump of the background process, as $\Delta \downarrow 0$,

$$\begin{aligned} \psi_{ss'}(d \mid \gamma, k, \ell) &= (1 + \Delta q_{ss})\, e^{-\gamma\Delta}\, \psi_{ss'}(d - v_{k\ell}(s)\Delta \mid k, \ell, \gamma) \\ &\quad + \sum_{\tilde{s} \neq s} \Delta q_{s\tilde{s}} \frac{q_{s\tilde{s}} e^{-\gamma\Delta}}{1 - e^{-q_{s\tilde{s}}\Delta}} \int_0^{\Delta} \psi_{\tilde{s}s'}(d - v_{k\ell}(s)t - v_{k\ell}(\tilde{s})(\Delta - t) \mid k, \ell, \gamma)e^{-q_{s\tilde{s}}t}\, \mathrm{d}t + o(\Delta). \end{aligned}$$

Subtracting $\psi_{ss'}(d - v_{k\ell}(s)\Delta \mid k, \ell, \gamma)$ from both sides and expanding $e^{-\gamma\Delta}$ as $1 - \gamma\Delta + o(\Delta)$ gives

$$\begin{aligned} \psi_{ss'}(d \mid k, \ell, \gamma) - \psi_{ss'}(d - v_{k\ell}(s)\Delta \mid k, \ell, \gamma) &= (-\gamma\Delta + \Delta q_{ss} - \gamma\Delta^2 q_{ss})\, \psi_{ss'}(d - v_{k\ell}(s)\Delta \mid k, \ell, \gamma) \\ &\quad + \sum_{\tilde{s} \neq s} \frac{(1 - \gamma\Delta)\Delta q^2_{s\tilde{s}}}{1 - e^{-q_{s\tilde{s}}\Delta}} \int_0^{\Delta} f(t, \Delta)\, \mathrm{d}t + o(\Delta), \end{aligned}$$

with now $f(t, \Delta) := \psi_{\tilde{s}s'}(d - v_{k\ell}(s)t - v_{k\ell}(\tilde{s})(\Delta - t) \mid k, \ell, \gamma)e^{-q_{s\tilde{s}}t}$. Dividing by Δ, letting $\Delta \downarrow 0$ and using both L'Hopital's and Leibniz' rule, we eventually obtain

$$v_{k\ell}(s)\, \psi'_{ss'}(d \mid k, \ell, \gamma) = (q_{ss} - \gamma)\, \psi_{ss'}(d \mid k, \ell, \gamma) + \sum_{\tilde{s} \neq s} q_{s\tilde{s}}\, \psi_{\tilde{s}s'}(d \mid k, \ell, \gamma).$$

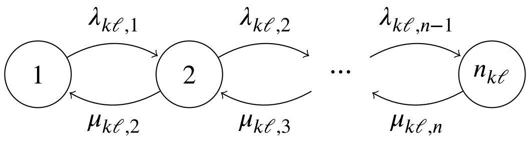


Fig. 16. Birth–death process.

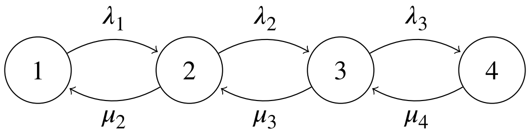


Fig. 17. Birth–death process with 4 states.

This completes the proof. □

Appendix C. Upper bounds probabilities

We will show how results on the sum of independent exponentially random variables can be used to derive an upper bound on the hitting probability

$$\mathbb{P}(\exists t \in [0, M] : B_{k\ell}(t) = s'_{k\ell} \mid B_{k\ell}(0) = s_{k\ell}), \tag{C.1}$$

in case $B_{k\ell}(t)$ is a birth–death process. Fig. 16 shows the outline of such a process with $n_{k\ell}$ states. By the structure of a birth–death process, it holds that

$$\mathbb{P}(\exists t \in [0, M] : B_{k\ell}(t) = s'_{k\ell} \mid B_{k\ell}(0) = s_{k\ell}) \leqslant \mathbb{P}(S_N \leqslant M),$$

with S_N the sum of N independent exponentially distributed random variables. Look for example at Fig. 17 with $\mathcal{S}_{k\ell} = \{1, 2, 3, 4\}$ and note that

$$\mathbb{P}(\exists t \in [0, M] : B_{k\ell}(t) = 3 \mid B_{k\ell}(0) = 1) \leqslant \mathbb{P}(E(\lambda_1) + E(\lambda_2) \leqslant M),$$

with $E(\lambda_1) \sim \mathrm{Exp}(\lambda_1)$ and $E(\lambda_2) \sim \mathrm{Exp}(\lambda_2)$ independent.

