Transportation Research Part E 164 (2022) 102790

Contents lists available at ScienceDirect

Transportation Research Part E

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

Online model-based reinforcement learning for decision-making in long distance routes

Juan J. Alcaraz *, Fernando Losilla, Luis Caballero-Arnaldos

Department of Information and Communications Technologies, Technical University of Cartagena (UPCT), Spain

ARTICLE INFO

Keywords:
Route scheduling
Reinforcement learning
Model predictive control
Monte Carlo tree search

ABSTRACT

In road transportation, long-distance routes require scheduled driving times, breaks, and rest periods, in compliance with the regulations on working conditions for truck drivers, while ensuring goods are delivered within the time windows of each customer. However, routes are subject to uncertain travel and service times, and incidents may cause additional delays, making predefined schedules ineffective in many real-life situations. This paper presents a reinforcement learning (RL) algorithm capable of making en-route decisions regarding driving times, breaks, and rest periods, under uncertain conditions. Our proposal aims at maximizing the likelihood of on-time delivery while complying with drivers' work regulations. We use an online model-based RL strategy that needs no prior training and is more flexible than model-free RL approaches, where the agent must be trained offline before making online decisions. Our proposal combines model predictive control with a rollout strategy and Monte Carlo tree search. At each decision stage, our algorithm anticipates the consequences of all the possible decisions in a number of future stages (the lookahead horizon), and then uses a base policy to generate a sequence of decisions beyond the lookahead horizon. This base policy could be, for example, a set of decision rules based on the experience and expertise of the transportation company covering the routes. Our numerical results show that the policy obtained using our algorithm outperforms not only the base policy (up to 83%), but also a policy obtained offline using deep Q networks (DQN), a state-of-the-art, model-free RL algorithm.

1. Introduction

Planning long-distance road transport routes involves scheduling driving hours, breaks, and rest periods in compliance with the regulations on working conditions for truck drivers, specifically, EC Regulation 561/2006 (EC Regulation 561/2006, 2006) in the European Union. When each delivery must be made within a customer-specific time window, this schedule must also ensure on-time delivery. This is a challenging problem even under deterministic conditions because, as shown in previous works (Goel, 2009, 2018), a given route can meet delivery times following some schedules but not others, and the set of possible schedules can be very large. To illustrate this with a couple of examples, consider a working day starting with a driving period of 4.5 h. According to EC Regulation 561/2006, after this period, the driver must either take an uninterrupted break of 45 min or rest for 3 h (split rest), which includes part of the daily rest time, meaning that the second rest period can last 9 h instead of 11. Each option results in a different schedule. Consider now that the vehicle stops after a driving period of 4.5 h, but now the driver has already completed a total number of 9 driving hours during the day and has not taken a split rest. The driver can: (1) start a rest period that will end when 24 h have elapsed since the end of the previous rest period; (2) start a rest period lasting the minimum time of a normal

* Corresponding author.
E-mail address: juan.alcaraz@upct.es (J.J. Alcaraz).

https://doi.org/10.1016/j.tre.2022.102790
Received 14 June 2021; Received in revised form 15 May 2022; Accepted 5 June 2022
Available online 25 June 2022

rest period (11 h); (3) start a reduced rest period (9 h); (4) take a break and drive for an additional hour (option 3 can be used 3 times a week, and option 4 only twice a week). Again, each decision generates a different schedule. In long distance routes like the ones considered in this paper (more than 4500 km and up to 8 visited nodes), the number of occasions when scheduling decisions should be made range roughly between 15 and 25, and the average number of options available at each decision stage is close to 4. Accordingly, the number of possible schedules for these routes ranges between 4^{15} and 4^{25} (i.e., approximately between 10^9 and 10^{15}). The objective of previous works like (Goel, 2009, 2018) was to find schedules ensuring that arrival and service times (loading and unloading) at every node were compatible with the time-window at each node. Of course, a predefined schedule can be planned only for deterministic travel speeds and service times.

In reality, however, the distance covered during each driving period and the service times at the visited nodes are non-deterministic and depend on diverse uncertainties, such as traffic intensity, waiting times at each location, or incidents causing unexpected delays. Let us consider a long-distance route with a regulation-compliant schedule ensuring on-time delivery under deterministic conditions. When the route is already in progress, the vehicle may arrive at a certain location later than planned. At that point, the predefined schedule may no longer be effective, and a specific unplanned decision should be made. This decision depends on the available options considering the current conditions of the driver (daily driving hours, uninterrupted driving and working hours, breaks taken, split rest taken, and so forth). We refer to these real-time decisions as *en-route* decisions. In fact, effective en-route decisions should take into account all the relevant variables of the route (e.g., the distance covered, customer locations, their time windows), as well as the uncertainty of future travel speeds and service times, to anticipate the possible outcomes of the current and future decisions. With all these considerations, making en-route decisions becomes a stochastic control problem where the decision maker observes all the relevant variables of the route/driver (which we refer to as the *state*) and selects the best possible regulation-compliant action to maximize the chances of serving every customer on time. This is the problem addressed in this paper.

It could be argued that most freight companies have been successfully operating long-haul routes for a long time without needing to solve such a challenging stochastic control problem. En-route decisions can be based on a predefined set of common-sense decision rules (a base heuristic policy). Such a policy can select actions that make efficient use of the available time (e.g., take a break when arriving to a customer location ahead of the time-window) and try to arrive earlier to the next node when the vehicle is running short of time for the next delivery (e.g., by extending the daily driving time and/or reducing the rest period) (Alcaraz et al., 2019). However, a base heuristic policy capable of generating an effective schedule under deterministic conditions can experience difficulties when faced with the randomness and unpredictability of real-life long-distance journeys. Under uncertain conditions, a sub-optimal policy may lead to unexpected out-of-window deliveries and/or make an excessive of use of driving time extensions and rest period reductions. As we show in this paper, all these undesirable circumstances can be mitigated by more effective policies. In this paper, we present an online reinforcement learning (RL) algorithm to generate these policies.

Our proposal follows a model-based RL approach according to the definition in Sutton and Barto (2018), incorporating a *sample model*, (i.e., it uses a characterization of the route uncertainties, such as velocities and service times, as random variables), to simulate possible sequences of future events (trajectories) at each decision stage. It is also a fully online mechanism, since learning and planning are conducted simultaneously during decision time. At each decision stage, the algorithm observes the current state and then estimates (learns) the expected cost of the trajectories (cost-to-go) of each decision. This implies that the policy is generated in real time, in contrast to offline RL, where the policies are learned before starting the route by means of previous (offline) simulations. More specifically, our proposal combines several model-based RL methods for *decision-time planning* (Sutton and Barto, 2018; Bertsekas, 2019). The first one is model predictive control (MPC), where the controller searches for the best actions within a limited number of future stages (lookahead horizon), then takes the estimated best action for the first stage and discards the following ones. Second, to estimate the cost-to-go, we use a rollout strategy, which consists of using a predefined base policy to generate the route decisions beyond the lookahead horizon. Third, we use a Monte Carlo tree search (MCTS) method to decide how many trajectories are generated for each action at each stage of the lookahead horizon.

We have evaluated the effectiveness of our proposal by simulating realistic routes based on the operations of a Spanish long-distance transport company. The results verify the *policy improvement principle* (Bertsekas, 2019), which states that if we obtain a new policy that optimizes the decisions for the lookahead stages, assuming that the cost-to-go beyond the lookahead horizon is determined by a suboptimal base policy, the new policy will outperform the base policy. We use two algorithms as benchmarks. The first algorithm is the specific base policy used in our rollout scheme. This base policy is shown to be effective for every route under deterministic conditions but is clearly outperformed by our proposal when the environment is stochastic. The second baseline is deep Q-networks (DQN) (Mnih et al., 2015), a state-of-the-art, model-free RL algorithm. This allows us to discuss the benefits and drawbacks of our model-based approach compared to a model-free one. We also evaluate the role of the base policy in our proposal, the tradeoff between prediction accuracy and computational overhead, and the effect of uncertainty on performance.

The remainder of this paper is organized as follows. Section 2 discusses related work and highlights the contributions of our proposal. Section 3 formulates the problem of en-route decision-making. Section 4 details our control algorithm. Section 5 describes our evaluation methodology and presents our numerical results. Finally, Section 6 outlines our conclusions.

2. Related work and contribution

Regulations on driving times and rest periods have been taken into account in several previous works, but only as an additional constraint in the formulation of the Vehicle Routing Problem (VRP), not for en-route decision-making. Consequently, the aim of these previous works is to generate routes with a predefined (static) schedule in compliance with the regulations. In this section, we

first discuss how previous works have handled these regulatory constraints in diverse VRP variants. Then, we will review how RL is gradually being adopted as a tool for solving different VRP variants, since it is becoming a promising approach for combinatorial optimization (Mazyavkina et al., 2021). Note that none of these previous works address the specific problem of this paper: optimizing en-route decisions with uncertainty on a given route. Thus, this section only considers antecedents of our approach in a related domain, the VRP, which shares elements with our problem but is different in terms of objectives and methods.

The work by Kok et al. (2010) formulates a Vehicle Routing with Time Windows (VRPTW) considering EU legislation on working and driving hours and addresses it by using the dynamic scheduling algorithm developed in Gromicho et al. (2012), which they modify to insert breaks in the journeys between customers. Goel (2009) proposes an extension of the Large Neighborhood Search (LNS) heuristic for a VRPTW in a way that each insertion or removal of a customer from a route must be feasible within the regulatory framework. A similar approach is used in Prescott-Gagnon et al. (2010), where an LNS heuristic is also employed, but, in this case, the main contribution is an efficient heuristic procedure to check the viability of a route after each customer insertion. Bernhardt et al. (2016) propose a Mixed Integer Linear Programming (MILP) model considering EU regulations, which they extend in Bernhardt et al. (2017) to integrate refueling decisions. They also show that incorporating these regulations in the problem significantly increases the computational time required to generate feasible solutions. Kleff (2019) uses a local search based algorithm and considers different variants of the VRPTW with drivers' labor regulations, showing that its computational cost can become too high as more aspects are taken into account. In Zäpfel and Bögl (2008) and Kovacs et al. (2012), the break and rest periods are inserted using a predefined sequence, while (Alcaraz et al., 2019) considers a predefined set of decision rules equivalent to the base policy used in our rollout scheme. A common conclusion in these previous works is that scheduling decisions should receive a closer attention in long-haul transport. This is because the legislation allows breaks and rest periods to be distributed over time with some flexibility (scheduling decisions), and sometimes a given route can be feasible under some scheduling decisions but not under others. Consequently, finding the optimal solution to a VRP compatible with driver rules implies considering all the possible schedules for every possible route, which might be in the order of 10^{15} for a single long-haul route, as discussed in the introduction. Therefore, adding regulation constraints to a VRP notably increases the inherent complexity of the problem, even when the remaining constraints are standard (e.g., vehicle capacity and time windows). We refer interested readers to Laporte (2016), Goel (2018) for more extensive coverage of the regulatory impact on long-haul VRP.

Travel times uncertainty is another aspect that must be addressed. Some of the previously mentioned works consider travel times variations throughout the day (Kok et al., 2010; Gromicho et al., 2012; Kleff, 2019). For example, in Kleff (2019), the driving time along a segment of the network depends on the time of day and, in Kok et al. (2010), the travel speeds on customer-to-customer routes are assumed to be available. However, traffic congestion is generally unpredictable, and deterministic routing and scheduling can lead to suboptimal solutions that require frequent route re-planning to ensure that customers' demands are met. Stochastic travel and service times can be incorporated into the VRP by using random variables subject to probability distributions, such as normal (Li et al., 2010), Li and Li (2020), Gamma (Taş et al., 2014), and log-normal (Gutierrez et al., 2018) distributions. Incidences on routes are also taken into account (Jabali et al., 2015; Vareias et al., 2019) to generate routes with more resilient schedules. In general, considering stochasticity provides better solutions in terms of robustness but makes the problems considerably harder to solve (Gutierrez et al., 2018).

