Lane-Level Route Planning for Autonomous Vehicles

Mitchell Jones, Maximilian Haas-Heger, and Jur van den Berg

Nuro
{mijones, mheger, jvandenberg}@nuro.ai

Abstract. We present an algorithm that, given a representation of a road network in lane-level detail, computes a route that minimizes the expected cost to reach a given destination. In doing so, our algorithm allows us to solve for the complex trade-offs encountered when trying to decide not just which roads to follow, but also when to change between the lanes making up these roads, in order to —for example— reduce the likelihood of missing a left exit while not unnecessarily driving in the left-most lane. This routing problem can naturally be formulated as a Markov Decision Process (MDP), in which lane change actions have stochastic outcomes. However, MDPs are known to be time-consuming to solve in general. In this paper, we show that —under reasonable assumptions— we can use a Dijkstra-like approach to solve this stochastic problem, and benefit from its efficient $O(n \log n)$ running time. This enables an autonomous vehicle to exhibit natural lane-selection behavior as it efficiently plans an optimal route to its destination.[1]

1 Introduction

Consider the scenario in which an autonomous vehicle traversing a multi-lane road network must reach a given destination via a series of lane changes. Such a scenario occurs when the vehicle prefers to stay in the rightmost lane, but must sometimes either take an exit on the left, or make a left turn in an adjacent lane. As lane changes are not always guaranteed to succeed at a given moment due to external factors (i.e. traffic in the target lane) it is important to start attempting to make the lane change neither too early nor too late. This is just one example of the subtle choices that need to be made when determining which roads to follow, and how to navigate the lanes making up those roads.

The various routes an autonomous vehicle may take on the multi-lane road network are computed by a component of the autonomy system typically called the *routing module*. The routing module will take as input an offline map (which is precomputed and contains information about lanes, road boundaries, traffic controls, and more), and a destination within the offline map defined by the user. The router is responsible for computing a route from the current position of the autonomous vehicle to its destination. This route is then passed as input to the

[1] The contents of this paper are covered by US Patent 11,199,841 [5].

rest of the autonomous vehicle's decision making system, which is responsible for computing the current driving behavior and translating that desired behavior into a trajectory, which can be executed in real time (see e.g. [3, 23]).

Several previous works on route planning for (autonomous) vehicles model the problem as a shortest path search on a graph with deterministic edge weights, and focus on how to handle the potentially very large graph in a memory- and compute-constrained system [4, 13]. Another set of articles focus on finding an optimal route in a network with *stochastic* edge weights, where the edge weights model the uncertain travel time of road segments due to traffic and congestion conditions [1, 14, 24, 29]. Some of these works consider the routing problem up to lane-level detail [9, 20]. While these works are complementary to ours, we focus on a distinct challenge in vehicle routing: instead of stochastic edge weights, we consider stochastic outcomes of (lane change) actions, as the autonomous vehicle's motion planning module that attempts to execute the route may not be able to change lanes at any given moment due to other traffic present. This allows us to determine when the vehicle should start to attempt changing lanes in order to reduce the risk of missing turns and exits while not unduly impeding traffic in faster lanes (this is of particular concern when the autonomous vehicle is a class-A truck or a low-speed vehicle, as is the case for the authors). The result is an algorithm that not only determines the optimal route through a road network, but also enables autonomous vehicles to exhibit lane selection behavior that is natural and intuitive to other traffic participants.

The main contributions of this paper are as follows. Firstly, we formulate our lane-level routing problem as a Markov Decision Process (MDP). The states correspond to sections of lanes in the road network, and the actions model how one navigates on these lanes; either by staying in the current lane, or deciding to try a lane change action which may or may not succeed. Secondly, we present an efficient algorithm to solve the MDP, where the running time is near-linear in the number of states of the MDP. While the optimal policy for MDPs with stochastic actions (such as the lane change action) cannot be computed efficiently in general, the key result of this paper is that under reasonable conditions we can compute the optimal policy for our routing problem using a Dijkstra-like algorithm. More specifically, if the cost formulation satisfies a *monotonicity* requirement — analogous to the nonnegative edge weight requirement in a deterministic graph search problem— we can use Dijkstra's algorithm to find the optimal policy in a single pass. We prove that for our problem reasonable conditions imply this monotonicity, and derive them constructively.

There is a large body of work on heuristic methods to efficiently solve MDPs in practice (see e.g. [2, 8, 18, 21]). For a subset of MDPs that have a single target state out of which one cannot transition (this includes ours), the problem is sometimes referred to as the *stochastic shortest path* (SSP) problem [7, 17]. It is in general not possible to apply Dijkstra's algorithm to SSPs, with the exception of some special cases [22, 26]. An implicit sufficient condition has been established previously for Dijkstra's applicability [6] (effectively the monotonicity requirement mentioned above), and in some cases explicit sufficient conditions can also

be formulated [27]. In this paper we do not study general MDPs or SSPs, but rather a special class for the particularly relevant application of autonomous vehicle routing. We constructively derive a reasonable explicit condition such that the implicit sufficient conditions are satisfied and Dijkstra's algorithm can in fact be applied.

The remainder of this paper is organized as follows. In Section 2 we describe the lane graph, the stochastic model for lane changes, and the Markov Decision Process that defines our problem. In Section 3 we show that our MDP fulfills a monotonicity condition (Theorem 1), which allows for the use of an efficient Dijkstra-like algorithm. We conduct experiments in Section 4 that illustrate the spectrum of routing policies one can obtain in representative environments. We conclude in Section 5.

2 Preliminaries

In order to define the state and action space of the MDP for our application of interest—routing an autonomous vehicle on multi-lane roads—we first define the *lane graph* and present a stochastic model for changing lanes within it.