With S_N the sum of N independent exponentially distributed random variables with rates λ_i (where $\lambda_i \neq \lambda_j$ for $i \neq j$) and density functions $f_i(z) =: \lambda_i\, e^{-\lambda_i z}$, we have (Bibinger, 2013)

$$f_{S_N}(z) = \sum_{n=1}^{N} \left(\prod_{j=1, j\neq n}^{N} \frac{\lambda_j}{\lambda_j - \lambda_n} \right) f_n(z) =: \sum_{n=1}^{N} c_n f_n(z).$$

This implies for $M < \infty$ that

$$\mathbb{P}(S_N \leqslant M) = \int_0^M \sum_{n=1}^{N} c_n f_n(z)\,\mathrm{d}z = \sum_{n=1}^{N} c_n \int_0^M f_n(z)\,\mathrm{d}z = \sum_{n=1}^{N} c_n (1 - e^{-\lambda_n M}).$$

These results can now directly be used to find an upper bound on the probabilities in (C.1).

References

Asmussen, S., 2003. Applied Probability and Queues, second ed. Springer-Verlag.
Bander, J.L., White III, C.C., 2002. A heuristic search approach for a nonstationary stochastic shortest path problem with terminal cost. Transp. Sci. 36 (2), 218–230.
Bellman, R.E., 1957a. Dynamic Programming. Princeton University Press, Princeton.
Bellman, R.E., 1957b. A Markovian decision process. J. Math. Mech. 6 (5), 679–684.
Bellman, R.E., 1958. On a routing problem. Quart. Appl. Math. 16 (1), 87–90.
Bertsekas, D.P., 1976. Dynamic Programming and Stochastic Control. In: Mathematics in Science and Engineering, Academic Press.
Bertsekas, D.P., 2005. Dynamic Programming and Optimal Control, Vol. 1, third ed. Athena Scientific.
Bertsekas, D.P., 2012. Dynamic Programming and Optimal Control, Vol. 2, fourth ed. Athena Scientific.
Bertsekas, D.P., Tsitsiklis, J.N., 1991. An analysis of stochastic shortest path problems. Math. Oper. Res. 16 (3), 580–595.
Bibinger, M., 2013. Notes on the sum and maximum of independent exponentially distributed random variables with different scale parameters. Probability arXiv.
Bonet, B., 2007. On the speed of convergence of value iteration on stochastic shortest-path problems. Math. Oper. Res. 32 (2), 365–373.
Boriboonsomsin, K., Barth, M., 2008. Impacts of freeway high-occupancy vehicle lane configuration on vehicle emissions. Transp. Res. D 13 (2), 112–125.

Boucherie, R.J., van Dijk, N.M. (Eds.), 2017. Markov Decision Processes in Practice. In: International Series in Operations Research & Management Science, vol. 248, Springer.

CBS, 2020. Statline open data. https://opendata.cbs.nl/statline/#/CBS/nl/dataset/70806ned/table?ts=1609750215691. Accessed: 12-12-2020.

Chang, M., Wiegmann, J., Smith, A., Bilotto, C., 2008. A review of HOV lane performance and policy options in the United States. Technical report, US DOT FHWA.

Cheung, R.K., 1998. Iterative methods for dynamic stochastic shortest path problems. Nav. Res. Logist. 45 (8), 769–789.

Davies, C., Lingras, P., 2003. Genetic algorithms for rerouting shortest paths in dynamic and stochastic networks. European J. Oper. Res. 144, 27–38.

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numer. Math. 1, 269–271.

Fan, Y.Y., Kalaba, R.E., Moore II, J.E., 2005. Arriving on time. J. Optim. Theory Appl. 127 (3), 497–513.

Ferris, M.C., Ruszczyński, A., 2000. Robust path choice in networks with failures. Networks 35 (3), 181–194.

Ford Jr., L.R., 1956. Network Flow Theory. Technical report, Rand Corp. Paper.

Frank, H., 1969. Shortest paths in probabilistic graphs. Oper. Res. 17 (4), 583–599.

Fu, L., Rilett, L.R., 1998. Expected shortest paths in dynamic and stochastic traffic networks. Transp. Res. B 32 (7), 499–516.

Fu, L., Sun, D., Rilett, L., 2006. Heuristic shortest path algorithms for transportation applications: state of the art. Comput. Oper. Res. 33 (11), 3324–3343.

Gao, S., Chabini, I., 2006. Optimal routing policy problems in stochastic time-dependent networks. Transp. Res. B 40, 93–122.

Gao, S., Huang, H., 2012. Real-time traveler information for optimal adaptive routing in stochastic time-dependent networks. Transp. Res. C 21, 196–213.

Goldberg, A.V., Tarjan, R.E., 1996. Expected Performance of Dijkstra's Shortest Path Algorithm. Technical report, NEC Research Institute.

Güner, A.R., Murat, A., Chinnam, R.B., 2012. Dynamic routing under recurrent and non-recurrent congestion using real-time ITS information. Comput. Oper. Res. 39 (2), 358–373.

Guo, S., Zhou, D., Fan, J., Tong, Q., Zhu, T., Lv, W., Li, D., Havlin, S., 2019. Identifying the most influential roads based on traffic correlation networks. EPJ Data Sci. 8, 28:1–28:17.

