A Hybrid Learning-Based Matheuristic to Solve the Vehicle Routing Problem with Stochastic Demands

Gaël Reynal[1(✉)], Quentin Cappart[1,3,4], Guy Desaulniers[1,2], and Louis-Martin Rousseau[1,3]

[1] Polytechnique Montréal, Montreal, Canada
gael-simon.reynal@etud.polymtl.ca
[2] GERAD, Montreal, Canada
[3] CIRRELT, Montreal, Canada
[4] UCLouvain, Louvain-la-Neuve, Belgium

Abstract. The vehicle routing problem with stochastic demands is a combinatorial optimization problem that arises in industrial applications such as waste management and facility replenishment. In these applications, one could aim at using a specific number of vehicles to better align with available resources, thereby including a fixed-fleet constraint in the problem. Standard column generation heuristics, that are usually efficient in this context, struggle to handle this additional constraint and cannot quickly produce good feasible solutions, mainly because the labeling algorithm used during the pricing becomes inefficient. We introduce a hybrid pricing heuristic that generates columns by combining a greedy component aiming for a quick generation of good columns, a reinforcement learning module to compute critical routes disregarded by the greedy construction, and a tabu search procedure to explore the search space around the generated routes. We embed our method within an existing restricted master heuristic framework: we first perform a column generation phase using our pricing heuristic to quickly generate a set of high-quality columns, which we then complete with a greedy randomized adaptive search procedure. The resulting restricted master problem is then solved as a mixed-integer program. We evaluate our approach on 40 benchmark instances with up to 60 customers and achieve an average optimality gap of 1% within a 5-min total computation time. Our matheuristic also provides more best average solution cost and optimal solutions than the competing heuristics considered.

Keywords: vehicle routing · stochastic demands · restricted master heuristic · pricing heuristics · reinforcement learning

1 Introduction

Vehicle routing problems (VRPs) are widely applicable in industrial contexts, including freight/parcel delivery, waste management, and facility replenishment.

T. Guns (Ed.): CPAIOR 2026, LNCS 16595, pp. 453–469, 2026.
https://doi.org/10.1007/978-3-032-27242-3_27

For this reason, numerous variants of this problem have been studied. The basic situation involves a fleet of vehicles of known capacity departing from a single depot to serve a set of customers with known demands. Additional constraints can be added to better represent the context of the problem at hand. For instance, customer demands may be unknown in contexts such as waste collection [17]. In this case, the demands are stochastic, and only their probability distribution is known beforehand; their realization is revealed only upon reaching the customer. This still allows for *a priori* optimization: the routes are created with a *recourse strategy* to handle the possibility of overloads, a case in which the realized demand exceeds the remaining capacity in the vehicle. This variant is referred to as the *vehicle routing problem with stochastic demands* (VRPSD) [30].

In this work, we introduce an additional constraint to the VRPSD, namely, a fixed-fleet constraint, which specifies a predetermined fleet size. Although the common practice is to impose a lower bound on the number of vehicles used in a solution [10,14], in industrial settings, solutions employing a fixed, given number of vehicles may be preferred because they better correspond to the resources actually available. For example, a firm might want to use all the vehicles at its disposal or provide work to each of its employees. Consequently, the fixed-fleet constraint remains highly relevant in practice.

Looking at the solution approaches, *path-flow formulations* have proven to be an effective way to address this variant [10,12,20]. In these formulations, each feasible route is represented by a variable, which leads to an exponential number of variables overall. This naturally motivates the use of *column generation* [10–12] to solve the linear relaxation. The algorithm iteratively solves a *restricted master problem* (RMP), a restriction of the complete model limited to a subset of its variables, and a *pricing problem* that identifies variables with negative reduced costs to add to the RMP at each iteration. When no such variables remain, the RMP solution is optimal for the linear relaxation. Because this solution is not necessarily feasible for the original problem, the column generation procedure must be integrated into a broader algorithmic framework. To achieve this, one can use a *branch-and-price* algorithm that integrates column generation within a branch-and-bound procedure, possibly enhanced with cutting planes [10–12]. While this method guarantees finding an optimal solution if the pricing problem is solved exactly, it typically produces only a limited number of feasible solutions during the search and can be computationally demanding. As a result, it is less suitable in contexts where the goal is to obtain a good integer solution quickly.

Another option is to solve the RMP as a mixed-integer program (MIP) once the column generation process has converged, a technique commonly known as the *restricted master heuristic* (RMH, see Sadykov et al. [26]). In theory, it should yield a good feasible solution; however, since column generation is not designed to produce complementary columns that support integrality, solving the RMP as a MIP alone is often insufficient. To address this limitation, completion procedures, such as the one proposed by Reynal et al. [25], focus on leveraging the columns identified during column generation to generate complementary columns through a fast heuristic before solving the RMP as a MIP.

However, the introduction of the fixed-fleet constraint causes classical RMHs, such as that of Reynal et al. [25], to struggle even with solving the linear relaxation, often preventing the final MIP from yielding a feasible solution. This difficulty is particularly pronounced in instances with long routes and a small fleet size, as the underlying labeling algorithm becomes computationally complex When the goal is to obtain a good feasible solution rapidly, alternative pricing strategies must be considered. Metaheuristics are well known for their ability to quickly generate high-quality solutions, which can guide the column generation process toward more feasible outcomes [1,15,24].

