Les Cahiers du GERAD ISSN: 0711–2440

Improving column complementarity in a restricted master heuristic with a GRASP-guided completion: Application to the vehicle routing problem with stochastic demands

G. Reynal, Q. Cappart, G. Desaulniers, L.-M. Rousseau

G–2025–56

August 2025

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Improving column complementarity in a restricted master heuristic with a GRASP-guided completion: Application to the vehicle routing problem with stochastic demands

Gaël Reynal [a, b]

Quentin Cappart [a, c]

Guy Desaulniers [a, b]

Louis-Martin Rousseau [a, c]

[a] *Bibliothèque, HEC Montréal, Montréal (Qc), Canada, H3T 2A7*

[b] *GERAD, Montréal (Qc), Canada, H3T 1J4*

[c] *CIRRELT, Montréal (Qc), Canada, H3T 1J4*

gael-simon.reynal@polymtl.ca
quentin.cappart@polymtl.ca
guy.desaulniers@polymtl.ca
louis-martin.rousseau@polymtl.ca

August 2025
Les Cahiers du GERAD
G–2025–56

Abstract : Many combinatorial optimization problems, such as vehicle and crew scheduling, can be modeled using path-flow formulations, where each variable represents a feasible route. A major challenge in these formulations lies in efficiently generating high-quality integer solutions from a model with exponentially many variables. We propose a hybrid restricted master heuristic that combines column generation with a greedy randomized adaptive search procedure (GRASP) to improve solution quality in these settings. The method first uses dynamic programming-based pricing to generate routes guided by dual variables from the linear relaxation. It then applies GRASP to complete partial solutions and introduce new routes that complement those with positive reduced costs, thereby improving column complementarity before solving the resulting model as a mixed-integer program. This integration leverages both the global guidance of column generation and the diversification strength of GRASP. We illustrate the approach on the vehicle routing problem with stochastic demands, where customer demands are modeled as random variables revealed upon arrival. Experimental results on 40 benchmark instances with up to 60 customers show that the method reliably produces high-quality solutions, achieving optimality gaps below 5% in all cases and averaging under 2% within 5 minutes of computation, with a higher proportion of best solutions returned than any other variant considered.

Keywords : Vehicle routing; stochastic demands; restricted master heuristic; GRASP

1 Introduction

Vehicle routing problems (VRPs) have been extensively studied due to their broad practical relevance (see, e.g., Mor and Speranza, 2022). In a typical VRP, a fleet of vehicles departs from a central depot to serve customers with known demands. Each vehicle has a limited capacity and must return to the depot before exceeding it. Various extensions to the classical VRP have been proposed to better capture real-world complexities.

Path-flow models offer a compact and powerful formulation by associating a binary variable with each feasible route (see Gauvin et al., 2014; Florio et al., 2023). Since the number of such routes grows exponentially, these models are typically solved using column generation, which iteratively adds new variables (columns) to a restricted master problem (RMP), i.e., a relaxed version of the full model. At each iteration, a pricing problem—guided by the dual variables of the RMP—is solved, often via dynamic programming. While column generation excels at solving the linear relaxation, it seldom produces good integer solutions directly.

To address this, column generation can be embedded in exact methods such as branch-price-and-cut (BPC) algorithms (see Costa et al., 2019), which are computationally intensive, or in matheuristics like restricted master heuristics (RMHs) (also known as price-and-branch, see Sadykov et al., 2019). RMHs generate a subset of promising columns, then solve the resulting RMP as a mixed-integer program (MIP) using a commercial solver. However, because column generation is not designed to produce complementary columns, the final MIP solutions are often inferior to those obtained by standalone metaheuristics.

Metaheuristics are generally faster and can produce high-quality solutions but depend heavily on effective guidance. In contrast, column generation provides global, high-level information through dual variables, which can help steer the search. However, column generation often reaches a plateau where dual variables and objective values stabilize, and additional iterations yield diminishing returns. Hence, there is a trade-off: a short column generation phase risks unstable duals, while an extended phase wastes time on marginal improvements to the linear relaxation.

