

A Deep Reinforcement Learning Algorithm for the Vehicle Routing Problem with Stochastic Demands and Outsourcing

Mohsen Dastpak^{a,b,c,*}, Fausto Errico^{a,b,c}, Ola Jabali^{c,d}

^a*Department de génie de la construction, École de technologie supérieure, , Montréal, H3C 1K3, QC, Canada*

^b*GERAD, , Montréal, H3T 1J4, QC, Canada*

^c*CIRRELT, , Montréal, H3C 3J7, QC, Canada*

^d*Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci 32, Milano, 20133, , Italy*

Abstract

We introduce the vehicle routing problem with stochastic demands and outsourcing options (VRP-SDO). In this problem, we consider a logistics service provider (LSP) that must partition a set of customer requests with uncertain demands into customers outsourced to a common carrier and customers committed to its own fleet. The former are exclusively served by the common carrier, whereas the LSP fully serves its committed customers with a fixed fleet. The latter induces a vehicle routing problem with stochastic demand (VRP-SD), which is solved dynamically while allowing multiple vehicle visits to a customer. Specifically, the demand of committed customers is revealed once visited. If the vehicle capacity is not sufficient to serve it, the residual demand may be served by other vehicles or by the same vehicle after having restocked at the depot. Moreover, vehicles are allowed to replenishment trips to the depot. Vehicles exceeding regular work-shift duration incur an overtime cost. The unit outsourcing cost is inversely correlated with the expected total demand of the outsourced customers. The VRP-SDO objective is to minimize expected total travel and overtime costs, as well as the outsourcing costs.

We propose an iterative two-level solution methodology for the VRP-SDO. The first level partitions customers into committed and outsourced subsets, while the second level solves the resulting VRP-SD and estimates its expected routing cost. Solving this stochastic dynamic problem for a fixed set of committed customers is computationally demanding. Given that the VRP-SD must be evaluated repeatedly for different subsets of committed customers throughout the iterative search, solving it from scratch at every iteration is computationally impractical. Therefore, we propose learning an offline routing policy which estimates routing costs almost instantaneously for any subset of committed customers. We demonstrate our methodology using an iterated local search (ILS) algorithm that primarily establishes the first-level partitions.

We formulate the second level problem as a Markov decision process and solve it using a deep Q-network (DQN). To represent the state of the system, we propose a graph attention network (GAT) that aggregates customer and vehicle information based on their relevance to the vehicle selecting an action, capturing the spatial structure induced by their relative locations. The DQN is trained offline on instances with variable customer cardinality and location, yielding a routing policy applicable to any daily realization of customers. We further introduce an online fine-tuning step that improves the accuracy of the routing cost approximation during the search.

Extensive experiments show that our DQN for the VRP-SD reduces routing costs by 19.6% relative to a state-of-the-art methodology, and by at least 29.6% when compared to classical routing heuristics. Our overall solution algorithm for the VRP-SDO yields savings of 13.7% on average relative to the version whose routing policy does not use the GAT representation. Furthermore, our method generates the high-quality decisions within minutes, whereas the benchmarks that do not use an offline-trained cost estimator require more than one hour on average.

Keywords: Stochastic Vehicle Routing Problem, Optional Outsourcing, Deep Reinforcement Learning

*Corresponding author

Email address: mohsen.dastpak.1@ens.etsmtl.ca (Mohsen Dastpak)

1. Introduction

The volume of last mile delivery services has significantly increased over the past years. In 2024 the number of packages delivered in the US was 22.4 billion, up from 15.4 billion in 2019 (Pitney Bowes 2024). Logistics service providers (LSPs) operating next-day delivery services typically manage a fixed set of resources (e.g., trucks). These are often insufficient to handle varying demand volumes. Furthermore, expanding resources to serve areas away from an LSP’s center of activity is often economically unviable. As a result, many LSPs structurally embed outsourcing services to fully satisfy the demand of their customers. For example, while Amazon has significantly grown its internal logistics network, analyses indicate that third-party carriers continue to handle a substantial number of packages, estimated at nearly 30% of total volume as of late 2021 (Tarasov 2021).

The use of outsourcing services by LSPs is governed by contractual agreements. In the context of next-day delivery, operational practices described in the literature indicate that an LSP is often obliged to communicate its *outsourced customers* of a given day by the morning of that day (see Canada Post Corporation (2025) for the *Parcel Services Customer Guide* by Canada Post). In turn, the LSP commits to serving the remainder set of customers, which we refer to as *committed customers*. This setting has been treated in the scientific literature as the VRP with private fleet and common carrier (VRP-PFCC) (Gahm et al. 2017, Baller et al. 2020, e.g.). As most routing problems, the VRP-PFCC is subject to uncertainty on multiple dimensions in practice (e.g., travel times and customer demands).

Routing under uncertainty may be cast under multiple modeling frameworks, such as stochastic programming, robust optimization, and stochastic dynamic programming (SDP), see Oyola et al. (2017, 2018). The latter has been gaining increased attention, as LSPs are gradually accepting to take decisions dynamically. Moreover, the methodological advances in stochastic dynamic routing problems have made the theoretical advantages of such models a tangible reality. Such advances are largely attributed to the integration of reinforcement learning in SDP problems. In this paper, we consider a stochastic dynamic variant of the VRP-PFCC, where customer demands are uncertain. Accordingly, we introduce the vehicle routing problem with stochastic demands and outsourcing options (VRP-SDO). In this problem, the LSP receives customer requests at the beginning of the operational day, requiring a rapid decision regarding which customers to outsource to a third-party carrier and which to commit to. As in Dabia et al. (2019), we consider that the cost per unit of outsourcing is inversely correlated with the expected total demand of the outsourced customers. Furthermore, the demand of customers is uncertain, and we assume that outsourced customers are exclusively served by the common carrier, whereas the LSP fully serves its committed customers, while possibly incurring overtime costs. In particular, we consider that vehicles are dynamically assigned to customers, and allow different vehicles to visit a customer, if a single visit is not sufficient.

The LSP must solve the VRP-SDO every day within a limited computing budget. From an application standpoint, the major limitation to the computing budget is the need to communicate the set of outsourced customers early in the day. The cost of partitioning a set of customers between outsourced ones and committed ones is not trivial to compute. In particular, the set of committed customers induces a stochastic dynamic problem, which we denote as the vehicle routing problem with stochastic demands (VRP-SD). Therefore, we propose an iterative two-level solution methodology. Such a hierarchy is common in the deterministic VRP-PFCC literature (Stenger et al. 2013, Vidal et al. 2016, Dabia et al. 2019). Specifically, the first-level partitions the customers into the two groups, allowing the immediate cost evaluation of the outsourced customers. In order to maintain a viable iterative procedure between the two stages, evaluating the costs of various sets of committed customers needs to be efficiently done. Indeed, any algorithm based on the exploration of the first-level decision space would require many evaluations of the second-level decision problem. However, solving the second-level routing problem (VRP-SD) for any fixed set of committed customers is computationally intensive. This subproblem, especially due to its stochastic and dynamic nature, may require seconds to several minutes for its solution (Erera et al. 2010, Mendoza et al. 2016). While these run times may seem acceptable in isolation, they become a significant bottleneck when the problem must be solved repeatedly during the execution of the solution algorithm. The strategy proposed in this paper aims then at transferring a consistent part of the computational burden from everyday online execution to an offline learning approach.

The choice of an offline learning approach is motivated by the fact that the complete set of customers to serve in the VRP-SDO differs from day to day. Furthermore, this complete set of customers is typically known shortly prior to establishing the sets of committed and outsourced customers. Thus, it is impractical to retrain a reinforcement learning algorithm on a new set of customers every day. As our methodology relies on deploying a trained algorithm, we propose training the algorithm on a VRP-SD while also considering the set of customers to be variable in terms

of cardinality and location. This allows estimating the routing cost of a VRP-SD in instant time for any daily realization of customers. We showcase our methodology by using an iterated local search (ILS) algorithm that primarily establishes the first-level partitions. However, it is worth mentioning that our methodology can be applied to other metaheuristics.

We formulate the VRP-SD as a Markov decision process (MDP), whose value function approximates the routing costs associated with a given set of committed customers as an input. To solve the resulting MDP, we build upon the methodology put forward by [Dastpak et al. \(2023\)](#). The uncertain parameters considered in that paper resemble those of the VRP-SD. However, [Dastpak et al. \(2023\)](#) deal with maximizing the expected served demands while meeting vehicle capacity and time restrictions. One of the main methodological contributions of that paper lies in using an observation function to guide decision-making. The proposed observation function represents the state of the system as a fixed-size vector, comprising a limited yet relevant set of features. These features were hand-selected based on various experiments. Moreover, the observation function heuristically chooses a set of target customers, which may be visited. As such, the proposed observation function does not capture the underlying graph structure induced by customer locations and therefore fails to represent spatial relationships such as proximity and neighborhood interactions. Lastly, customer information is discretized using a grid-based technique, which may greatly impact the precision of the overall methodology. To address these limitations, we propose a graph attention network (GAT) architecture to represent the observation function. This architecture employs an attention mechanism ([Vaswani et al. 2017](#)) to aggregate customers' and vehicles' information into a fixed-size vector by considering their relevance to the vehicle that needs to take an action. One of the core contributions of this paper is to demonstrate the power of GATs to represent a concise but informative observation vector.

Considering the GAT, we propose a deep Q-network (DQN) for the VRP-SD. DQNs are reinforcement learning algorithms combining Q-learning with deep neural networks. We propose a Q-learning algorithm using a two-layer neural network to approximate the value of observation-action pairs, also referred to as the Q-network. In particular, we train the Q-network alongside the proposed GAT architecture of the observation function, resulting in a deep Q-network. We denote the proposed routing method for solving the VRP-SD as DQNCO. We employ several techniques to enhance the overall performance of the Q-learning algorithm, including replay memory ([Mnih et al. 2015](#)) and double Q-network ([Van Hasselt et al. 2016](#)). We name the entire solution method, which is the ILS algorithm with the embedded routing policy, as IDQNCO. To further enhance the performance of the IDQNCO, we introduce an online fine-tuning step inside the ILS algorithm to improve the accuracy of the routing cost value function. The IDQNCO with this additional step is denoted as IDQNCO⁺.

We conduct comprehensive experiments to evaluate the performance of the proposed algorithm, comparing it with six benchmark methods. We first show the superiority of the GAT-based architecture introduced in the DQNCO over all the other benchmarks, including SQNCO, the routing policy from [Dastpak et al. \(2023\)](#), based on the hand-engineered observation function. We then demonstrate that IDQNCO⁺ significantly outperforms all the other benchmarks. Notably, our method reduces overall costs by more than 22.3% compared to benchmarks with basic routing policies and by an average of 13.7% compared to ISQNCO⁺, which combines the same ILS algorithm and fine-tuning step with the SQNCO routing policy. Summarizing, the contributions of this paper are as follows:

- We introduce the VRP-SDO, as a stochastic variant of the VRP-PFCC, and propose a two-level formulation for it.
- We formulate the outsourcing problem as a binary integer programming problem and the VRP-SD as an MDP.
- We propose a DQN-based algorithm that is trained offline and rapidly estimates the routing costs of any committed customer set within the ILS.
- We propose a GAT architecture to represent the active vehicle's observation, taking into account the graph-like structure of customers and vehicles in the service area, as well as their relevance to the active vehicle.
- We conduct extensive experiments and demonstrate that our method outperforms six benchmarks.

The rest of the paper is organized as follows. We survey the literature around our problem in Section 2. In Section 3, we describe the VRP-SDO and provide its formulation. In Section 4 we describe the proposed solution methodology, and we present computational results in Section 5. Finally, we draw our conclusions in Section 6.

2. Literature Review

In this section we provide an overview of the scientific literature related to the VRP-SDO and to our chosen methodology. In Section 2.1, we position the VRP-SDO within the landscape of relevant problems. In Section 2.2 we survey the main solution methods pertaining to closely related variants of the VRP-SDO and position our methodological contribution.

2.1. Related problems

The VRP-SDO encompasses two main complicating dimensions: stochastic customer demands and the determination of outsourced customers. With the latter being an option prior to commencing the routes and revealing any customer demand. As the combination of both dimensions is relatively rare in the literature, we survey the relevant contributions of each in separation.

Uncertain demands have been widely explored in the routing literature (see [Parada et al. 2024](#), for a comprehensive survey), where it is typically assumed that customer demand is revealed upon the first arrival of a vehicle. As a result, if the available vehicle capacity is not sufficient to satisfy customer demand, a *classical* recourse action is predominantly applied, whereby the vehicle restocks its capacity at the depot and returns to the customer ([Laporte et al. 2002](#), [Jabali et al. 2014](#)). This recourse action is often complemented by allowing vehicles to perform preventive returns to the depot (e.g., [Louveau and Salazar-González \(2018\)](#), [Florio et al. \(2020\)](#)).

The problem settings using classical recourse, as well as preventive returns, implicitly assume that each customer is served exclusively by a single vehicle. Thus, any recourse action is performed by the vehicle that had visited the customer for the first time. This assumption facilitates modeling the resulting problem as a two-stage stochastic program, where the first stage establishes fixed routes for the vehicles, which are maintained irrespectively of the outcome of the second stage, thus having second stage optimization problems that are separable by vehicle (e.g., [Hoogendoorn and Spliet \(2023\)](#)). Such unique vehicle-to-customer assignment assumptions have been relaxed in the literature (e.g., [Goodson et al. \(2016\)](#)) by assuming that all vehicles may serve a customer in additional visits, if needed. This flexibility often comes in the form of dynamically establishing the customer visits for each vehicle. Considering the committed customers, we dynamically allocate vehicles to customers while allowing customers to be visited by multiple vehicles, in case a single visit is not sufficient. We consider classical recourse as well as preventive returns.

Irrespective of the adopted recourse strategy, the variability in customer demand leads to variability in route duration. This introduces challenges when vehicles are constrained by regular working hours. For example, [Erera et al. \(2010\)](#), [Goodson et al. \(2016\)](#), and [Dastpak et al. \(2023\)](#) assume hard duration limits for each vehicle trip. [Erera et al. \(2010\)](#) consider a variable number of vehicles to ensure serving all customers under the duration limit constraints, while the latter two works maintain a fixed set of vehicles and aim to maximize the total served demand within the duration limit. In VRP-SDO, similar to [Mendoza et al. \(2016\)](#), we consider a fixed set of available vehicles with a soft duration limit, where exceeding regular working hours is possible at an additional cost.

A common approach for controlling costs in routing problems with limited resources, such as fleet size, vehicle capacity, and trip duration, involves outsourcing the service of a subset of customers. This is often addressed as a deterministic problem, which is referred to as the VRP with private fleet and common carrier (VRP-PFCC). For instance, [Ceschia et al. \(2011\)](#), [Moon et al. \(2012\)](#), [Brito et al. \(2015\)](#), [Gahm et al. \(2017\)](#) investigate cases in which the LSP can rent vehicles in addition to its own fleet. In these cases, a daily cost or a distance-dependent cost for rented vehicles is optimized. Alternatively, [Chu \(2005\)](#), [Bolduc et al. \(2008\)](#), [Côté and Potvin \(2009\)](#), and [Poon et al. \(2022\)](#) study problems in which outsourced customers have a given outsourcing cost. In contrast, [Baller et al. \(2020\)](#) and [Zhu et al. \(2026\)](#) consider a cost per unit of outsourced demand. In particular, [Baller et al. \(2020\)](#) consider outsourcing a portion of a customer’s demand. In real-world applications, common carrier companies may offer volume-based discounts based on either individual customer demands ([Stenger et al. 2013](#)) or aggregated volume of outsourced demands ([Dabia et al. 2019](#)). Similar to the latter, we consider that outsourcing rates decrease as the total quantity of outsourced demands increases. Given that in the VRP-SDO outsourcing decisions are to be established prior to revealing the customer demand values, the considered outsourcing costs are based on the total expected demand values. Furthermore, we acknowledge that the VRP-SDO shares modeling features with the capacitated profitable routing problem ([Archetti et al. 2009](#)), the team orienteering problem ([Boussier et al. 2007](#), [Vidal et al. 2016](#)), and the prize-collecting problem ([Moradi and Boroujeni 2025](#)).