On another front, *machine learning* has shown promising results when applied to routing problems, including within column generation frameworks. For example, Morabit et al. [21] introduced a bipartite graph neural network at each column generation iteration to select which columns to add to the RMP from a larger candidate set. Their model was trained to imitate a MIP formulation containing all candidate columns that aims at selecting a minimal set of columns that maximizes the objective function decrease. Later, Chi et al. [6] enhanced this approach by employing *reinforcement learning* (RL) to predict the most promising columns to add to the RMP from the candidate set. Their method outperformed the expert model proposed by Morabit et al. [21] on VRPTW instances with up to 50 customers. However, it required an efficient algorithm to generate candidate columns, a challenging task, particularly when long routes are involved. More recently, Morabit et al. [22] proposed a pricing heuristic based on supervised learning for the VRPTW. In many VRP variants, including the VRPSD, the pricing problem can be formulated as an NP-hard elementary shortest path problem with resource constraints. Their method reduces the network size through arc selection, performed by either a random forest or a deep neural network, achieving a reduction of approximately 40% in computational time compared to traditional column generation. A similar approach was designed in [13] for an electric bus scheduling problem. Beyond column generation, Cappart et al. [4] combined RL with constraint programming to address the *traveling salesman problem with time windows*. This line of research has since been extended to further improve efficiency and generality [5,18,19].

These recent works suggest that incorporating a RL model can help the algorithm identify additional high-quality columns to add to the RMP, which motivates our main contribution. We propose a new pricing heuristic designed for the VRPSD with a fixed-fleet constraint. Our heuristic consists of three complementary components:

1. A *greedy module* that rapidly generates columns during the initial iterations.
2. A *RL module* that enables the discovery of high-quality columns even when their search is computationally demanding, and
3. A *tabu search* that enhances diversification within the batch of generated columns, thereby promoting a better exploration of the search space.

We integrate this pricing heuristic into the framework introduced by Reynal et al. [25]. Specifically, our method follows the RMH structure: a column generation phase, followed by the same GRASP-based completion phase (with the

fixed-fleet constraint handled as a soft constraint), and finally, the resolution of the MIP. We experiment on the same set of 40 instances as they did except that we impose the fixed-fleet constraint together with a fixed time limit. Our approach achieves superior performance compared to Reynal et al. [25], which fails to find a feasible solution in 60% to 35% of the instances depending on the available computation time, and also outperforms the GRASP of Mendoza et al. [20] in terms of the number of optimal solutions found across all tested computation time limits.

2 Problem Description

The vehicle routing problem with stochastic demands can be formulated as follows. Let $\mathcal{N}$ be a set of n_C independent customers, each of them having an uncertain demand following a Poisson distribution of parameter μ_j (see, e.g., [7,12,25]). All customers must be served using exactly one of K identical vehicles of capacity Q departing from the depot, denoted as 0. Because the demands are uncertain, we compute the probability $P(OV \mid j,k,n)$ that the n^{th} overload occurs (denoted OV) when reaching customer j with total expected load k :

$$P(OV \mid j,k,n) = 1 - \sum_{i=0}^{nQ-k} \frac{(\mu_j)^i}{i!} e^{-\mu_j}. \tag{1}$$

The overloads are handled using a classical detour-to-depot recourse policy as in [7,12,25], thus designing the routes a priori. In this case, the realization of a customer demand is only revealed upon reaching it. If it results in an overload, the vehicle makes a round trip to the depot to replenish before continuing the route normally.

A vehicle route, denoted by $r = (j_0^r, j_1^r, j_2^r, \ldots, j_{|r|+1}^r)$, starts and ends at the depot $j_0^r = j_{|r|+1}^r = 0$ and visits an ordered list of $|r|$ customers $\mathcal{V}_r = (j_1^r, j_2^r, \ldots, j_{|r|}^r)$ with $|r| \geq 1$. To be feasible, it must respect the constraints:

$$\sum_{l=1}^{|r|} \mu_{j_l^r} \leq Q, \tag{2}$$

$$j_{i_1}^r \neq j_{i_2}^r, \qquad \forall (i_1,i_2) \in \{1,\ldots,|r|\}, i_1 < i_2. \tag{3}$$

Capacity constraint (2) ensures that the total expected demand of a route is less than the vehicle capacity, which limits the number of expensive potential overloads occurring in a route as in several previous works (see, e.g., [12]). The *elementarity constraints* (3) forbid a customer to be visited more than once in the same route. We denote by $\mathcal{R}$ the set of *feasible routes*.

Let $d_{i,j} \in \mathbb{R}^+$ be the Euclidean distance between any pair of nodes i and j, customer or depot. We also introduce a failure penalty $p_{j,k,n} \in \mathbb{R}^+$ accounting for the cost of applying the recourse policy when the n^{th} overload occurs upon reaching customer j with an expected load of k:

$$p_{j,k,n} = 2\, d_{0,j}\; P(OV \mid j,k,n) \tag{4}$$

with $p_{0,k,n} = 0$ for all possible values of k and n. We are now able to compute the expected cost of a route r, denoted c_r, as its total expected travel distance. Let $\kappa_l^r = \sum_{i=1}^{l-1} \mu_{j_i^r}$ be the expected load of a vehicle following route r when it reaches the l^{th} node j_l^r. Then,

$$c_r = \sum_{l=1}^{|r|+1} \left(d_{j_{l-1}^r, j_l^r} + \sum_{n=1}^{\infty} \left(p_{j_l^r, \kappa_l^r, n} - p_{j_{l-1}^r, \kappa_{l-1}^r, n} \right) \right). \tag{5}$$

In practice, the probability that an overload occurs is only taken into account if it is greater than 10^{-4}, as in previous works [12,25]. The VRPSD with the fixed-fleet constraint consists in finding exactly K feasible routes such that each customer in $\mathcal{N}$ is served by exactly one route, while minimizing the sum of the expected route costs. This problem is illustrated in Fig. 1.