This work aims to bridge the gap between RMHs and metaheuristics by embedding a greedy randomized adaptive search procedure (GRASP) into a column generation heuristic. We focus on the vehicle routing problem with stochastic demands (VRPSD), a well-studied variant where customer demands are modeled probabilistically and revealed upon arrival (Tillman, 1969). This setting allows for *overloads*, i.e., situations where actual demand exceeds a vehicle's remaining capacity.

Following Yee and Golden (1980), we consider a two-stage stochastic programming framework with recourse, where expected costs are minimized under a recourse policy. The most common strategy is reactive: a vehicle returns to the depot only when an overload occurs, although recent work (e.g., Florio et al., 2020) has explored preventive returns.

Our contribution is a fast RMH that uses dynamic programming-based pricing (as in Gauvin et al., 2014) and solves the final RMP as a MIP. To improve column complementarity before MIP resolution, we apply a GRASP-inspired completion algorithm (following Mendoza et al., 2016), which builds complete solutions from the linear relaxation's columns. This reduces the problem size on which GRASP operates, enhancing its efficiency while retaining its speed. The new columns are added to the RMP, which is then solved as an MIP in order to maximize the chances of identifying high-quality integer solutions within limited time.

To our knowledge, this is the first work to enhance column complementarity through post-processing in the context of RMHs. We test our method on 40 VRPSD instances with up to 60 customers and long feasible routes, a scenario where dynamic programming-based pricing is less efficient.

The paper is structured as follows. Section 2 reviews relevant literature. Section 3 defines the VRPSD mathematically. Section 4 describes the column generation approach. Section 5 presents the

GRASP completion strategy and its integration in Section 6. Section 7 reports experimental results, and Section 8 concludes.

2 Literature review

This section presents the literature relevant to our work. We first present an overview of classical RMHs for routing problems in Section 2.1, followed by a review of key methods for specifically solving the VRPSD in Section 2.2.

2.1 Restricted master heuristics for routing problems

Restricted master heuristics are commonly used to obtain feasible solutions without an optimality guarantee while keeping column generation in the process. Recent works focusing on variants of routing problems can adopt the basic RMH approach by generating columns and solving the resulting RMP as a MIP. For example, He et al. (2023) study an aircraft routing problem with maintenance tasks. They solved it using column generation, exploiting two dominance strategies during the pricing, and solving the last RMP as a MIP. However, their approach can only find a solution for small problems. For this reason, the RMH is almost always combined with an additional optimization process. As mentioned by Sadykov et al. (2019) in their study of diving methods, a classical sub-MIP approach is inefficient in obtaining good feasible solutions rapidly. Among all the methods they cover, it was the worst regarding computation time and solution quality. The approach calls for an additional method to account for feasibility.

Sadykov et al. (2019) combined their RMH with a diving heuristic, highly improving the method's performance: a feasible initial solution is generated by rounding the values of the columns in the RMP and heuristically exploring the resulting branching nodes. Later, Bianchessi et al. (2022) introduced a RMH for the close-enough-arc routing problem. They embed several MIP resolutions in a branch-and-price algorithm that aims to provide an upper bound at each branching node. They can also solve the RMP as a MIP if the algorithm has performed a sufficiently large number of column generation iterations since the last resolution. Finally, an iterated local search heuristic generates additional upper bounds. This method allows them to reach an average 1.7% optimality gap in 40 minutes for instances with 50 to 196 customers. More recently, Petris et al. (2024) considered the commodity-constrained split delivery vehicle routing problem and proposed a RMH featuring an additional local search performed on the solution returned by the MIP. For instances of 40 to 60 customers, they achieve an optimality gap of 2 to 5% depending on the set in an average of 15 minutes.