The previously mentioned VRP-PFCCs deal with deterministic problems. To the best of our knowledge, only [Zhu et al. \(2026\)](#) address a dynamic VRP with stochastic demands where outsourcing is performed when a vehicle cannot fully serve a customer. Therefore, outsourcing is not a primary assignment decision, but is rather a recourse penalty per unit. However, in the VRP-SDO, the outsourced customers are determined at the beginning of the day, which is aligned with common practice. To the best of our knowledge, no prior work jointly addresses vehicle routing with demand uncertainty where outsourcing decisions are made a priori. Indeed, the VRP-SDO combines complex modeling features from two strands of the literature. Vehicles are dynamically dispatched to customers, thus impeding the separation of subproblem on a per vehicle basis. Moreover, the outsourcing costs are a function of the total expected demand value of outsourced customers, which creates significant dependencies among these decisions.

2.2. Solution methods

The solution approaches to the VRP-PFCC vary depending on the type of outsourcing service considered by the LSP. In problems involving renting vehicles, a common approach is to establish routes for both the private and rental fleets ([Brito et al. 2015](#), [Sawadsitang et al. 2017](#), [Gahm et al. 2017](#)). Consequently, the outsourcing decisions become an integral part of the routing process. In problems where the LSP chooses a subset of customers to outsource and does not route them, a virtual vehicle route is typically used to model the common carrier ([Stenger et al. 2013](#)). Specifically, in these cases, solution methods often assign and route a subset of customers to m private vehicles, whereas the remaining subset is assigned to a virtual route representing the outsourcing decisions.

[Goeke et al. \(2019\)](#), [Dabia et al. \(2019\)](#), [Su et al. \(2023\)](#), and [Moradi and Boroujeni \(2025\)](#) have proposed branch-price-and-cut algorithms to address deterministic VRP-PFCC variants. Additionally, several authors have proposed heuristic methods for the VRP-PFCC. For example, [Chu \(2005\)](#) consider outsourcing with a given cost per customer, employing a savings-based constructive procedure and local improvement. [Bolduc et al. \(2008\)](#) introduce a heuristic for the same problem, integrating a local descent on various neighborhood structures with a perturbation mechanism involving the swapping of customer pairs, and an improvement procedure. [Côté and Potvin \(2009\)](#) propose a tabu search to address the same problem and achieved improvements over previous results. Additionally, [Potvin and Naud \(2011\)](#) present an enhanced tabu search heuristic with a neighborhood structure based on ejection chains. To address a multi-depot extension of the VRP-PFCC, [Stenger et al. \(2013\)](#) proposed an adaptive variable neighborhood search, incorporating an adaptive mechanism for routing and customer selection within the shaking phase. Subsequently, [Vidal et al. \(2016\)](#) introduce large neighborhoods for this problem, resulting in higher-quality solutions than those of [Stenger et al. \(2013\)](#). Considering a VRP-PFCC with multiple common carrier types, [Gahm et al. \(2017\)](#) explored variants of variable neighborhood search.

The previously mentioned methodologies address deterministic problems. In the VRP-SDO the customer demands are stochastic and committed customers must be fully served by the LSP fleet. Our solution methodology relies on an ILS algorithm, which is commonly used for routing problems, e.g., ([Maximo et al. 2024](#)). At each iteration, our methodology first determines the committed and outsourced customers, where evaluating the cost of the former is a vehicle routing problem with stochastic demands. We model the VRP-SD as an MDP considering classical recourse as well as preventive returns, while allowing multiple vehicles to serve the same customer. Two primary methodologies have been adopted to solve MDP-based SVRP. Approximate policy iteration methods (e.g., policy gradient), which aim at directly developing the desired policy by iteratively improving an initial policy ([Li et al. 2022](#)). Conversely, approximate value iteration methods, such as Q-learning, compute a value function for state-action values (e.g., [Li et al. \(2021\)](#), [Chen et al. \(2022\)](#)). We adopt a Q-learning algorithm, where we develop a value function estimating routing costs.

MDP formulations are typically addressed using dynamic programming algorithms, yet these algorithms are susceptible to the curse of dimensionality. This challenge is prevalent in stochastic routing problems, which are characterized by expansive combinatorial state and action spaces. Various mechanisms have been proposed to address these complexities, such as decomposing the problem into multiple single-vehicle subproblems ([Fan et al. 2006](#), [Goodson et al. 2013](#)), or decentralizing (i.e., computing value functions for individual vehicles rather than the joint fleet) while either neglecting ([Oda and Joe-Wong 2018](#), [Oroojlooy and Hajinezhad 2023](#)) or considering ([Li et al. 2019](#), [Chen et al. 2019](#), [Kullman et al. 2022](#)) inter-vehicle collaboration. For a comprehensive review, we direct readers to [Dastpak et al. \(2023\)](#). In this paper, to alleviate the issues associated with large state and action spaces, we adopt two methodological features from their work. Namely, we use an MDP with consecutive action selection and an observation function (MDP-CO). Consecutive action selection entails that only one vehicle makes a decision at each decision

epoch, whereas the entire state of the system is aggregated into the vehicle’s observation. Specifically, [Dastpak et al. \(2023\)](#) proposed a simple observation function based on grid-based aggregation of all customers (see, for example [Chen et al. 2019](#), [Kullman et al. 2022](#)) and a basis function representation (see, for example [Joe and Lau 2020](#)) of *target customers*, who are heuristically identified as promising candidates for the active vehicle.

In this paper, we improve the observation function introduced by [Dastpak et al. \(2023\)](#) in two major and complementary ways, both realized within a single graph attention network (GAT). The first improvement concerns how node information is represented. In our setting, the nodes of the graph are the customers, each described by its own raw features (e.g., location and expected demand). [Dastpak et al. \(2023\)](#) treat these nodes in isolation and thereby overlook the spatial dependencies and topological relationships among them. Such relationships are naturally captured by graph neural networks (GNNs), which represent a node by aggregating information from its neighbors ([Scarselli et al. 2009](#)) and have proven effective across a range of combinatorial optimization problems ([Cappart et al. 2021](#)). An early example of such a GNN is the “structure2vec”, which represents each node as a learned combination of its neighborhood features (e.g., coordinates and demand) and has been applied to the traveling salesman problem (TSP) ([Dai et al. 2016](#)) and the VRP ([Lin et al. 2022](#)). A more advanced way to perform such neighborhood aggregation is through attention mechanisms, which learn the relevance of each neighbor to the node being represented. This mechanism was first introduced by [Bahdanau et al. \(2015\)](#) to help long short-term memory (i.e., LSTM) and recurrent neural network (i.e., RNN)-based encoder-decoders to cope with the long-range sequences in natural language processing, as later scaled up in systems such as Google’s neural machine translation ([Wu et al. 2016](#)). Although originally designed for sequences, the underlying principle of attention (i.e., dynamically weighting the importance of input elements) transfers naturally to graph-structured data. The transformer encoder ([Vaswani et al. 2017](#)) is the architecture that applies this principle to a set of nodes, encoding each one as a weighted combination of all the others. In a vehicle routing context, transformer encoders have been employed to generate representations that capture each customer’s relative position within the graph ([Xu et al. 2022](#)). We build our GAT from such transformer-style attention layers. Accordingly, each node is encoded as a weighted combination of all other nodes, where the weights express the learned relevance (or alignment) between node pairs.

The proposed GAT addresses the second limitation of [Dastpak et al. \(2023\)](#), which concerns how the node representations are reduced to the fixed-size observation of the active vehicle. To represent a varying number of customers into a fixed-size vector, [Dastpak et al. \(2023\)](#) used a heuristic rule to select a subset of promising customers of fixed cardinality (i.e., target customers) in the observation. By retaining only a subset of customers, this rule produces an incomplete observation that may omit important information for routing decisions. We instead let the GAT decide, through another attention mechanism, how to represent the information of all customers and vehicles into a single fixed-size observation. Specifically, the GAT assigns every customer and vehicle a weight that reflects its relevance to the active vehicle, and then combines all of them into a single fixed-size vector. This way of summarizing a variable set of nodes into one fixed-size vector through attention follows [Kool et al. \(2019\)](#), whose encoder-decoder architecture consolidates the customer set into a single vector before constructing a TSP tour, and [Nazari et al. \(2018\)](#), who extend the idea to the VRP. The approaches of [Bono et al. \(2021\)](#) and [Zhang et al. \(2020\)](#) are the most closely related to ours, as they also address multi-vehicle VRPs and embed customers and vehicles jointly through attention mechanisms. However, in both works the attention layers directly parameterize a routing policy by scoring customers for the next visit, whereas our GAT builds a fixed-size observation on which a value function is learned to estimate routing costs.

In conclusion, to the best of our knowledge, no existing solution method in the literature can be trivially adapted to the VRP-SDO. We present a novel solution method that decomposes the problem into outsourcing and routing components. The latter component is modeled as an MDP and solved with a deep Q-network based algorithm. This yields a fast and reliable mechanism for estimating routing costs, thus allowing an iterative evaluation of a large number of decomposed solutions. The pillar of this mechanism is a novel GAT structure for representing the system’s state. Furthermore, we propose an online fine-tuning step inside the local search heuristic to further enhance the performance of the developed value function in real-time.

3. The Vehicle Routing Problem with Stochastic Demands and Outsourcing Options

In Section 3.1, we formally define the VRP-SDO. In Section 3.2 we present a high-level mathematical formulation for it based on a two-level decision structure, namely, outsourcing decisions at the first level, and routing at the second

(the VRP-SD). In Section 3.3 we provide an MDP formulation of the VRP-SD.

3.1. Problem Description

The LSP operates m vehicles, denoted by $\mathcal{V} = \{1, 2, \dots, m\}$, all initially located at the depot. We denote the depot and its location by 0 and l_0 , respectively. We assume that vehicles are identical and have capacity Q . Every morning, the LSP receives service requests from a set of customers $C = \{1, \dots, n\}$. The requests are to be served during the day. Thus, all service requests are known before routing and outsourcing decisions are made. Although pick-up and delivery interpretations are mathematically equivalent in our setting, we adopt the pick-up interpretation for consistency of exposition. We assume that the set of customer requests varies from one day to another. Each customer $c \in C$ is characterized by its location l_c and a stochastic demand d_c , whose expected value $\bar{d}_c$ is assumed to be known. In particular, the demand of each customer $c \in C$ is assumed to follow a known probability distribution denoted $\Gamma_c^D = \Gamma^D(\bar{d}_c)$, where $\Gamma^D(\bar{d}_c)$ is a function that returns a probability distribution for a given input value $\bar{d}_c$. We define a complete graph $G = (N, E)$ with the set of nodes $N = \{0\} \cup C$ and the set of arcs $E = \{(i, j) \mid i, j \in N, i \neq j\}$, which represents potential visit sequences between nodes. The travel time between two nodes i and j , denoted by τ_{ij} , is defined for each $(i, j) \in E$.

At the beginning of the operating day, the LSP determines the subset of outsourced customers $C^o \subset C$. We let $\bar{C}^o = C \setminus C^o$ denote the set of committed customers, which will be served by the LSP's fleet. Partially outsourcing a customer's demand is not allowed, i.e., a customer's demand is either fully outsourced or fully served by the LSP. Similar to [Dabia et al. \(2019\)](#), we assume an outsourcing cost $\Psi(C^o)$ to be a piecewise linear function of the expected total outsourced demand $\sum_{c \in C^o} \bar{d}_c$.

The LSP's vehicles begin and end their routes at the depot. As customary in the literature, we assume that a customer's actual demand is observed when a vehicle visits the customer ([Florio et al. 2023](#)). We allow the demand of each committed customer $c \in \bar{C}^o$ to be collected over multiple visits. Specifically, if a vehicle is unable to completely serve a customer during a visit, it serves it to the fullest possible extent, and any remaining demand is collected by the same or other vehicles at a later time. Vehicles are also allowed to perform a *preventive unloading* action, meaning that they can visit the depot to unload even if their capacity has not been fully used yet. Vehicle operation time is charged at one unit of cost per unit of time up to a regular work-shift duration L , and at an *overtime* cost of ϕ per time unit beyond it. We assume that a vehicle may remain at the depot and thus terminates its operation, provided that at least one vehicle remains operative to attend remaining customers. The objective is to minimize the total expected cost, including outsourcing, travel, and overtime cost.

3.2. A high-level formulation of the VRP-SDO

We now present a two-level mathematical formulation for the VRP-SDO. The first-level problem, referred to as the outsourcing problem, consists of partitioning the set of customers into two subsets: committed customers and outsourced customers. To this end, we introduce an n -dimensional binary variable $x = (x_c)_{c \in C}$, where $x_c = 1$ ($x_c = 0$) indicates that customer $c \in C$ is committed (outsourced). The subsets of committed and outsourced customers can be defined based on the decision variable x as $C^o = \{c \in C \mid x_c = 0\}$ and $\bar{C}^o = C \setminus C^o$.

The second-level problem, which we refer to as the Vehicle Routing Problem with Stochastic Demands (i.e., VRP-SD), consists of routing the LSP's fleet to serve the committed customers. Therefore, the second-level problem is stochastic and dynamic, and its solution is expressed in terms of a policy rather than a fixed routing plan. We let π_r denote a routing policy used to serve the committed customers $\bar{C}^o$. Let $w = \{w^c\}_{c \in \bar{C}^o}$ be the vector of realized customer demands, where each w^c is a demand realization distributed according to Γ_c^D . For a given routing policy π_r , committed customer set $\bar{C}^o$, vehicle $v \in \mathcal{V}$, and demand realization vector w , let $T_{\pi_r}(\bar{C}^o, v, w)$ denote the duration of the route performed by vehicle v . Accordingly, we define $R(\bar{C}^o)$ as the minimum expected routing cost of serving the

committed customers, including both travel and overtime costs. We formalize the VRP-SDO as follows:

$$\min_x \Psi(C^o) + R(\bar{C}^o), \quad (1)$$

s.t.:

$$C^o = \{c \in C | x_c = 0\}, \quad (2)$$

$$\bar{C}^o = \{c \in C | x_c = 1\}, \quad (3)$$

$$R(\bar{C}^o) = \min_{\pi_r} \mathbb{E}_w \left[\sum_{v \in \mathcal{V}} \left[\min(T_{\pi_r}(\bar{C}^o, v, w), L) + \phi \max(T_{\pi_r}(\bar{C}^o, v, w) - L, 0) \right] \right], \quad (4)$$

$$x_c \in \{0, 1\}, \forall c \in C. \quad (5)$$

The challenge of formulation (1)-(5) is the inner optimization problem (4), which hides substantial complexity. Indeed, for any fixed first-level decision x , evaluating the objective function requires computing $R(\bar{C}^o)$, that is, solving the second-level problem. This problem is itself a stochastic, dynamic, and combinatorial optimization problem: routing decisions must adapt in real time to revealed demands, multiple vehicles must be coordinated, and operational features such as vehicle capacity and overtime penalties must be taken into account. As a result, accurately evaluating a single candidate outsourcing decision is computationally very demanding. The ability to efficiently solve problem (4) is a key component of the overall solution method. We model VRP-SD as an MDP, as detailed in Section 3.3. This provides a suitable framework for capturing the sequential and stochastic nature of the problem and for later developing a reinforcement learning approach to approximate routing costs efficiently for the evaluation of the first-level outsourcing decisions.