2.1 The Lane Graph Representation

The lane graph is a directed graph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$, where the vertices correspond to a set of *cells* $\mathcal{X}$, with each cell representing a portion of a lane that can be driven by an autonomous vehicle, and $\mathcal{E}$ is the set of directed edges representing specific relationships between cells. There are four types of relationships between cells: left neighbor, right neighbor, successor, and predecessor. We construct the lane graph such that cells have one-to-one neighbor relationships, resulting in lane graphs with a rectangular pattern like the ones shown in Fig. 3. More formally, the lane graph has the following properties:

- Each cell $x \in \mathcal{X}$ has at most one left neighbor, denoted $\mathrm{lnb}(x)$, and at most one right neighbor, denoted $\mathrm{rnb}(x)$. If a cell x_2 is a neighbor of cell x_1, then x_2 is accessible from x_1 through a lane change along the entire extent of cell x_1. The neighbor relationship is symmetric: $x_1 = \mathrm{lnb}(x_2) \iff x_2 = \mathrm{rnb}(x_1)$.
- Each cell x has a set $\mathrm{succ}(x)$ of successors and a set $\mathrm{pred}(x)$ of predecessors. If cell x_2 is a successor of cell x_1, then x_2 is accessible from x_1 by continuing to drive in the same lane. The successor and predecessor relationships are symmetric: $x_1 \in \mathrm{succ}(x_2) \iff x_2 \in \mathrm{pred}(x_1)$.
- Each cell x has an associated *length* $\ell(x) > 0$.
- Each cell x has an associated *cost* $c(x) > 0$ of traversing the cell.

For our analysis we assume that $|\mathcal{E}| = O(|\mathcal{X}|)$, i.e. every cell has on average a constant number of successors and predecessors (in addition to at most two neighbors). Note that in most lane graphs the majority of lane cells will have just one successor and predecessor, unless a lane forks or merges, in which case it will have more than one successor or predecessor, respectively.

There are some practical considerations to take into account when deciding on the length of the cells making up the lane graph. They must be divided such that they support the one-to-one neighbor relationships, meaning that a merge, a fork, or any other change in neighbor relations among lanes will force cell boundaries across the width of the road. The cells should also not be too short, as that will increase the total number of cells, and therefore the computational expense in computing an optimal policy. At the same time, they should not be too long, as the cells effectively discretize the policy we are computing.

2.2 A Stochastic Model for Lane Changes

It is possible to change lanes from cell x_1 to cell x_2 if x_2 is a neighbor of x_1 in the lane graph. A lane change may in practice not always succeed however, because at any given time traffic in neighboring cells may make a lane change impossible. We therefore define a stochastic model for lane changes. Obviously, a lane change between two longer cells has a higher probability of eventually succeeding than a lane change between shorter cells.

Let $f : \mathbb{R}^+ \to [0, 1]$ define the probability that a lane change between two lane graph cells of length ℓ succeeds. This probability function should have the following properties:

$$f(0) = 0, \qquad f(\infty) = 1, \qquad f(\ell_1 + \ell_2) = f(\ell_1) + (1 - f(\ell_1))f(\ell_2). \tag{1}$$

That is, a lane change between two cells of zero length will never succeed, and a lane change between two cells of infinite length will surely succeed. The last property of Eq. (1) guarantees that the probability function is invariant to partitioning lanes into multiple cells of different length.

It can easily be verified that

$$f(\ell) = 1 - \exp(-\alpha\ell), \qquad \alpha > 0, \tag{2}$$

is the unique solution satisfying these properties, and one readily recognizes in $f(\ell)$ the cumulative density function of an exponential distribution with rate parameter α. In our case, α can be interpreted as the average number of successful lane changes per unit of length if one were to constantly try to change lanes.

2.3 The Markov Decision Process

We formulate our problem as a Markov Decision Process (MDP): The set of states are exactly the set of cells $\mathcal{X}$, the set of actions $\mathcal{A}(x)$ are the actions one can take from each cell $x \in \mathcal{X}$ (to be defined shortly), the cost function $c(x, a, x')$ defines the cost of moving from $x \in \mathcal{X}$ to $x' \in \mathcal{X}$ under action $a \in \mathcal{A}(x)$, and $p(x'|x, a)$ is a transition probability function defining the probability one arrives at $x' \in \mathcal{X}$ when taking action $a \in \mathcal{A}(x)$ from cell $x \in \mathcal{X}$.

Given a goal cell $x_g \in \mathcal{X}$, the objective is to compute for each cell $x \in \mathcal{X}$ the optimal *expected* cost $g(x)$ to reach the goal from x when taking optimal

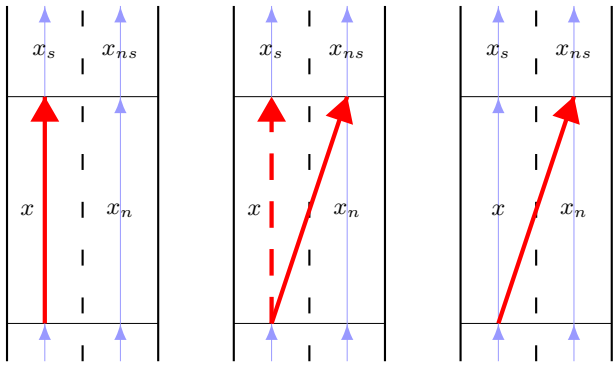


Fig. 1. Notation of possible actions. (a) a stay-in-lane action a_s, (b) a lane change action a_{lc}, and (c) a forced lane change action a_{flc}. We visualize the policy by drawing arrows from the beginning of a cell to the beginning of the target cell under the policy. This arrow, however, only denotes the action we want the motion planning framework to attempt while traversing the cell. It does not indicate for example the trajectory we expect the robot to take in changing lanes. The actual lane change maneuver may be completed within a fraction of a cell, or over the space of several successive cells, depending on the length of the cells and external factors such as traffic.

actions (referred to in this paper as the *value* or *cost-to-go*), and to compute the optimal action $\pi(x)$ one should take for each cell. By definition, $g(x_g) = 0$ and cells x from which the goal cannot be reached have $g(x) = \infty$.

The solution is generally defined by the Bellman equation:

$$q(x,a) = \sum_{x' \in \mathcal{X}} p(x'|x,a)(c(x,a,x') + g(x')), \tag{3}$$

$$g(x) = \min_{a \in \mathcal{A}(x)} q(x,a), \qquad g(x_g) = 0, \tag{4}$$

$$\pi(x) = \operatorname*{argmin}_{a \in \mathcal{A}(x)} q(x,a), \qquad \pi(x_g) = \varnothing. \tag{5}$$

where $q(x,a)$ is the expected cost to reach the goal when taking action a from x and optimal actions thereafter. Let us precisely define the set of actions $\mathcal{A}(x)$ one can take from a cell x, and their associated costs and transition probabilities.