These results underline the importance of combining the RMH with an external process since column generation is not designed to produce complementary columns that can yield a feasible solution. This addition can come as a repair procedure afterward as in Petris et al. (2024) or intervene during the branch-and-price to reduce the size of the search tree as in Bianchessi et al. (2022) and Sadykov et al. (2019). However, repairing the solution obtained from the MIP appears less promising. Due to the lack of compatibility between columns, the MIP solution discards nearly all of the best columns from the RMP and builds its solution around only one or two. Petris et al. (2024) successfully adopt a local search applied to a single solution, which allows for a deeper search but starts from a less accurate upper bound. We therefore favor the first approach, which builds a solution based on each column of the RMP solution using an efficient and fast method. This enables us to retain the full potential of these high-quality columns, solve the MIP more quickly thanks to warm starting, and keep open the possibility that the few additional columns generated this way may lead to an even better combination, further improving the solution.

2.2 Solving methods for the VRPSD

Christiansen and Lysgaard (2007) developed the first exact branch-and-price algorithm for the VRPSD with a reactive restocking policy. Later, Gauvin et al. (2014) built on this work, proposing an exact dynamic programming algorithm and a tabu search heuristic to solve the pricing problem while applying valid inequalities to yield a BPC algorithm. They considered a set of 40 instances from Christiansen and Lysgaard (2007) with up to 60 clients and solved 38 of them in less than 20 minutes, twice the number previously solved to optimality. For the VRPSD with a preventive (or optimal) restocking policy, Florio et al. (2020) developed a BPC algorithm that incorporates stochastic dynamic programming during the pricing phase, which has proven to be highly efficient for instances with a large vehicle-to-customer ratio (i.e., with short routes). They considered instances from Uchoa et al. (2017) and found the optimal solution for instances with up to 76 nodes in less than 5 hours. More recently, Florio et al. (2022) allowed to reorder pairs of consecutive customers on a route under a preventive policy and developed a BPC algorithm for this strategy. For their tests, they considered 21 instances with up to 50 customers from Uchoa et al. (2017) and nine different load factors and demand distributions for each instance to yield 189 instances. They solved 90 of them within a 2-hour time limit. Florio et al. (2023) provide a recent survey on the VRPSD, where they also introduced a BPC algorithm for the case with an optimal restocking policy capable of handling demand correlation. Their method outperforms the one by Florio et al. (2020): on the same set of 45 instances, they optimally solve 8 additional instances and reduce the computational time by more than 60%. Finally, Hoogendoorn and Spliet (2025) evaluate several classical modeling choices for the VRPSD, namely, limiting the total expected demand in a route and imposing a fixed number of routes. They show that the worst-case scenario for adding both constraints results in an arbitrarily large increase of the cost. However, their empirical results show that adding the capacity constraint on the expected demand results in a significant computational benefit while keeping the increase of the objective value marginal. For this reason, we choose to include this capacity constraint.

Mendoza et al. (2011) explored metaheuristics for the VRPSD and introduced two constructive heuristics for the VRPSD with multiple compartments: a stochastic Clarke-and-Wright heuristic, a greedy procedure to merge routes greedily, and a two-phase approach that first builds a tour using several constructive heuristics and then optimally divides it into clusters while accounting for stochasticity. They achieved an average gap of around 5% to the best-known solution with a computational time of approximately one minute for instances with up to 200 customers. Later, Mendoza et al. (2016) introduced a GRASP with heuristic concentration for the VRPSD with duration constraints. They first construct a giant tour, split it into feasible routes, and then apply a variable neighborhood descent, repeating this to generate numerous candidate routes. They obtain the final solution by solving a set covering problem. Tested on 40 instances with up to 60 customers, their heuristic computed the best-known solution in at least one of 10 runs, averaging 1 minute per run.