3.3. MDP formulation for the VRP-SD

For a given set of committed customers $\bar{C}^o$, the VRP-SD dynamically determines the sequence of customer visits and depot returns for each vehicle $v \in \mathcal{V}$ so as to minimize the expected routing and overtime costs. We formulate this problem as an MDP. We define a *decision epoch* as a point in time at which a vehicle is ready to depart from its current location, that is, either the depot or a customer location. At each decision epoch k , the system state s_k is defined as

$$s_k = (F^C, F^V, t_k), \quad (6)$$

where $F^C = [(l_c, h_c, \bar{d}_c, \hat{d}_c)]_{c \in \bar{C}^o}$ and $F^V = [(l_v, a_v, q_v, g_v)]_{v \in \mathcal{V}}$ represent the state of customers and vehicles, respectively. We note that F^C and F^V are matrices with dimensions of $|\bar{C}^o| \times 4$ and $|\mathcal{V}| \times 4$, respectively. The last component t_k denotes the time of decision epoch k . The state of each customer $c \in \bar{C}^o$ is described by its location l_c , availability h_c , expected demand $\bar{d}_c$, and unserved demand $\hat{d}_c$. We note that $h_c = 0$ if customer c is either currently assigned to a vehicle or it is fully served. The state of each vehicle $v \in \mathcal{V}$ is described by its destination l_v , arrival time at destination a_v , current available capacity q_v , and a binary variable g_v indicating whether the vehicle is in operation. We note that $g_v = 0$ indicates that the operation of vehicle v is terminated and it cannot be used in the current or in future decision epochs. For vehicles that have already arrived at their destination, l_v indicates their current location.

In this formulation we define the action of each vehicle as its next location to visit, namely, a customer or the depot, (see, for example, [Maxwell et al. 2010](#), [Chen et al. 2019](#), [Dastpak et al. 2023](#)). In particular, we define the joint action at decision epoch k by the m -dimensional vector of $y_k = (y_k^1, \dots, y_k^m)$, where y_k^v denotes the action of vehicle v . Let $\bar{\mathcal{V}}_k = \{v \in \mathcal{V} \mid a_v = t_k \wedge g_v = 1\}$ be the set of vehicles that have arrived at their destinations and are available to take an action at decision epoch k . We call $\bar{\mathcal{V}}_k$ the set of active vehicles. Since service is assumed to be instantaneous, the arrival time of a vehicle at a customer coincides with the time at which the vehicle is ready to depart. We now describe the feasible actions for vehicles $v \in \bar{\mathcal{V}}_k$, that is, the possible values of y_k^v . Let $\bar{C}_k^o = \{c \in \bar{C}^o \mid h_c = 1\}$ be the set of available customers at the decision epoch k . If vehicle v is located at a customer, it has three possible actions. It may visit another customer $c \in \bar{C}_k^o$ *directly*, denoted by y_c^D , visit another customer $c \in \bar{C}_k^o$ *indirectly*, denoted by y_c^I , that is, by first performing an unloading action at the depot, or select the depot l_0 as its next destination. We note that returning to the depot without obligation, that is, not because of a route failure or because no customer is currently available, is interpreted as terminating the trip of the vehicle. On the contrary, if the vehicle is at the depot because it was obliged to, it may either choose a customer c to visit directly, that is, select action y_c^D , or terminate its operation

by choosing action l_0 . When the vehicle is at the depot, indirectly visiting a customer c , that is, choosing y_c^I , is not applicable. The joint action space $A(s_k)$ is then defined as follows:

$$A(s_k) = \left\{ y_k \in \{l_0\} \cup \{y_c^D, y_c^I | c \in \bar{C}_k^o\}^m : \right. \quad (7)$$

$$y_k^v = l_v \quad \forall v \in \mathcal{V} \setminus \bar{\mathcal{V}}_k, \quad (8)$$

$$y_k^v = l_0 \quad \forall \{v \in \mathcal{V} | q_v = 0 \vee \bar{C}_k^o = \emptyset\}, \quad (9)$$

$$y_k^v \neq y_c^I \quad \forall \{v \in \mathcal{V} | l_v = l_0\}, \forall c \in \bar{C}_k^o, \quad (10)$$

$$y_k^v \neq l_0 \quad \forall \{v \in \mathcal{V} | \sum_{v \in \mathcal{V}} g_v = 1 \wedge \bar{C}_k^o \neq \emptyset\}, \quad (11)$$

$$y_k^v \neq y_k^{v'} \quad \forall \{v, v' \in \mathcal{V} | v \neq v' \wedge y_k^v \neq l_0\}. \quad (12)$$

In the action space $A(s_k)$, Condition (8) obliges unavailable vehicles at time t_k to continue to their current assigned destination. Condition (9) enforces a return to the depot when a vehicle has no remaining capacity or when no customer is available to visit. Condition (10) ensures that indirect service of a customer is not allowed when the vehicle is already at the depot. Condition (11) implies that the vehicle cannot terminate its trip (travel to the depot without obligation), if customers are still available to be served and no other vehicle is in operation. Finally, Condition (12) ensures that no two vehicles choose the same destination, unless that destination is the depot.

We now describe the evolution of the system from the initial decision epoch onward. The state at decision epoch zero, denoted by s_0 , represents the initial configuration of the system before any routing actions are taken. It is constructed from the committed customer set $\bar{C}^o$, which is determined by the first-level outsourcing decision. At this stage, customer demands have not yet been realized, all vehicles are located at the depot, and no service has started. The initial state s_0 is defined as follows:

$$s_0 = [(l_c, 1, \bar{d}_c, ?)]_{c \in \bar{C}^o}, [(l_0, t_0, Q, 1)]_{v \in \mathcal{V}, t_0}. \quad (13)$$

At this decision epoch, the customer's availability h_c and unserved demand $\hat{d}_c$ are initially set to 1 (i.e., available) and "?" (i.e., unknown), respectively. If customer c is already fully served or a vehicle is assigned to serve it, then h_c takes the value of 0. The unserved demand $\hat{d}_c$ is updated to the actual demand upon the first visit. The state of the fleet is initialized by setting l_v, a_v, q_v , and g_v to l_0, t_0, Q , and 1, respectively.

In state s_k , taking an action $y_k \in A(s_k)$ transitions the system to the next state s_{k+1} according to the state transition function $S^M(s_k, y_k, w_{k+1})$. Recalling that w is a given realization of the stochastic demands, we denote by $w_k \subset w$ the set of demand realizations corresponding to customers that are visited for the first time at decision epoch k .

The transition function is divided into two conceptual steps. The first step updates the state s_k to the post-decision state s_k^y , which reflects the status of the system immediately after action y_k is taken but before the realization of the exogenous information w_{k+1} . The state is updated as follows:

$$h_c = 0, \forall \{c \in \bar{C}^o | \exists v \in \bar{\mathcal{V}}_k, y_k^v = y_c^D \vee y_k^v = y_c^I\}, \quad (14)$$

$$g_v = 0, \forall \{v \in \bar{\mathcal{V}}_k | y_k^v = l_0 \wedge q_v > 0 \wedge \bar{C}_k^o \neq \emptyset\}, \quad (15)$$

$$l_v = \begin{cases} l_c & y_k^v = y_c^D \vee y_c^I, \\ l_0 & \text{otherwise} \end{cases}, \forall v \in \bar{\mathcal{V}}_k, \quad (16)$$

$$a_v = t_k + \begin{cases} \tau_{v,0} & y_k^v = l_0 \\ \tau_{v,c} & y_k^v = y_c^D, \forall v \in \bar{\mathcal{V}}_k, \\ \tau_{v,0} + \tau_{0,c} & y_k^v = y_c^I \end{cases} \quad (17)$$

$$t_{k+1} = \min_{v \in \mathcal{V}} a_v. \quad (18)$$

Equation (14) updates the selected customers as unavailable. Equation (15) updates the in-operation status of vehicles that terminate their trip. Equations (16) and (17) update the location and arrival time of each vehicle according to its action. Lastly, Equation (18) advances the time to the next decision epoch, at which the exogenous information w_{k+1} is revealed. All other state components remain unchanged.

The second step of the transition function occurs at the beginning of decision epoch $k + 1$ and transforms the post-decision state s_k^v into s_{k+1} . Let $\tilde{C}_{k+1} = \{c \in \bar{C}^o | \exists v \in \bar{V}_{k+1} \wedge l_c = l_v\}$ be the set of customers that are served at decision epoch $k + 1$. If the actual demand of a customer in $\tilde{C}_{k+1}$ has not yet been realized (i.e., $\hat{d}_c = ?$), then $\hat{d}_c$ is set to the observed demand w_{k+1}^c . Thus,

$$\hat{d}_c = w_{k+1}^c, \forall \{c \in \tilde{C}_{k+1} | \hat{d}_c = ?\}. \quad (19)$$

Let η_v^{k+1} denote the demand volume served by vehicle $v \in \bar{V}_{k+1}$ at decision epoch $k + 1$, defined as:

$$\eta_v^{k+1} = \min\{\hat{d}_{c_v}, q_v\}, \forall \{v \in \bar{V}_{k+1} | l_v \neq l_0\}, \quad (20)$$

where c_v denotes the customer $c \in \tilde{C}_{k+1}$ served by vehicle v (i.e., $l_c = l_v$). The unserved demands of customers in $\tilde{C}_{k+1}$ and the available capacity of vehicles $v \in \bar{V}_{k+1}$ are then updated as follows:

$$\hat{d}_{c_v} = \hat{d}_{c_v} - \eta_v^{k+1}, \forall \{v \in \bar{V}_{k+1} | l_v \neq l_0\}. \quad (21)$$

$$q_v = \begin{cases} q_v - \eta_v^{k+1} & y_k^v = y_c^D, \\ Q - \eta_v^{k+1} & y_k^v = y_c^I, \\ Q & \text{otherwise} \end{cases}, \forall v \in \bar{V}_{k+1}. \quad (22)$$

Finally, customers in $\tilde{C}_{k+1}$ whose demands are not fully served become available again for future service. Thus,

$$h_c = 1, \forall \{c \in \tilde{C}_{k+1} | \hat{d}_c > 0\}. \quad (23)$$

We define the cost function as $C(s_k, y_k) = \sum_{v \in \bar{V}_k} C_v(s_k, y_k^v)$, representing the routing cost incurred by taking action y_k in state s_k . Here, $C_v(s_k, y_k^v)$ denotes the individual routing cost associated with vehicle v taking action y_k^v in state s_k , and is defined as follows:

$$C_v(s_k, y_k^v) = (\min\{a_v, L\} - \min\{t_k, L\}) + \phi(\max\{a_v, L\} - \max\{t_k, L\})$$

In this MDP formulation, the routing policy π_r determines the action $y_k \in A(s_k)$ when state s_k is observed (i.e., $y_k = \pi_r(s_k)$). Accordingly, we define the value of being at state s_k as follows:

$$V^{\pi_r}(s_k) = \mathbb{E}_w[C(s_k, y_k) + \gamma V^{\pi_r}(s_{k+1}) | s_k], \quad \forall s_k \in S_{\bar{C}^o}, \quad (24)$$

where γ is a discount factor that reflects the relative importance of near-future costs. In addition, $S_{\bar{C}^o}$ denotes the state space associated with the committed customer set $\bar{C}^o$. We seek an optimal routing policy π_r^* that minimizes the value function $V^{\pi_r}(s_k)$:

$$\pi_r^* = \arg \min_{\pi_r} V^{\pi_r}(s_k), \forall s_k \in S_{\bar{C}^o}. \quad (25)$$

The value function corresponding to the optimal policy π_r^* is denoted $V^*(s_k)$ for all states s_k . For a given committed customer set $\bar{C}^o$, the value function in Equation (24) and its corresponding decision policy in Equation (25) address the routing problem in Equation (4). In particular, $V^*(s_0)$ is used as an approximation of the expected routing cost $R(\bar{C}^o)$. The proposed MDP formulation remains computationally intractable because of the size of the state and action spaces. In particular, the associated state-action space is too large to allow an explicit evaluation of the value function $V^{\pi_r}(s_k)$ for every state $s_k \in S_{\bar{C}^o}$. This challenge is commonly referred to as the curse of dimensionality (Powell 2022).

A further difficulty arises in the overall two-level problem, since the committed customer set $\bar{C}^o$ depends on the first-level outsourcing decision. As a result, a routing policy that is optimal for one committed customer set may no longer be optimal for another, or may not even be defined. Since a given initial customer set C may induce an exponential number of committed customer subsets, solving a distinct second-level problem for each such subset is not practical. This is also impractical when using an iterative algorithm that alternates between the two-levels of

the VRP-SDO. Therefore, in Section 4.1.1, we generalize the MDP formulation so as to handle arbitrary committed customer sets. Then, in Section 4.1.2, we address the curse of dimensionality by developing a deep reinforcement learning algorithm.

We then focus on the solution of the second-level problem, the VRP-SD, in Section 4.1, and present our ILS algorithm in Section 4.2.

4. Solution Method

Our solution method is designed to be executed on each day, before the beginning of the operations. The ILS, presented in Section 4.2, explores and selects candidate outsourcing decisions. However, this algorithm relies on the existence of an *oracle* able to provide an estimation of the expected cost of candidate outsourcing decisions, i.e., the value of the second-level problem. The method to obtain such an oracle is described in Section 4.1. First, in Section 4.1.1 we generalize the formulation provided in Section 3.3 so that it operates over a distribution of possible customer subsets, rather than on a specific subset of committed customers $\bar{C}^o$. Thus, eliminating the need to retrain our offline algorithm for each instance. The overarching assumption is that, while the set of customers varies at each day, LSPs typically possess historical information on previous days of operations, which allows them to deduce empirical distributions of customer locations and demands. Although formally simple, this generalization is conceptually important because it decouples the second-level model from the first-level decision, opening the door to developing a learning-based algorithm able to return a reusable value function that is entirely computed offline. Second, following Dastpak et al. (2023), Section 4.1.2 simplifies the obtained formulation by applying the concepts of *consecutive action space* and *observation function*. As it will be clear later on, the resulting simplified MDP formulation allows only one vehicle to be active at a given decision epoch, and adopts an observation function mapping the original large state space into a simplified state tailored to the vehicle that is active at that decision epoch.

Differently from Dastpak et al. (2023) where the observation function is comprised of hand selected features, in Section 4.1.3 we propose presenting the observation function via a GAT. We then embed the GAT architecture into a deep reinforcement learning approach based on a DQN (Section 4.1.4). The full network is trained offline, and enables fast and scalable estimation of the routing cost for any subset of committed customer, thus providing the desired oracle to be used within the ILS algorithm. The details of the ILS are presented in Section 4.2. In particular, Section 4.2.1 introduces a core implementation of the algorithm while Section 4.2.2 introduces an online fine-tuning step of the DQN to enhance the accuracy of the offline-trained value function.

4.1. Solving the vehicle routing problem with stochastic demands

4.1.1. A generalized MDP formulation

The MDP formulation presented in Section 3.3 is formalized on a specific customer set $\bar{C}^o$ with corresponding state space $S_{\bar{C}^o}$. It is important to note that the corresponding value function $V^*(s_k)$ becomes sub-optimal or even undefined for any change in the set of the committed customers $\bar{C}^o$, which directly impacts $S_{\bar{C}^o}$. In our solution method this is an important limitation due to two main factors. First, given a set of customers C for a given day, we aim at using an ILS to search into possible outsourcing decisions, potentially including any possible subset of C . Solving a separate instance of the problem described in 3.3 for every possible subset of $\bar{C}^o \subset C$ is not practical, especially in presence of a limited computational budget, as in our application. This points towards a solution strategy where part of the computations is done offline, before the set of customers C is known, which brings us to the second challenge. Given that the customer set C is unknown when performing the offline computations, we need to consider C as a random variable with a related probability distribution.

The proposed modeling strategy is to generalize the MDP in Section 3.3 by allowing the initial set of committed customers to be a random variable. We formalize this by assuming that the set of daily customers is a random variable distributed as $C \sim \Gamma^C$. The ILS will then explore the subset of committed customers according to an endogenous conditional probability distribution $\bar{C}^o | C \sim \Gamma_{ILS}^{\bar{C}^o | C}$. We assume then that the set of committed customers is distributed according to the marginal distribution $\bar{C}^o \sim \Gamma^{\bar{C}^o} = \mathbb{E}_{C \sim \Gamma^C} [\Gamma_{ILS}^{\bar{C}^o | C}(\cdot | C)]$. In consequence, we rewrite the value function and the optimal routing policy in Equations (24) and (25) as:

$$V^{\pi_r}(s_k) = \mathbb{E}_w[C(s_k, y_k) + \gamma V^{\pi_r}(s_{k+1}) | s_k], \quad \forall s_k \in \bar{S}, \quad (26)$$

$$\pi_r^* = \arg \min_{\pi_r} V^{\pi_r}(s_k), \forall s_k \in \bar{S}, \quad (27)$$

where $\bar{S}$ is the union of state-spaces corresponding to all customer subsets $\bar{C}^o$ considered under the distribution $\Gamma^{\bar{C}^o}$, i.e., $\bar{S} = \bigcup_{\bar{C}^o \sim \Gamma^{\bar{C}^o}} S_{\bar{C}^o}$.