In recent years, Reinforcement Learning (RL) techniques have being gradually adopted as additional tools for solving diverse variants of the VRP. Early works (Mao and Shen, 2018; Nazari et al., 2018) show how a model-free RL (MFRL) framework can generate good quality solutions to basic VRP instances with little computation time (after training). J. Poullet (Poullet, 2020) extends this framework to more complex VRP variants. These works show the potential RL has to address stochastic VRPs and make real-time en-route decisions under uncertainty, although the latter application is still an open issue. It should be noted that, unlike our online approach, MFRL algorithms require previous training, involving considerable computational effort, or a representative training dataset, which can be hard to obtain. In some cases, RL algorithms are used in combination with other algorithms, such as neural networks, metaheuristics, or classical optimization methods. For example, James et al. (2019) uses a deep neural network model for fast route generation, and a deep RL (DRL) strategy to determine the model parameters. Zhao et al. (2020) present a hybrid approach in which a DRL algorithm generates VRP solutions that are then improved with a local search method. Liu et al. (2020) combine deep inverse reinforcement learning with Dijkstra's algorithm to recommend routes for food delivery drivers. We can find additional examples of the use RL in other related areas of transportation. In taxis dispatching, model predictive control is a common strategy (Miao et al., 2016), and RL has been combined with an integer programming model of vehicle-customer matching in Liang et al. (2021).

In conclusion, although there are an increasing number of recent works applying RL to address diverse transportation problems, RL has not yet been explored for en-route decision-making in long-haul transportation considering EU regulations on drivers' working conditions.

2.1. Contribution

Here we detail the contributions of this paper:

1. We address a problem of significant practical interest for trucking companies: en-route decision-making for long-haul routes, considering driving, breaks and rest periods in compliance with EU regulations. Decisions should be made based on the current state of the journey, considering the uncertainty of future travel and service times, and aiming to deliver every transported good within the predefined time window. To the best of our knowledge, this is still an open problem.

2. We propose an online, model-based RL framework, which is a novel approach in this environment. It has two main advantages: First, it does not require prior training, which makes our approach more flexible compared to offline model-free approaches (e.g. it can adapt, without additional training, to changes in optimization criteria, regulations, or the statistic characterization of the route uncertainties). Second, it uses an existing base policy, which our algorithm is guaranteed to improve according to the *policy improvement principle*. This base policy can be a set of decision rules based on the experience of the road transport company. The performance of the base policy provides a lower bound on the performance of our algorithm.
3. We combine the following techniques: model predictive control to make decisions by anticipating their future consequences; a rollout strategy to approximate the cost-to-go using a base policy; and a Monte Carlo tree search (MCTS) method to efficiently sample future trajectories. We provide a novel implementation of MCTS based on a multi-armed bandit algorithm aimed at maximizing the probability of selecting the best action given a limited number of samples.
4. We have empirically verified that the obtained policy consistently improves the base policy under stochastic conditions, and it also outperforms an offline trained DQN agent, at the expense of higher computation at decision time. We quantify the performance and the per-stage computation time under different configurations and different degrees of uncertainty, to assess the feasibility of the proposal. Our results show that efficient decisions can be made within computation times that are compatible with real-time operation.

3. Problem statement

As described in the introduction, this work addresses the problem of selecting driver decisions on long-distance routes, during events (such as stops on the route, or the end of a delivery) where more than one decision is available. The occurrence of one of these events is a *decision stage* in our model. Our objective is to find a decision rule (policy) that selects, at consecutive stages, actions that comply with regulations on driving and rest periods, maximize the expected amount of goods delivered within their respective time windows, and make a moderate use of last resort decisions (driving extensions and reductions in rest time). In this section, we explain this problem in detail, starting with a description of the regulations involved.

3.1. Driving and working hours regulation

Drivers' working conditions in the European Union are regulated by EC Regulation 561/2006 (EC Regulation 561/2006, 2006), which came into effect in April 2007. According to this regulation, driving periods, breaks, and rest periods must be scheduled as follows:

1. After a driving period of 4.5 h, drivers must take an uninterrupted break of at least 45 min, unless they take a rest period.
2. Daily driving time must not exceed 9 h. A regular daily rest period is any rest period of at least 11 h.
3. The daily rest period may be reduced to 9 h no more than three times a week.
4. Daily rest need not be completed at the end of the 24-h period provided that more than 11 (or 9) hours of daily rest have elapsed within the 24-h period. In other words, normal daily rest must begin no later than 13 h after the end of the last rest period. In the case of rest reduction, daily rest must begin no later than 15 h after the end of the last rest period. These values (13 and 15 h) are known as the maximum daily duration and the maximum extended daily duration.
5. In each 24-h period after the end of the previous daily rest period, the driver must take a new daily rest period.
6. Weekly driving time must not exceed 56 h.
7. Daily driving time may be extended to a maximum of 10 h not more than twice a week.
8. The break may be replaced by a break of at least 15 min followed by a break of at least 30 min.
9. The daily rest period may be taken in two parts, the first of which must be an uninterrupted period of at least 3 h and the second an uninterrupted period of at least 9 h.

In addition, the European Directive 2002/15/EC (2002), which regulates uninterrupted working periods (i.e., driving plus loading or unloading the vehicle), must be observed. This directive introduces two additional restrictions:

1. Working time intervals must not exceed six hours.
2. Weekly working time must not exceed 60 h.

3.2. Elements of the problem

The problem addressed is essentially a stochastic control problem where, at each decision stage, the decision maker (agent) observes the state of the controlled system and makes a decision based on this observation. The next state will be determined by the current state and decision and, in general, by the outcome of a random variable (thus, the stochastic nature of the problem). This subsection describes all the elements that set up this problem.

System state. The state of the system is defined by the following set of route and driving variables.

- The distance d covered by the vehicle up to the current stage
- The total time t elapsed since departure from the depot
- The number of days elapsed since departure, $D = 1, 2, \ldots$

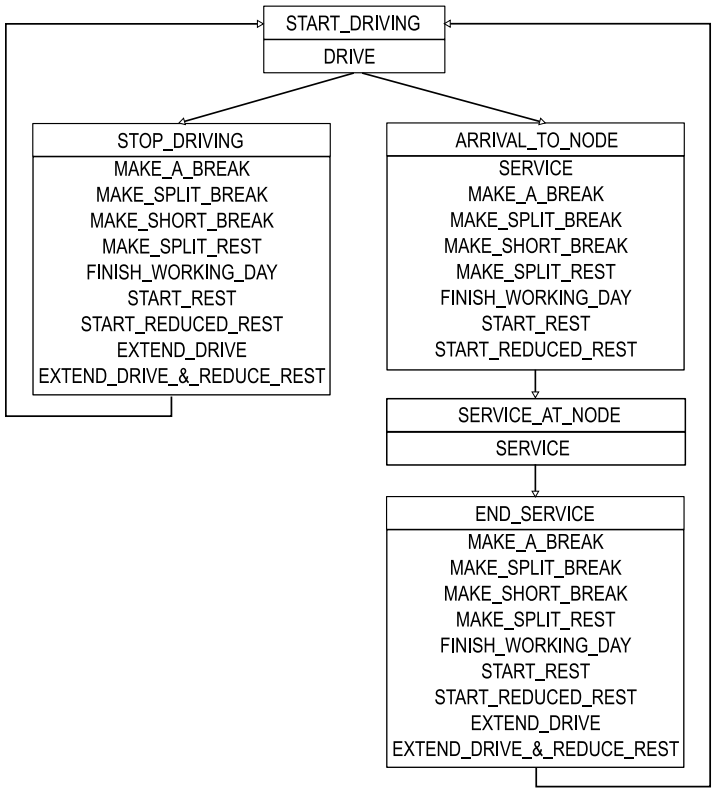


Fig. 1. Sequence of events within a route and the possible actions associated with each event.

- The next location to be visited $j \in \mathcal{L}$, where $\mathcal{L}$ denotes the set of locations visited along the route
- The event associated with the current stage k, belonging to the following set: {`START_DRIVING, STOP_DRIVING, ARRIVAL_TO_NODE, SERVICE_AT_NODE, END_SERVICE`}
- Driving time during the current day, t_{drive}
- Current uninterrupted driving time, $t_{\text{on road}}$
- Current uninterrupted work time, t_{work}
- A boolean value indicating if at least one break has been done during the current day, `BREAK_DONE`
- A boolean value indicating if the daily rest time has been split, `SPLIT_REST`
- A boolean value indicating if the daily rest time has been reduced, `REDUCED_REST`
- A boolean value indicating if the driving time has been extended, `EXTENDED_DRIVE`
- The number of daily rest reductions used, R_{rest}
- The number of driving time extensions used, D_{ext}

The above variables are grouped into the system state x_k, where $k = 0, 1, \ldots, N$ denotes consecutive decision stages.

Actions and action subsets. At each stage, the agent selects an action u_k based on the observed state x_k. The action u_k is constrained to take values from a given subset $U_k(x_k)$, which depends on the variables in x_k and the regulations. For example, consider a stop in which the daily driving time t_{drive} is equal to 9 h (the maximum without an extension). If the variable D_{ext}, which accounts for the driving time extensions used during the route, is less than 2, extending the driving time will be possible. If $D_{\text{ext}} = 2$, this option will not be available. Fig. 1 shows the possible actions that could belong to $U_k(x_k)$ depending on the event reported in x_k. Appendix A details how the specific actions within $U_k(x_k)$ are determined.

Decision Stages. As shown in Fig. 1, there are three events (`STOP_DRIVING, ARRIVAL_TO_NODE, END_SERVICE`) associated with multiple actions. Therefore, decision stages are defined as the moments these events occur.

System dynamics. We use this term to refer to the function f_k that governs the evolution of the state from stage k to stage $k+1$. This function depends on the current state x_k, the selected action u_k, and a random variable ω_k, which is characterized by a probability distribution that depends on x_k and u_k, and gives the system its stochastic nature. In our problem, there are two sources of randomness: vehicle speed at each driving period and service (loading or unloading) times at each node. Therefore, if the current decision u_k is to start driving, ω_k will be sampled from the probability distribution characterizing vehicle speed. If u_k dictates starting the service at the current node n (specified by x_k), ω_k will be sampled from the probability distribution characterizing the service time at n. In Section 5, we provide a specific example of the characterization of these route variables.

Given the described elements, the system dynamics are as follows:

$$x_{k+1} = f_k(x_k, u_k, \omega_k), \text{ for } k = 0, 1, \dots, N \tag{1}$$

where u_k is constrained to take values from $U_k(x_k)$.

Total cost of a route. To evaluate the effectiveness of a given policy, we need to define a numerical measure of the cost of a route governed by this policy. We will associate the overall cost of a route with three variables: (1) the amount of pallets Q_{late} that could not be delivered within their respective time widows, (2) the number of daily rest reductions used, R_{rest}, and (3) the number of driving time extensions used, D_{ext}. Note that shortening rest periods or extending driving times should be considered last resort measures since they may worsen drivers' well-being and may represent an additional expense to the company, depending on contractual agreements. In general, the objective is to minimize a weighted sum of these three variables. We consider equal weight in our numerical experiments, thus, the total cost of a route is given by $Q_{\text{late}} + R_{\text{rest}} + D_{\text{ext}}$. Note that this cost is a random variable that depends on the values taken by x_k, u_k, ω_k, for $k = 0, 1, \dots, N$.

Cost per stage. This is denoted by $g_k(x_k, u_k, \omega_k)$, and it is defined as the sum of the increments in the cost variables Q_{late}, R_{rest}, and D_{ext}, occurring during the transition from stage k to stage $k+1$.