Stay-in-lane actions: For each successor $x_s \in \text{succ}(x)$ of x, we have a stay-in-lane action a_s that would route the vehicle from cell x to cell x_s (see Fig. 1(a)). We have the following cost and transition probabilities for action a_s:

$$c(x, a_s, x_s) = c(x), \qquad p(x_s|x, a_s) = 1, \tag{6}$$

where $c(x)$ is the cost to traverse cell x.

Lane change actions: For each neighbor $x_n \in \text{lnb}(x) \cup \text{rnb}(x)$ of x, we have a lane change action a_{lc} for each pair of successors $(x_s, x_{ns}) \in \text{succ}(x) \times \text{succ}(x_n)$, which would route the vehicle from cell x to cell x_{ns} if the lane change

is successful and to x_s if the lane change is unsuccessful (see Fig. 1(b)). We have the following costs and transition probabilities for action a_{lc}:

$$c(x, a_{\text{lc}}, x_{ns}) = c_{\text{lc}} + c(x), \qquad p(x_{ns}|x, a_{\text{lc}}) = f(\ell(x)), \tag{7}$$

$$c(x, a_{\text{lc}}, x_s) = c(x), \qquad p(x_s|x, a_{\text{lc}}) = 1 - f(\ell(x)), \tag{8}$$

where $\ell(x)$ is the length of cell x, $f(\cdot)$ is as in Eq. (2), and $c_{\text{lc}} \geq 0$ is the cost for making a lane change. This cost is applied in order to discourage the vehicle from changing lanes unless necessary.

Forced lane change actions: Cells from which the goal cannot be reached will have a cost-to-go of ∞. Since our model of lane changes is stochastic, large parts of our lane graph will have a nonzero probability, however small, of arriving in one of these cells, and would therefore have an expected cost-to-go of infinity as well. This would render the formulation of our problem useless. To avoid this, we define an additional action we call a "forced" lane change, which is guaranteed to succeed, but comes at a large but finite additional cost.

For each neighbor $x_n \in \text{lnb}(x) \cup \text{rnb}(x)$ of x, we have a forced lane change action a_{flc} for each successor $x_{ns} \in \text{succ}(x_n)$, which would route the vehicle from cell x to cell x_{ns} regardless of whether the lane change would normally succeed (see Fig. 1(c)). However, a (large) extra cost of $c_{\text{flc}} \geq 0$ is applied if the lane change would normally not succeed. We have the following cost and transition probabilities for action a_{flc}:

$$c(x, a_{\text{flc}}, x_{ns}) = c_{\text{lc}} + c(x) + (1 - f(\ell(x)))c_{\text{flc}}, \qquad p(x_{ns}|x, a_{\text{flc}}) = 1. \tag{9}$$

3 Computing the Optimal Policy

The Markov Decision Process as defined above, can be solved in its general form using the standard value iteration or policy iteration algorithms. Both approaches, however, suffer from a running time that is at least quadratic in the number of states [21]. Fortunately, under reasonable conditions, we can solve the problem using a Dijkstra-like approach that is non-iterative and runs in $O(n \log n)$ time, where $n = |\mathcal{X}|$.

3.1 Monotonicity Conditions

For a Dijkstra-like approach to work, the cost formulation must be *monotone* [6, 21]. That is, the cost-to-go $g(x)$ of each cell x must be larger than the cost-to-go $g(x')$ of the cells x' one could arrive at from x when taking the optimal action $\pi(x)$. The monotonicity requirement is formally stated as follows:

$$\forall \{x, x' \,|\, p(x'|x, \pi(x)) > 0\} :: g(x) > g(x'). \tag{10}$$

This is the equivalent of the requirement of nonnegative edge costs in Dijkstra's algorithm. It guarantees that the optimal value of x depends only on cells that

have been previously visited in Dijkstra's algorithm (we remark that the monotonicity condition requires a strict inequality to avoid cyclic dependencies [27]). Therefore, a single pass of the algorithm suffices to find the optimal policy.

Let us consider under what conditions our cost formulation is monotone. For stay-in-lane actions and forced lane change actions, the monotonicity requirement trivially holds. If stay-in-lane action a_s is optimal, we have x_s, a successor of x as the only cell one can arrive at. And trivially:

$$g(x) = q(x, a_s) = c(x) + g(x_s) > g(x_s). \tag{11}$$

If a forced lane change action a_{flc} is optimal, we have x_{ns}, the successor of a neighbor of x as the only cell one can arrive at. And trivially:

$$g(x) = q(x, a_{\text{flc}}) = c_{\text{lc}} + c(x) + (1 - f(\ell(x)))c_{\text{flc}} + g(x_{ns}) > g(x_{ns}). \tag{12}$$

For regular lane change actions we have two potential next cells: x_s, a successor of x, and x_{ns}, a successor of a neighbor of x. We show that the monotonicity requirement for regular lane change actions holds under the following condition:

Lemma 1. *If $\forall x \in \mathcal{X} :: c(x)/\ell(x) \geq \alpha c_{\text{flc}}$, where α is the lane change success rate, then for any cell x from which a lane change action a_{lc} is optimal, we have $g(x) > g(x_s) \wedge g(x) > g(x_{ns})$.*

The above result gives a very reasonable condition for monotonicity. For instance, setting $c_{\text{flc}} = 1/\alpha$ and requiring $c(x) \geq \ell(x)$ for all cells x would satisfy the monotonicity requirement, and allows us to use a Dijkstra-like approach. The above Lemma is proved in two parts.

Lemma 2. *For any cell x from which a lane change action a_{lc} is optimal, we have $g(x) > g(x_{ns})$.*

Proof. As a_{lc} is optimal, we have $g(x) = q(x, a_{\text{lc}})$, and we must have that $q(x, a_{\text{lc}}) \leq q(x, a_s)$ (a stay-in-lane action). This provides an upper bound on the value of $g(x_{ns})$:

$$\begin{aligned}
& q(x, a_{\text{lc}}) \leq q(x, a_s) \\
\overset{\text{i}}{\iff}\ & f(\ell(x))(c_{\text{lc}} + c(x) + g(x_{ns})) + (1 - f(\ell(x)))(c(x) + g(x_s)) \leq c(x) + g(x_s) \\
\overset{\text{ii}}{\iff}\ & f(\ell(x))(c_{\text{lc}} + g(x_{ns})) \leq f(\ell(x))g(x_s) \\
\overset{\text{iii}}{\iff}\ & g(x_{ns}) \leq g(x_s) - c_{\text{lc}}.
\end{aligned} \tag{13}$$

Equivalence (i) follows from expanding both sides using Eq. (3) and respectively Eqs. (7)–(8) and Eq. (6). Equivalence (ii) follows from rearranging and eliminating terms, and equivalence (iii) follows from dividing both sides by $f(\ell(x))$.