Formulation (26) and (27) is independent of the specific day of operation and the corresponding initial set of customers C . This fact comes with the additional benefit that once optimized, the optimal value function (26) and policy (27) can be applied to any given day, for any given subset of customers without the need for additional computation. On the other hand, challenges may come from the estimation of the conditional probability distribution $\Gamma^{\bar{C}^o}$, given that it is hard to estimate a priori the behavior of the ILS algorithm. In Section 5.1.2 we explain the choices made in our implementation.

4.1.2. The consecutive action rule and the observation function (Dastpak et al. 2023)

Solving problem (26) and (27) is extremely challenging, mainly due to the so called curse of dimensionality (Powell 2022). In particular we face 1) an extremely large action space, due to the potential presence of multiple active vehicles at a given decision epoch, resulting in mutually interdependent actions, see Equations (7) – (12) describing the mutual decision space, and 2) an extremely large state space, as defined in (6).

For a similar problem, Dastpak et al. (2023) proposed to approximate the original problem formulation by a simpler one obtained by following two main strategies: 1) to restrict the set of feasible actions by imposing a *consecutive action* rule, i.e., by imposing that only one vehicle at time is allowed to be active, and 2) to introduce an aggregation of the original system state called the *observation function*, whose purpose is to filter the information in the original state that is more relevant when establishing the next action to assign to the currently active vehicle. In this section, we briefly describe the application of these ideas to our problem.

The consecutive action rule requires that only one vehicle can be active. This is aligned with the specific characteristic of our problem where it is unlikely that several vehicles simultaneously arrive at their respective destinations and thus are simultaneously active. However, in cases where this happens, such as at decision epoch zero, when a decision must be taken for all vehicles, a random order is imposed among them by assigning each vehicle to a different decision epoch. This results in a singleton active vehicle set $\bar{\mathcal{V}}_k$, with $|\bar{\mathcal{V}}_k| = 1$ and $\bar{v} \in \bar{\mathcal{V}}_k$ represents the *active vehicle*. Given the consecutive action rule, the active vehicle $\bar{v}$ for a given state s_k is uniquely defined by the state itself. With this formalism, we set $y_k = (y_k^{\bar{v}})$ and we define the action space $\bar{A}(s_k)$ as follows:

$$\bar{A}(s_k) = \left\{ y_k^{\bar{v}} \in \{l_0\} \cup \{y_c^D, y_c^I | c \in \bar{C}_k^o\} : \right. \quad (28)$$

$$y_k^{\bar{v}} = l_0 \quad q_{\bar{v}} = 0 \vee \bar{C}_k^o = \emptyset, \quad (29)$$

$$y_k^{\bar{v}} \neq y_c^I \quad l_{\bar{v}} = l_0, \forall c \in \bar{C}_k^o, \quad (30)$$

$$y_k^{\bar{v}} \neq l_0 \quad \left. \sum_{v \in \bar{\mathcal{V}}} g_v = 1 \wedge \bar{C}_k^o \neq \emptyset, \right\} \quad (31)$$

where Condition (29) forces the active vehicle to visit the depot if either it has no capacity or there is no available customer to serve. Condition (30) ensures that indirect visits of customers $c \in \bar{C}_k^o$ are not feasible if the active vehicle is located at the depot. The last condition implies that the vehicle $\bar{v}$ cannot terminate its trip if it is the only vehicle in-operation and there are still customers to be served.

The second strategy consists in replacing the full system state s_k by a fixed-size aggregated representation, called observation, tailored to the active vehicle $\bar{v}$. Specifically, the observation $o_{k,\bar{v}}$ is obtained via an observation function $O(s_k, \bar{v})$; i.e., $o_{k,\bar{v}} = O(s_k, \bar{v})$. Decisions are then made based on this reduced representation, which yields an approximate MDP formulation, called MDP-CO, on the observation space.

Equations (14)–(23) remain applicable to MDP-CO, if we account for the new definition of $\bar{\mathcal{V}}$. We can then rewrite equations (26) and (27) by adopting the observation as the representation of the system state:

$$V^{\pi_r}(o_{k,\bar{v}}) = \mathbb{E}_w[C(s_k, y_k) + \gamma V^{\pi_r}(o_{k+1,\bar{v}'}) | o_{k,\bar{v}}], \quad (32)$$

$$\pi_r^* = \arg \min_{\pi_r} V^{\pi_r}(o_{k,\bar{v}}), \forall o_{k,\bar{v}} \in \Omega, \quad (33)$$

where $\bar{v}'$ is the active vehicle in decision epoch $k + 1$, and $y_k = \pi_r(o_{k,\bar{v}})$. In addition, Ω represents the set of all possible observations $o_{k,\bar{v}}$. It is worth noting that in equations (32) and (33), the value function $V^{\pi_r}(\cdot)$ is defined on the observation $o_{k,\bar{v}}$, whereas in equations (26) and (27), the same notation is used for the value function defined on the system state s_k . We retain this notation for simplicity, since the intended meaning is clear from the context. With this convention, the optimal state-based value $V^*(s_k)$ is approximated by the observation-based value $V^*(o_{k,\bar{v}})$. The approximated expected routing costs $R(\bar{C}^o)$ for any tentative set of committed customers $\bar{C}^o$ is simply retrieved in the value $V^*(s_0)$.

4.1.3. Learning the observation function: the Graph Attention Network

The careful design of the observation function is crucial when solving problem (32) and (33). Dastpak et al. (2023) proposed an observation function based on hand-crafted features of the full state space, to return a fixed-size vector representation. In particular, they replaced the customer component of the system state F^C , by two simpler vectors. The first vector provides a general overview of all customers' state by aggregating information using a grid-like discretization technique. The second vector is a subset of F^C built by filtering only a fixed number of *target customers* chosen according to a heuristic metric. While simple to implement, this observation function has several limitations. First, since target customers are heuristically selected, good candidate customers may be excluded. Second, there is a trade-off in the grid-like discretization of customer information: a finer grid yields higher precision, but also increases the size of the state representation and introduces an additional design parameter, namely the grid resolution. Third, the observation function $o_{k,\bar{v}}$ is defined relative to the index of the active vehicle $\bar{v}$. Since the vehicles are homogeneous, this representation does not explicitly exploit the permutation symmetry of vehicle identities. As a result, states that are operationally equivalent up to a relabeling of vehicles may still need to be treated separately by the value function approximation. To address these limitations, in this section we propose to replace the hand-crafted observation function in Dastpak et al. (2023) with a learned representation based on a GAT. A GAT is a neural architecture designed to process graph-structured data by learning how to aggregate information across related entities. This is particularly appropriate in our setting, where decisions depend on both customer and vehicle attributes and on network information such as travel times and spatial proximity. The attention mechanism allows the model to identify the entities that are most relevant to the active vehicle and to combine their information into a fixed-size observation vector. In this way, the representation is learned directly from data, rather than imposed through manual target-customer selection, grid-based discretization, or an index-dependent treatment of homogeneous vehicles.

The proposed GAT is based on Transformer Encoders introduced in Vaswani et al. (2017), which use an attention mechanism to generate contextualized representations of their inputs. In essence, the attention mechanism is a neural operator that takes as input a *query* vector and a *context* matrix, and computes *alignment scores* between the query vector and each row vector of the context matrix. The scores are then used to produce a new *context-aware* representation of the query vector, often referred to as the *attention output*. Higher alignment scores correspond to context rows that receive greater importance in the construction of the query representation. The resulting attention output is a compact, continuous vector representation of the query, capturing its relationship to the information contained in the context matrix.

In our setting, for example, given a customer and its state representation (the query vector), we may aim to produce a compact alternative representation of that customer relative to the states of all vehicles (the context matrix). Generally, we may wish to compute representations for all customers (a query matrix) relative to all vehicle states. More formally, we consider a query matrix $F \in \mathbb{R}^{|F| \times e_F}$, where each row $F_i \in \mathbb{R}^{e_F}$ represents a query vector. We seek to compute alternative representations for these query vectors. To this scope, we consider a context matrix $\bar{F} \in \mathbb{R}^{|\bar{F}| \times e_F}$, whose rows $\bar{F}_i \in \mathbb{R}^{e_F}$ are context vectors. Given a query matrix F and a context matrix $\bar{F}$, the attention mechanism computes a new output matrix $H = \text{Att}(F, \bar{F}) \in \mathbb{R}^{|F| \times e}$, where e is a parameter defining the dimension of the attention output, and each row H_i is the contextualized representation of the corresponding query vector F_i . When $F = \bar{F}$, the mechanism becomes a self-attention operation, where each entity is contextualized relative to all others in the same set. In our setting, for example, we may be interested in obtaining a representation of the customer set relative to the customer set itself.

The attention mechanism function $\text{Att}(F, \bar{F})$ works as follows: it first projects the query matrix F into a new matrix $U^q \in \mathbb{R}^{|F| \times e}$, with $U^q = FW^q$, and $W^q \in \mathbb{R}^{e_F \times e}$ a learnable weight matrix. Similarly, the context matrix $\bar{F}$ is projected into two separate matrices, $U^k, U^v \in \mathbb{R}^{|\bar{F}| \times e}$, with $U^k = \bar{F}W^k$ (keys), $U^v = \bar{F}W^v$ (values), and $W^k, W^v \in \mathbb{R}^{e_F \times e}$

learnable weight matrices. The alignment between each query vector in U^q and the keys in U^k is computed using scaled dot-product similarity. Specifically, the attention score matrix $M \in \mathbb{R}^{|F| \times |\bar{F}|}$ is calculated as:

$$M = \text{softmax}\left(\frac{U^q(U^k)^\top}{\sqrt{e}}\right).$$

The final attention output matrix $H \in \mathbb{R}^{|F| \times e}$ is then computed as $H = MU^v$, where each row H_i is a weighted combination of the value vectors in U^v , contextualized by the alignment scores of F_i with the components in $\bar{F}$. This completes the computation of $H = \text{Att}(F, \bar{F})$. Figure 1 illustrates the schematic structure of the attention mechanism.

Matrices $W^i, \forall i \in \{q, k, v\}$ are the only trainable parameters in the attention mechanism. Their dimensions depend solely on the embedding sizes e, e_F , and $e_{\bar{F}}$, which are fixed. Consequently, the number of trainable parameters is independent of $|F|$ and $|\bar{F}|$. This implies that, once trained, the attention mechanism can process inputs with different sizes (e.g., different number of customers or vehicles) while producing a fixed-size representation of the system state.

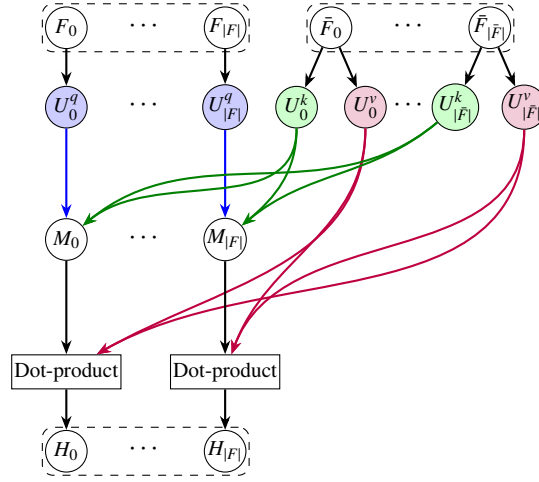


Figure 1: Attention mechanism $\text{Att}(F, \bar{F})$

In the following, we present the implementation of the observation function $O(s_k, \bar{v})$ via the attention mechanism. The general overview is illustrated in Figure 2. The proposed implementation is comprised of three main attention mechanism blocks, namely Node Embedder, Vehicle Embedder, and Graph Embedder. Each of these blocks are illustrated in Figure 3. The Node Embedder, illustrated in Figure 3(a), takes as input the states of all customers and

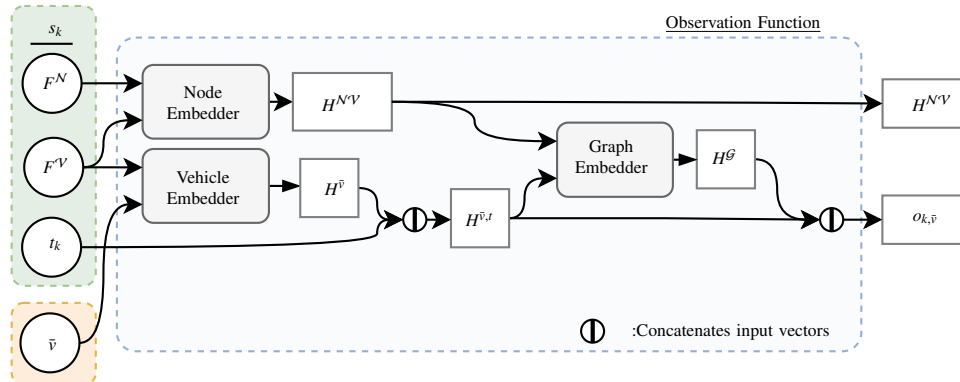


Figure 2: Schematic structure of proposed Observation Function

vehicles. For implementation purposes, we assume that the number of customers in a given operational day does not

exceed a fixed upper bound, denoted by $n_{\max}$; that is, $n \leq n_{\max}$ for any realization of C . This value is specified in the experimental setup described in Section 5.1.1. Accordingly, we define a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N} = \{0, 1, \dots, n_{\max}\}$ is the set of $n_{\max} + 1$ nodes, which includes the depot, customers $\bar{C}^o$, and dummy nodes if $|\bar{C}^o| < n_{\max}$. Furthermore $\mathcal{E}$ is the set of edges between every two nodes in $\mathcal{N}$, representing potential consecutive visits of a given vehicle. We further define $F^N \in \mathbb{R}^{(n_{\max}+1) \times 4}$ as the fixed-size matrix that includes the state representation of the depot and all potential customer nodes at a given decision epoch. In this matrix, the first row corresponds to the state of the depot, rows 1 to $|\bar{C}^o|$ correspond to F^C , and rows $|\bar{C}^o| + 1$ to $n_{\max}$ are padded with zero vectors (i.e., $(0, 0, 0, 0)$). Additionally, a masking strategy is applied to F^N such that the feature vectors of unavailable customers (i.e., $c \in \bar{C}^o \setminus \bar{C}_k^o$) are set to zero. This masking strategy ensures that both unavailable and non-existent nodes are represented by zero vectors, which effectively removes their influence in the attention mechanism. The node embedder also takes as input the state of all vehicles, represented by $F^V \in \mathbb{R}^{|\mathcal{V}| \times 4}$.

Structurally, the Node Embedder features two parallel attention blocks that are later concatenated into a single attention output. The first self-attention block, $\text{Att}(F^N, F^N)$, generates a matrix $H^N \in \mathbb{R}^{(n_{\max}+1) \times e}$, where each row H_i^N is a representation of node $i \in \mathcal{N}$ contextualized with respect to the other customers. The second attention block, $\text{Att}(F^N, F^V)$, generates a matrix $H^V \in \mathbb{R}^{(n_{\max}+1) \times e}$, where each row H_i^V is a representation of node i contextualized with respect to the fleet of vehicles. The final attention output H^{NV} has dimension $(n_{\max} + 1) \times 2e$ and is obtained by concatenating H^N and H^V . As a result, each node in $\mathcal{N}$ is now represented by a vector of size $1 \times 2e$, providing an enriched representation that captures its relationships with both other nodes in the graph and all vehicles in the system.