3 Problem statement and mathematical formulation

The VRPSD can be defined as follows. Let $\mathcal{N}$ be a set of n_C customers to serve using at least K identical vehicles of capacity Q housed in a single depot, denoted 0. Each customer must be served by exactly one vehicle. As in Christiansen and Lysgaard (2007), Gauvin et al. (2014) and Florio et al. (2023), we assume that the demand of each customer $j \in \mathcal{N}$ is uncertain and follows a Poisson distribution of known parameter μ_j. The customer demands are considered independent. This allows to compute the probability $P(OV \mid j, k, n)$ that the n^{th} overload occurs (denoted by OV) when a vehicle reaches customer j with an expected load k:

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

To handle overloads, we adopt the following detour-to-depot policy (see, e.g., Gauvin et al., 2014), and design the vehicle routes *a priori*. The demand of a customer is observed when a vehicle reaches it during operations. If it yields an overload, the vehicle makes a roundtrip to the depot to fully replenish before completing its route as planned.

A vehicle route $r = (j_0^r, j_1^r, j_2^r, \ldots, j_{|r|}^r)$ starts and ends at depot $j_0^r = j_{|r|}^r = 0$ and visits an ordered list of $|r| - 1$ customers $\mathcal{V}_r = (j_1^r, j_2^r, \ldots, j_{|r|-1}^r)$ with $|r| \geq 2$. It is feasible if it respects the following constraints:

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

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

The capacity constraint (2) ensures that the expected demand serviced along a route does not exceed the vehicle capacity. As in several previous works (see, e.g., Gauvin et al., 2014), it is imposed to limit the length of the routes and the number of potential overloads. The elementarity constraints (3) forbid visiting the same customer more than once in a route. We denote the set of all feasible routes by $\mathcal{R}$. Let $d_{i,j}$ be the traveled distance between any pair of distinct locations i and j in $\mathcal{N} \cup \{0\}$. For each customer, we introduce the failure penalty $p_{j,k,n}$ for the n^{th} overload to occur when 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. The cost c_r of a route r is then given by its total expected traveled distance, where $\kappa_l^r = \sum_{i=1}^{l-1} \mu_{j_i^r}$ is the expected load with which a vehicle following route r reaches the l^{th} customer of the route:

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

In practice, we only compute the probability for the n^{th} overload to occur if it is greater than 10^{-4}. The VRPSD consists in finding at least K feasible vehicle routes such that each customer in $\mathcal{N}$ is visited exactly once while minimizing the sum of the costs of the selected routes. We use a path-flow formulation that relies on the following additional notation. For each route $r \in \mathcal{R}$ and customer $j \in \mathcal{N}$, let $\nu_{j,r}$ be a binary parameter equal to 1 if route r visits customer j and 0 otherwise. Furthermore, for each route $r \in \mathcal{R}$, we define a binary variable x_r that takes value 1 if route r is selected in the solution and 0 otherwise. This gives the following integer program:

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

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

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

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

The objective function (6) minimizes the total expected cost of the solution. Constraints (7) ensure that each customer is visited exactly once, whereas constraint (8) requires the solution to use all available vehicles. Finally, constraints (9) impose binary requirements on the route variables. Figure 1 provides a representation of a toy example of a VRPSD instance along with a feasible solution for this instance.

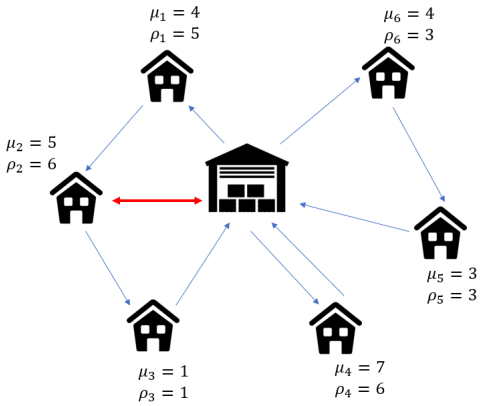

Figure 1: A toy example of a VRPSD instance with 3 vehicles of capacity 10 and a possible solution. μ_j is the expected demand of customer j while ρ_j is its actual realization, only known upon visiting this customer. An overload occurs at customer 2 triggering a back-and-forth return to the depot before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