We complete the proof by showing that this bound implies $q(x, a_{\text{lc}}) > g(x_{ns})$:

$$q(x, a_{\text{lc}}) > g(x_{ns})$$

$$
\begin{aligned}
&\overset{\text{i}}{\iff} f(\ell(x))(c_{\text{lc}} + c(x) + g(x_{ns})) + (1 - f(\ell(x)))(c(x) + g(x_s)) > g(x_{ns}) \\
&\overset{\text{ii}}{\iff} c(x) + f(\ell(x))c_{\text{lc}} + (1 - f(\ell(x)))g(x_s) > (1 - f(\ell(x)))g(x_{ns}) \\
&\overset{\text{iii}}{\impliedby} c(x) + f(\ell(x))c_{\text{lc}} + (1 - f(\ell(x)))g(x_s) > (1 - f(\ell(x)))(g(x_s) - c_{\text{lc}}) \\
&\overset{\text{iv}}{\iff} c(x) + c_{\text{lc}} > 0.
\end{aligned}
$$

Equivalence (i) follows from expanding the left hand side using Eq. (3) and Eqs. (7)–(8). Equivalence (ii) follows from rearranging terms. Implication (iii) follows from inequality (13). Equivalence (iv) follows from rearranging and eliminating terms. The last sum is positive as $c_{\text{lc}} \geq 0$ and $c(x) > 0$ for all $x \in \mathcal{X}$. □

Lemma 3. *If* $\forall x \in \mathcal{X} :: c(x)/\ell(x) \geq \alpha c_{\text{flc}}$*, then for any cell* x *from which a lane change action* a_{lc} *is optimal, we have* $g(x) > g(x_s)$*.*

Proof. As a_{lc} is optimal, we have $g(x) = q(x, a_{\text{lc}})$, and we must have that $q(x, a_{\text{lc}}) \leq q(x, a_{\text{flc}})$ (a forced lane change action). This provides an upper bound on the value of $g(x_s)$:

$$
\begin{aligned}
& q(x, a_{\text{lc}}) \leq q(x, a_{\text{flc}}) \\
\overset{\text{i}}{\iff}\ & f(\ell(x))(c_{\text{lc}} + c(x) + g(x_{ns})) + (1 - f(\ell(x)))(c(x) + g(x_s)) \leq \\
& \quad c_{\text{lc}} + c(x) + (1 - f(\ell(x)))c_{\text{flc}} + g(x_{ns}) \\
\overset{\text{ii}}{\iff}\ & (1 - f(\ell(x)))g(x_s) \leq (1 - f(\ell(x)))(c_{\text{lc}} + c_{\text{flc}} + g(x_{ns})) \\
\overset{\text{iii}}{\iff}\ & g(x_s) \leq g(x_{ns}) + c_{\text{flc}} + c_{\text{lc}}. \qquad (14)
\end{aligned}
$$

Equivalence (i) follows from expanding both sides using Eq. (3) and respectively Eqs. (7)–(8) and Eq. (9). Equivalence (ii) follows from rearranging and eliminating terms, and equivalence (iii) follows from dividing both sides by $(1 - f(\ell(x)))$.

We complete the proof by showing that this bound and the conditions of the lemma imply $q(x, a_{\text{lc}}) > g(x_s)$:

$$
\begin{aligned}
& q(x, a_{\text{lc}}) > g(x_s) \\
\overset{\text{i}}{\iff}\ & f(\ell(x))(c_{\text{lc}} + c(x) + g(x_{ns})) + (1 - f(\ell(x)))(c(x) + g(x_s)) > g(x_s) \\
\overset{\text{ii}}{\iff}\ & c(x) + f(\ell(x))(c_{\text{lc}} + g(x_{ns})) > f(\ell(x))g(x_s) \\
\overset{\text{iii}}{\impliedby}\ & c(x) + f(\ell(x))(c_{\text{lc}} + g(x_{ns})) > f(\ell(x))(g(x_{ns}) + c_{\text{flc}} + c_{\text{lc}}) \\
\overset{\text{iv}}{\iff}\ & c(x) > f(\ell(x))c_{\text{flc}} \\
\overset{\text{v}}{\impliedby}\ & c(x) \geq \alpha\ell(x)c_{\text{flc}} \\
\overset{\text{vi}}{\iff}\ & c(x)/\ell(x) \geq \alpha c_{\text{flc}}.
\end{aligned}
$$

Equivalence (i) follows from expanding using Eq. (3) and Eqs. (7)–(8). Equivalence (ii) follows from rearranging terms. Implication (iii) follows from inequality (14). Equivalence (iv) follows from eliminating terms. Implication (v) follows as $f(\ell) = 1 - \exp(-\alpha\ell) < \alpha\ell$ for $\ell > 0$, and $\ell(x) > 0$ for all $x \in \mathcal{X}$. Through dividing by $\ell(x)$ (equivalence (vi)) we arrive at the condition of the lemma. □

The proof of Lemma 1 now follows directly by combining Lemma 2 and Lemma 3. The discussion above culminates in the following Theorem.

Theorem 1. *Let $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ be the lane graph. If $\forall x \in \mathcal{X} :: c(x)/\ell(x) \geq \alpha c_{\text{flc}}$, then the monotonicity requirement of Eq.* (10) *is satisfied:*

$$\forall \{x, x' \,|\, p(x'|x, \pi(x)) > 0\} :: g(x) > g(x').$$

3.2 Dijkstra-Like Algorithm

Given the conditions for monotonicity stated in the previous section, we now give a Dijkstra-like algorithm to find the optimal costs-to-go (values) and actions for all cells in the lane graph given a goal cell x_g (see Algorithm 1). Initially, the value of all cells is set to infinity, except for the goal cell, whose value is set to zero (line 1). The algorithm maintains a priority queue $\mathcal{Q}$ of all *open* cells that have been given a value, but whose optimal value has yet to be confirmed. Initially, the queue only contains the goal cell (line 2).