To model the VRPSD, we adopt a path-flow formulation. For each feasible route $r \in \mathcal{R}$ and each customer $j \in \mathcal{N}$, let $\nu_{j,r}$ be a binary parameter equal to 1 if route r serves customer j. Moreover, let x_r be a binary variable that takes value 1 if route r is part of the solution and 0 otherwise. The proposed integer program is

$$\min_{x} \quad \sum_{r \in \mathcal{R}} c_r x_r \tag{6}$$

$$\text{s.t.} \quad \sum_{r \in \mathcal{R}} \nu_{j,r} x_r = 1, \qquad \forall j \in \mathcal{N} \tag{7}$$

$$\sum_{r \in \mathcal{R}} x_r = K \tag{8}$$

$$x_r \in \{0, 1\}, \qquad \forall r \in \mathcal{R}. \tag{9}$$

The objective function (6) minimizes the sum of the route expected costs. Constraints (7) ensure that each customer is served by exactly one vehicle, whereas constraint (8) sets the number of routes to select to K. Finally, constraints (9) ensure the integrality of the variables.

3 Technical Background on Restricted Master Heuristics

RMHs form a class of matheuristics typically designed for large-scale MIPs. They rely on column generation to build a reduced-sized MIP that can be solved more efficiently. Column generation is an iterative process that aims to solve a linear relaxation, called the *master problem*. At each iteration, it begins by solving the RMP, a version of the master problem limited to a subset $\hat{\mathcal{R}} \subset \mathcal{R}$ of its variables. Solving the RMP yields a primal solution and a dual vector $\left(\gamma_0, (\gamma_j)_{j \in \mathcal{N}}\right)$, corresponding to the dual variables associated with constraints (7) and (8), respectively. The current primal solution is optimal for the master problem if no variable $x_r \in \mathcal{R} \setminus \hat{\mathcal{R}}$ has a negative reduced cost $\bar{c}_r$, defined as

$$\bar{c}_r = c_r - \sum_{j \in \mathcal{N}} \nu_{j,r} \gamma_j - \gamma_0. \tag{10}$$

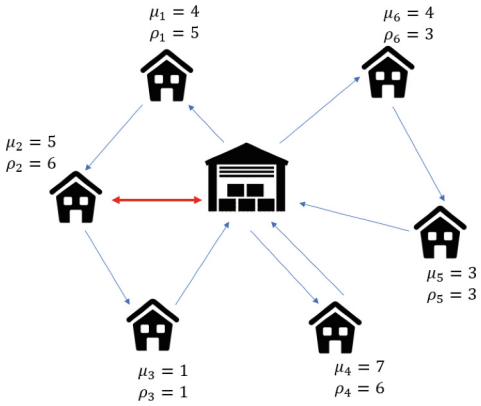

Fig. 1. A toy example of a VRPSD instance with 3 vehicles of capacity 10 and a possible solution. μ_j is the expected demand of customer j while ρ_j is its actual realization, only known upon visiting this customer. An overload occurs at customer 2, triggering a return to the depot before the route is normally completed.

To verify this condition, the algorithm solves the following pricing problem:

$$\min_{r \in \mathcal{R} \setminus \hat{\mathcal{R}}} \bar{c}_r. \tag{11}$$

If variables with a negative reduced cost are found, they are added to the RMP, and a new iteration is performed. Otherwise, the algorithm establishes that the current primal solution is optimal for the master problem and terminates.

For the VRPSD, the pricing problem is formulated as an *elementary shortest path problem with resource constraints* (ESPPRC) [8] on an acyclic network $G = (V, A)$ [7]. The set of vertices V includes a source node and a sink node, denoted as $\langle 0, 0\rangle$ and $\langle 0, Q\rangle$, respectively, to represent the depot. It also contains nodes of the form $\langle j, k\rangle$, where j denotes a customer and $k \in [0, Q - \mu_j]$ represents a cumulative expected demand when reaching customer j. The arc set A is partitioned into three subsets:

$$\begin{array}{ll} A^1 = \{\langle 0,0\rangle, \langle j,0\rangle | j \in \mathcal{N}\} & \text{(leaving source)} \\ A^2 = \{(\langle j,k\rangle, \langle i,k+\mu_j\rangle) \mid j,i \in \mathcal{N}, j \neq i, k \in [0, Q-\mu_j-\mu_i]\} & \text{(linking nodes)} \\ A^3 = \{(\langle j,k\rangle, \langle 0,Q\rangle) \mid j \in \mathcal{N}, k \in [0, Q-\mu_j]\} & \text{(entering sink)} \end{array} \tag{12}$$

Each arc in the above sets has an adjusted cost computed as

$$\bar{c}_a = \begin{cases} d_{0,j} + \sum_{n=1}^{\infty} p_{j,0,n} - \gamma_j & \text{if } a = \big(\langle 0,0\rangle, \langle j,0\rangle\big) \in A^1 \\ d_{j,i} + \sum_{n=1}^{\infty} \big(p_{i,k+\mu_j,n} - p_{j,k,n}\big) - \gamma_i & \text{if } a = \big(\langle j,k\rangle, \langle i,k+\mu_j\rangle\big) \in A^2 \\ d_{j,0} - \gamma_0 & \text{if } a = \big(\langle j,k\rangle, \langle 0,Q\rangle\big) \in A^3, \end{cases} \tag{13}$$

guaranteeing that the cost of a source-to-sink path in G is equal to the reduced cost of the corresponding route.

Because a customer is represented by multiple nodes, a feasible path in G may not correspond to a feasible route in $\mathcal{R}$ due to potential violations of the elementarity constraints (3). To address this, one resource constraint per customer must be added, ensuring that at most one of the nodes representing a given customer can be included in any feasible path.