4 Column generation

Column generation is an iterative algorithm used to solve the linear relaxation of a large-scale MIP that involves an exponential number of variables like model (6)–(9). In this context, the linear relaxation is called the *master problem*. At each iteration, the algorithm starts by solving the master problem restricted to a subset $\hat{\mathcal{R}}$ of its variables, i.e., a RMP, providing a primal solution and a dual solution $\big((\gamma_j)_{j\in\mathcal{N}}, \gamma_0\big)$, where $(\gamma_j)_{j\in\mathcal{N}}$ and γ_0 are the dual variables associated with constraints (7) and (8), respectively. This primal solution is also optimal for the master problem if there are no unknown variables x_r, $r \in \mathcal{R} \setminus \hat{\mathcal{R}}$, with a negative reduced cost $\bar{c}_r$, where this reduced cost $\bar{c}_r$ expresses as:

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

The algorithm then solves the following pricing problem:

$$\min_{r\in\mathcal{R}} \bar{c}_r, \tag{11}$$

which considers the whole set $\mathcal{R}$ (which is represented implicitly) and aims at determining if there exist negative reduced cost variables (columns). If the pricing problem returns one or several negative reduced cost variables, it adds them to the RMP, and a new iteration begins. Otherwise, it proves that the current RMP solution is optimal for the master problem, and the algorithm stops.

For the VRPSD, the pricing problem can be formulated as an elementary shortest path problem with resource constraints (ESPPRC) (see, e.g., Costa et al., 2019) on an acyclic network $G = (V, A)$ proposed by Christiansen and Lysgaard (2007), where V and A are its node and arc sets, respectively. Node set V contains a pair of source and sink nodes $\langle 0, 0\rangle$ and $\langle 0, Q\rangle$ representing the depot, and nodes $\langle j, k\rangle$ where $j \in \mathcal{N}$ is a customer and $k \in [0, Q - \mu_j]$ an expected cumulative demand along a feasible route before reaching j. Arc set A is divided into three disjoint arc subsets: $A = A^1 \cup A^2 \cup A^3$, where $A^1 = \{\big(\langle 0, 0\rangle, \langle j, 0\rangle\big) \,|\, j \in \mathcal{N}\}$ contains the arcs leaving the source node, $A^2 = \{\big(\langle j, k\rangle, \langle i, k+\mu_j\rangle\big) \,|\, j, i \in \mathcal{N}, j \neq i, k \in [0, Q - \mu_j]\}$ the arcs linking two customer nodes, and $A^3 = \{\big(\langle j, k\rangle, \langle 0, Q\rangle\big) \mid j \in \mathcal{N}, k \in [0, Q - \mu_j]\}$ the arcs entering the sink node. To ensure that the cost of a source-to-sink path is the reduced cost of its corresponding route variable, the ESPPRC considers the following adjusted arc costs:

$$\bar{c}_a = \begin{cases} d_{0,j} + \sum_{n=1}^{\infty} p_{j,0,n} & \text{if } a = \big(\langle 0, 0\rangle, \langle j, 0\rangle\big) \in A^1 \\ d_{j,i} + \sum_{n=1}^{\infty} \big(p_{i,k+\mu_j,n} - p_{j,k,n}\big) - \gamma_i & \text{if } a = \big(\langle j, k\rangle, \langle i, k+\mu_j\rangle\big) \in A^2 \\ d_{j,0} & \text{if } a = \big(\langle j, k\rangle, \langle 0, Q\rangle\big) \in A^3. \end{cases} \tag{12}$$