In each iteration, the cell $\bar{x}$ in the queue with the minimum value is taken and removed from the queue (lines 4 and 5). At this point, $\bar{x}$'s optimal value has been confirmed and is said to be *closed*. The optimality follows from Theorem 1, as the cost-to-go function g is monotone. We then look at all of the children of $\bar{x}$ (line 6)—defined as all x from which an action a can be taken that has a nonzero probability of taking us to $\bar{x}$. If the value of the child x can be decreased by taking action a (line 8), then we update its value (line 9) and set its optimal action (line 10). Subsequently, x is either inserted into the priority queue with key $g(x)$ or if it was already in the priority queue, its key is decreased (line 11). It should be noted that any cell whose value is decreased must either be newly visited (i.e. its prior value was infinity), or it is open and therefore in the queue. If somehow a cell whose value is updated is neither in the queue nor had a prior value of infinity, it must have been closed before, which means that the cost formulation is not monotone. One can check for this explicitly in any implementation.

The algorithm continues until the queue is empty (see line 3) and the optimal value of all cells has been confirmed. The stored optimal actions $\pi(x)$ can now be used to execute the route.

In Algorithm 2 we present a less abstract implementation of lines 6 and 7 of Algorithm 1 to find the children of a given cell x (i.e., the cell-action pairs through which one can arrive at x with nonzero probability) and their associated q-values. Fig. 2 depicts the four types of children a cell x may have. In Algorithm 2 we precisely identify the actions by annotating (superscripting) them with the cells one can arrive at through the action. It should be noted that the algorithm will return all cell-action pairs for which one can arrive at x, but an efficient implementation would only return for each child cell the action with the lowest q-value. Also, in line 8 one only needs to consider successors x_s that are closed (x_s has a finite value and it is not in the queue).

The above discussion and Theorem 1 imply the following result.

Algorithm 1 ROUTER(x_g)

$\forall x : \pi(x) \leftarrow \varnothing$; $\forall x : g(x) \leftarrow \infty$; $g(x_g) \leftarrow 0$
$\mathcal{Q} \leftarrow \{x_g\}$
while $\mathcal{Q} \neq \varnothing$ **do**
  $\bar{x} \leftarrow \operatorname{argmin}_{x \in \mathcal{Q}} g(x)$
  $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{\bar{x}\}$
  for all $(x, a) \in \{(x, a) \,|\, p(\bar{x}|x, a) > 0\}$ **do**
    $q(x, a) \leftarrow \sum_{x'} p(x'|x, a)(c(x, a, x') + g(x'))$
    if $q(x, a) < g(x)$ **then**
      $g(x) \leftarrow q(x, a)$
      $\pi(x) \leftarrow a$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x\}$
return $g(\cdot), \pi(\cdot)$

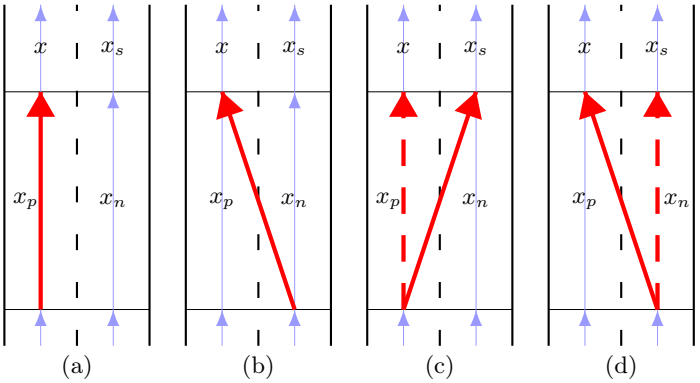


Fig. 2. The possible children of a cell x in Dijkstra's algorithm. These comprise of all cell-action pairs through which one can arrive at x with nonzero probability. (a) a stay-in-lane action from a predecessor x_p, (b) a forced lane change action into a predecessor x_p from a neighbor x_n of that predecessor, (c) a lane change action from a predecessor x_p into a neighbor x_n of that predecessor, and (d) a lane change action into a predecessor x_p from a neighbor x_n of that predecessor.

Theorem 2. *Given a lane graph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ with $|\mathcal{X}| = n$ and $|\mathcal{E}| = O(n)$, a goal cell $x_g \in \mathcal{X}$, and the property $\forall x \in \mathcal{X} :: c(x)/\ell(x) \geq \alpha c_{\text{flc}}$, one can compute the optimal cost-to-go $g(x)$ and action $\pi(x)$ for all cells $x \in \mathcal{X}$ in time $O(n \log n)$.*

The running time follows from the standard analysis of Dijkstra's algorithm [11].

3.3 Potential Heuristics

Since we can successfully employ a variant of Dijkstra's algorithm to solve our routing problem, it is natural to wonder whether we can augment the algorithm with a *heuristic* to focus the search and speed up the algorithm in a similar way A* [19] improves upon the running time of Dijkstra's algorithm in deterministic graph search. Our initial analysis reveals that this is unlikely.

Algorithm 2 FINDCHILDREN(x)

1: $\mathcal{C} \leftarrow \varnothing$
2: **for all** $x_p \in \text{pred}(x)$ **do**
3: $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{(x_p, a_s^x)\}$ ▷ Fig. 2(a)
4: $\quad q(x_p, a_s^x) \leftarrow c(x_p) + g(x)$
5: $\quad$ **for all** $x_n \in \text{lnb}(x_p) \cup \text{rnb}(x_p)$ **do**
6: $\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{(x_n, a_{\text{flc}}^x)\}$ ▷ Fig. 2(b)
7: $\quad\quad q(x_n, a_{\text{flc}}^x) \leftarrow c_{\text{lc}} + c(x_n) + (1 - f(\ell(x_n)))c_{\text{flc}} + g(x)$
8: $\quad\quad$ **for all** $x_s \in \text{succ}(x_n)$ **do**
9: $\quad\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{(x_p, a_{\text{lc}}^{x_s,x})\}$ ▷ Fig. 2(c)
10: $\quad\quad\quad q(x_p, a_{\text{lc}}^{x_s,x}) \leftarrow c(x_p) + f(\ell(x_p))(c_{\text{lc}} + g(x_s)) + (1 - f(\ell(x_p)))g(x)$
11: $\quad\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{(x_n, a_{\text{lc}}^{x,x_s})\}$ ▷ Fig. 2(d)
12: $\quad\quad\quad q(x_n, a_{\text{lc}}^{x,x_s}) \leftarrow c(x_n) + f(\ell(x_n))(c_{\text{lc}} + g(x)) + (1 - f(\ell(x_n)))g(x_s)$
return $\mathcal{C}$