Most previous works (e.g., [12,25]) solve the pricing problem using a labeling algorithm, often relying on relaxations such as the ng-ESPPRC [2] to accelerate computation. However, for instances involving longer routes, particularly under the fixed-fleet constraint with a tight fleet size, such dynamic programming algorithms tend to lose efficiency, making them irrelevant for a RMH. We thus adapt the RMH from Reynal et al. [25] by replacing their labeling algorithm by a new pricing heuristic that aims to generate high-quality columns efficiently. Once done, the column generation phase is followed by a GRASP completion phase in which the fixed-fleet constraint has been passed as a soft constraint. The resulting RMP is then solved as a MIP.

4 Design of Our Pricing Heuristic

This section presents our core contribution, a hybrid pricing heuristic for the VRPSD. It is formalized in Algorithm 1. First, we generate an initial pool of routes using a greedy procedure (line 2). Then, if there are not enough columns with a negative reduced cost (parametrized with a value Θ in line 3), we complete the generation with a reinforcement learning construction (line 4). At this step, the generated routes may be too similar and we propose to diversify them using a tabu search (line 7). The improved routes are then returned. The next paragraphs describe the three main procedures: the greedy construction, the RL module, and the tabu search.

Algorithm 1. HYBRIDPRICING(.) - Hybrid pricing heuristic

Require: Θ: threshold defining if learning should be used for generating new routes.
Ensure: $\mathcal{R}_{\text{HP}}$: set of generated routes

1: $\mathcal{R}_{\text{HP}} \leftarrow \emptyset$
2: $\mathcal{R} \leftarrow$ GREEDY()
3: **if** $\left|\{r \in \mathcal{R} \mid \bar{c}_r < 0\}\right| < \Theta \cdot |\mathcal{R}|$ **then**
4: $\quad \mathcal{R}_{\text{LEARNING}} \leftarrow$ LEARNING()
5: $\quad \mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_{\text{LEARNING}}$
6: **for** $r \in \mathcal{R}$ **do**
7: $\quad r' \leftarrow$ TABUSEARCH(r)
8: $\quad \mathcal{R}_{\text{HP}} \leftarrow \mathcal{R}_{\text{HP}} \cup \{r'\}$
9: **return** $\mathcal{R}_{\text{HP}}$

4.1 Greedy Construction GREEDY(.)

The greedy component aims to quickly generate columns using two heuristics, namely, *nearest-neighbor* and *best-insertion*, each of which builds routes by adding one customer at a time. We start by describing how they build one route at a time.

Nearest-Neighbor Heuristic. The procedure begins with a route departing from the depot and iteratively selects the next location (customer or depot) to visit. At each step, the route must remain feasible, i.e., satisfy the capacity and elementarity constraints (2) and (3). The chosen extension is the one yielding the least reduced cost. The route is completed when it includes a return to the depot.

Best-Insertion Heuristic. The construction process starts with an empty route. At each iteration, it evaluates the insertion of each remaining customer into all possible positions along the current route. The selected insertion is the one that results in the feasible route with the least reduced cost. The algorithm terminates when no further feasible insertion can improve the solution.

Improvement with Beam-Search. To enable these two heuristics to generate multiple columns simultaneously, we integrate them into a beam-search procedure. Instead of constructing a single route, each heuristic maintains a pool of n_G routes, where n_G is a user-defined parameter. At each iteration, a candidate pool is formed by considering all possible extensions (for the nearest-neighbor heuristic) or insertions (for the best-insertion heuristic) of the routes currently in the pool. The pool is then updated with the n_G candidates exhibiting the least reduced costs. Routes that are already complete are added to the candidate pool without generating new extensions or insertions. The algorithm terminates when the pool contains only complete routes. The beam-search width n_G is set to $5n_C$, where n_C is the number of customers. We observed that larger n_G values lead to larger RMPs, which are more time-consuming to solve while providing only marginal performance improvements.

4.2 Learning Module - LEARNING(.)

Solving a problem with deep reinforcement learning requires to model it as a *Markov decision process* and to design a neural procedure to solve it. Let us define the set of states (S), the set of actions (A), the transition function ($T : S \times A \rightarrow S$) and the reward function ($R : S \times A \rightarrow \mathbb{R}$), as the tuple $\langle S, A, T, R \rangle$. Our model is as follows:

States. We define a state $s \in S$ as the triplet $(j_s, k_s, \mathcal{V}_s)$ where j_s is the last customer visited by the route (i.e. the current location of the agent), k_s is the expected cumulated load gathered in the route up to this point and $\mathcal{V}_s$ is the set of customers already visited by the route. By the problem definition, we know that $k_s = \sum_{j \in \mathcal{V}_s} \mu_j$. A state is terminal if the agent is back to the depot.

Actions. An action corresponds to the selection of the customer $j \in \mathcal{V}$ to visit next. We note that the selection must satisfy the capacity constraint (2) and the elementarity constraints (3). Formally, the set of actions $\mathcal{A}_s$ available at a state $(j_s, k_s, \mathcal{V}_s)$ is as follows:

$$\mathcal{A}_s = \{j \in \mathcal{V} \backslash \mathcal{V}_s \mid k_s + \mu_j \leq Q\}. \tag{14}$$

Transition Function. Executing an action $a \in \mathcal{A}_s$ at state $s = (j_s, k_s, \mathcal{V}_s)$ leads the agent to the state $s' = (a, k_s + \mu_a, \mathcal{V}_s \cup \{a\})$.