Policy. Formally, a policy is a sequence of functions $\pi = \{\mu_0, \dots, \mu_N\}$, where μ_k maps states x_k into actions, $u_k = \mu_k(x_k)$. We say that a policy is feasible if $\mu_k(x_k) \in U_k(x_k)$ for any x_k at $k = 0, 1, \dots, N$.

Cost-to-go. At a given state x_k, observed at stage k, the cost-to-go function under policy π is defined as the expected cost starting from x_k:

$$J_{k,\pi}(x_k) = \mathbb{E}\left[\sum_{n=k}^{N} g_n(x_n, \mu_n(x_n), \omega_n)\right] \tag{2}$$

where the expectation $\mathbb{E}[\cdot]$ is defined over the random variables ω_n for $n = k, \dots, N$ (whose distributions depend on the visited state–action pairs x_n, u_n, for $n = k, \dots, N$). Note that the cost-to-go functions can be defined recursively as follows:

$$J_{k,\pi}(x_k) = \mathbb{E}\left[g_k(x_k, u_k, \omega_k) + J_{k+1,\pi}(f_k(x_k, u_k, \omega_k))\right], \text{ for } k = 0, \dots, N \tag{3}$$

where each expectation is defined over the corresponding random variable ω_k for $k = 0, \dots, N$, and by convention it is assumed that $J_{N+1,\pi}(\cdot) = 0$.

3.3. Problem formulation

Given an initial state x_0 at $k = 0$, the goal is to find an optimal policy π^*, that minimizes the cost-to-go starting at x_0. This is formulated as the following optimization problem:

$$J_{0,\pi^*}(x_0) = \min_{\pi \in \Pi} J_{0,\pi}(x_0) \tag{4}$$

where Π denotes the set of feasible policies (i.e., complying with regulation constraints).

Dynamic Programming formulation. We will now provide a more suitable formulation of the problem (4) based on the Dynamic Programming algorithm (Bertsekas, 2019). First, we can define the optimal cost-to-go function at k recursively, as we did for the cost-to-go function in (3):

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \mathbb{E}\left[g_k(x_k, u_k, \omega_k) + J_{k+1}^*(f_k(x_k, u_k, \omega_k))\right], \text{ for } k = 0, \dots, N \tag{5}$$

where, as in (3), each expectation is defined over ω_k for $k = 0, \dots, N$, and $J_{N+1}^*(\cdot) = 0$. If the functions J_0^*, J_1^*, …, J_N^* were known, we could obtain an optimal policy with

$$\mu_k^*(x_k) \in \arg\min_{u_k \in U_k(x_k)} \mathbb{E}\left[g_k(x_k, u_k, \omega_k) + J_{k+1}^*(f_k(x_k, u_k, \omega_k))\right] \tag{6}$$

for each state x_k encountered at $k = 0, 1, \dots, N$.

Q-functions. The function minimized on the right hand side of (5) and (6) at each k is often referred to as the Q-function (and also as the Q-value or Q-factor):

$$Q_k(x_k, u_k) = \mathbb{E}\left[g_k(x_k, u_k, \omega_k) + J_{k+1}^*(f_k(x_k, u_k, \omega_k))\right] \tag{7}$$

Let us assume that we have access to an estimation $\tilde{J}_{k+1}$ instead of the exact function J_{k+1}^*. In this case, we define the approximated Q-function:

$$\tilde{Q}_k(x_k, u_k) = \mathbb{E}\left[g_k(x_k, u_k, \omega_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, \omega_k))\right]. \tag{8}$$

Estimating (8) to obtain a suboptimal policy $\tilde{\mu}_k(x_k) \in \arg\min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k)$ constitutes a general reinforcement learning (RL) strategy known as *approximation in value space*. In the next section we explain the specific RL algorithm used in our problem.

4. Online reinforcement learning

The general operation of our online reinforcement learning agent is given by the following steps. At each $k = 0, 1, \dots, N$:

1. Observe state x_k
2. Obtain an approximation $\tilde{Q}_k(x_k, u_k)$ of the Q-function $Q_k(x_k, u_k)$ for $u_k \in U_k(x_k)$
3. Compute and apply the action $\tilde{\mu}_k(x_k) \in \arg\min_{u \in U_k(x_k)} \tilde{Q}_k(x_k, u_k)$
4. Return to step 1 until $k = N$.

It is important to highlight the difference between an offline strategy and our online proposal. When a simulator of the environment is available, an offline RL algorithm will be trained using this simulator in order to obtain an approximated Q-function capable of estimating the Q-function at any visited state–action pair. Once trained, the algorithm will be deployed on the real environment and it will use its Q-function estimator to select the most appropriate action at each online decision stage, just as in step 3 above. In contrast, our online RL scheme does not approximate the Q-functions in advance. Thus, no previous offline training is required. Instead, $\tilde{Q}_k(x_k, \cdot)$ is estimated once x_k is observed.

The main challenges of Q-function estimation are obtaining the $\tilde{J}_k$ functions for $k = 1, \dots, N$, and computing the expectation in (8). The following subsections present the proposed techniques to address these two challenges.

4.1. Rollout algorithm with multistage lookahead

Let us assume that a heuristic policy $\pi = \{\mu_0, \mu_1, \dots, \mu_N\}$, called the *base policy*, is provided. The rollout algorithm aims to improve the performance of the base policy, producing an improved policy called the *rollout policy*. In its ℓ-stage lookahead form, this policy improvement relies on the following approximation of the optimal cost-to-go function for $\ell > 1$:

$$\tilde{J}_{k+1}(x_{k+1}) = \min_{\substack{cu_n \in U_n(x_n) \\ n=k+1,\dots,k+\ell-1}} \mathbb{E}\left[\sum_{n=k+1}^{k+\ell-1} g_n(x_n, u_n, \omega_n) + J_{k+\ell,\pi}(x_{k+\ell})\right] \tag{9}$$

and $\tilde{J}_{k+1}(x_{k+1}) = J_{k+1,\pi}(x_{k+1})$ if $\ell = 1$. With this approximation, the computation of $\tilde{\mu}_k(x_k)$ involves the minimization of the aggregated cost over a limited lookahead horizon of ℓ stages followed by the cost-to-go $J_{k+\ell,\pi}$ of the base policy π. In other words, this approximation assumes that the base policy takes control after ℓ stages and then runs until the Nth stage. Therefore, for $\ell > 1$, the following stochastic optimization problem must be solved at each stage k:

$$\begin{aligned} &\underset{u_k, u_{k+1}, \dots, u_{k+\ell-1}}{\text{minimize}} \ \mathbb{E}\left[g_k(x_k, u_k, \omega_k) + \sum_{n=k+1}^{k+\ell-1} g_n(x_n, u_n, \omega_n) + J_{k+\ell,\pi}(x_{k+\ell})\right] \\ &\text{subject to} \\ &\quad u_n \in U_n(x_n), \text{ for } n = k, \dots, k+\ell-1 \\ &\quad x_{n+1} = f_n(x_n, u_n, \omega_n), \text{ for } n = k, \dots, k+\ell-1 \end{aligned} \tag{10}$$

where the expectation is taken with respect to the trajectories induced by the random process $\omega_k, \dots, \omega_{k+\ell-1}$. If $\ell = 1$, the summation term is removed from the objective function in (10). The first action of the obtained sequence $u_k, u_{k+1}, \dots, u_{k+\ell-1}$ is then applied at stage k, $\tilde{\mu}_k(x_k) = u_k$, while the remaining ones are discarded.

4.2. Simulation-based implementation

The problem (10) can be solved approximately by generating simulated trajectories starting from x_k. However, implementing a lookahead of $\ell > 1$ stages is not straightforward because each trajectory requires the generation of an approximately optimal sequence of ℓ consecutive actions $u_k, \dots, u_{k+\ell-1}$. One possible approach is to proceed backwards in time using a dynamic programming strategy for each generated trajectory. This implies obtaining recursive approximations of the optimal cost-to-go. At stage $k + \ell - 1$, the approximation is defined as:

$$\begin{aligned} &\hat{J}_{k+\ell-1}(x_{k+\ell-1}) \approx \\ &\quad \min_{u_{k+\ell-1} \in U_{k+\ell-1}(x_{k+\ell-1})} \mathbb{E}\left[g_{k+\ell-1}(x_{k+\ell-1}, u_{k+\ell-1}, \omega_{k+\ell-1}) + J_{k+\ell,\pi}(x_{k+\ell})\right] \end{aligned} \tag{11}$$

and for $n = k+1, \dots, k+\ell-2$, the approximations are defined as:

$$\hat{J}_n(x_n) \approx \min_{u_n \in U_n(x_n)} \mathbb{E}\left[g_n(x_n, u_n, \omega_n) + \hat{J}_{n+1}(x_{n+1})\right] \tag{12}$$

where expectations at each recursive step are estimated by sample averaging.

The general strategy involves generating a lookahead tree as follows. The root node of the tree is the observed state x_k, from which $S|U(x_k)|$ trajectories are initiated, where $|U(x_k)|$ denotes the cardinality of the action set $U(x_k)$ (i.e., the number of available actions at x_k), and S denotes the per-action sample budget at each node of the tree. S is a configurable parameter of the algorithm, whose influence will be discussed later in Section 5.

Each of the edges leaving the root node corresponds to one specific action $u_k^s \in U(x_k)$ and one sample of the random variable ω_k^s. Each edge connects x_k with one child node given by $x_{k+1}^s = f_k(x_k, u_k^s, \omega_k^s)$ and is associated with one sample of the per-stage

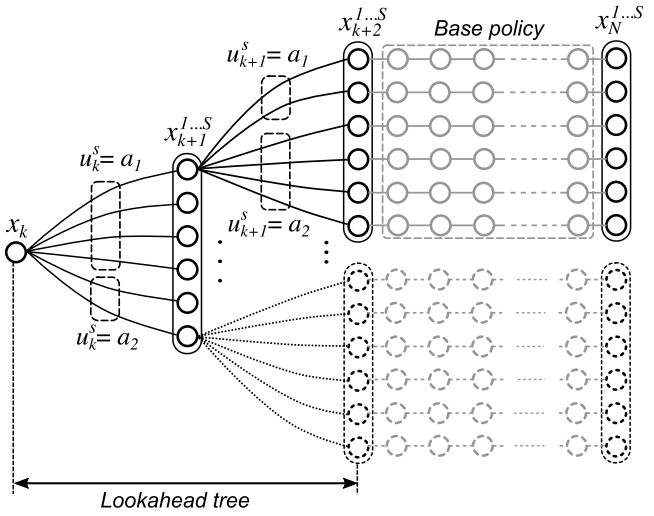


Fig. 2. Schematic representation of a 2-stage lookahead tree built from x_k with the rollout algorithm. The sets of available actions at both x_k and x_{k+1}^1 are equal to $\{a_1, a_2\}$, and the per-action budget is $S = 3$ (although the samples are not evenly assigned among actions). From states x_{k+2}^s (for $s = 1, \ldots, S$), the base policy is used until reaching the last stage.

cost $g_k^s = g_k(x_k, u_k^s, \omega_k^s)$. From every child node, $S|U(x_{k+1}^s)|$ new edges come out, resulting in $S|U(x_{k+1}^s)|$ variations on each of the trajectories initiated at x_k. Each variation is generated by one sample of the random variable ω_{k+1}^s and one action $u_{k+1}^s \in U(x_{k+1}^s)$. The tree generation proceeds similarly up to stage $k + \ell - 1$, as detailed in Algorithm 1. As a result, the tree contains ℓ depth levels, one per lookahead stage, such that the edges on level i are defined by pairs (x_{k+i}^s, x_{k+i+1}^s) for $i = 0, \ldots, \ell - 1$, and each edge is associated with one action u_{k+i}^s and one cost sample g_{k+i}^s. See Fig. 2 for an illustration of a lookahead tree with $\ell = 2$ and $S = 3$.