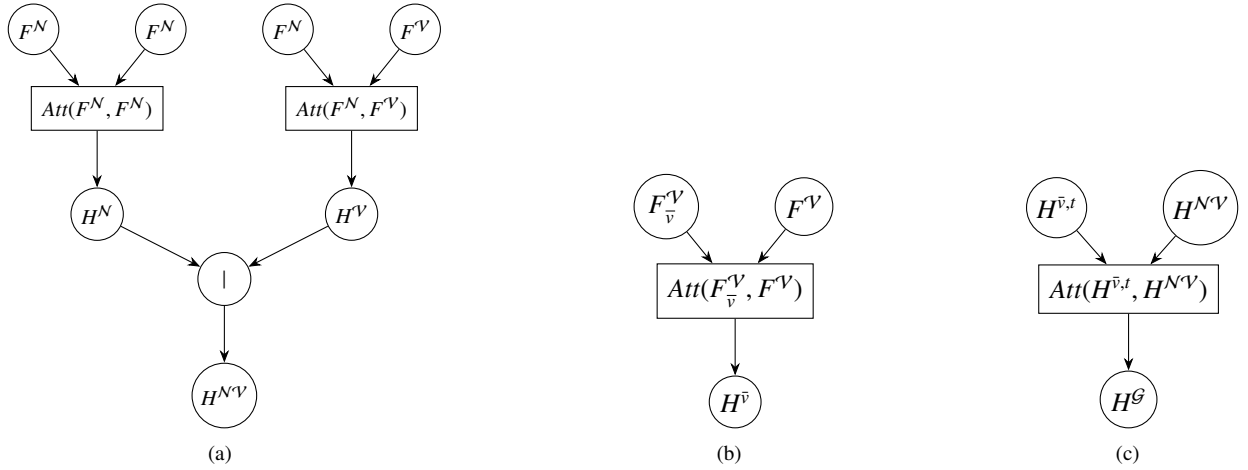


Figure 3: (a) Node Embedder, (b) Vehicle Embedder, (c) Graph Embedder

The Vehicle Embedder, illustrated in Figure 3(b), takes the state of the active vehicle F_v^V as the query vector, and state of all vehicles F^V as the context matrix. The resulting attention mechanism $\text{Att}(F_v^V, F^V)$ outputs a vector $H^v \in \mathbb{R}^{1 \times e}$ representing the active vehicle contextualized with respect to the rest of the fleet. Importantly, this operation does not assume any particular ordering of the vehicles. This ensures permutation invariance, and thus accounts for symmetry among homogeneous vehicles.

The final component of the observation function is the Graph Embedder. As illustrated in Figure 3(c), it takes as input the active vehicle embedding H^v , the current time t_k , and the node embedding H^{NV} . In particular, the query vector $H^{v,t} = [H^v | t_k]$ concatenates the active vehicle's embedding with the current time. The context matrix is provided by the node embedding H^{NV} . The corresponding attention mechanism $\text{Att}(H^{v,t}, H^{NV})$ produces a graph-level embedding $H^G \in \mathbb{R}^{1 \times e}$ providing a representation of the active vehicle contextualized with respect to the embedding of the full set of nodes in the graph. We note that the rationale of including t_k in the query vector, is to allow the attention mechanism to assess the relevance of each node by considering the remaining operational time.

Finally, we define the active vehicle's observation of a given state s_k as:

$$o_{k,\bar{v}} = [H^G | H^{v,t}]. \quad (34)$$

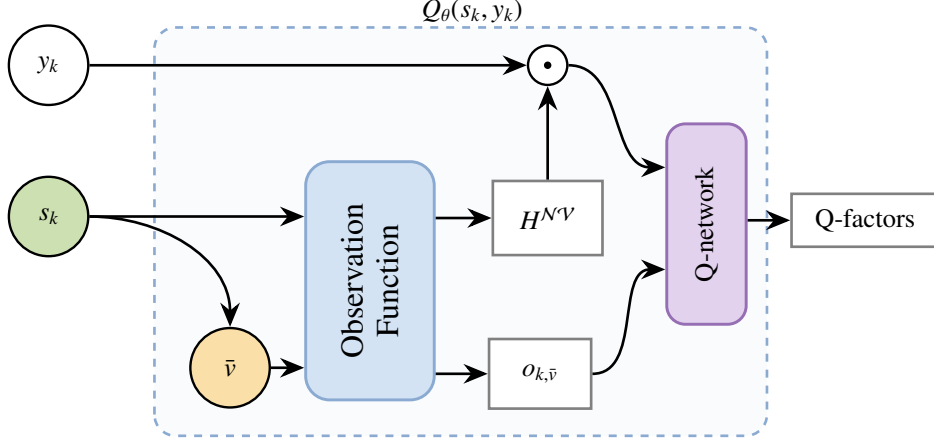


Figure 4: Schematic overview of the proposed DQNCO

4.1.4. The deep Q-learning Algorithm

In this section, we apply the Q-learning algorithm, which is a model-free version of approximate value iteration (Watkins and Dayan 1992), to solve the MDP-CO, specialized to the observation function presented in the previous section. We call the corresponding algorithm DQNCO. In our setting the Q-factor $Q^{\pi_r}(o_{k,\bar{v}}, y_k)$ represents the expected cost-to-go of performing the action y_k when $o_{k,\bar{v}}$ is observed, and then applying policy π_r . We formally define it as follows:

$$Q^{\pi_r}(o_{k,\bar{v}}, y_k) = \mathbb{E}_w[C(s_k, y_k) + \gamma V^{\pi_r}(o_{k+1,\bar{v}'} | o_{k,\bar{v}}, y_k)], \quad (35)$$

where $\bar{v}'$ is the active vehicle in decision epoch $k + 1$. We then rewrite the value function in Equation (32) in terms of Q-factors as follows:

$$V^{\pi_r}(o_{k,\bar{v}}) = Q^{\pi_r}(o_{k,\bar{v}}, \pi_r(o_{k,\bar{v}})), \quad \forall o_{k,\bar{v}} \in \Omega. \quad (36)$$

Therefore, the optimal policy π_r^* is rewritten as:

$$\pi_r^* = \arg \min_{\pi_r} Q^{\pi_r}(o_{k,\bar{v}}, \pi_r(o_{k,\bar{v}})), \quad \forall o_{k,\bar{v}} \in \Omega. \quad (37)$$

Consistently with the convention previously introduced for the value function $V^{\pi_r}(\cdot)$, we use the notation $Q^{\pi_r}(\cdot)$ both for Q-factors defined on the system state s_k and for Q-factors defined on the observation $o_{k,\bar{v}}$. The Q-factors corresponding to the optimal policy π_r^* are denoted $Q^*(\cdot)$. Thus, the optimal state-based Q-factor $Q^*(s_k, y_k)$ is approximated by the observation-based Q-factor $Q^*(o_{k,\bar{v}}, y_k)$.

The optimal policy π_r^* under Q-learning is found through a sequence of interactions of an agent with the environment, the so-called experiences, and by iteratively updating an estimate of the Q-factors. We define an *experience* as a tuple of (s_k, y_k, c_k, s_{k+1}) , describing the agent at the decision epoch k in state s_k , taking action y_k and observing a cost $c_k = C(s_k, y_k)$. The agent then transitions to the state s_{k+1} . The proposed DQNCO is illustrated in Figure 4. The Q-network receives as input the observation $o_{k,\bar{v}}$ and the embedding $H_i^{N^V}$ of a node $i \in \mathcal{N}$ representing the next visit, i.e., y_k , and returns the corresponding Q-factors. More specifically, as defined in Equations (28)–(31), the active vehicle may travel to a customer $c \in \bar{\mathcal{C}}^o$ either directly, denoted as y_c^D , or indirectly (by passing at the depot), denoted as y_c^I . Accordingly, we associate two Q-factors with each customer $c \in \bar{\mathcal{C}}^o$. Coherently, the proposed Q-network returns two Q-factors (Q_c^D, Q_c^I) for each input customer c . Note that to ensure action feasibility, we set Q_c^I to $+\infty$, if serving that customer indirectly is not feasible, i.e., $y_c^I \notin \bar{A}(s_k)$. To compute the Q-factor of returning to the depot, we follow a similar procedure, by passing the node embedding of the depot, i.e., $H_0^{N^V}$, to the Q-network. In this case, the second Q_0^I is not applicable, thus we set it to $+\infty$.

We designed the Q-network as a two-layer artificial neural network, connected by Rectified Linear Unit (ReLU) activation functions, denoted by σ . Thus, Q-factors for a given observation $o_{k,\bar{v}}$ and a node $i \in \mathcal{N}$ with the correspond-

ing node embedding of $H_i^{N\mathcal{V}}$ can be computed as follows:

$$[Q_i^D, Q_i^I] = \sigma(\sigma([o_{k,\bar{v}}|H_i^{N\mathcal{V}}] W_1^Q + b_1^Q) W_2^Q + b_2^Q), \quad (38)$$

where W_1^Q, W_2^Q, b_1^Q , and b_2^Q are trainable parameters.

Since both the observation function and the Q-network are based on neural networks, we train both networks together in a single deep neural network structure characterized by a parameter θ . We thus introduce a function $Q_\theta(s_k, y_k)$ representing the combination of the observation function and the Q-network, where θ expresses the dependence on the values of the trainable parameters. For a given θ , $Q_\theta(s_k, y_k)$ represents an estimate of the Q-factors. The training procedure aims at minimizing a loss function. We define the loss function as the Mean Squared Error (MSE) of Δ , where Δ is the temporal difference between the approximated Q-factor given by the network $Q_\theta(s_k, y_k)$ and the estimated expected Q-factor $\hat{Q}(s_k, y_k)$ computed for each experience (s_k, y_k, c_k, s_{k+1}) . $\hat{Q}(s_k, y_k)$ is defined as follows:

$$\hat{Q}(s_k, y_k) = c_k + \gamma \min_{y_{k+1} \in \bar{A}(s_{k+1})} Q_\theta(s_{k+1}, y_{k+1}). \quad (39)$$

Let θ^* be a minimizer of the loss function Δ . We then approximate the optimal Q-factor $Q^*(s_k, y_k)$ by $Q_{\theta^*}(s_k, y_k)$.

After exploring different enhancement techniques, preliminary experiments revealed that using the Double Q-network (Van Hasselt et al. 2016) and replay memory help our method to reach better convergence, as well as improving its overall performance. The double Q-network considers a secondary network parameterized by $\bar{\theta}$. This network enables us to evaluate the expected Q-factors ($\hat{Q}$) based on a stable target, resulting in a more stable behavior in the training process. Therefore, we replace the secondary network with parameters $\bar{\theta}$, by a copy of the primary network with parameters θ , every P^d trials. The traditional Q-network methods train the network for the most recent experiences. However, the memory replay technique buffers new experiences into a fixed-size list, denoted as B , and enables us to train the network periodically over a subset of experiences randomly sampled from the buffer list. The buffer list has the FIFO property which discards older experiences when its capacity is reached. With the probability P^l at each decision epoch, we uniformly sample a minibatch of experiences, denoted $\tilde{B}$, from B and use it to update the network parameters θ . Using the replay memory (Mnih et al. 2015) has a number of benefits, including breaking the correlation between successive experiences and giving more opportunities to those that occur often. The choice of the enhancement techniques explained above is determined empirically by performing a series of preliminary experiments. We detail the DQNCO in Algorithm 1.

4.2. The ILS algorithm

4.2.1. The core implementation

In this section we present a meta-heuristic algorithm to solve the VRP-SDO. Our method is based on the principles of the Iterated Local Search. The main idea is that the ILS searches over the set of possible outsourced customer subsets C^o and takes advantage of the offline learning algorithm, described in Section 4.1, to obtain nearly-instantaneous routing cost estimates of the corresponding committed customer subset $\bar{C}^o$. A general overview of the algorithm is provided in Algorithm 2.

The ILS takes the customer set C as input and produces an outsourcing solution x^* as output. It begins by generating an initial solution x defining a subset of C . The `initial_solution()` function assigns $x_c = 1$ to a random subset of customers in C and $x_c = 0$ for the remaining customers. The `evaluate_solution(x)` function estimates the objective value of solution x by replacing the expected routing cost $R(\bar{C}^o)$ in (1) and (4), by its approximation $V^*(s_0)$, which is computed offline and retrieved by solving $V^*(s_0) = \min_{y_k \in \bar{A}(s_0)} Q_\theta(s_0, y_k)$, as defined in (36). Specifically,

$$\text{evaluate_solution}(x) = \Psi(C^o) + V^*(s_0).$$

The ILS algorithm then employs a Variable Neighborhood Search (VNS) procedure (lines 7–16 in Algorithm 2). The VNS algorithm seeks the local optimum around x by exploring two neighborhoods. First, the “Add” neighborhood structure generates neighbors of solution x by converting an outsourced customer ($x_c = 0$) in x to a committed customer ($x_c \leftarrow 1$). Second, the “Swap” neighborhood structure generates neighbors of x by simultaneously converting an outsourced customer ($x_c = 0$) to a committed customer ($x_c \leftarrow 1$) and setting a committed customer ($x_{c'} = 1$) to an outsourced customer ($x_{c'} \leftarrow 0$). The algorithm begins with the “Add” neighborhood (lines 8–11), identifying the

Algorithm 1: DQNCO

```
1 Initialize  $\theta, \bar{\theta}$ 
2 while  $trials < Trials\_MAX$  do
3   Observe a new set of customers  $\bar{C}^o$ 
4   Generate a sample demand scenario  $w$ 
5    $k \leftarrow 0$ 
6   while true do
7      $\bar{v} \leftarrow \bar{V}_k[0]$ 
8     if  $k > 0$  then
9       Given the demand scenario  $w$ , transition from  $s_{k-1}^y$  to  $s_k$ 
10      Append  $(s_{k-1}, y_{k-1}, c_{k-1}, s_k)$  to  $B$ 
11    end
12     $y_k = \begin{cases} \text{random action in } \bar{A}(s_k) & rand[0, 1] < \epsilon \\ \arg \min_{y \in \bar{A}(s_k)} Q_\theta(s_k, y) & \text{otherwise} \end{cases}$ 
13     $c_k = C(s_k, y_k)$ 
14    Transition from  $s_k$  to  $s_k^y$ 
15    if  $rand[0, 1] < P^t$  then
16      Sample the minibatch  $\tilde{B}$  from  $B$ 
17       $\Delta \leftarrow [0]_{|\tilde{B}|}$ 
18       $i \leftarrow 0$ 
19      foreach  $(s, y, c, s') \in \tilde{B}$  do
20        if  $s'$  is terminal then
21           $\Delta[i] = Q_\theta(s, y) - c$ 
22        else
23           $\Delta[i] = Q_\theta(s, y) - [c + \gamma \min_{y' \in \bar{A}(s')} Q_{\bar{\theta}}(s', y')]$ 
24        end
25         $i \leftarrow i + 1$ 
26      end
27      Compute the loss function and update the network  $\theta$ :  $Loss = \mathbb{E}_{\tilde{B}}(\Delta^2)$ ,  $\theta = \theta - \alpha \nabla_\theta Loss$ 
28    end
29    if  $k = K$  then
30      break
31    end
32    Set  $k = k + 1$ 
33  end
34  Set  $trials = trials + 1$ 
35  Every  $P^d$  trials, set  $\bar{\theta} = \theta$ 
36  Decay the learning rate  $\alpha$  and the exploration rate  $\epsilon$ 
37 end
38 return  $Q_\theta(., .)$ 
```

best neighborhood solution (x', f') , and replacing it with the current solution (x, f) if it has a lower objective value (i.e., $f' < f$). If no improvement occurs, the algorithm explores the 'Swap' neighborhood of the current solution x (lines 12–15) and repeats the procedure. If no improvement is found in both neighborhoods, the VNS loop terminates, and solution (x, f) is returned. At the end of the VNS step, we devise an optional step, called *online fine-tuning* (lines 17–20), which is detailed in Section 4.2.2. In the next step (lines 21–23), the current solution (x, f) is compared with the best solution (x^*, f^*) . If $f < f^*$, the best solution x^* is updated; otherwise, we increment *no_change* by one. The VNS algorithm is repeated while $itr < \xi$ and *no_change* < 15, each time perturbing the best solution x^* to generate a new solution x . The *perturbation* (x^*, κ) function shakes the best solution by randomly removing κ customers from

$\overline{C}^o$ and assigning them to the outsourced set. This function reduces the risk of getting stuck in local optima.