Reward Function. Given a state $s = (j_s, k_s, \mathcal{V}_s)$ corresponding to partial route r, choosing to perform action $a \in \mathcal{A}_s$ leading to route $r' = r :: a$ comes with the reward $\bar{c}_{(\langle j_s, k_s \rangle, \langle a, k_s + \mu_a \rangle)}$, which is the adjusted cost of the arc associated with the transition as defined in Eq. (13)

Features. The next question is how to represent a state $s \in S$ in a structure amenable to learning. Following related works in the field of learning for combinatorial optimization, we propose to leverage an architecture based on a graph neural network. Briefly, this architecture operates on inputs having an underlying graph structure. We refer to the work of Cappart et al. [3] for an extended description of this architecture and its use in the context of combinatorial optimization. Let $G = (V, E)$ be a graph representing a current state of the pricing problem. We define the set of vertices V as the set of customers and the set of edges E as the possible connections between customers. Besides, each vertex and edge is decorated with a set of features. They are summarized in Table 1 and mainly corresponds to information either related to the instance (e.g., distances to the depot) or to the pricing state (e.g., the customer resources). The depot is represented by node 0 and corresponds to a customer with a dual variable γ_0 and an expected demand $\mu_0 = 0$.

Neural Architecture. Our architecture is based on an *edge-featured graph attention network* [28] (4 layers of 96 neurons) followed by a multilayer perceptron (3 layers of 96 neurons) with a ReLU activation function. The final activation is a sigmoid that we use to obtain a value comprised between 0 and 1 for each vertex v, which we interpret as the quality of the decision of going to the location related to vertex v, given the current state.

Training Phase. The training data are obtained by solving VRPSD instances from [7]. As done by Reynal et al. [25], we also include variations of these instances with a smaller fleet size K and larger capacity Q to create instances featuring longer routes, yielding a total of 149 instances. Among those, 40 were kept as the dataset for this work and thus not included in the training.

For the remaining 109 instances, we collect dual variable values obtained at each iteration of a column generation algorithm applied to solve the root node of the BPC algorithm, without adding any cutting planes. Each of the 4386 pricing problem instances obtained this way is a training instance for our model, with 60% being used for training, 20% for validation and 20% for testing.

Table 1. Definition of our input graph $G = (V, E)$ representing a state $(j_s, k_s, \mathcal{V}_s)$.

Features on the vertices $j \in V$	
Description	Value
Current location of the agent	1 if $j = j_s$ else 0
Customer resources	1 if $j \in \mathcal{V}_s$ else 0
Possible customers to visit next	$\mathcal{A}_s$ as defined by (14)
Percentage of remaining capacity	$\frac{k_s + \mu_j}{Q}$
Distance to the depot	$d_{0,j}$
Distance from j_s to j	$d_{j_s,j}$
Cost from j_s to j	$d_{j_s,j} + \sum_{n=1}^{\infty} \left(p_{j,k_s+\mu_j,n} - p_{j_s,k_s,n} \right)$
Features on the edges $(j_1, j_2) \in E$	
Description	Value
Distance to the depot from the head node	d_{0,j_2}
Length of (j_1, j_2)	d_{j_1,j_2}
Cost of (j_1, j_2) when leaving with charge k_s	$d_{j_1,j_2} + \sum_{n=1}^{\infty} \left(p_{j_2,k_s+\mu_{j_2},n} - p_{j_1,k_s,n} \right)$
Dual variable at the head node	γ_{j_2}

We train our model using a standard *proximal policy optimization* (PPO) algorithm [27] with a batch size of 32, and a convergence criterion set to a loss reduction below 10^{-6} over 5 iterations. We use Adam optimizer [16] with default parameters, except for the learning rate, which we set at 10^{-5}. The implementation is done using PyTorch and DGL libraries [23,29]. Training ran for 24,587 epochs, representing around a day of computation time.

Once trained, the model is employed to generate new columns. At each step, it provides the k_L most probable customers to visit next, where k_L is a user-defined parameter. To evaluate each potential extension, the resulting partial route is completed using a nearest-neighbor heuristic until it returns to the depot, and the reduced cost of the completed route is used as the evaluation criterion. The extension selected is the one leading to the completed route with the least reduced cost. As before, route feasibility must be maintained at every step. A beam-search strategy is also applied to generate a pool of columns simultaneously. The construction process terminates once n_L complete routes have been generated. We set $n_L = 50$ and $k_L = 3$. These relatively small values, compared to those used in the greedy procedure, stem from the fact that calling the graph neural network is a computationally expensive operation, which requires us to be parsimonious with the number of calls. In line with this rationale, the threshold Θ is set to 0.05, such that the majority of routes are produced by the greedy procedure, while the learning component focuses on identifying critical routes that the greedy construction fails to capture.

4.3 Diversification with a Tabu Search - TabuSearch(.)

A recurrent issue with the previous constructions is that the columns tend to be highly similar, which reduces the effectiveness of the pricing heuristic. To address this limitation, we introduce a diversification mechanism based on a tabu search procedure. Specifically, we apply local perturbations to each route $r = (j_0^r, \ldots, j_{k-1}^r, j_k^r, j_{k+1}^r, \ldots, j_{|r|+1}^r)$ produced by the greedy and learning-based procedures. Two types of moves are considered:

1. *Removal moves*, which remove a customer from the route. Removing the k^{th} customer of route r yields the route $(j_0^r, \ldots, j_{k-1}^r, j_{k+1}^r, \ldots, j_{|r|+1}^r)$.
2. *Insertion moves*, which insert a customer into the route at the best possible position while maintaining route feasibility. Inserting customer j at the k^{th} position of route r yields the route $(j_0^r, \ldots, j_{k-1}^r, j, j_k^r, j_{k+1}^r, \ldots, j_{|r|+1}^r)$.