Algorithm 1 Generate lookahead tree

Input: Current state x_k, depth of the tree ℓ, per-action budget S.
Output: Tree data structure containing state, actions, and cost samples.
Generate action set $U_k(x_k)$
for $s = 1, \ldots, S|U(x_k)|$ **do**
    Select action $u_k^s \in U_k(x_k)$
    Generate sample ω_k^s
    Obtain cost sample $g_k^s = g_k(x_k, u_k^s, \omega_k^s)$
    Produce state sample $x_{k+1}^s = f_k(x_k, u_k^s, \omega_k^s)$
    Associate u_k^s and g_k^s with edge (x_k, x_{k+1}^s)
end for
for $i = 1, \ldots, \ell - 1$ **do**
    for each x_{k+i}^s **do**
        Generate action set $U_{k+i}(x_{k+i}^s)$
        for $s = 1, \ldots, S|U(x_{k+i}^s)|$ **do**
            Select action $u_{k+i}^s \in U_{k+i}(x_{k+i}^s)$
            Generate sample ω_{k+i}^s
            Obtain cost sample $g_{k+i}^s = g_{k+i}(x_{k+i}^s, u_{k+i}^s, \omega_{k+i}^s)$
            Produce state sample $x_{k+i+1}^s = f_{k+i}(x_{k+i}^s, u_{k+i}^s, \omega_{k+i}^s)$
            Associate u_{k+i}^s and g_{k+i}^s with edge (x_{k+i}^s, x_{k+i+1}^s)
        end for
    end for
end for

From each leaf node $x_{k+\ell}^s$, the base policy π must run from stage $k + \ell$ to the final stage N, completing the trajectory and obtaining a sample of the cost-to-go function $J_{k+\ell,\pi}(x_{k+\ell}^s)$ for the base policy, as detailed in Algorithm 2. Recall that taking one sample of the cost-to-go requires generating all the samples ω_n^s for $n = k + \ell, \ldots, N$.

To determine an upper bound on the number of generated trajectories from x_k, let $\bar{U}$ denote the maximum number of available actions at any state (in our case $\bar{U} = 9$). In the first stage, up to $S\bar{U}$ trajectories are initiated. Each one of them branches into a maximum of $S\bar{U}$ trajectories in the second stage, resulting in $\left(S\bar{U}\right)^2$ trajectories. Generalizing, we conclude that the sample complexity (number of generated trajectories) at each decision stage is $O\left((S\bar{U})^\ell\right)$.

Algorithm 2 Generation of a trajectory with the base policy

Input: Base policy $\pi = \{\mu_0, \dots, \mu_N\}$, state $x^s_{k+\ell}$
Output: Cost-to-go sample $J_{k+\ell,\pi}(x^s_{k+\ell})$
$J_{k+\ell,\pi}(x^s_{k+\ell}) = 0$
$x_n = x^s_{k+\ell}$
for $n = k+\ell, \dots, N$ **do**
    Obtain action $u_n = \mu_n(x_n)$
    Generate sample ω_n
    Update cost-to-go $J_{k+\ell,\pi}(x^s_{k+\ell}) = J_{k+\ell,\pi}(x^s_{k+\ell}) + g_n(x_n, u_n, \omega_n)$
    if $n < N$ **then**
        Generate next state $x_{n+1} = f_n(x_n, u_n, \omega_n)$
    end if
end for

Note that all the available actions need to be sampled at every visited state to estimate their performances. However, the principal aim is to use the available sample budget efficiently to estimate the best action for each state. This is a non-trivial problem that will be discussed in the next subsection.

Given the data stored in the lookahead tree, we can obtain the approximate value $\hat{J}_{k+i}$ of the optimal cost-to-go using Monte Carlo averaging, starting at level $i = \ell - 1$ and proceeding backwards until $i = 1$. This procedure is described in Algorithm 3, where $\delta(u' = u)$ is an indicator function that equals 1 when $u' = u$ and 0 otherwise.

Algorithm 3 Computation of the Q-function

Input: Lookahead tree starting at x_k
Output: Approximated Q-function $\hat{Q}_k(x_k, u)$ for $u \in U_k(x_k)$.
for each leaf node $x^s_{k+\ell}$ **do**
    $\hat{J}_{k+\ell}(x^s_{k+\ell}) = J_{k+\ell,\pi}(x^s_{k+\ell})$
end for
for each depth level of the lookahead tree $i = \ell - 1, \dots, 0$ **do**
    for each node x^s_{k+i} at the i-th depth level of the tree **do**
        for each child node x^s_{k+i+1} of x^s_{k+i} **do**
            Get u^s_{k+i} and g^s_{k+i} associated with edge (x^s_{k+i}, x^s_{k+i+1})
            $q^s_{k+i}(x^s_{k+i}, u^s_{k+i}) = g^s_{k+i} + \hat{J}_{k+i+1}(x^s_{k+i+1})$
        end for
        for each $u \in U_{k+i}(x^s_{k+i})$ **do**
            $$\hat{Q}_{k+i}(x^s_{k+i}, u) = \frac{\sum_{s=1}^{S} q^s_{k+i}(x^s_{k+i}, u^s_{k+i})\delta(u^s_{k+i} = u)}{\sum_{s=1}^{S} \delta(u^s_{k+i} = u)}$$
        end for
        if $i > 0$ **then**
            $$\hat{J}_{k+i}(x^s_{k+i}) = \min_{u \in U_{k+i}(x^s_{k+i})} \hat{Q}_{k+i}(x^s_{k+i}, u)$$
        end if
    end for
end for

4.3. Sampling strategy

In the rollout algorithm described above, every time a new state sample x^s_{k+i} is generated for $i = 0, \dots, \ell - 1$, it is necessary to generate trajectories from all the actions in $U_{k+i}(x^s_{k+i})$ to obtain a sample of $\tilde{J}_{k+i}(x^s_{k+i})$ (the approximation of the optimal cost-to-go). Because the algorithm operates online, the sampling process necessary to build a lookahead tree at each stage must be accomplished in reasonable and bounded time. To address this limitation, we introduced the sample budget S in the previous subsection, resulting in $S|U_{k+i}(x^s_{k+i})|$ total samples at a given node x^s_{k+i} of the lookahead tree. Let $A = |U_{k+i}(x^s_{k+i})|$ denote the cardinality of the action set at this node, and let $B = SA$ denote the total sample budget at the node.

Because all the actions in $U_{k+i}(x^s_{k+i})$ need to be evaluated, the problem is how to allocate the B samples among the A actions. Sampling all the actions equally often (i.e., taking S samples per action) can provide estimations of similar accuracy for all the actions. However, this approach can be inefficient because many samples can be wasted in poor performing actions, while finding the best action may require sampling the better ones more often. In fact, we should aim for two objectives: first, to accurately identify the best action for each state, and second, to obtain more samples of the best action to increase the accuracy of the Q-function approximation.

This problem fits into the category of multi armed bandits (MABs) (Lattimore and Szepesvári, 2020). In a general setting, MABs describe sequential decision problems where an agent takes samples from a set of actions (arms) and observes their performance. The arm selected at each step is determined by the history of past actions and observations. There are various types of MAB problems depending on their objective. In our case, we consider a best arm identification problem that aims to accurately find the best arm with a given sample budget.

Several algorithms addressing this problem have been proposed: Sequential Halving (SH) (Karnin et al., 2013), Successive Rejects (SR) (Audibert et al., 2010) and UCB-E (Audibert et al., 2010). The first two are parameter-free, meaning they do not require previous off-line tuning of any parameter. Our algorithm implements the more recent algorithm, SH, described in Algorithm 4, which operates as follows. First, SH divides the sample budget B into $T = \lceil \log_2(A) \rceil$ elimination rounds of identical numbers of samples. At the end of each round, the algorithm discards half of the arms with the lowest empirical performances. During each round, each arm that has not yet been discarded is selected equally often. The selected action $\hat{u}$ is the last surviving arm.

Algorithm 4 Sequential Halving

Input: Set of actions $U_{k+i}(x^s_{k+i})$, per action sample budget S
Output: Best empirical action $\hat{u} \in U_{k+i}(x^s_{k+i})$
$A = |U_{k+i}(x^s_{k+i})|$, $B = SA$, $T = \lceil \log_2(A) \rceil$
$\mathcal{U}^{(0)} = U_{k+i}(x^s_{k+i})$
for each elimination round $t = 0, \dots, T-1$ **do**
  for each $u \in \mathcal{U}^{(t)}$ **do**
    Select arm u for $\left\lfloor \frac{B}{\lceil \log_2(A) \rceil |\mathcal{U}^{(t)}|} \right\rfloor$ consecutive samples
    Update the performance estimation of u
  end for
  Let $\mathcal{U}^{(t+1)}$ contain the $\lceil |\mathcal{U}^{(t)}|/2 \rceil$ actions with better performance in $\mathcal{U}^{(t)}$
end for
$\hat{u}$ is the only element in $\mathcal{U}^{(T)}$

One benefit of SH over randomized MAB algorithms like UCB is that SH allows us to compute the exact number of samples obtained from the best empirical arm, as follows:

$$n_{\text{BEST}} = \sum_{t=0}^{T-1} \left\lfloor \frac{B}{\lceil \log_2(A) \rceil \lceil \frac{A}{2^t} \rceil} \right\rfloor \tag{13}$$

The work that presented SH (Karnin et al., 2013) also provided an upper bound on the probability of finding the best arm, but it showed that its empirical performance is notably better than this bound. Section 5 numerically evaluates how estimation accuracy, implicitly determined by S and ℓ, affects the performance of our proposal.

The use of an MAB-based sampling strategy at each visited state x^s_{k+i}, imposes a subtle but relevant constraint on the generation of the lookahead tree. The actions selected from $U^s_{k+i}(x^s_{k+i})$ cannot be determined a priori. Instead, each selected action depends on the actions previously selected from $U^s_{k+i}(x^s_{k+i})$ and their performance observations. Recall that these observations are samples of the Q-function $Q_{k+i}(x^s_{k+i}, u)$, not the per-stage cost. Therefore, taking one sample of $Q_{k+i}(x^s_{k+i}, u)$ implies generating a new edge (x^s_{k+i}, x^s_{k+i+1}) and then generating the whole tail of the lookahead tree from x^s_{k+i+1}. For the sake of clarity, this aspect is not reflected in Algorithm 1 but must be taken into account if multistage lookahead is combined with an MAB algorithm, as in our implementation.

5. Numerical results

5.1. Methodology

In this section, we evaluate the performance of our online RL algorithm by simulating complete realizations of realistic long-haul routes based on typical operations of a real freight company in Spain.[1] These routes pick up goods in southeastern Spain and deliver them to customers in Germany, with each route characterized by its delivery day and time-window. For each route, we estimate the cost attained with our algorithm, the cost obtained with the base policy detailed in Appendix B, and the cost obtained with a state-of-the-art, model-based RL algorithm known as Deep Q-Networks (DQN) (Mnih et al., 2015). Unlike the base policy and our proposal, DQN needs to be trained offline to find an effective policy. Later in this section, we will discuss how DQN was configured and trained.

In all our experiments, each algorithm is evaluated in 200 simulations of each route (1200 runs in total). Each simulation run involves the generation of random travel speeds for all the driving periods and all the service times at the nodes. In these test runs, the simulator may insert random incidents along the journey, affecting either travel speeds or service times at any point along the route. These incidents are inserted for testing purposes, and the algorithms are assumed to be unaware of them during

[1] The simulation environment and the RL agents developed for this section are available in https://github.com/jjalcaraz-upct/mbrl-route.