Algorithm 2: Iterated Local Search

```

1  $(x^*, f^*) \leftarrow (\emptyset, \infty)$ 
2  $itr \leftarrow 0, no\_change \leftarrow 0$ 
3 while  $itr < \xi$  and  $no\_change < 15$  do
4    $\kappa \leftarrow \min(\lceil \frac{no\_change+1}{5} \rceil, 3)$  //  $\kappa \in \{1, 2, 3\}$ 
5    $x \leftarrow \begin{cases} \text{initial\_solution}(C) & x^* = \emptyset \\ \text{perturbation}(x^*, \kappa) & \text{otherwise} \end{cases}$  // generate  $x$  by initializing or shaking  $x^*$ 
6    $f \leftarrow \text{evaluate\_solution}(x)$ 
7   while true do
8      $(x', f') \leftarrow \text{best\_improvement}(x, \text{Add})$  // explore the ‘Add’ neighborhood of  $x$ 
9     if  $f' < f$  then
10        $(x, f) \leftarrow (x', f')$  // update the current solution
11       Continue
12      $(x', f') \leftarrow \text{best\_improvement}(x, \text{Swap})$  // explore the ‘Swap’ neighborhood of  $x$ 
13     if  $f' < f$  then
14        $(x, f) \leftarrow (x', f')$  // update the current solution
15       Continue
16     break
17   if online_fine_tuning then
18      $f^r \leftarrow \text{simulate\_routing}(x)$  // estimate the routing costs via simulation
19      $f \leftarrow \Psi(\{c \in C | x_c = 0\}) + f^r$  // update the costs of solution  $x$ 
20      $\text{fine\_tuning}(x, f^r)$  // fine-tune the value function
21   if  $f < f^*$  then
22      $(x^*, f^*) \leftarrow (x, f)$  // update the best solution
23      $no\_change \leftarrow 0$ 
24   else
25      $no\_change \leftarrow no\_change + 1$ 
26    $itr \leftarrow itr + 1$ 
27 return  $(x^*, f^*)$ 

```

4.2.2. ILS algorithm enhancement

Preliminary experiments revealed some differences between routing costs estimated via the value function $V^*(s_0)$ (36) and those computed by simulating the routing policy π_r^* (37) on the customer set $\overline{C}^o$. Specifically, on small instances the observed gaps were relatively modest on average, however, for some $\overline{C}^o$, this gap was significant. As a result, some outsourcing decisions looked considerably more attractive than what they really were. We noticed that the ILS algorithm sometimes invested considerable effort exploring neighborhoods of these solutions, disregarding potentially better alternatives. Lines 17–20 of Algorithm 2 refer to an optional procedure, called online fine-tuning, to address this issue. If the procedure is active, it first ensures that the estimated routing cost $V^*(s_0)$ of the selected solution accurately reflects the performance of the routing policy. This involves simulating the routing policy (37) for a number of demand scenarios and confronting the observed routing costs with $V^*(s_0)$. We then perform a fine-tuning step, which consists in a partial re-training of the deep Q-Network. The main purpose is to locally improve the estimate of the value function.

The procedure $\text{fine_tuning}(x, f^r)$ takes as input the outsourcing solution x and its simulated routing costs f^r . It then creates a copy of the pre-trained Q-network, denoted by $Q_{\theta'}(s_k, y)$, and updates its parameters θ' using the simulated costs, while keeping the original network $Q_{\theta}(s_k, y)$ fixed for routing committed customers. The fine-tuning

step is developed as a supervised learning procedure. Each call to `fine_tuning( $x, f^r$ )` adds the pair (x, f^r) to a training set. The model is then trained to minimize the error between the predicted routing cost $Q_\theta(s_0, y_0)$ and the corresponding simulated cost f^r , where $y_0 = \arg \min_y Q_\theta(s_0, y)$. The parameters are updated via backpropagation for a fixed number of epochs (10 in our implementation).

It is important to note that repeatedly simulating policy (37) is time-consuming and cannot be executed every time the ILS requires evaluating routing costs. In consequence, the function `best_improvement()`, where the ILS algorithm explores the neighborhood of the current solution, completely relies on the cost estimation provided by the value function $V^*(s_k)$. We use the simulation-based estimation only at the end of every round of neighborhood search. We demonstrate the impact of adopting the fine-tuning step in Section 5.2.3.

5. Computational Results

In this section we assess the performance of the proposed solution method through a set of computational experiments. This section is organized as follows. Section 5.1 describes the experimental setup, namely the instances and the test and training protocol. Section 5.2 reports the computational experiments, in which we assess the ILS algorithm and the routing policy DQNCO both in isolation and as components of the complete solution method, comparing each against custom benchmarks. Finally, Section 5.3 draws managerial insights from the results.

5.1. Experimental setup

Section 5.1.1 describes the instances used in our experiments and the procedure by which they are generated. Section 5.1.2 then specifies the test and training protocol, including the hyperparameters of the learning-based methods.

5.1.1. Instances

Since no instance set for the VRP-SDO is available in the literature, we constructed one. To this end, we built on the instance generation procedure of Dastpak et al. (2023), proposed for a related problem, and adapted it to our setting. The problem studied in Dastpak et al. (2023), denoted VRP-VCSD, has a hard duration limit constraint, so that no overtime operation is allowed, its objective is to maximize the demand served within that limit, and unserved customers are treated as missed opportunities rather than outsourced. Despite these differences, VRP-VCSD and VRP-SD share the following features: variable customer sets, stochastic demands, and a fixed capacitated fleet operating under a duration limit. These commonalities allow us to adopt their generation procedure and extend it to account for the overtime and outsourcing costs of the VRP-SDO.

Following Dastpak et al. (2023), each instance i is defined by a customer density $D \in \mathcal{D} = \{\text{Low, Moderate, High}\}$ and a vehicle capacity $Q \in \mathcal{Q} = \{25, 50, 75\}$. A customer density D corresponds to a distribution function Γ^C that governs the number of customers in the service area, their locations, drawn uniformly over that area, and their expected demands, each assigned uniformly at random from $\{5, 10, 15\}$. The average number of customers $\bar{n}$, the number of vehicles m , and the duration limit L associated with each density D are reported in Table 1; we refer the reader to Dastpak et al. (2023) for a complete description of the procedure. In Section 4.1.3, we defined and used the parameter $n_{\max}$ as an upper bound for the number of customers in a given day. In our experiment we set $n_{\max} = 1.25\bar{n}$. The demand of each customer c follows a uniform distribution Γ_c^D on $[\bar{d}_c - \vartheta, \bar{d}_c + \vartheta]$, with $\vartheta = 5$ for $\bar{d}_c \in \{10, 15\}$ and $\vartheta = 4$ otherwise.

D	$\bar{n}$	m	L
Low (L)	23	3	221.47
Moderate (M)	53	7	195.54
High (H)	83	11	187.29

Table 1: Values of $\bar{n}$, m , and L for each density level D

Following Mendoza et al. (2016), we set the overtime penalty to $\phi = 2$. For the outsourcing cost function $\Psi(\cdot)$, we adopt, as in Dabia et al. (2019), a piecewise linear function of cost per unit of expected total outsourced demand, illustrated in Figure 5. Specifically, the numbers on each piece of this function represent the outsourcing cost rate. For example, if the planner decides to outsource a total of 300 demand volume, the outsourcing cost will be $200 * 10.0 + (300 - 200) * 9.0 = 2900$. Under this function the LSP receives a discount when it outsources a larger expected volume

of customer demand. The specific outsourcing cost rates are calibrated through a series of preliminary experiments on our instances, so as to balance outsourcing costs against routing costs, the latter comprising both travel and overtime costs. The rates must not be too low, as outsourcing costs well below routing costs would make outsourcing every customer the dominant decision for the LSP; conversely, they must not be too high, so that outsourcing part of the demand remains a viable option. In Section 5.3, we further examine scenarios with lower and higher rates to analyze their impact on the performance of the solution.

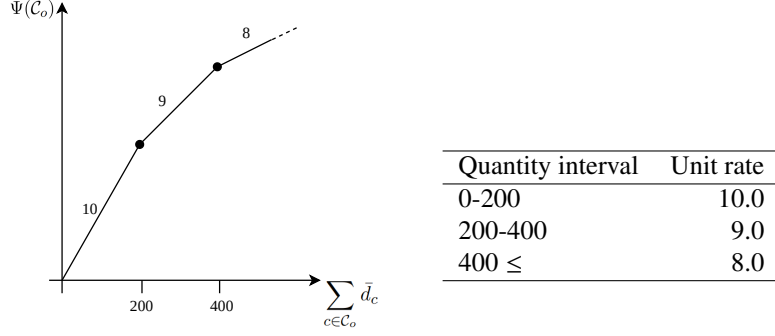


Figure 5: Outsourcing cost function per unit of expected total outsourced demand

In total, this yields nine instances, one for each pair $i \in \mathcal{D} \times \mathcal{Q}$, on which we compared our solution method against the benchmarks.

5.1.2. Test and training protocol

We evaluate all methods on a common test set. For each instance i , a realization $\hat{i}$ is obtained by sampling a customer set from Γ^C and, conditional on it, the customer demands from Γ^D . Each method was tested on every instance i over 250,000 realizations, formed by 500 customer sets sampled from Γ^C and, for each such set, 500 demand realizations sampled from Γ^D . In the ILS algorithm, the maximum number of iterations ξ is set to 100. In the fine-tuning step, we use 50 randomly generated demand scenarios to re-evaluate the performance of the routing policy via simulation.

The Q-learning based methods require a training phase to develop their routing policy. Normally, the training set would be drawn from the marginal distribution $\Gamma^{\bar{C}^o}$ of the committed set $\bar{C}^o$. This is however not feasible in practice, since $\Gamma^{\bar{C}^o}$ is induced by the outsourcing decisions searched by the ILS algorithm (as discussed in Section 4.1.1). We consequently construct the training set by adopting a surrogate procedure. Specifically, a realization $\hat{i}$ in the training set is obtained as follows. We first sample a customer set C from Γ^C . We then emulate the outsourcing decisions searched by the ILS algorithm by removing $\hat{n}$ customers at random from C . The remaining customers are interpreted as the committed set $\bar{C}^o$, thus forming the realization $\hat{i}$. The value $\hat{n}$ is drawn uniformly from $[0, n-1]$. This procedure is repeated for all realizations in the training set. We note that test instances and a code to generate train instances are available at <https://github.com/moda707/vrp-sdo>.

We now specify the hyperparameters of DQNCO. In Q-learning, the discount factor γ is commonly set slightly below one to speed up convergence; however, as discussed earlier, $\gamma < 1$ discounts the value function and thereby underestimates the routing costs. We therefore set $\gamma = 1.0$. To further aid convergence, we employ a double Q-network, in which the target network is replaced by a copy of the primary network every $P^d = 1000$ trials, together with a replay memory of size 50,000. At each decision epoch, the network parameters θ are updated with probability $P^t = 0.05$, and the minibatch size $|\bar{B}|$ and embedding size e are set to 32 and 128, respectively.

We trained DQNCO on five million realizations $\hat{i}$; each such realization used in training is referred to as a trial. The exploration rate ϵ decays linearly from 1.0 to 0.1 over the first third of the training trials, from 0.1 to 0.05 over the second third, and remains at 0.05 thereafter, while the learning rate decays linearly from 10^{-3} to 10^{-4} over the first third.

All Q-learning-based methods are trained on GPUs, whereas all other components of method are executed on CPUs. Specifically, the former were trained on Compute Canada GPU clusters equipped with NVIDIA Tesla V100

GPUs, while the latter were performed on the Béluga cluster of Compute Canada using one Intel Xeon Gold 6148 CPU core and 32 GB of RAM. Computational costs and training times are discussed in Section 5.3.

5.2. Performance Analysis

To the best of our knowledge, no solution method in the literature can be applied directly to the VRP-SDO. We therefore assess our method against a set of suitable custom benchmarks. More specifically, we analyze the performance of the proposed solution method in three steps, progressively combining its two levels. In Section 5.2.1, we isolate the first level by assessing the ILS algorithm against a full-enumeration approach. In Section 5.2.2, we isolate the second level by comparing the routing policy DQNCO against a set of benchmark routing policies on the routing problem alone. Finally, in Section 5.2.3, we evaluate the complete method against benchmarks obtained by combining the ILS algorithm with each of these routing policies, all solving the VRP-SDO.

5.2.1. Performance of the ILS algorithm

We first assess the efficiency of the ILS algorithm employed in IDQNCO. To this end, we compare it against a benchmark that fully enumerates the possible outsourcing solutions and, as in our method, uses DQNCO to evaluate their routing costs, so that the two differ only in how the outsourcing problem is searched. Since this benchmark has $O(2^n)$ time complexity, its computation time grows exponentially with the number of customers; we therefore restrict this experiment to low-density instances.

We ran the full-enumeration benchmark on instances with $D = \text{Low}$ and $Q \in \{25, 50, 75\}$, each over 10 sampled customer sets, imposing a runtime limit of 48 hours per customer set. Within this limit, the enumeration returned the optimal outsourcing solution in 29 of the 30 customer sets. In every case, the optimal solution coincided with the solution returned by IDQNCO, at a considerably lower computational cost: the full-enumeration benchmark required more than four hours on average, whereas the ILS algorithm obtained the same solution in eight seconds on average. This experiment confirms the effectiveness of the ILS algorithm. Table 2 reports the results, where “Enum.” relates to the full enumeration results.

Q	Computation Time		Enum. found Opt. Sol.	Enum. & ILS Sol. matched
	IDQNCO (sec)	Enum. (sec)		
25	12	11820	9/10	9/9
50	6	15407	10/10	10/10
75	5	19927	10/10	10/10
Avg:	8	15718	29/30	29/29

Table 2: Comparing the performance of the ILS with the full enumeration benchmark on instances with Low density

5.2.2. Performance of DQNCO in VRP-SD

We now isolate the second level and assess the routing policy DQNCO on the VRP-SD, comparing it against four benchmark routing policies: random policy (RP), greedy policy (GP), hyper-greedy policy (HP), and SQNCO. In RP, the next location for the active vehicle in decision epoch k is chosen uniformly at random from the set of available customers $\bar{C}_k^o = \{c \in \bar{C}^o | h_c = 1\}$. In GP, the active vehicle is sent to the nearest available customer, that is, $y_k^v = y_c^D$ with $c = \arg \min_{c \in \bar{C}_k^o} \tau_{v,c}$. In HP, the active vehicle is sent to the customer with the highest ratio of served demand to travel distance, that is, $y_k^v = y_c^D$ with $c = \arg \max_{c \in \bar{C}_k^o} \frac{\min(\hat{d}_c, q_v)}{\tau_{v,c}}$, so that customers with larger demands and shorter distances are prioritized. Under all three policies, the active vehicle returns to the depot whenever $|\bar{C}_k^o| = 0$ or $q_v = 0$; staying at the depot, that is, early trip termination, and preventive restocking are not available. The fourth benchmark routing policy is SQNCO, which is the Q-learning method of Dastpak et al. (2023) adapted to our problem. Specifically, it differs from DQNCO in how the observation function is constructed: as detailed in Section 4.1.2, DQNCO relies on a more sophisticated (i.e., deeper) neural network architecture that can capture more relevant information about the active vehicle. Accordingly, SQNCO uses a shallower network and does not employ the graph-attention observation function of DQNCO. SQNCO shares the same exploration and learning rate schedules as DQNCO (Section 5.1.2) but is trained on three million realizations instead of five million.