At each iteration, we perform the first non-tabu improving move found, or otherwise the best non-tabu deteriorating move, using the reduced cost $\bar{c}_r$ as the evaluation function. Once a move is executed, it is added to a tabu list τ of fixed size T to prevent it from being undone by its opposite move during the next T iterations. The tabu list is updated in a first-in, first-out manner: when its maximum capacity is reached, the oldest move is removed to make room for the new one. We perform n_T iterations of this process on each route and return the route with the least reduced cost encountered during the search. In our experiments, we set $n_T = 20$ and $T = 10$. Preliminary tests indicated that increasing these values, up to 50 iterations and a tabu list length of 20, does not lead to significant performance improvements.

5 Experimental Results

In this section, we evaluate our new pricing heuristic and analyze the contribution of each component to its overall performance.

5.1 Experimental Protocol

The tests are conducted on the same set of instances as Reynal et al. [25], except that we consider the fixed-fleet constraint. The fleet size is set to the lower bound on the number of vehicles required to serve all customers $\lceil (\sum_{j \in \mathcal{V}} \mu_j) \setminus Q \rceil$. All instances remain feasible under this configuration. The column generation phase is performed using Gencol 4.5, the MIP resolution with CPLEX 22.1.1, and all experiments are executed on an Intel(R) Core(TM) i7-8700 CPU running at 3.20 GHz. The dataset includes 40 instances of the VRPSD, with 39 to 60 customers. These instances cannot be solved to optimality by the branch-price-and-cut (BPC) algorithm of [12] in one hour.

We evaluate our method (referred to as hybridPricing) using the same computation times as Reynal et al. [25], namely 2, 5, 10, and 15 min, with approximately 50% of the time allocated to column generation and the remaining 50% to the completion phase and MIP solving. We compare our approach with the following baselines:

- *Reynal et al.* [25]. This RMH leverages first a dynamic programming-based pricing to generate routes guided by dual variables from the RMP. It then applies a GRASP metaheuristic to complete partial solutions and introduce new routes that complement those used in the solution of the current RMP. Finally, it solves the resulting model as a MIP.
- *Mendoza et al.* [20]. This method generates routes with a GRASP for 90% of the total computation time and solves a MIP with all generated routes for the remaining 10%.

All reported results are computed by averaging the cost of the solutions obtained by each method over 5 runs.

5.2 Main Result: Comparison with State-of-the-Art Heuristics

The results over the whole dataset are summarized in Table 2 and in Fig. 2 by means of performance profiles [9]. Because most optimal solutions for these instances are unknown, the gaps are computed with respect to the final lower bound returned by the BPC algorithm of [12] after running for 2 h.

As shown by these results, the RMH of Reynal et al. [25] fails to achieve competitive results. It reaches a feasible solution for 40% of the instances for the 2-min timeout. This proportion only increases to 65% with more computation time available as in the 15-min case. Due to the small fleet size, allowing for longer routes these instances are already challenging for the dynamic programming pricing. The addition of the fixed-fleet constraint (8) prevents the column generation from yielding a feasible solution in a reasonable amount of time.

As in Reynal et al. [25], the GRASP of Mendoza et al. [20] performs better with less computation times. For the 2-min case, it achieves a smaller average optimality gap and a larger number of optimal solutions and best average solution cost. As shown on the performance profiles, its best and worst-case scenarios over 5 runs are also better than our algorithm. As available computation time increases, our method HYBRIDPRICING becomes competitive. We achieve an average optimality gap of 1% in the 5 and 10-min cases, outperforming the GRASP. Both methods yield similar worst-case scenarios, but our algorithm provides a better best-case scenario than the GRASP. We account for stochasticity between the 5 runs used to produce these results by conducting two-sided Wilcoxon signed-rank tests, with the null hypothesis H_0 "*both methods yield identical performance*". We obtain p-values of 0.014 and 0.003 respectively for the 5 and 10-min cases, thus rejecting the null in these configurations.

Finally, the GRASP and our method yield almost identical best-case scenarios for the 15-min timeout case. Moreover, the longer computation time makes HYBRIDPRICING much more robust, thus improving its worst-case scenario. The same two-sided Wilcoxon signed-rank test with the same null hypothesis however yields a p-value of 0.055, preventing from confidently rejecting the null in the 15-min case, even if the p-value is only slightly above the 5% threshold.

These results show that our method requires time to be efficient: it heavily relies on the quality of the routes generated during the column generation phase.

Table 2. Average optimality gap and number of instances solved under certain gaps for all baselines and computation times. Average optimality gap is computed on the instances for which a method finds at least one feasible solution. The number of such instances is provided in parentheses. Bold values indicate the best result per row.

		hybridPricing	Mendoza et al.	Reynal et al.
2 minutes	*Avg optimality gap*	1.43%(40)	**1.34%**(40)	1.46%(16)
	# best avg solution cost	14	**28**	5
	# optimal solutions	2	**3**	1
	# < 1% optimality gap	**18**	**18**	7
	# < 5% optimality gap	**40**	**40**	16
5 minutes	*Avg optimality gap*	**1.01%**(40)	1.06%(40)	0.95%(24)
	# best avg solution cost	**27**	24	10
	# optimal solutions	**6**	5	3
	# < 1% optimality gap	24	**25**	15
	# < 5% optimality gap	**40**	**40**	24
10 minutes	*Avg optimality gap*	**0.95%**(40)	1.02%(40)	1.62%(25)
	# best avg solution cost	**27**	23	12
	# optimal solutions	**6**	5	3
	# < 1% optimality gap	**26**	**26**	17
	# < 5% optimality gap	**40**	**40**	24
15 minutes	*Avg optimality gap*	**0.90%**(40)	0.94%(40)	0.89%(26)
	# best avg solution cost	**26**	13	21
	# optimal solutions	**10**	6	3
	# < 1% optimality gap	**27**	**27**	18
	# < 5% optimality gap	**40**	**40**	26

In the 2-min case, it is too short to find high-quality routes to complete with the GRASP. However, the results in the 5-, 10-, and 15-min cases show that the column generation phase identifies critical columns that the GRASP disregards, yielding better solutions.