Hall, R.W., 1986. The fastest path through a network with random time-dependent travel times. Transp. Sci. 20 (3), 182–188.

Ibe, O., 2013. Markov Processes for Stochastic Modeling, second ed. Elsevier.

Kaufman, D.E., Smith, R.L., 1993. Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. IVHS J. 1 (1), 1–11.

Kharoufeh, J.P., Gautam, N., 2004. Deriving link travel-time distributions via stochastic speed processes. Transp. Sci. 38 (1), 97–106.

Kim, S., Lewis, M.E., White III, C.C., 2005a. Optimal vehicle routing with real-time traffic information. IEEE Trans. Intell. Transp. Syst. 6 (2), 178–188.

Kim, S., Lewis, M.E., White III, C.C., 2005b. State space reduction for non-stationary stochastic shortest path problems with real-time traffic information. IEEE Trans. Intell. Transp. Syst. 6 (3), 273–284.

Lerner, J., Wagner, D., Zweig, K.A. (Eds.), 2009. Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation. In: Lecture Notes in Computer Science, vol. 5515, Springer.

Loui, R.P., 1983. Optimal paths in graphs with stochastic or multidimensional weights. Commun. ACM 26 (9), 670–676.

Mao, C., Shen, Z., 2018. A reinforcement learning framework for the adaptive routing problem in stochastic time-dependent network. Transp. Res. C 93, 179–197.

Miller-Hooks, E., 2001. Adaptive least-expected time paths in stochastic, time-varying transportation and data networks. Networks 37 (1), 35–52.

Miller-Hooks, E., Mahmassani, H.S., 2000. Least expected time paths in stochastic, time-varying transportation networks. Transp. Sci. 34 (2), 198–215.

Mirchandani, B.P., Soroush, H., 1986. Routes and flows in stochastic networks. In: Angrealtta, G., Mason, F., Serafini, P. (Eds.), Advanced Schools on Stochastic in Combinatorial Optimization. World Scientific Publishing Company, pp. 129–177.

Murthy, I., Sarkar, S., 1996. A relaxation-based pruning technique for a class of stochastic shortest path problems. Transp. Sci. 30 (3), 220–236.

Nie, Y.M., Wu, X., 2009. Shortest path problem considering on-time arrival probability. Transp. Res. B 43 (6), 597–613.

Norris, J.R., 1997. Markov Chains. Cambridge University Press.

Pease III, M.C. (Ed.), 1965. Methods of Matrix Algebra. Academic Press.

Pedersen, S.A., Yang, B., Jensen, C.S., 2020. Fast stochastic routing under time-varying uncertainty. VLDB J. 29, 819–839.

Polychronopoulos, G.H., Tsitsiklis, J.N., 1996. Stochastic shortest path problems with recourse. Networks 27, 133–143.

Powell, W.B., 2007. Approximate Dynamic Programming: Solving the Curses of Dimensionality. In: Wiley Series in Probability and Statistics, John Wiley & Sons.

Priambodo, B., Ahmad, A., Kadir, R.A., 2020. Prediction of average speed based on relationships between neighbouring roads using k-NN and neural network. Int. J. Online Biomed. Eng. 16 (1), 18–33.

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

Psaraftis, H.N., Tsitsiklis, J.N., 1993. Dynamic shortest paths in acyclic networks with Markovian arc costs. Oper. Res. 41 (1), 91–101.

Puterman, M.L., 1994. Markov Decision Processes: Discrete Stochastic Dynamic Programming. In: Wiley Series in Probability and Statistics, John Wiley & Sons.

Ross, S., 1983. Introduction to Stochastic Dynamic Programming. Academic Press.

Sever, D., Dellaert, N., van Woensel, T., de Kok, T., 2013. Dynamic shortest path problems: Hybrid routing policies considering network disruptions. Comput. Oper. Res. 40 (12), 2852–2863.

Snelder, M., Bakri, T., van Arem, B., 2013. Delays caused by incidents: Data-driven approach. Transp. Res. Rec. 2333, 1–8.

Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction, second ed. MIT Press.

Sýkora, O., 2008. State-space dimensionality reduction in Markov decision processes. In: WDS'08 Proceedings of Contributed Papers: Part I - Mathematics and Computer Sciences. pp. 165–170.

Thomas, B.W., White III, C.C., 2007. The dynamic shortest path problem with anticipation. European J. Oper. Res. 176 (2), 836–854.

Waller, S.T., Ziliaskopoulos, A.K., 2002. On the online shortest path problem with limited arc cost dependencies. Networks 40 (4), 216–227.

Wellman, M.P., Ford, M., Larson, K., 1995. Path planning under time-dependent uncertainty. In: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. pp. 532–539.

Wolshon, B., Lambert, L., 2006. Reversible lane systems: Synthesis of practice. J. Transp. Eng. 132 (12), 933–944.

Yen, J.Y., 1970. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. Quart. Appl. Math. 27 (4), 526–530.

Yen, J.Y., 1971. Finding the k shortest loopless paths in a network. Manage. Sci. 17 (11), 712–716.