We solved the VRP-SD for each instance i under all five routing policies, assuming that no customer is outsourced. For each instance, we evaluated the policies on the 250,000 realizations $\hat{i}$ generated as described in Section 5.1.2. Table

3 reports the results. The first two columns identify the instance; the next five report the routing cost — travel plus overtime — under RP, GP, HP, SQNCO, and DQNCO; and the final four columns report the performance gap between DQNCO and benchmark policy X , computed as

$$\%X = \frac{DQNCO - X}{X} \times 100.$$

The results show that DQNCO outperforms all four benchmark routing policies on every instance. On average, it achieves routing costs that are 66.0%, 29.6%, and 34.6% lower than those of RP, GP, and HP, and 19.6% lower than those of SQNCO. This gap demonstrates the effectiveness of the GAT-based observation function adopted in DQNCO.

D	Q	RP	GP	HP	SQNCO	DQNCO	%RP	%GP	%HP	%SQNCO
L	25	2859.3	1835.8	1937.7	1702.3	1343.0	-53.0%	-26.8%	-30.7%	-21.1%
	50	2183.8	1119.5	1242.5	1031.6	748.1	-65.7%	-33.2%	-39.8%	-27.5%
	75	1943.1	857.9	964.1	874.4	577.8	-70.3%	-32.6%	-40.1%	-33.9%
M	25	6987.6	4065.9	4264.9	3572.3	2945.1	-57.9%	-27.6%	-30.9%	-17.6%
	50	5394.5	2455.5	2703.3	2174.0	1660.6	-69.2%	-32.4%	-38.6%	-23.6%
	75	4831.9	1853.1	1987.9	1704.1	1343.7	-72.2%	-27.5%	-32.4%	-21.1%
H	25	11226.6	6122.1	6448.1	4967.4	4483.5	-60.1%	-26.8%	-30.5%	-9.7%
	50	8707.8	3701.3	4025.0	2873.2	2298.5	-73.6%	-37.9%	-42.9%	-20.0%
	75	7832.6	2779.2	2928.8	2221.9	2181.0	-72.2%	-21.5%	-25.5%	-1.8%
Avg:							-66.0%	-29.6%	-34.6%	-19.6%

Table 3: Comparing the performance of the routing policy DQNCO with the other four routing policies RP, GP, HP, and SQNCO

5.2.3. Performance of the full solution method

Having assessed the ILS algorithm and the routing policy DQNCO in isolation, we now evaluate the full solution method, in which the two levels operate jointly to solve the VRP-SDO. We combine the ILS algorithm with each of the routing policies introduced in Section 5.2.2, yielding the methods IRP, IGP, IHP, ISQNCO, and IDQNCO. Applying the online fine-tuning step of Section 4.2.2 to the two Q-learning-based methods, ISQNCO and IDQNCO, yields ISQNCO⁺ and IDQNCO⁺.

We evaluated all seven methods on the test set described in Section 5.1.2. For each of the 500 customer sets, which defines a daily VRP-SDO, every method produces an outsourcing solution by searching with the ILS algorithm. During this search, the ILS estimates the routing cost of each candidate solution as follows: for IRP, IGP, and IHP, by simulating the corresponding policy over 50 demand realizations on the committed customer set $\bar{C}^o$ and averaging; for the Q-learning-based methods, by using the trained value function evaluated at the initial post-decision state, $V^*(s_0)$. We then evaluate the chosen outsourcing solution by simulating the method’s routing policy on the committed customer set over 500 demand realizations, computing the outsourcing cost via the function $\Psi(\cdot)$ introduced in Section 5.1.1, and reporting their sum as the total cost.

Table 4 reports the total cost of each method on every instance, where the total cost is the sum of the outsourcing cost and the estimated routing cost. We also report performance gaps relative to IDQNCO⁺. For each method $X \in \{\text{IRP, IGP, IHP, ISQNCO, ISQNCO}^+, \text{IDQNCO}\}$, the gap is computed as

$$\%X = \frac{\text{IDQNCO}^+ - X}{X} \times 100.$$

The results show that IDQNCO⁺ achieves the lowest total cost on every instance, with average gaps of -45.3%, -22.3%, -27.1%, -19.0%, -13.7%, and -1.2% over IRP, IGP, IHP, ISQNCO, ISQNCO⁺, and IDQNCO, respectively. The gap between IDQNCO and ISQNCO, the two methods that differ only in the routing policy used, confirms the effect of the GAT-based observation function already observed in Section 5.2.2 for DQNCO on VRP-SD. Online fine-tuning improves both ISQNCO and IDQNCO, more substantially for the shallower one: an average gap of -6.3% for ISQNCO⁺ over ISQNCO, compared to -1.2% for IDQNCO⁺ over IDQNCO. We examine the fine-tuning step effects on the first-level decisions separately in the next experiment. A potential interpretation is that the more accurate value-function estimates of DQNCO leave less residual error for the fine-tuning step to correct.

D	Q	IRP	IGP	IHP	ISQNCO	ISQNCO ⁺	IDQNCO	IDQNCO ⁺	%IRP	%IGP	%IHP	%ISQNCO	%ISQNCO ⁺	%IDQNCO
L	25	1675.3	1520.8	1548.0	1475.9	1371.1	1325.6	1278.6	-23.7%	-15.9%	-17.4%	-13.4%	-6.7%	-3.5%
	50	1429.9	1003.3	1096.7	1000.2	878.2	742.3	737.7	-48.4%	-26.5%	-32.7%	-26.2%	-16.0%	-0.6%
	75	1289.0	753.3	804.1	867.2	732.1	577.1	572.3	-55.6%	-24.0%	-28.8%	-34.0%	-21.8%	-0.8%
M	25	3962.5	3584.9	3629.5	3406.5	3322.5	2939.6	2913.1	-26.5%	-18.7%	-19.7%	-14.5%	-12.3%	-0.9%
	50	3435.3	2263.8	2549.7	2172.1	2057.6	1661.4	1643.8	-52.1%	-27.4%	-35.5%	-24.3%	-20.1%	-1.1%
	75	3163.5	1629.4	1721.4	1707.4	1587.2	1338.2	1316.4	-58.4%	-19.2%	-23.5%	-22.9%	-17.1%	-1.6%
H	25	6136.5	5617.9	5710.5	4971.4	4932.2	4482.8	4478.5	-27.0%	-20.3%	-21.6%	-9.9%	-9.2%	-0.1%
	50	5399.3	3589.7	4394.6	2871.0	2825.3	2304.1	2298.6	-57.4%	-36.0%	-47.7%	-19.9%	-18.6%	-0.2%
	75	5050.0	2409.4	2524.0	2227.5	2123.2	2130.0	2095.8	-58.5%	-13.0%	-17.0%	-5.9%	-1.3%	-1.6%
Avg:									-45.3%	-22.3%	-27.1%	-19.0%	-13.7%	-1.2%

Table 4: Comparing the performance of the IDQNCO⁺ with the other six methods in terms of the total costs

5.3. Managerial insights

The experiments of Section 5.2 compare the solution methods against one another. We now examine two questions that an LSP would face when deploying these methods in practice. First, how sensitive are the resulting decisions to the outsourcing rates charged by the common carrier, and how does this sensitivity depend on the LSP’s fleet capacity? Second, how long does each method take to run, and how does that constrain the choice of method given the LSP’s daily decision window?

We consider four scenarios for the outsourcing rates charged by the common carrier, denoted Very Low, Low, Moderate, and High, with unit rates equal to 0.2, 0.5, 1, and 2 times the rates used in Section 5.2, respectively. For each scenario, we ran IDQNCO⁺ on the three instances with Moderate density and $Q \in \{25, 50, 75\}$, on the first 50 customer sets of the test set described in Section 5.1.2. Table 5 reports, for each scenario–instance pair, the percentage of outsourced customers (% Out. Cust.), the percentage of travel time spent in overtime (%Overtime), the routing cost (RC), the outsourcing cost (OC), the total cost (TC), and the percentage of total cost paid for outsourcing (% OC/TC).

Across all capacities, higher rates lead to higher total costs, fewer outsourced customers, and a lower percentage of total cost spent on outsourcing. The intensity of this response, however, depends on the vehicle capacity. As rates rise from Very Low to High, the percentage of outsourced customers drops from 96.5% to 3.0% at $Q = 25$, but only from 26.5% to 1.1% at $Q = 75$. An LSP with a fleet composed of vehicles with a small capacity is therefore exposed to the carrier’s pricing: most of its operation runs through the carrier at low rates and almost none at high rates, so its operational mode heavily depends on the carrier’s price level. An LSP with a fleet composed of large-capacity vehicles, in contrast, is largely rate-insensitive, since even at the lowest rates only about a quarter of demand is outsourced and at the highest rates outsourcing is essentially abandoned.

We now turn to the computation time required to obtain the outsourcing decision each day, and how it constrains the choice of method. We measured the average CPU time required by each method to produce the outsourcing solution on each instance, with an imposed time limit of two hours per run. This time limit was chosen such that, on average, more than 90% of the experiments (57 out of 63, corresponding to 9 instances and 7 methods) were able to find an outsourcing solution within the given time limit. Table 6 reports the results. The three simulation-based methods, IRP, IGP, and IHP, are considerably slower than the learned ones: on the high-density instances the time limit is reached on almost every run, and on average across all instances they require more than an hour each. ISQNCO and IDQNCO, in contrast, complete in 18 and 43 seconds on average, respectively, with the difference reflecting the deeper network used by DQNCO and therefore the higher per-call cost of estimating routing costs. Applying the online fine-tuning step raises these times to 3.2 minutes for ISQNCO⁺ and 8.5 minutes for IDQNCO⁺. These daily times do not include the offline training required by the Q-learning-based methods. Training SQNCO takes between one and two days per instance $i \in \mathcal{D} \times Q$, and training DQNCO between four and sixteen days; both were trained on Compute Canada GPUs. This time is spent once before deployment and does not recur.

Generally speaking, simulation-based methods, i.e., IRP, IGP, and IHP, rely on online simulation and become slower as the problem size grows, which may render them unsuitable when the LSP’s daily decision window is limited. Learning-based methods avoid this online cost by transferring computation effort to the offline training phase, and consequently appear better suited to the operational needs of an LSP, at the price of a substantial one-time training investment that they require beforehand.

Ψ	Q	% Out. Cust.	%Overtime	RC	OC	TC	% OC/TC
[2, 1.8, 1.6]	25	96.51%	1.09%	44.9	922.0	966.9	95.36%
	50	53.36%	7.70%	659.8	498.2	1158.0	43.02%
	75	26.45%	5.78%	861.0	243.4	1104.4	22.04%
	Avg:	58.77%	6.44%	521.9	554.5	1076.5	51.52%
[5, 4.5, 4]	25	47.02%	15.72%	1365.2	1004.7	2369.9	42.39%
	50	8.68%	17.58%	1431.2	150.5	1581.7	9.51%
	75	6.07%	8.77%	1150.8	116.0	1266.8	9.16%
	Avg:	20.59%	14.25%	1315.7	423.7	1739.5	24.36%
[10, 9, 8]	25	11.06%	31.89%	2517.2	377.0	2894.2	13.03%
	50	3.26%	19.03%	1540.4	97.0	1637.4	5.92%
	75	2.23%	11.54%	1252.3	66.0	1318.3	5.01%
	Avg:	5.52%	22.76%	1770.0	180.0	1950.0	9.23%
[15, 13.5, 12]	25	2.99%	35.80%	2801.8	144.0	2945.8	4.89%
	50	1.54%	20.09%	1583.2	61.5	1644.7	3.74%
	75	1.11%	11.31%	1272.8	45.0	1317.8	3.41%
	Avg:	1.88%	25.03%	1885.9	83.5	1969.4	4.24%

Table 5: Sensitivity analysis on the outsourcing costs structure

D	Q	IRP	IGP	IHP	ISQNCO	ISQNCO ⁺	IDQNCO	IDQNCO ⁺
L	25	276	328	473	5	52	12	140
	50	265	236	382	4	34	6	78
	75	225	303	299	3	38	5	73
	Avg:	255	289	385	4	41	8	97
M	25	3566	6215	6698	31	204	95	644
	50	4167	2858	5191	9	109	30	343
	75	3442	3101	3831	7	183	18	424
	Avg:	3725	4058	5240	16	166	48	471
H	25	7200	7200	7200	72	684	124	1055
	50	7200	6919	7200	17	221	44	923
	75	7200	7033	7095	13	192	56	905
	Avg:	7200	7051	7165	34	366	75	961
Total Avg:		3727	3799	4263	18	191	43	510

Table 6: Comparing the computation time (in Seconds) required to solve each instance

6. Conclusions

In this study, we addressed a stochastic variant of the VRP with outsourcing options, in which a set of customers along with their demand distributions are revealed at the beginning of each day. Given this set, an LSP must determine the committed and outsourcing sets of customers. The LSP may pay overtime costs in case of violating a daily work shift limit. The objective is to determine the set of outsourced customers and find the optimal routing policy to serve committed customers such that travel, overtime, and outsourcing costs are minimized.

We formalized this problem as a two-level problem, where the first level identifies committed and outsourced customers, while the second level estimates routing costs. To solve this problem, we introduce the IDQNCO, a heuristic algorithm featuring an ILS algorithm. It evaluates various customer partitions (i.e., committed vs outsourced) and solves a dynamic VRP with stochastic demands for each explored partition to compute its routing costs. To efficiently handle the routing costs estimation, we formalized the second level routing problem as an MDP and proposed the DQNCO, which is a DQN-based algorithm, to solve it. The DQNCO required an intensive training phase, which was in conflict with the need to solve the VRP-SDO on a daily basis. To overcome this limitation, we modified the MDP formulation of the routing problem in such a way as to enable the DQNCO to handle random customer sets drawn from a suitable probability distribution. By doing so, our DQNCO was trained offline once, allowing the IDQNCO to be readily used on a daily basis. Additionally, we proposed IDQNCO⁺, which enhanced IDQNCO with an on-line fine-tuning step that improved the accuracy of routing costs estimation as the ILS algorithm explores different customer partitions.

We compared IDQNCO⁺ against six benchmarks, all employing the same ILS algorithm but differing in their routing policy and, for Q-learning-based methods, in whether the online fine-tuning step was applied. The results revealed that IDQNCO⁺ consistently outperformed IDQNCO and all other benchmarks, reducing overall costs by more than 22.3% relative to benchmarks with basic routing policies and by 13.7% on average relative to ISQNCO⁺.

Notably, IDQNCO⁺ demonstrated superior performance compared to ISQNCO, illustrating the effectiveness of the proposed GAT structure introduced in DQNCO. Additionally, the superiority of IDQNCO⁺ over IDQNCO shows the effectiveness of online fine-tuning in improving the accuracy of the pre-trained value function for estimating routing costs.

From a managerial standpoint, our results offer practical guidance for LSPs considering the VRP-SDO operational setting. The sensitivity analysis on outsourcing rates shows that the impact of a common carrier’s pricing on an LSP’s operations depends heavily on its fleet composition. LSPs operating small capacity vehicles see their outsourced share shift dramatically with rate changes, while LSPs operating large capacity vehicles remain comparatively insensitive to the carrier’s pricing. This suggests that fleet sizing decisions should account not only for expected demand, but also for the LSP’s exposure to third-party pricing dynamics. Furthermore, our results show that the DQN-based methods generate outsourcing decisions in under a minute on average, well within an LSP’s daily decision window, whereas classical approaches are often too slow for this purpose, sometimes taking up to two hours. This computational advantage comes at the cost of an offline training phase of several days, which is incurred once before deployment and does not recur, making the proposed methodology particularly attractive for LSPs seeking a scalable, reusable solution to a problem they must solve every day.

This study demonstrates that an offline-trained Q-network can successfully provide nearly instantaneous and accurate estimations of second-level problems. These estimations can be integrated into traditional heuristic schemes, such as an ILS in our case, which explore the first-level decision space. Therefore, it is promising to explore the application of this framework to other complex optimization problems, particularly those where the second-level problem is a stochastic dynamic problem.