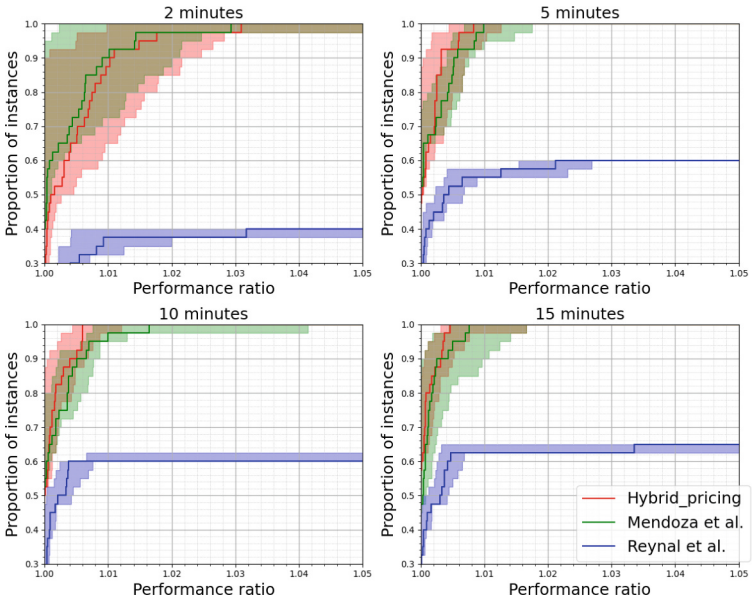

Fig. 2. Performance profiles on the optimality gap for all baselines. Plain line is the average performance over 5 runs while the colored zone around it represents the minimal and maximal performance. Performance ratios are computed with regard to the best solution obtained by all considered methods over all 5 runs.

5.3 Analysis: Ablation Study

To better assess the contribution of each component in our RMH, we conducted an ablation study. Specifically, we independently removed each module of the hybrid pricing framework (i.e. greedy construction, learning component, and tabu search) and computed the number of instances for which they yield the best average solution cost, when benchmarked against the RMH of [25] and the GRASP of [20]. The corresponding results are presented in Table 3.

We first observe from the first three rows that removing any single module leads to a noticeable degradation in performance, indicating that all three components play complementary roles in achieving competitive results. Among them, the tabu search is particularly critical: in all cases, its removal causes a significant drop in performance. This can be explained by the fact that the routes generated by the greedy construction tend to be similar (as is also the case for the routes generated by the RL module), thus emphasizing the importance of diversification mechanisms introduced by the tabu search.

Another noteworthy observation concerns the learning component. When used as the only module to generate columns (variant - Greedy and Tabu), the performance of the RMH is poor for the 2-min timeout (1 best average solution cost), but as more computation time becomes available, it steadily increases to reach 15 best average solution costs for a time of 15 min. This suggests that

the learning mechanism requires sufficient time to fully exploit its potential and guide the search process effectively.

Finally, the greedy construction complements the RL module. When used alone (variant - Learning and Tabu), it quickly reaches its limit in terms of quality, yielding poor performance even when combined with tabu search (variant - Learning) for timeouts of less than 15 min. The combination with the learning module (variant - Tabu) allows for going through the first easy column generation iterations without wasting time on them, leaving more time to handle the last difficult iterations with the learning module, thus increasing the performance of the algorithm. However, the lack of diversification brought by the tabu search in hybridPricing prevents it from yielding competitive results.

Table 3. Results of the ablation study. #: number of best avg solution cost; %: number of best avg solution cost in percentage; Δ: loss (in percentage) on the number of best avg solution cost with respect to hybridPricing.

	2 minutes			5 minutes			10 minutes			15 minutes		
Variant	#	%	Δ	#	%	Δ	#	%	Δ	#	%	Δ
hybridPricing	14	35	-	27	67.5	-	27	67.5	-	26	65	-
- Learning	4	10.0	-71.4	6	15.0	-77.7	9	22.5	-66.7	14	35.0	-46.1
- Greedy	7	17.5	-50.0	8	20.0	-70.3	12	30.0	-55.6	17	42.5	-34.6
- Tabu	4	10.0	-71.4	6	15.0	-77.7	8	20.0	-70.3	12	30.0	-53.8
- Learning and Tabu	4	10.0	-71.4	4	10.0	-74.1	7	17.5	-79.4	9	22.5	-65.3
- Greedy and Tabu	1	2.5	-92.8	7	20.0	-74.1	13	32.5	-51.9	15	37.5	-42.3

6 Conclusion

This paper introduced a new pricing heuristic for the VRPSD under a fixed-fleet constraint, integrating three complementary components: (1) a greedy construction procedure to rapidly generate initial solutions and stabilize the dual variables in the early iterations of column generation; (2) a deep reinforcement learning agent that exploits problem-specific knowledge acquired during training to produce high-quality columns; and (3) a tabu search mechanism that enhances diversification among the routes generated by the other two components.

Experimental results show that our RMH performs competitively with the recent one of Reynal et al. [25], while achieving a higher ratio of optimal solutions, more best average solution costs, and smaller average optimality gaps than the classical GRASP of Mendoza et al. [20]. The ablation study confirms that each component contributes to the overall performance of the method, highlighting the synergy between learning, constructive, and diversification strategies.

Future works could focus on improving the greedy construction. In its current state, it allows the column generation to quickly go through the early iterations in which the generated columns are less relevant, but the resulting routes are of poor quality and thus rarely contribute to the returned solution. Another option would be to consider a different construction method for the learning module, which only builds the routes by choosing which client to visit next. One could instead consider a clustering algorithm in which the model would take a route as input and return, for each other customer, if it should also be part of the route, inserting the most promising ones at the best possible place.