Table 1
Configuration of the simulation parameters.

Velocity during each driving period	
Distribution	Truncated Gaussian
Max speed	100 km/h
Mean speed	80 km/h
Min speed	60 km/h
Standard deviation	5 km/h
Service time (load or delivery) at locations	
Distribution	Truncated Gaussian
Min time	30 min
Mean time	3 min per pallet
Standard deviation	10% of mean time
Incidences	
Incidence probability	0.1 per drive/service period
Maximum incidences	1 per route
Max speed	40 km/h
Mean speed	25 km/h
Min speed	10 km/h
Standard deviation	1 km/h
Mean service time	6 min per pallet
Online RL algorithm	
Budget	10 samples/action
Lookahead tree depth	2 levels

Table 2
Route 1. Units: Distances (km/h), load (euro-pallets), time window, arrival and departure time (hours since the start of the departure day), margin (hours).

Nodes	Distance	Load	Window	Arrival	Depart	Margin
Murcia	0	9.4			5.5	
Alicante	204.8	17.3		8.3	9.26	
Valencia	362.8	4.8		12	12.52	
Gaimersheim	2517.6	−5.6	74–77	75.33	75.83	1.16
Eching	2687.2	−1.7	76–83	78.7	79.33	3.66
Marktredwitz	2861.3	−13.9	78–83	81.35	82.03	0.96
Langenberg	3130.9	−10.3	98–101	97.79	98.5	2.5
Murcia	5417.8			162		

Table 3
Route 2. Units: Distances (km/h), load (euro-pallets), time window, arrival and departure time (hours since the start of the departure day), margin (hours).

Nodes	Distance	Load	Window	Arrival	Depart	Margin
Murcia	0	11.8			7.5	
Alicante	204.8	6.6		10.46	10.97	
Valencia	362.8	9.4		13.71	14.21	
Melsungen	1777.1	−4.4	50–59	57.08	57.64	1.36
Trostberg	2520.7	−6.8	74–77	75.5	76	0.99
Bielefeld	2624.1	−12.1	76–83	77.97	78.65	4.34
Neumunster	3038.8	−4.5	102–107	99.8	102.5	4.5
Murcia	5450.2			166.6		

Table 4
Route 3. Units: Distances (km/h), load (euro-pallets), time window, arrival and departure time (hours since the start of the departure day), margin (hours).

Nodes	Distance	Load	Window	Arrival	Depart	Margin
Murcia	0	15.9			7.78	
Alicante	204.8	6.1		10.66	11.18	
Valencia	362.8	9.8		13.93	14.43	
Grunheide	2238.5	−10.1	78–83	76.89	78.5	4.49
Landsberg	2319.8	−1.5	78–83	79.52	80.05	2.94
Coswig	2841.7	−8.6	102–107	97.25	102.5	4.5
Tuningen	3452.7	−11.6	126–131	124.3	126.6	4.36
Murcia	6157.1			193.6		

Table 5
Route 4. Units: Distances (km/h), load (euro-pallets), time window, arrival and departure time (hours since the start of the departure day), margin (hours).

Nodes	Distance	Load	Window	Arrival	Depart	Margin
Murcia	0.0	31.2			8.54	
Valencia	270.1	1.6		12.35	1.85	
Balingen	1967.0	−11.5	74–77	73.65	74.63	2.37
Lauenau	2430.1	−5.3	76–83	80.74	81.24	1.75
Meckenheim	2487.2	−8.3	76–83	81.93	82.57	0.42
Striegistal	2715.7	−2.7	98–101	96.23	98.5	2.5
Gonitz	2897.7	−5	98–107	100.6	101.1	5.9
Murcia	4879.8					

Table 6
Route 5. Units: Distances (km/h), load (euro-pallets), time window, arrival and departure time (hours since the start of the departure day), margin (hours).

Nodes	Distance	Load	Window	Arrival	Depart	Margin
Murcia		6.4			5.5	
Alicante	204.8	12.3		8.38	8.89	
Valencia	362.8	0.5		10.88	11.63	
Heddesheim	2464.3	−2.5	74–83	77.48	78.11	4.88
Valluhn	2827.0	−11	100–107	83.39	100.5	6.46
Erharting	3031.2	−5.7	100–107	102.7	103.3	0.56
Murcia	5055.4			163.2		

Table 7
Route 6. Units: Distances (km/h), load (euro-pallets), time window, arrival and departure time (hours since the start of the departure day), margin (hours).

Nodes	Distance	Load	Window	Arrival	Depart	Margin
Murcia		1.9			4.5	
Alicante	204.8	5.9		7.38	8	
Valencia	362.8	8		9.99	11.25	
Schwabach	2306.7	−11.1	74–77	74.62	75.19	1.08
Straubing	2430.6	−1.2	77–83	76.74	77.29	5.7
Malchow	2839.7	−3.2	77–83	82.07	82.63	0.37
Murcia	4828.9			142.6		

training/planning periods. Table 1 summarizes the simulation setup for these runs. In Section 5.4, we consider scenarios with a different degree of uncertainty.

The simulated routes correspond to a fleet of vehicles delivering to multiple customers, so each vehicle visits a subset of nodes. Vehicles pick up goods from some specific nodes in Spain (Murcia, Alicante, and Valencia), and deliver them to destination nodes in Germany. The parameters of these routes, shown in Tables 2, 3, 4, 5, 6, and 7, are: the distance to the origin of each location along the route, the amount of loaded goods (or delivered goods if the value is negative) at each location, and the delivery time window (in hours counted from the start of the departure day). As an example, Fig. 3 depicts *route 1*, which covers 5417.8 km in 162 h following the base policy with deterministic travel and service times. Additionally, to illustrate the adequacy of the base policy, these tables include the arrival and departure times with the base policy for a nominal speed of 80 km/h and a deterministic, load-dependent service time. The last column shows the time margins from the completion of each delivery to the end of its corresponding time window. As will become clear in the following subsection, the base policy is effective for these routes as long as there are no uncertainties in travel and service times on the routes.

5.1.1. Base policy

The base policy is both an element of our algorithm and a baseline in our numerical experiments that allows us to validate our algorithm by assessing the policy improvement principle. The base policy was designed with these goals in mind: first, it needed to be a lightweight procedure since it has to be executed thousands of times during each run; second, it should be effective at least under deterministic conditions; and third, it should be compliant with EU regulations on drivers' working conditions. In Section 5.3, we evaluate the impact of using alternative base policies.

The base policy detailed in Appendix B operates on the following principles: Decisions are mostly based on the upcoming customer, j, on the route, and, more specifically, on the customer's delivery time-window, which is delimited by a start time $t_{\text{start}}(j)$ and an end time $t_{\text{end}}(j)$. At each decision stage, the algorithm estimates the driving time $t_{\text{est}}(j)$ required to arrive at j at a constant speed and the service time $t_{\text{serv}}(j)$, considering a constant service time per pallet. Using these variables, the decisions are made as follows:

- When the vehicle stops, if the current working day has not finished, the decision is simply to take a break. Otherwise, the algorithm successively assesses which decision ensures on time delivery, starting from those providing a longer daily rest and

Fig. 3. Illustration of route 1, one long-haul route considered for the evaluation of the decision-making policies. The vehicle picks up goods from Murcia, Alicante, and Valencia, in Spain, and delivers them to several German locations.

finishing with a rest reduction and a driving extension (provided that both options are available). The policy selects the first decision capable of delivering the goods within the time window, aiming to avoid unnecessary rest reductions and/or driving extensions.

- Upon an arrival at a node, if the current time t is earlier than $t_{\text{start}}(j)$, the policy successively assesses which interruption fits in the time available until the window opens: a daily rest, a split rest (if available), or a break, selecting the first one that fits. A break will be taken if required to comply with the maximum 6 h of continuous work. Otherwise, the service starts.
- After finishing a service, if the working day has not ended, the selected action is to continue driving unless a break must be taken to comply with the maximum working hours. If the day has finished, the rest period starts.
- Whenever a decision to take a break is made, a split break (lasting 15 min) is taken if possible, i.e., if the current uninterrupted driving time is less than 4.5 h and if a break has not been taken previously.

In summary, the base policy tries to meet the delivery of the upcoming node on time while making efficient use of the available resources (time, driving extensions, and rest reductions).

5.1.2. DQN description and configuration

DQN is a state-of-the art, model-free deep RL algorithm especially suitable for controlling environments with discrete actions (like the one in this paper). DQN belongs to the family of Q-learning algorithms, which learn an approximation of the Q-function for all the state action pairs. DQN approximates the Q-function with a deep neural network (known as *Q-network*). To stabilize the learning process, it includes an *experience replay buffer* and a duplicate of the Q-network, which is used as the learning target during a predefined amount of training episodes (see Sutton and Barto (2018) and Mnih et al. (2015) for more details on Q-learning and DQN).

One of the drawbacks of DQN, and of model-free RL algorithms in general, is that it requires a non-negligible amount of hyperparameter tuning and offline training before it can be deployed on the controlled environment. For our experiments, we tuned and trained DQN for each one of the routes considered (a DQN agent capable of generalizing any possible route is a challenging problem that is out of the scope of this paper). The configuration parameters that remained fixed for all the routes are the following:

- The Q-network contains 2 hidden layers of 128 neurons each and rectified linear unit (ReLU) activation.
- The replay buffer stores up to 50 000 transitions.
- The batch size (number of samples taken from the replay buffer at each update) is 64.
- The target Q-network is updated every 100 episodes.
- The optimization algorithm is Adam, and the loss function is the mean squared error.

The above setting is fairly standard. In our experiments, increasing the number of neurons per hidden layer to 256 and the size of the replay buffer to 100 000 samples did not provide noticeable improvements.

The two hyperparameters that need to be carefully tuned are the learning rate α, which determines the step size at each update, and the epsilon decaying rate ϵ_r, which determines how the exploration parameter (ϵ) decays from its initial value (generally 1) to its minimum value (0.01 in our case). To find the best configuration of α and ϵ_r, we trained DQN on each route for all the combinations of $\alpha \in \{1 \cdot 10^{-7}, 5 \cdot 10^{-7}, 1 \cdot 10^{-6}, \ldots, 5 \cdot 10^{-3}, 1 \cdot 10^{-3}\}$ and $\epsilon_r \in \{0.99, 0.995, 0.999\}$. Each training experiment involved the simulation of 50 000 episodes, which was found to be long enough for DQN to converge. As a result of this campaign of training experiments, we found the following best configuration pairs (α, ϵ_r) for each route: Route 1: $(1 \cdot 10^{-5}, 0.995)$, Route 2: $(5 \cdot 10^{-5}, 0.995)$, Route 3: $(5 \cdot 10^{-5}, 0.99)$, Route 4: $(5 \cdot 10^{-5}, 0.995)$, Route 5: $(5 \cdot 10^{-5}, 0.999)$, Route 6: $(1 \cdot 10^{-5}, 0.995)$. It should be noted that training each configuration of the DQN agent took our server (an HP Proliant DL380 with 2 Intel Xeon E5-2650V3 CPUs) between 8 and 70 h, depending on the available computational resources. This volume of computation might render offline training impractical for freight companies that need to plan tens or even hundreds of routes daily.

5.2. Efficiency comparison

Fig. 4 shows the histograms for the 200 cost values obtained by each algorithm for each route. Complementing the results shown in this figure, Table 8 summarizes the statistical estimations of the mean and the standard deviation of the cost of each algorithm on each route, including the p-values for the null hypothesis that the performance of the considered algorithm is equivalent to ours. All the p-values are below 0.05, showing sufficient statistical evidence that our rollout policy outperforms both baselines in all the routes. Recall that the cost of a route is determined by the pallets delivered late, plus a penalty that accounts for the number of driving extensions and rest reductions. We can consider the route costs of the base policy under deterministic conditions to provide a reference for these estimated costs. In this ideal case, the costs are: 1 for routes 1 and 2 (both routes included one extension of the driving time), and 0 for routes 3, 4, 5, and 6.