References

- Archetti, C., Feillet, D., Hertz, A., Speranza, M.G., 2009. The capacitated team orienteering and profitable tour problems. *Journal of the Operational Research Society* 60, 831–842.
- Bahdanau, D., Cho, K., Bengio, Y., 2015. Neural machine translation by jointly learning to align and translate, in: 3rd International Conference on Learning Representations, ICLR 2015.
- Baller, A.C., Dabia, S., Dullaert, W.E., Vigo, D., 2020. The vehicle routing problem with partial outsourcing. *Transportation Science* 54, 1034–1052.
- Bolduc, M.C., Renaud, J., Boctor, F., Laporte, G., 2008. A perturbation metaheuristic for the vehicle routing problem with private fleet and common carriers. *Journal of the Operational Research Society* 59, 776–787.
- Bono, G., Dibangoye, J.S., Simonin, O., Matignon, L., Pereyron, F., 2021. Solving multi-agent routing problems using deep attention mechanisms. *IEEE Transactions on Intelligent Transportation Systems* 22, 7804–7813.
- Boussier, S., Feillet, D., Gendreau, M., 2007. An exact algorithm for team orienteering problems. *4OR* 5, 211–230.
- Brito, J., Martínez, F.J., Moreno, J.A., Verdegay, J.L., 2015. An ACO hybrid metaheuristic for close-open vehicle routing problems with time windows and fuzzy constraints. *Applied Soft Computing Journal* 32, 154–163.
- Canada Post Corporation, 2025. Parcel Services Customer Guide. Technical Report. Canada Post Corporation. URL: <https://www.canadapost-postescanada.ca/cpc/doc/en/support/customer-guide/amalgamated-parcel-services-guide.pdf>. accessed: 2025-12-14.
- Cappart, Q., Chételat, D., Khalil, E.B., Lodi, A., Morris, C., Velickovic, P., 2021. Combinatorial optimization and reasoning with graph neural networks, in: IJCAI International Joint Conference on Artificial Intelligence, pp. 4348–4355.
- Ceschia, S., Di Gaspero, L., Schaerf, A., 2011. Tabu search techniques for the heterogeneous vehicle routing problem with time windows and carrier-dependent costs. *Journal of Scheduling* 14, 601–615.
- Chen, X., Ulmer, M.W., Thomas, B.W., 2022. Deep Q-learning for same-day delivery with vehicles and drones. *European Journal of Operational Research* 298, 939–952.
- Chen, Y., Qian, Y., Yao, Y., Wu, Z., Li, R., Zhou, Y., Hu, H., Xu, Y., 2019. Can sophisticated dispatching strategy acquired by reinforcement learning? a case study in dynamic courier dispatching system, in: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, pp. 1395–1403.
- Chu, C.W., 2005. A heuristic algorithm for the truckload and less-than-truckload problem. *European Journal of Operational Research* 165, 657–667.

- Côté, J.F., Potvin, J.Y., 2009. A tabu search heuristic for the vehicle routing problem with private fleet and common carrier. *European Journal of Operational Research* 198, 464–469.
- Dabia, S., Lai, D., Vigo, D., 2019. An exact algorithm for a rich vehicle routing problem with private fleet and common carrier. *Transportation Science* 53, 986–1000.
- Dai, H., Dai, B., Song, L., 2016. Discriminative embeddings of latent variable models for structured data, in: 33rd International Conference on Machine Learning, ICML 2016, pp. 3970–3986.
- Dastpak, M., Errico, F., Jabali, O., 2023. Off-line approximate dynamic programming for the vehicle routing problem with a highly variable customer basis and stochastic demands. *Computers & Operations Research* 159, 106338.
- Erera, A.L., Morales, J.C., Savelsbergh, M., 2010. The vehicle routing problem with stochastic demand and duration constraints. *Transportation Science* 44, 474–492.
- Fan, J., Wang, X., Ning, H., 2006. A multiple vehicles routing problem algorithm with stochastic demand, in: 2006 6th World Congress on Intelligent Control and Automation, pp. 1688–1692.
- Florio, A.M., Gendreau, M., Hartl, R.F., Minner, S., Vidal, T., 2023. Recent advances in vehicle routing with stochastic demands: Bayesian learning for correlated demands and elementary branch-price-and-cut. *European Journal of Operational Research* 306, 1081–1093.
- Florio, A.M., Hartl, R.F., Minner, S., 2020. Optimal a priori tour and restocking policy for the single-vehicle routing problem with stochastic demands. *European Journal of Operational Research* 285, 172–182.
- Gahm, C., Brabänder, C., Tuma, A., 2017. Vehicle routing with private fleet, multiple common carriers offering volume discounts, and rental options. *Transportation Research Part E: Logistics and Transportation Review* 97, 192–216.
- Goeke, D., Gschwind, T., Schneider, M., 2019. Upper and lower bounds for the vehicle-routing problem with private fleet and common carrier. *Discrete Applied Mathematics* 264, 43–61.
- Goodson, J.C., Ohlmann, J.W., Thomas, B.W., 2013. Rollout policies for dynamic solutions to the multivehicle routing problem with stochastic demand and duration limits. *Operations Research* 61, 138–154.
- Goodson, J.C., Thomas, B.W., Ohlmann, J.W., 2016. Restocking-based rollout policies for the vehicle routing problem with stochastic demand and duration limits. *Transportation Science* 50, 591–607.
- Hoogendoorn, Y., Spliet, R., 2023. An improved integer L-shaped method for the vehicle routing problem with stochastic demands. *INFORMS Journal on Computing* 35, 423–439.
- Jabali, O., Rei, W., Gendreau, M., Laporte, G., 2014. Partial-route inequalities for the multi-vehicle routing problem with stochastic demands. *Discrete Applied Mathematics* 177, 121–136.
- Joe, W., Lau, H.C., 2020. Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers, in: Proceedings of the International Conference on Automated Planning and Scheduling, pp. 394–402.
- Kool, W., van Hoof, H., Welling, M., 2019. Attention, learn to solve routing problems!, in: 7th International Conference on Learning Representations, ICLR 2019.
- Kullman, N.D., Cousineau, M., Goodson, J.C., Mendoza, J.E., 2022. Dynamic ride-hailing with electric vehicles. *Transportation Science* 56, 775–794.
- Laporte, G., Louveaux, F.V., van Hamme, L., 2002. An integer L-shaped algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research* 50, 415–423.
- Li, J., Ma, Y., Gao, R., Cao, Z., Lim, A., Song, W., Zhang, J., 2022. Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem. *IEEE Transactions on Cybernetics* 52, 13572–13585.
- Li, M., Qin, Z., Jiao, Y., Yang, Y., Wang, J., Wang, C., Wu, G., Ye, J., 2019. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning, in: The World Wide Web Conference, pp. 983–994.
- Li, X., Luo, W., Yuan, M., Wang, J., Lu, J., Wang, J., Lü, J., Zeng, J., 2021. Learning to optimize industry-scale dynamic pickup and delivery problems, in: 2021 IEEE 37th International Conference on Data Engineering (ICDE), pp. 2511–2522.
- Lin, B., Ghaddar, B., Nathwani, J., 2022. Deep reinforcement learning for the electric vehicle routing problem with time windows. *IEEE Transactions on Intelligent Transportation Systems* 23, 11528–11538.
- Louveaux, F.V., Salazar-González, J.J., 2018. Exact approach for the vehicle routing problem with stochastic demands and preventive returns. *Transportation Science* 52, 1463–1478.
- Maximo, V.R., Cordeau, J.F., Nascimento, M.C.V., 2024. AILS-II: An adaptive iterated local search heuristic for the large-scale capacitated vehicle routing problem. *INFORMS Journal on Computing* 36, 974–986.
- Maxwell, M.S., Restrepo, M., Henderson, S.G., Topaloglu, H., 2010. Approximate dynamic programming for ambulance redeployment. *INFORMS Journal on Computing* 22, 266–281.

- Mendoza, J.E., Rousseau, L.M., Villegas, J.G., 2016. A hybrid metaheuristic for the vehicle routing problem with stochastic demand and duration constraints. *Journal of Heuristics* 22, 539–566.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. *Nature* 518, 529–533.
- Moon, I., Lee, J.H., Seong, J., 2012. Vehicle routing problem with time windows considering overtime and outsourcing vehicles. *Expert Systems with Applications* 39, 13202–13213.
- Moradi, N., Boroujeni, N.M., 2025. Prize-collecting electric vehicle routing model for parcel delivery problem. *Expert Systems with Applications* 259, 125183.
- Nazari, M., Oroojlooy, A., Snyder, L., Takác, M., 2018. Reinforcement learning for solving the vehicle routing problem. *Advances in Neural Information Processing Systems* 31, 9839–9849.
- Oda, T., Joe-Wong, C., 2018. MOVI: A model-free approach to dynamic fleet management, in: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 2708–2716.
- Oroojlooy, A., Hajinezhad, D., 2023. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence* 53, 13677–13722.
- Oyola, J., Arntzen, H., Woodruff, D.L., 2017. The stochastic vehicle routing problem, a literature review, Part II: solution methods. *EURO Journal on Transportation and Logistics* 6, 349–388.
- Oyola, J., Arntzen, H., Woodruff, D.L., 2018. The stochastic vehicle routing problem, a literature review, Part I: models. *EURO Journal on Transportation and Logistics* 7, 193–221.
- Parada, L., Legault, R., Côté, J.F., Gendreau, M., 2024. A disaggregated integer L-shaped method for stochastic vehicle routing problems with monotonic recourse. *European Journal of Operational Research* 318, 520–533.
- Pitney Bowes, 2024. Parcel shipping index. URL: <https://www.pitneybowes.com/content/dam/pitneybowes/us/en/shipping-index/dmr-2334-parcel-shipping-index-ebook-finalv3.pdf>. accessed: 2025-11-25.
- Poon, M., Gu, R., Yuan, Y., 2022. A vehicle routing problem with option for outsourcing and time-dependent travel time. *IEEE Access* 10, 49757–49770.
- Potvin, J.Y., Naud, M.A., 2011. Tabu search with ejection chains for the vehicle routing problem with private fleet and common carrier. *Journal of the Operational Research Society* 62, 326–336.
- Powell, W.B., 2022. *Reinforcement Learning and Stochastic Optimization*. John Wiley & Sons, Ltd.
- Sawadsitang, S., Kaewpuang, R., Jiang, S., Niyato, D., Wang, P., 2017. Optimal stochastic delivery planning in full-truckload and less-than-truckload delivery, in: *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, pp. 1–5.
- Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G., 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 61–80.
- Stenger, A., Vigo, D., Enz, S., Schwind, M., 2013. An adaptive variable neighborhood search algorithm for a vehicle routing problem arising in small package shipping. *Transportation Science* 47, 64–80.
- Su, X., Xu, G., Huang, N., Qin, H., 2023. A branch-and-price-and-cut for the manpower allocation and vehicle routing problem with staff qualifications and time windows. *Advanced Engineering Informatics* 57, 102093.
- Tarasov, K., 2021. Inside the rapid growth of Amazon Logistics and how it's taking on third-party shipping. URL: <https://www.cnbc.com/video/2021/09/04/amazon-now-ships-for-other-merchants-to-compete-with-fedex-ups.html>. accessed: 2025-12-14.
- Van Hasselt, H., Guez, A., Silver, D., 2016. Deep reinforcement learning with double Q-learning, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2094–2100.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need, in: *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6000–6010.
- Vidal, T., Maculan, N., Ochi, L.S., Penna, P.H.V., 2016. Large neighborhoods with implicit customer selection for vehicle routing problems with profits. *Transportation Science* 50, 720–734.
- Watkins, C.J.C.H., Dayan, P., 1992. Q-learning. *Machine Learning* 8, 279–292.
- Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., Dean, J., 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR* abs/1609.08144.

- Xu, Y., Fang, M., Chen, L., Xu, G., Du, Y., Zhang, C., 2022. Reinforcement learning with multiple relational attention for solving vehicle routing problems. *IEEE Transactions on Cybernetics* 52, 11107–11120.
- Zhang, K., He, F., Zhang, Z., Lin, X., Li, M., 2020. Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach. *Transportation Research Part C: Emerging Technologies* 121, 102861.
- Zhu, L., Adulyasak, Y., Rousseau, L.M., 2026. Partial-outsourcing strategy for the vehicle routing problem with stochastic demands. *Networks* 87, 266–288.

Appendix A. Table of notations

Symbol	Description
Problem	
$c, C, \bar{C}^o, C^o$	Customer c and the set of realized, committed and outsourced customers
$v, \mathcal{V}$	Vehicle v and the set of vehicles
G, N, E	A complete graph G with nodes N and arcs E
l_0, l_c, l_v	Location of the depot, customer c , and vehicle v
τ_{ij}	Travel time between locations i and j
h_c, g_v	Availability of customer c and operational status of vehicle v
$\bar{d}_c, d_c, \hat{d}_c$	Expected, realized, and unserved demand of customer c
Γ_c^D, w	Probability distribution of demand for customer c and a vector of demand realizations
Q, q_v	Total capacity of each vehicle and the remaining capacity of vehicle v
a_v, L	Arrival time of vehicle v at its next location and the shift duration of each vehicle
$\Psi(C^o), \phi$	Cost function to outsource customers C^o and the overtime cost penalty
π_r, π_r^*	A routing policy and the optimal routing policy
$T_{\pi_r}(\bar{C}^o, v, w)$	Duration of the route performed by v , given $\bar{C}^o, w$ and π_r
$R(\bar{C}^o)$	Minimum expected routing costs for serving $\bar{C}^o$
x_c	A binary decision variable for the first level problem
MDP	
k, t_k, s_k	Decision epoch, its time, and the system state
$O(s_k, \bar{v}), o_{k,\bar{v}}$	Observation function and observation of active vehicle $\bar{v}$ at decision epoch k
F^C, F^N, F^V	State of customers, all nodes (customers + depot), and vehicles
$\bar{V}_k, \bar{v}$	Set of vehicles active at decision epoch k and the active vehicle
$\bar{C}_k^o, \tilde{C}_k$	Set of available customers and set of customers being served at decision epoch k
y_k	Action vector at decision epoch k
y_c^D, y_c^I	Direct and indirect visit actions to customer c
$A(s_k), \bar{A}(s_k)$	Action space at state s_k for MDP and MDP-CO
γ	Discount factor in the MDP
$C(s_k, y_k)$	Cost of taking action y_k in state s_k
$\Gamma^C, \Gamma^{\bar{C}^o}$	Distribution of daily customer realizations and committed customer subsets $\bar{C}^o$
$S_{\bar{C}^o}, \bar{S}, \Omega$	State space for a fixed customer subset, generalized state space, and active vehicle's observation space
$V^{\pi_r}(s_k), V^{\pi_r}(o_{k,\bar{v}})$	Value function under routing policy π_r for both state and observation
$Q^{\pi_r}(s_k, y_k), Q^{\pi_r}(o_{k,\bar{v}}, y_k)$	Q-factors under routing policy π_r for both state and observation
$\hat{Q}(s_k, y_k)$	Estimated expected Q-factor of state s_k and action y_k
$V^*(s_k), V^*(o_{k,\bar{v}})$	Optimal value function for a given state s_k and observation $o_{k,\bar{v}}$
$Q^*(s_k, y_k), Q^*(o_{k,\bar{v}}, y_k)$	Q-factors of action y_k for a given state s_k and observation $o_{k,\bar{v}}$
Neural network and ILS	
e	Embedding dimension
H^N, H^V	Nodes and vehicles embeddings
$H^{\bar{v}}, H^{\mathcal{G}}$	Embeddings of the active vehicle and the graph
$Att(F, \bar{F})$	Attention mechanism on query F and context $\bar{F}$

continued on next page

Table of Notations (continued)

Symbol	Description
$Q_\theta(s, y)$	Q-factor function for state s , action y , and deep neural network parameters of θ
$\theta, \bar{\theta}$	Parameters of the primary and target deep neural networks
B	Memory replay buffer
P^d, P^t	Double Q-network replacement interval and Q-factors update probability at each decision epoch
ξ	Maximum allowed iterations for the ILS algorithm
Instances	
$\mathcal{D}, Q$	Sets of customer densities and vehicle capacities

Table A.7: Table of Notations