The difference between A* and Dijkstra's is that instead of expanding the node x with minimal $g(x)$-value as in Dijkstra's (see line 4 of Algorithm 1), A* expands the node with a minimal value of $g(x) + h(x)$, where $h(x)$ is a given *heuristic* function, that in a goal-initiated search (as in our case) would give an estimate of the cost to reach x from the start node. The heuristic is said to be *consistent* if the following holds:

$$\forall\{x, x' \,|\, x' \in \text{succ}(x)\} :: h(x') \leq h(x) + c(x, x'), \tag{15}$$

where $c(x, x')$ is the edge cost between x and x'. If the heuristic is consistent, all nodes will be expanded at most once in the A* algorithm and an optimal path is found in optimal running time [12]. An example of a consistent heuristic is the Euclidean distance when the search problem is situated on a graph embedded in Euclidean space with edge costs equal to their Euclidean length.

Since in our case we have nondeterministic actions, one cannot speak of an "edge cost" between two nodes (cells in our case). An appropriate equivalent definition of consistency for our case would be:[2]

$$\forall\{x, x' \,|\, p(x'|x, \pi(x)) > 0\} :: h(x') \leq h(x) + (g(x) - g(x')). \tag{16}$$

Note that it follows from the monotonicity requirement of Eq. (10), proved in Theorem 1, that $g(x) - g(x') > 0$.

Since our problem is embedded in the plane (or on the surface of the earth), and the length of the lane cells is the basis for cost, the Euclidean distance (or great circle distance) would be a sensible candidate for a consistent heuristic in our problem. However, it is not. The only trivial heuristic that is globally consistent appears to be $h(x) = 0$ for all $x \in \mathcal{X}$, which would render an A*-like algorithm equivalent to the Dijkstra-like algorithm. We can see this as follows. Let us consider a simple straight section of a highway of length ℓ with two lanes,

[2] We can use a non-strict inequality here, as long as nodes in the queue with equal $g(x) + h(x)$ are tie-broken by their $g(x)$ value.

and let the goal point be the end of the left lane, and the start point be the beginning of the right lane. Let the cost of traversing each cell be equal to its length. Let x be the distance away from the goal point along either lane, and let us consider the situation in which we have infinitesimally short cells. Let $g(x)$ be the cost to the goal from the point x along the right lane. It can be shown that $g(x) = x + c_{\text{lc}} + c_{\text{flc}}(1 - f(x))$. By choosing the maximum value $c_{\text{flc}} = 1/\alpha$ for the forced lane change cost, this simplifies to $g(x) = x + c_{\text{lc}} + \exp(-\alpha x)/\alpha$. It can be seen that the derivative of $g(x)$ at $x = 0$ equals 0. So, we have $g(\Delta x) \approx g(0)$ for small Δx. If we take as our heuristic the Euclidean distance from the start point, then $h(0) = \ell$ and $h(\Delta x) = \ell - \Delta x$. However, unfortunately, $h(0) \not\leq h(\Delta x) + (g(\Delta x) - g(0))$ as the consistency condition would require. Obviously, $h(x) = g(x_{\text{start}}) - g(x)$ would be globally consistent, but it is not clear that any trivial explicit heuristic exists that is consistent other than $h(x) = 0$.

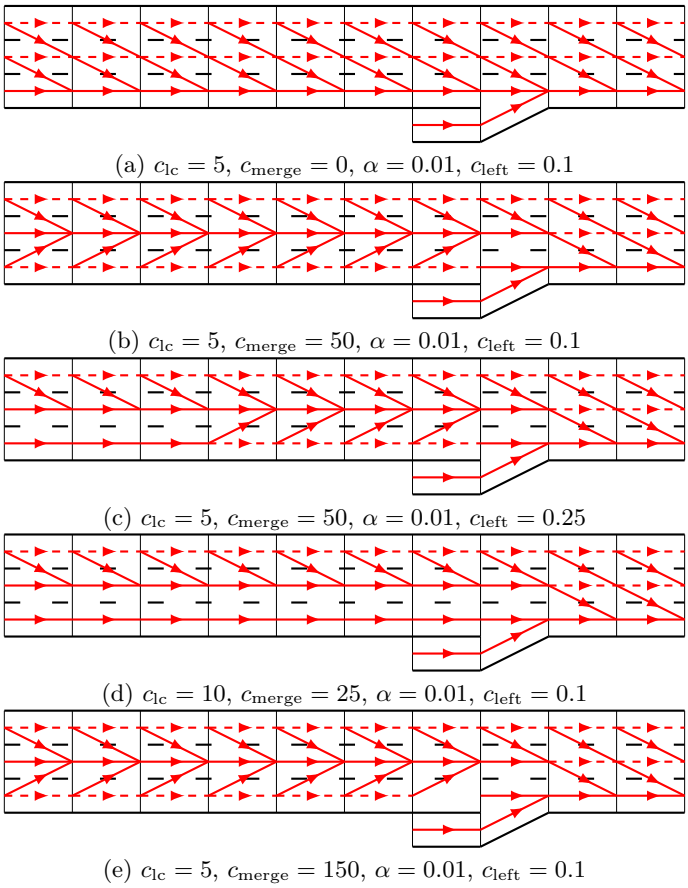
Our situation is in a sense similar to that of the Fast Marching Method (FMM) [25], which implements Dijkstra's algorithm to propagate the Eikonal equation on a grid. In the standard case of a 4-connected grid, FMM does not allow for any consistent heuristic other than $h(x) = 0$ [10, 28]. A number of papers appear in the literature that suggest to use heuristics in FMM, without discussing whether any notion of consistency is satisfied (e.g. [16]). Using inconsistent heuristics would lead either to the computation of incorrect values, or the revisiting of nodes an unbounded number of times. The latter could negate any potential speed-up that an A*-like approach may provide, although works like [15] show that in certain cases inconsistent heuristics may still be beneficial.