We can draw the following conclusions from these results: First, it is evident that randomness in velocities and service times makes the problem more challenging, as evidenced by worse performance of the base policy. To illustrate the negative effect of uncertainty, let us consider route 1. In a deterministic scenario, the base policy attains a cost of 1. However, in a realistic route, a succession of unfavorable travel times could cause a late delivery to the third customer, increasing the cost from 1 to 8.6. Also, since the delay is cumulative, a late delivery to the third customer makes it more likely to arrive late to the fourth customer, raising the cost to 24.2. Thus, a slight inaccuracy in planning can have a very large impact on cost, hence, the significant differences in performance observed among policies. Second, even though our proposal uses the base policy for its rollout scheme, the resulting rollout policy clearly outperforms the base policy, reducing the cost by a factor of between 5 and 20. This empirical verification of the *policy improvement principle* (Bertsekas, 2019) validates our implementation. Third, although DQN is more resilient than the base policy, it is still less effective than the proposed algorithm. One possible explanation for this is that although both algorithms leverage the same prior knowledge about the environment (i.e., route details and the statistic characterization of velocities and service times), DQN, as all model-free algorithms, uses this information to learn an approximation of the Q-function *in advance*. This approximation should be valid for *any* state, not just for the specific states that will be visited during the route and which are impossible to know during offline training. In contrast, our proposal exploits this information in decision time so that once a state is observed, our online algorithm can devote all available resources to estimating the Q-function *at this particular state*. As a result, the online estimation can be more accurate than one that generalizes from visits to other states, enabling more effective decision-making.

The strategy of estimating the Q-function once the state is observed is known as *decision-time planning* (Sutton and Barto, 2018) and has the well-known drawback of increased computational cost per decision (due to the simulation of the multiple trajectories across the lookahead tree), which limits its use to applications that do not require fast responses. This is precisely the case of our en-route decision-making problem: in any of the events where a decision should be made (`STOP_DRIVING`, `ARRIVAL_TO_NODE`, `END_SERVICE`), a latency of around 1 min can be considered acceptable. Section 5.4 discusses this issue in further detail. Table 9 summarizes the main differences between DQN, a model-free RL approach, and our model-based online RL proposal.

5.3. Influence of the base policy

While the policy improvement principle guarantees that the rollout policy will outperform the base policy, it does not clarify to what extent the base policy determines the performance of the rollout policy. In this subsection, we address this issue by evaluating and comparing the performance of our proposal under three different base policies: (1) base policy 1, detailed in Appendix B, is the one used in the previous section; (2) base policy 2 is a simplification of base policy 1, from which it removes the options of extending driving time and reducing rest periods; and (3) base policy 3, which simplifies base policy 2 by removing the options of taking split breaks or split rests. We have repeated the performance evaluation experiments of the rollout policy using base policies 2 and 3. Table 10 shows the average cost and confidence interval (with a confidence level of 95%) for each route. Also, for policies 2 and 3, we show the p-value for the null hypothesis that using the alternative base policy provides similar performance to using base policy 1.

We see that the performance of the rollout policy is very similar with the three base policies. The confidence intervals are mostly overlapping, and most p-values are greater than 0.05, exceeding 0.3 in many cases. In conclusion, it cannot be stated that the design of the base policy has a notable influence on rollout policy performance. Recall that by using a base policy to approximate the cost-to-go of the trajectories, we introduce a pessimistic bias into the Q-value estimations. Nevertheless, these results suggest that this bias has little influence on the comparison of the actions, i.e., base policies tend to worsen the Q-values of all the actions in such a way that their relative differences in performance are kept, at least for the best actions, from one base policy to another. Consequently, the rollout policy tends to select similar actions (similar behavior) under different base policies.

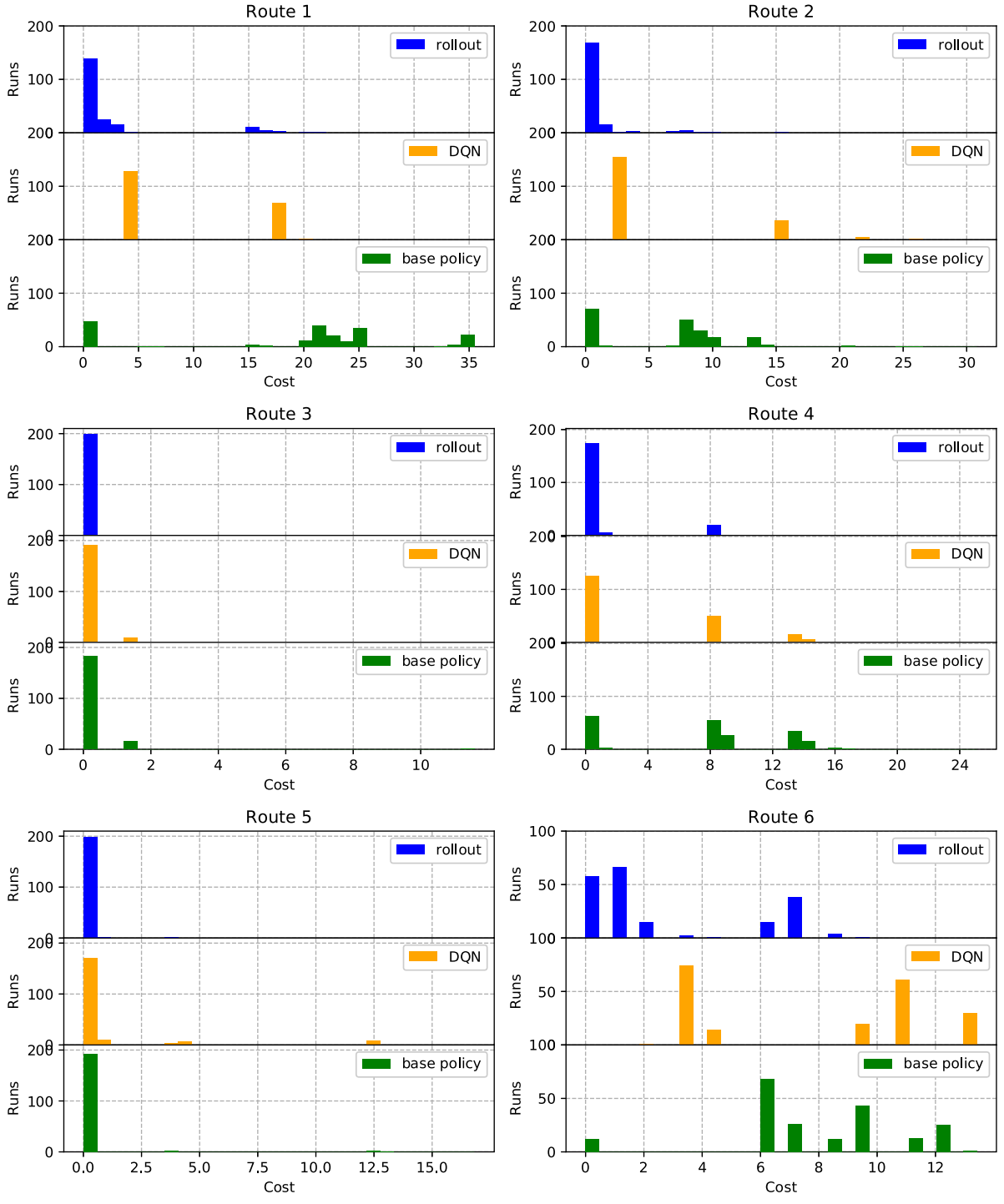


Fig. 4. Histograms of the 200 cost values obtained with each algorithm on each route.

5.4. Parameter configuration and computational overhead

As discussed in Section 4.3, the prediction accuracy of our online RL algorithm is determined by two parameters: the depth of the lookahead tree ℓ and the sample budget S. Increasing the values of these parameters enhances prediction accuracy at each decision stage, allowing the algorithm to make more effective decisions at the cost of greater computational effort. Therefore, we need to quantify the accuracy-computation tradeoff to assess the feasibility of our proposal. Moreover, we evaluate its robustness against different degrees of uncertainty in travel and service times.

Table 8
Statistical comparison of route costs obtained with the base policy, the DQN policy and, our rollout policy.

Route	Base policy			DQN policy			Rollout policy	
	Mean	var	p-value	Mean	var	p-value	Mean	var
1	18.9	121.45	$8.34 \cdot 10^{-60}$	8.95	44.79	$5.86 \cdot 10^{-26}$	2.38	23.41
2	6.56	24.43	$2.41 \cdot 10^{-36}$	6.11	36.5	$1.44 \cdot 10^{-23}$	1.35	3.78
3	0.178	0.825	0.0043	0.0675	0.097	0.0052	0.0075	0.011
4	7.23	30.6	$2.22 \cdot 10^{-40}$	3.82	28.93	$3.03 \cdot 10^{-12}$	0.85	6.21
5	0.27	2.67	0.013	0.37	2.96	0.0022	0.016	0.0512
6	7.26	7.14	$1.57 \cdot 10^{-60}$	7.06	13.3	$6.96 \cdot 10^{-44}$	2.04	7.09

Table 9
Quantitative and qualitative comparison of DQN vs. our model-based online RL proposal combining MCP, rollout and MCTS.

Feature	DQN	MPC + rollout + MCTS
Training time	8–70 h per route, depending on the available computational resources	No offline training is required
Decision time	0.6 ms	25 s ($S = 10$, $\ell = 2$), 0.67 s ($S = 10$, $\ell = 1$)
Performance (improvement over base policy)	22%	83% ($S = 10$, $\ell = 2$), 79% ($S = 10$, $\ell = 1$)
Hyperparameters	α, ϵ_r, replay buffer size, Q-network update frequency, multiple deep neural network hyperparameters (e.g. layers, dimensions, activation function, optimizer, batch size)	S and ℓ
Configuration	Per-route hyperparameter tuning is required	Larger S and ℓ values improve performance but increase decision time
Flexibility (to changes in the environmental conditions)	The policy must be re-trained using a route simulator that includes the changes in the environment	Only requires a simulator update (no re-train needed)

Table 10
Statistical comparison of route costs obtained by the rollout policy with three different base policies. Each statistical value is obtained for 200 runs of each route.

Route	Base policy 1			Base policy 2				Base policy 3			
	Mean	95% CI		Mean	95% CI		p-value	Mean	95% CI		p-value
		Low	Up		Low	Up			Low	Up	
1	2.39	1.71	3.06	2.30	1.68	2.92	0.428	2.61	1.91	3.32	0.322
2	1.35	1.08	1.62	1.00	0.79	1.21	0.022	1.20	0.92	1.32	0.088
3	0.01	0.00	0.02	0.00	0.00	0.00	0.159	0.00	0.00	0.00	0.159
4	0.86	0.51	1.20	1.23	0.84	1.67	0.073	0.80	0.46	1.14	0.417
5	0.10	0.00	0.23	0.02	0.00	0.05	0.109	0.16	0.00	0.03	0.309
6	2.24	1.84	2.66	1.62	1.26	1.98	0.012	1.95	1.59	2.31	0.141

We carried out a campaign of experiments in which we evaluated our online RL algorithm using all the (S, ℓ) pairs for $S \in \{5, 10, 15, 20\}$ and $\ell \in \{1, 2\}$ to quantify the computation overhead. We run 200 simulations of the 6 routes described previously for each configuration, measuring route costs and the elapsed time at each decision stage. Fig. 5 shows the average decision time per stage for each parameter setting. Given the time scale of the controlled process, where driving periods last several hours and service times last at least 30 min, the 25 second decision time of the selected configuration ($S = 10$, $\ell = 2$) is within acceptable margins for an en-route execution of the algorithm. Although it may be feasible to use configurations of greater depth and budget, it is advisable to be aware of the exponential increase in computational time that this entails.