References

1. Alvelos, F., de Sousa, A., Santos, D.: Combining column generation and metaheuristics. In: Talbi, E. (ed.) Hybrid Metaheuristics. Studies in Computational Intelligence, vol. 434, pp. 285–334. Springer, Berlin (2013)
2. Baldacci, R., Mingozzi, A., Roberti, R.: New route relaxation and pricing strategies for the vehicle routing problem. Oper. Res. **59**(5), 1269–1283 (2011)
3. Cappart, Q., Chételat, D., Khalil, E.B., Lodi, A., Morris, C., Veličković, P.: Combinatorial optimization and reasoning with graph neural networks. J. Mach. Learn. Res. **24**(130), 1–61 (2023)
4. Cappart, Q., Moisan, T., Rousseau, L.M., Prémont-Schwarz, I., Cire, A.A.: Combining reinforcement learning and constraint programming for combinatorial optimization. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 3677–3687 (2021)
5. Chalumeau, F., Coulon, I., Cappart, Q., Rousseau, L.-M.: SeaPearl: a constraint programming solver guided by reinforcement learning. In: Stuckey, P.J. (ed.) CPAIOR 2021. LNCS, vol. 12735, pp. 392–409. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78230-6_25
6. Chi, C., Aboussalah, A., Khalil, E., Wang, J., Sherkat-Masoumi, Z.: A deep reinforcement learning framework for column generation. In: Advances in Neural Information Processing Systems, vol. 35, pp. 9633–9644 (2022)
7. Christiansen, C.H., Lysgaard, J.: A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. Oper. Res. Lett. **35**(6), 773–781 (2007)
8. Costa, L., Contardo, C., Desaulniers, G.: Exact branch-price-and-cut algorithms for vehicle routing. Transp. Sci. **53**(4), 946–985 (2019)
9. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)
10. Florio, A.M., Gendreau, M., Hartl, R.F., Minner, S., Vidal, T.: Recent advances in vehicle routing with stochastic demands: Bayesian learning for correlated demands and elementary branch-price-and-cut. Eur. J. Oper. Res. **306**(3), 1081–1093 (2023)
11. Florio, A.M., Hartl, R.F., Minner, S.: New exact algorithm for the vehicle routing problem with stochastic demands. Transp. Sci. **54**(4), 1073–1090 (2020)
12. Gauvin, C., Desaulniers, G., Gendreau, M.: A branch-cut-and-price algorithm for the vehicle routing problem with stochastic demands. Comput. Oper. Res. **50**, 141–153 (2014)
13. Gerbaux, J., Desaulniers, G., Cappart, Q.: A machine-learning-based column generation heuristic for electric bus scheduling. Comput. Oper. Res. **173**, 106848 (2025)

14. Hoogendoorn, Y., Spliet, R.: An evaluation of common modeling choices for the vehicle routing problem with stochastic demands. Eur. J. Oper. Res. **321**(1), 107–122 (2025)
15. Hu, W., Du, B., Wu, Y., Liang, H., Peng, C., Hu, Q.: A hybrid column generation algorithm based on metaheuristic optimization. Transport **31**(4), 389–407 (2016)
16. Kingma, D.P.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
17. Liao, S., Xu, Y., Niu, Y., Cao, Z.: Learning-guided bi-objective evolutionary optimization for green municipal waste collection vehicle routing. J. Clean. Prod. **501**, 145316 (2025)
18. Marty, T., et al.: Learning and fine-tuning a generic value-selection heuristic inside a constraint programming solver. Constraints **29**(3), 234–260 (2024)
19. Marty, T., François, T., Tessier, P., Gautier, L., Rousseau, L.M., Cappart, Q.: Learning a generic value-selection heuristic inside a constraint programming solver. In: 29th International Conference on Principles and Practice of Constraint Programming (CP 2023), pp. 25–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2023)
20. Mendoza, J.E., Rousseau, L.M., Villegas, J.G.: A hybrid metaheuristic for the vehicle routing problem with stochastic demand and duration constraints. J. Heuristics **22**, 539–566 (2016)
21. Morabit, M., Desaulniers, G., Lodi, A.: Machine-learning-based column selection for column generation. Transp. Sci. **55**(4), 815–831 (2021)
22. Morabit, M., Desaulniers, G., Lodi, A.: Machine-learning-based arc selection for constrained shortest path problems in column generation. INFORMS J. Optim. **5**(2), 191–210 (2023)
23. Paszke, A., et al.: Automatic differentiation in PyTorch (2017)
24. Prescott-Gagnon, E., Desaulniers, G., Rousseau, L.M.: A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. Networks **54**(4), 190–204 (2009)
25. Reynal, G., Cappart, Q., Desaulniers, G., Rousseau, L.M.: Improving column complementarity in a restricted master heuristic with a grasp-guided completion: application to the vehicle routing problem with stochastic demands. Les Cahiers du GERAD G-2025-56, HEC Montréal (2025). https://www.gerad.ca/fr/papers/G-2025-57
26. Sadykov, R., Vanderbeck, F., Pessoa, A., Tahiri, I., Uchoa, E.: Primal heuristics for branch and price: the assets of diving methods. INFORMS J. Comput. **31**(2), 251–267 (2019)
27. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
28. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
29. Wang, M., et al.: Deep graph library: a graph-centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315 (2019)
30. Yee, J.R., Golden, B.L.: A note on determining operating strategies for probabilistic vehicle routing. Naval Res. Logistics Q. **27**(1), 159–163 (1980)