4 Experiments

We implemented the presented router in our autonomous vehicle software at Ike for routing class-A trucks on highways (Ike was acquired by Nuro in 2021) and at Nuro to route delivery robots on surface streets. As it is challenging to effectively visualize the routing results on expansive road networks, we illustrate the rich class of routing behavior our algorithm can generate in two constructed but representative scenarios for various parameter values.

In our first experiment we consider a straight three-lane highway that has an on-ramp merging with the rightmost lane (see Fig. 3). The goal cell is in the rightmost lane 5km down from the section of the highway shown in the figure. We show the spectrum of routing policies our algorithm can produce for varying parameter values. The length of each cell is 10m and the cost $c(x)$ of each cell x is equal to its length times a factor σ that penalizes not driving in the rightmost lane. We have $\sigma = 1 + mc_{\text{left}}$, where $m = 0, 1, 2$ for the rightmost, middle, and leftmost lane respectively; c_{left} is a parameter that we vary in the experiments. In addition, we penalize being routed through the merge, so each cell with a successor with multiple predecessors gets an additional cost of c_{merge}, which we vary in the experiments. In all experiments, we set $c_{\text{flc}} = 1/\alpha$.

In the first parameter setting (see Fig. 3(a)) we have no cost for going through a merge, and the resulting routing policy is to always try to lane change to the



(a) $c_{\text{lc}} = 5$, $c_{\text{merge}} = 0$, $\alpha = 0.01$, $c_{\text{left}} = 0.1$

(b) $c_{\text{lc}} = 5$, $c_{\text{merge}} = 50$, $\alpha = 0.01$, $c_{\text{left}} = 0.1$

(c) $c_{\text{lc}} = 5$, $c_{\text{merge}} = 50$, $\alpha = 0.01$, $c_{\text{left}} = 0.25$

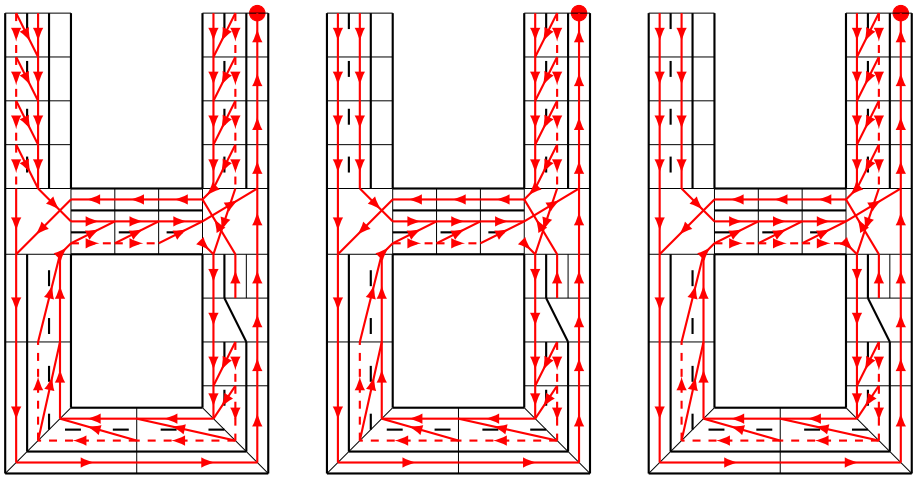
(d) $c_{\text{lc}} = 10$, $c_{\text{merge}} = 25$, $\alpha = 0.01$, $c_{\text{left}} = 0.1$

(e) $c_{\text{lc}} = 5$, $c_{\text{merge}} = 150$, $\alpha = 0.01$, $c_{\text{left}} = 0.1$

Fig. 3. Routing policy on a three-lane highway around a merge produced by our algorithm for various parameter values. The optimal actions are visualized as in Fig. 1. Each cell is 10m long. The figure shows a subset of the cells making up the highway; the goal cell is in the rightmost lane 5km down the road from the cells shown.

rightmost lane of the road. With a larger merge cost (see Fig. 3(b)), we see that the router will try to lane change out of the right lane into the middle lane before the merge in order to avoid it, and then lane change back to the right lane after the merge. If we modestly increase the penalty for not driving in the rightmost lane (see Fig. 3(c)) we see that the router policy waits with attempting to lane change out of the right lane until we are relatively close to the merge. Increasing the cost for a lane change and reducing the cost for the merge (see Fig. 3(d)) leads to a policy where we do not attempt to move out of the right lane to avoid the merge; however, if one is already in the middle lane, we would not move to the right lane until after the merge. If we make the merge cost very large (see Fig. 3(e)), the router tries to avoid the merge using a forced lane change.

In our second experiment we consider a somewhat more complex road network (see Fig. 4). The goal is in the top right corner. A vehicle starting out in



(a) low c_{lc} (b) high c_{lc} (c) $\alpha c_{\text{flc}} > c(x)/\ell(x)$

Fig. 4. Routing policy on a more complex road network produced by our algorithm for two different values of lane change cost, as well as a forced lane change cost that violates the monotonicity requirement. The goal is in the top-right corner.

the top left corner has two potential paths of reaching the goal. The first option is to perform a lane change into the left lane such that it can make a left turn in the intersection. It can then keep left to reach the goal with no further lane changes. The second option allows it to stay in the right lane. However it must then travel a longer path around the perimeter of this road map. For a low lane change cost our algorithm determines an optimal policy that prefers to change lanes and thus take the shorter of the two paths (see Fig. 4(a)). For a larger lane change cost however, the optimal policy takes the longer path in order to avoid a lane change (see Fig. 4(b)). This illustrates the ability of our algorithm to make both macroscopic decisions (which roads to take to the goal) as well as the microscopic decisions of when to change lanes.

Finally, we consider the case of a forced lane change cost so large that the monotonicity requirement is violated. In this case optimal policies may contain cycles (see Fig. 4(c)), and the problem can no longer be solved with a single pass of Dijkstra's algorithm. Instead, the queue never empties as we keep encountering (and having to reopen) previously closed nodes. Eventually the value of these nodes converges, as we are effectively performing value iteration (policy iteration would be a less inefficient choice in this case). This example illustrates that the computational benefit of using a monotone cost formulation comes somewhat at the expense of the richness of routing policies that may result.

5 Discussion

We presented an approach to provide lane-level routing for autonomous vehicles. We modeled the problem as an Markov Decision Process, while proving that we

can still solve it using an efficient Dijkstra-like algorithm. The algorithm allows autonomous vehicles, including Nuro's delivery robots, to make natural choices regarding what lane to drive in and when to change lanes.