We have defined two additional scenarios which are variations of the reference scenario specified in Table 1 to evaluate the performance of each configuration under different degrees of uncertainty:

1. High uncertainty scenario: the standard deviation of the random variables is doubled with respect to the reference scenario.
2. Low uncertainty scenario: the standard deviation of velocity is divided by 5, the standard deviation of service time is divided by 10, and no en-route incidents occur.

Fig. 6 shows the average costs (6 routes, 200 runs per route) at each scenario under different configurations of S and ℓ. As expected, higher uncertainty implies higher average costs. These results also confirm that the parameter configuration is relatively straightforward: higher values of S and ℓ provide better quality decisions (thus lower expected cost) at the expense of longer computation time (which is as shown in Fig. 5 for the three scenarios). Fortunately, even under high uncertainty conditions, configurations requiring a moderate computational effort attain notable cost reductions compared to the base policy: the average costs obtained by the base policy at each scenario are 2.23 for the low uncertainty scenario, 6.01 for the reference scenario, and 6.77 for the high uncertainty scenario. Note that our selected configuration ($S = 10$, $\ell = 2$) achieves a reasonable balance between computation time (25 seconds per decision) and performance since no significant improvements are observed beyond $S = 10$. However, we could greatly reduce the decision time without significantly increasing the cost. For example, by setting $\ell = 1$, we

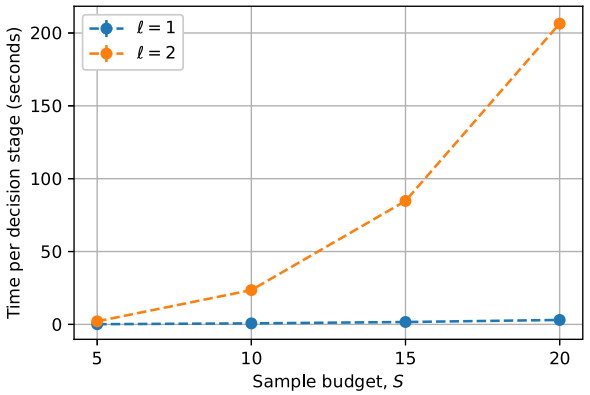
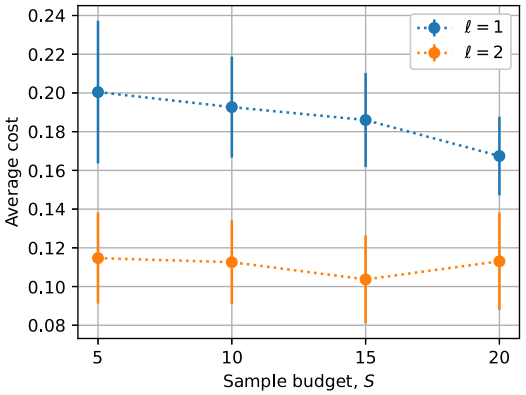
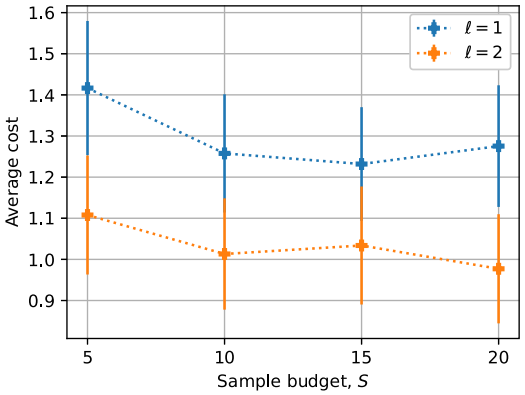


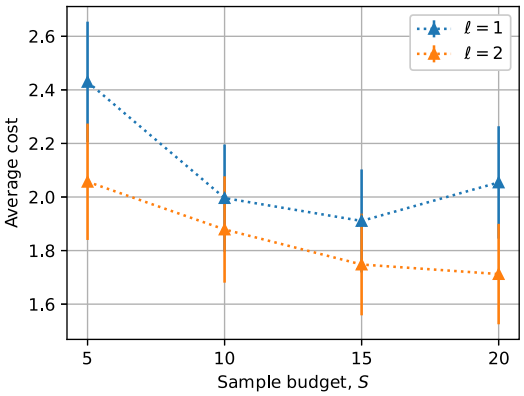
Fig. 5. Average elapsed time per decision stage as a function of the sample budget for $\ell = 1$ and $\ell = 2$.



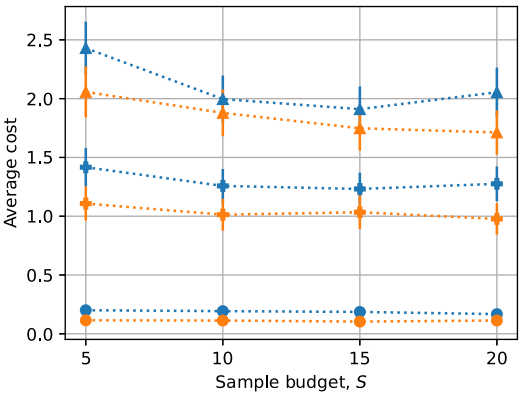
(a) Low uncertainty scenario.



(b) Reference scenario.



(c) High uncertainty scenario.



(d) Comparison of the three scenarios.

Fig. 6. Plots of the average cost achieved with our proposal in each scenario under different configurations of S and ℓ. Each estimation has been obtained from 1200 experiments. Confidence intervals are computed with a 95% confidence level.

reduce the decision time from 25 seconds to 0.69 s, while the cost increment is less than 0.25. Finally, it must be highlighted that having to configure only two parameters (of predictable effect) is an advantage over model-free methods, especially those using deep neural networks that involve multiple hyperparameters.

6. Conclusions and future work

In long-distance road transport, en-route decisions regarding the duration of driving and rest periods must assure the delivery of goods within their time windows while complying with regulations on driving time, breaks and rest periods. We have shown that, when travel and service times are deterministic, a set of heuristic decision rules can be sufficient to accomplish this task. However, in real-life journeys, these times are random, and unforeseen incidents may occur introducing additional delays. Under these circumstances these heuristics can be ineffective, resulting in higher route costs, measured in terms of out-of-time delivered goods, drive extensions and rest reductions.

We propose a model-based RL method combining MPC, rollout, and MTCS for en-route decision-making. At each decision stage, our method anticipates future events by generating a set of trajectories (simulations of the route) starting from the current state of the route. In these trajectories, the algorithm evaluates all the available actions at future stages up to a given horizon, after which the generated trajectory follows the base policy. Each trajectory provides a sample of the optimal cost-to-go from the current stage using a given action, allowing the algorithm to pick the best estimated action at the current stage. Since obtaining samples is computationally expensive, our method incorporates an MAB algorithm to maximize sample efficiency in the search for the optimal action at each level of the lookahead tree. We have conducted extensive experiments on realistic routes based on the real-life experience of a Spanish freight company. In our numerical experiments, our RL method reduced the cost of the base policy by a factor of between 5 and 20, in accordance with the policy improvement principle applicable to rollout methods. Compared to a state-of-the-art, model-free RL algorithm, such as DQN, our algorithm does not need to be trained offline, which can be a very time consuming process, especially if it includes hyperparameter tuning. Instead, our proposal integrates learning and control, thus requiring more computational effort at decision time. To assess the feasibility of this online operation, we have quantified the tradeoff between computational overhead and prediction accuracy, determined by the two main parameters of the algorithm: the sample budget per action and the length of the lookahead horizon. Our results show that notable cost reductions can be attained with computation times that are compatible with real-time operation.

Nevertheless, it should be noted that including additional degrees of freedom in en-route decisions will increase the computational cost of each decision. For example, some companies would be interested in adding tactical decisions, such as unloading some goods in a nearby logistic center to outsource the last-mile delivery, like in Alcaraz et al. (2019). The inclusion of additional actions and events in the problem is an interesting future research line. Another potential improvement would be to increase the accuracy of the estimations by obtaining empirical distributions for travel velocity and service times, which could be even particularized to road segments and locations, leveraging the experience and data gathered by the freight company.

Acknowledgments

This work has been funded by Consejeria de Desarrollo Economico, Turismo y Empleo, Region de Murcia, under the RIS3Mur project grant SiSPERT (ref. 2I16SAE00023), and by project Grant PID2020-116329GB-C22 funded by MCIN/AEI/10.13039/501100011033.

Appendix A. Creation of the action sets

In this appendix, we show how to determine the actions included in the set $U_k(x_k)$. Each action in $U_k(x_k)$ must comply with the regulations considering the driver's variables in state x_k. Fig. A.7 shows the conditions associated to each action at each state. Tables A.11 and A.12 summarize the variables and constants involved, respectively.

Table A.11
Variables considered when determining the action sets.

Variable	Definition
j	Next node to be served
$t_{\text{start}}(j)$	Start of j's time window
$t_{\text{end}}(j)$	End of j's time window
$t_{\text{service}}(j)$	Expected service time at j
t_{rest}	Available rest time for the current day (11 or 9 h)
$t_{\text{until\_rest}}$	Time until the start of the rest period (13 or 15 h)
t_{drive}	Driving time during the current day
$t_{\text{on\_road}}$	Current uninterrupted driving time
DRIVE_ALLOWED	Boolean indicating if driving is allowed (according to daily and weekly drive time, rest periods, work time, and so forth)
BREAK_DONE	Indicating whether a break (of any length) has been taken in the current day
t_{break}	Time of a break (45, 30 or 15 min for a normal, short, or split break respectively)
DRIVE_EXTENDED	Boolean indicating whether the driving extension option has been used in the current day
REST_REDUCED	Boolean indicating whether the rest reduce option has been used in the current day

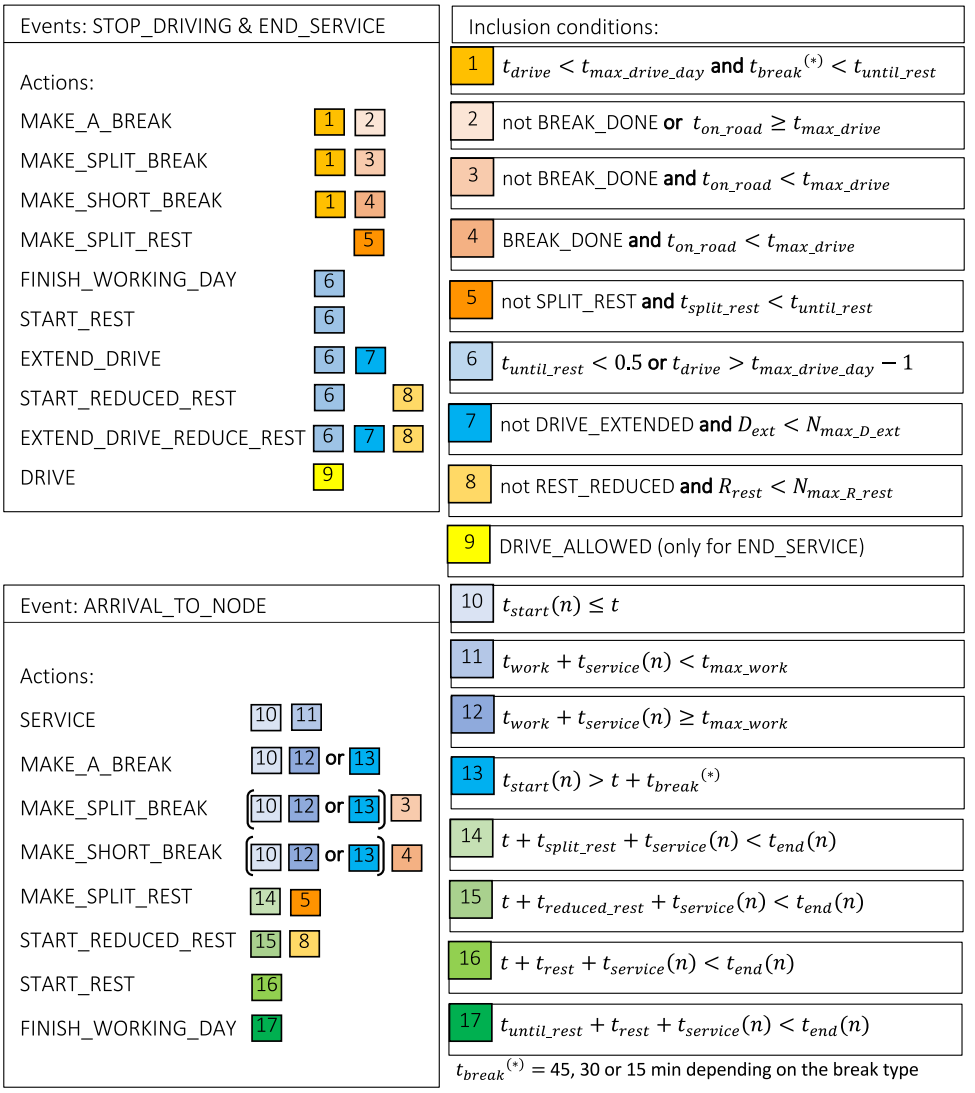


Fig. A.7. Diagram illustrating the conditions that should be met for each action to be included in the set of available actions.