A topic of ongoing research is how to extend the algorithm to make use of real-time traffic data. While in the experiments we let the cost of cells equal their lengths, we could instead use their travel time if an average speed is available for each cell. In that case we can set $c_{\text{flc}} = 1/(\alpha v_{\max})$ to ensure that the monotonicity condition holds. It should also be noted that all proofs regarding the monotonicity of our problem are local in nature. This means that α, the lane change success rate, can be locally varied without sacrificing monotonicity (as long as α and c_{flc} remain consistent locally). This can be used to reduce the probability of lane changes succeeding (and increase the cost of forced lane changes) in dense traffic conditions, which would cause our algorithm to adjust its behavior and perform necessary lane changes earlier in such conditions.

We showed that it is not trivial to augment our algorithm with a consistent heuristic to focus the search and reduce the computation time of the algorithm. Even though a simple example of a two-lane highway seems to rule out any heuristic other than $h(x) = 0$ to be globally consistent, it remains an unsatisfactory thought that on a higher level – e.g. planning a route from San Francisco to New York on the entire US interstate network – it would not be possible to focus the search in any way, particularly as the Euclidean distance (or great circle distance) would be "almost" consistent for the vast majority of states in such a network. We are continuing to explore ways to apply heuristics in some form. One idea is to use a hierarchical approach with different levels of granularity in each hierarchy. Another potential avenue for reducing the computation time is to use branch-and-bound techniques such as, e.g., domain restriction [10].

References

1. Ahmadi, K., Allan, V.H.: Stochastic path finding under congestion. In: CSCI. pp. 135–140. IEEE (2017)
2. Andre, D., Friedman, N., Parr, R.: Generalized prioritized sweeping. In: NIPS. pp. 1001–1007. The MIT Press (1997)
3. Badue, C., Guidolini, R., Carneiro, R.V., Azevedo, P., Cardoso, V.B., Forechi, A., Jesus, L.F.R., Berriel, R.F., Paixão, T.M., Mutz, F.W., de Paula Veronese, L., Oliveira-Santos, T., Souza, A.F.D.: Self-driving cars: A survey. Expert Syst. Appl. **165**, 113816 (2021)
4. Bast, H., Delling, D., Goldberg, A.V., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., Werneck, R.F.: Route planning in transportation networks. In: Algorithm Engineering, Lecture Notes in Computer Science, vol. 9220, pp. 19–80. Springer (2016)
5. van den Berg, J.: Methods and systems for determination of a routing policy for an autonomous vehicle (Dec 14 2021), US Patent 11,199,841
6. Bertsekas, D.: Dynamic programming and optimal control: Volume II. Athena Scientific (2012)
7. Bertsekas, D.P., Tsitsiklis, J.N.: An analysis of stochastic shortest path problems. Mathematics of Operations Research **16**(3), 580–595 (1991)

8. Bonet, B., Geffner, H.: Labeled RTDP: Improving the convergence of real-time dynamic programming. In: ICAPS. vol. 3, pp. 12–21 (2003)
9. Chen, B.: Autonomous vehicle routing at RideOS. Medium.com (2018)
10. Clawson, Z., Chacon, A., Vladimirsky, A.: Causal domain restriction for Eikonal equations. SIAM J. Sci. Comput. **36**(5) (2014)
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd Edition. MIT Press (2009)
12. Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of A*. J. ACM **32**(3), 505–536 (1985)
13. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering route planning algorithms. In: Algorithmics of Large and Complex Networks (2009)
14. Fan, Y., Kalaba, R., Moore, J.: Shortest paths in stochastic networks with correlated link costs. Computers & Mathematics with Applications **49**, 1549–1564 (2005)
15. Felner, A., Zahavi, U., Holte, R., Schaeffer, J., Sturtevant, N., Zhang, Z.: Inconsistent heuristics in theory and practice. Artificial Intelligence **175**(9-10), 1570–1603 (2011)
16. Ferguson, D., Stentz, A.: Field D*: An interpolation-based path planner and replanner. In: Robotics Research. Springer Tracts in Advanced Robotics, vol. 28, pp. 239–253. Springer (2005)
17. Guillot, M., Stauffer, G.: The stochastic shortest path problem: A polyhedral combinatorics perspective. Eur. J. Oper. Res. **285**(1), 148–158 (2020)
18. Hansen, E.A., Zilberstein, S.: LAO*: A heuristic search algorithm that finds solutions with loops. Artificial Intelligence **129**(1-2), 35–62 (2001)
19. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
20. Jiang, K., Yang, D., Liu, C., Zhang, T., Xiao, Z.: A flexible multi-layer map model designed for lane-level route planning in autonomous vehicles. Engineering **5**(2), 305–318 (2019)
21. LaValle, S.M.: Planning Algorithms. Cambridge University Press (2006)
22. McMahan, H.B., Gordon, G.J.: Fast exact planning in markov decision processes. In: ICAPS. pp. 151–160. AAAI (2005)
23. Paden, B., Čáp, M., Yong, S.Z., Yershov, D., Frazzoli, E.: A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Transactions on intelligent vehicles **1**(1), 33–55 (2016)
24. Rossi, F., Zhang, R., Hindy, Y., Pavone, M.: Routing autonomous vehicles in congested transportation networks: structural properties and coordination algorithms. In: Autonomous Robots (2018)
25. Sethian, J.A.: A fast marching level set method for monotonically advancing fronts. Proceedings of the National Academy of Sciences **93**(4), 1591–1595 (1996)
26. Tsitsiklis, J.N.: Efficient algorithms for globally optimal trajectories. IEEE Trans. Autom. Control. **40**(9), 1528–1538 (1995)
27. Vladimirsky, A.: Label-setting methods for multimode stochastic shortest path problems on graphs. Math. Oper. Res. **33**(4), 821–838 (2008)
28. Yershov, D.S., LaValle, S.M.: Simplicial Dijkstra and A* algorithms for optimal feedback planning. In: IROS. pp. 3862–3867. IEEE (2011)
29. Zheng, W., Thangeda, P., Savas, Y., Ornik, M.: Optimal routing in stochastic networks with reliability guarantees. In: ITSC. pp. 3521–3526 (2021)