Table A.12
Variables considered when determining the action sets.

Variable	Definition
$t_{\text{max\_drive\_day}} = 9$ h	Maximum allowed daily drive time, without drive extension
$t_{\text{split\_rest}} = 3$ h	Minimum duration of the first rest period in case of split break
$t_{\text{max\_work}} = 6$	Maximum duration of an uninterrupted work period
$N_{\text{max\_ D\_ext}} = 2$	Maximum number of driving extensions
$N_{\text{max\_R\_rest}} = 3$	Maximum number of rest reductions

Appendix B. Base policy

In this appendix, we detail the base policy in Algorithm 5. In addition to the variables defined in Appendix A, the base policy uses $t_{\text{est}}(j)$, which denotes the estimated time to arrive to the next node, and $t_{R\ \text{rest}} = 9$ h, corresponding to the duration of a reduced rest period, `REDUCIBLE_REST` is a boolean indicating if it is possible to reduce the rest time during the current day, `EXTENSIBLE_DRIVE` is a boolean indicating if it is possible to extend the driving time during the current day, and t_{ext} denotes the driving time added by a driving extension. An additional check is made after running Algorithm 5: whenever u_k is set to `MAKE_A_BREAK`, if $t_{\text{drive}} < t_{\text{max drive}}$ and `BREAK_DONE` are inactive, u_k is re-set to `MAKE_SHORT_BREAK`.

Algorithm 5 BasePolicy

Input: Current state x_k.
Output: Action u_k
if event variable in x_k = STOP_DRIVING **then**
  if $t_{\text{until\_rest}} > 0$ or $t_{\text{drive}} < t_{\text{max\_drive\_day}}$ **then**
    u_k = MAKE_A_BREAK
  else if $t + t_{\text{until\_rest}} + t_{\text{rest}} + t_{\text{est}}(j) < t_{\text{start}}(j)$ **then**
    u_k = FINISH_WORKING_DAY
  else if $t + t_{\text{rest}} + t_{\text{est}}(j) + t_{\text{service}}(j) < t_{\text{end}}(j)$ **then**
    u_k = START_REST
  else
    if $t + t_{R\ \text{rest}} + t_{\text{est}}(j) + t_{\text{service}}(j) < t_{\text{end}}(j)$ and REDUCIBLE_REST **then**
      u_k = START_REDUCED_REST
    else if $t_{\text{ext}} < t_{\text{est}}(j)$ and EXTENSIBLE_DRIVE **then**
      u_k = EXTEND_DRIVE
    else if EXTENSIBLE_DRIVE and REDUCIBLE_REST **then**
      u_k = EXTEND_DRIVE_&_REDUCE_REST
    else
      u_k = START_REST
    end if
  end if
else if event variable in x_k = ARRIVAL_TO_NODE **then**
  if $t_{\text{start}}(j) > t + t_{\text{rest}}$ **then**
    u_k = START_REST
  else if $t_{\text{start}}(j) > t + t_{\text{split\_rest}}$, $t + t_{\text{split\_rest}} < t_{\text{until\_rest}}$ and not SPLIT_REST **then**
    u_k = MAKE_SPLIT_REST
  else if $t_{\text{start}}(j) > t + t_{\text{break}}$ **then**
    u_k = MAKE_A_BREAK
  else if $t_{\text{work}} + t_{\text{service}}(j) < t_{\text{max\_work}}$ **then**
    u_k =SERVICE
  else
    u_k = MAKE_A_BREAK
  end if
else if event variable in x_k = END_SERVICE **then**
  if $t_{\text{until\_rest}} > 0$ and $t_{\text{drive}} < t_{\text{max\_drive\_day}}$ **then**
    if $t_{\text{work}} < t_{\text{max\_work}}$ **then**
      u_k = DRIVE
    else
      u_k = MAKE_A_BREAK
    end if
  else
    u_k = START_REST
  end if
end if

References

Alcaraz, J.J., Caballero-Arnaldos, L., Vales-Alonso, J., 2019. Rich vehicle routing problem with last-mile outsourcing decisions. Transp. Res. E 129, 263–286.

Audibert, J.-Y., Bubeck, S., Munos, R., 2010. Best arm identification in multi-armed bandits. In: COLT. pp. 41–53.

Bernhardt, A., Melo, T., Bousonville, T., Kopfer, H., 2016. Scheduling of Driver Activities with Multiple Soft Time Windows Considering European Regulations on Rest Periods and Breaks. Tech. Rep., Schriftenreihe Logistik der Fakultät für Wirtschaftswissenschaften der htw saar.

Bernhardt, A., Melo, T., Bousonville, T., Kopfer, H., 2017. Truck Driver Scheduling with Combined Planning of Rest Periods, Breaks and Vehicle Refueling. Tech. Rep., Schriftenreihe Logistik der Fakultät für Wirtschaftswissenschaften der htw saar.

Bertsekas, D.P., 2019. Reinforcement Learning and Optimal Control. Athena Scientific Belmont, MA.

2006. Regulation (EC) No 561/2006 of the European parliament and of the council of 15 March 2006 on the harmonisation of certain social legislation relating to road transport and amending Council Regulations (EEC) No 3821/85 and (EC) No 2135/98 and repealing council regulation (EEC) No 3820/85, OJ L 102, 1–14.

2002. Directive 2002/15/EC of the European parliament and of the council of 11 march 2002 on the organisation of the working time of persons performing mobile road transport activities, OJ L 80, 35–39.

Goel, A., 2009. Vehicle scheduling and routing with drivers' working hours. Transp. Sci. 43 (1), 17–26.

Goel, A., 2018. Legal aspects in road transport optimization in europe. Transp. Res. E 114, 144–162.

Gromicho, J., van Hoorn, J., Kok, A., Schutten", J., 2012. Restricted dynamic programming: A flexible framework for solving realistic VRPs. Comput. Oper. Res. 39 (5), 902–909.

Gutierrez, A., Dieulle, L., Labadie, N., Velasco, N., 2018. A multi-population algorithm to solve the VRP with stochastic service and travel times. Comput. Ind. Eng. 125, 144–156.
Jabali, O., Leus, R., Van Woensel, T., De Kok, T., 2015. Self-imposed time windows in vehicle routing problems. OR Spectrum 37 (2), 331–352.
James, J., Yu, W., Gu, J., 2019. Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. IEEE Trans. Intell. Transp. Syst. 20 (10), 3806–3817.
Karnin, Z., Koren, T., Somekh, O., 2013. Almost optimal exploration in multi-armed bandits. In: International Conference on Machine Learning. PMLR, pp. 1238–1246.
Kleff, A., 2019. Scheduling and Routing of Truck Drivers Considering Regulations on Drivers' Working Hours (Ph.D. thesis). Karlsruher Institut für Technologie (KIT).
Kok, A.L., Meyer, C.M., Kopfer, H., Schutten, J.M.J., 2010. A dynamic programming heuristic for the vehicle routing problem with time windows and European community social legislation. Transp. Sci. 44 (4), 442–454.
Kovacs, A.A., Parragh, S.N., Doerner, K.F., Hartl, R.F., 2012. Adaptive large neighborhood search for service technician routing and scheduling problems. J. Sched. 15 (5), 579–600.
Laporte, G., 2016. Scheduling issues in vehicle routing. Ann. Oper. Res. 236 (2), 463–474.
Lattimore, T., Szepesvári, C., 2020. Bandit Algorithms. Cambridge University Press.
Li, G., Li, J., 2020. An improved tabu search algorithm for the stochastic vehicle routing problem with soft time windows. IEEE Access 8, 158115–158124.
Li, X., Tian, P., Leung, S.C., 2010. Vehicle routing problems with time windows and stochastic travel and service times: Models and algorithm. Int. J. Prod. Econ. 125 (1), 137–145.
Liang, E., Wen, K., Lam, W.H., Sumalee, A., Zhong, R., 2021. An integrated reinforcement learning and centralized programming approach for online taxi dispatching. IEEE Trans. Neural Netw. Learn. Syst..
Liu, S., Jiang, H., Chen, S., Ye, J., He, R., Sun, Z., 2020. Integrating Dijkstra's algorithm into deep inverse reinforcement learning for food delivery route planning. Transp. Res. E 142, 102070.
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.
Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E., 2021. Reinforcement learning for combinatorial optimization: A survey. Comput. Oper. Res. 105400.
Miao, F., Han, S., Lin, S., Stankovic, J.A., Zhang, D., Munir, S., Huang, H., He, T., Pappas, G.J., 2016. Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach. IEEE Trans. Autom. Sci. Eng. 13 (2), 463–478.
Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.
Nazari, M., Oroojlooy, A., Takac, M., Snyder, L.V., 2018. Reinforcement learning for solving the vehicle routing problem. In: Adavances in Neural Network Information Processing, Vol. 31. Neural Networks Information Processing Systems (NIPS), pp. 9861–9871.
Poullet, J., 2020. Leveraging machine learning to solve The vehicle Routing Problem with Time Windows (Ph.D. thesis). Massachusetts Institute of Technology.
Prescott-Gagnon, E., Desaulniers, G., Drexl, M., Rousseau, L.-M., 2010. European driver rules in vehicle routing with time windows. Transp. Sci. 44 (4), 455–473.
Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. MIT Press.
Taş, D., Dellaert, N., van Woensel, T., De Kok, T., 2014. The time-dependent vehicle routing problem with soft time windows and stochastic travel times. Transp. Res. C 48, 66–83.
Vareias, A.D., Repoussis, P.P., Tarantilis, C.D., 2019. Assessing customer service reliability in route planning with self-imposed time windows and stochastic travel times. Transp. Sci. 53 (1), 256–281.
Zäpfel, G., Bögl, M., 2008. Multi-period vehicle routing and crew scheduling with outsourcing options. Int. J. Prod. Econ. 113 (2), 980–996.
Zhao, J., Mao, M., Zhao, X., Zou, J., 2020. A hybrid of deep reinforcement learning and local search for the vehicle routing problems. IEEE Trans. Intell. Transp. Syst. http://dx.doi.org/10.1109/TITS.2020.3003163, (Early Access).