Each feasible route $r \in \mathcal{R}$ corresponds to a source-to-sink path in G. However, some paths may represent infeasible routes as they can violate the elementarity constraints (3). Indeed, even if G is acyclic, a customer is represented by multiple nodes (with different expected cumulative demands). To impose that at most one of these nodes must be included in a feasible path, resource constraints, one per customer, are considered in the ESPPRC pricing problem. Given that the ESPPRC is NP-hard, we rather use, as in Gauvin et al. (2014), the well-known ng-path relaxation (ng-ESPPRC Baldacci et al., 2011) that allows the generation of routes that may contain some cycles. The ng-ESPPRC is usually solved by a labeling algorithm (see, e.g., Costa et al., 2019). In our case, we rely on the bidirectional labeling algorithm of Gauvin et al. (2014). Finally, to strengthen the master problem, capacity and subset row inequalities are added as cutting planes.

5 GRASP completion algorithm

The column generation approach described in Section 4 comes with the weakness that it does not aim to generate *complementary columns*, i.e., routes that can be combined to form an integer solution. For example, if two routes visit a customer, they cannot be both used in an integer solution due to covering constraints (7). The consequence is a lack of feasible solutions that can be obtained by solving the RMP as a MIP. To help finding integer solutions, we propose to add a post-processing phase that searches for additional columns that are complementary with the ones used in the last solution of the RMP. More precisely, for each route used in the final RMP solution, we apply a heuristic to build a complete solution containing this route and add to the RMP all columns of this solution that are not yet in the RMP. In this way, when solving the MIP resulting from the RMP, we are guaranteed to find a solution which is at least as good as the best solution found by the heuristic, and possibly a better one if the whole set of columns in the RMP allows it. For the rest of this section, we denote by

$$\mathcal{C}_s = \sum_{r \in s} c_r \tag{13}$$

the cost of a solution s, which is computed as the sum of the expected cost of its routes.

Let us describe the heuristic used to compute a complete solution for each route in the RMP solution. First, this route is set apart and the problem is reduced by removing the customers covered by this route. The heuristic is then applied to construct a partial solution covering all remaining customers. We have reimplemented the GRASP from the work of Mendoza et al. (2016). This type of method makes several runs of a two-phase process: in the first phase, an initial solution is built using a randomized weighted construction heuristic; in the second phase, a local search heuristic tries to improve this initial solution. Let $\mathcal{V}$ be the set of customers to be visited by a tour τ, $\mathcal{V}^{in}$ the set of customers it already visits and $\mathcal{E}_\tau$ be the set of its edges.

The construction phase starts by building a giant tour τ visiting all the customers that have to be visited by the partial solution through procedure GENERATE_TOUR(.), using one of four construction heuristics, namely *random nearest neighbor* (RNN), *random nearest insertion* (RNI), *random farthest insertion* (RFI) and *random best insertion* (RBI). The set of construction heuristics is denoted by $\mathcal{H}$. Each heuristic $h \in \mathcal{H}$ takes an integer L_h as a parameter defining the neighborhood's size to be considered during the construction. Random nearest neighbor is initialized by a tour only containing the depot and is completed by returning to the depot once all customers have been visited. All other heuristics are initialized by doing a back-and-forth from the depot to a random customer in $\mathcal{V}$. The heuristics add customers to the tour iteratively, one customer at a time, and stop when all customers are inserted. To do so, et each step, they sample a random integer l in the range $[1, \ldots, \min\{L_h, |\mathcal{V} \backslash \mathcal{V}^{in}|\}]$ and adds the l^{th} nearest customer according to a metric depending on the heuristic. The metrics are as follows.

RNN: The distance from the last customer of the tour.
RNI: $d_{min}(v) = \min\{d_{u,v} | u \in \mathcal{V}^{in}\}$.

RFI: $d_{max}(v) = \max\{d_{u,v} | u \in \mathcal{V}^{in}\}$
RBI: $d_{best}(v) = \min\{d_{u,v} + d_{v,w} - d_{u,w} | (u,w) \in \mathcal{E}_\tau\}$.

The resulting tour $\tau = \{0, j_1, \ldots, j_{|\mathcal{V}|}\}$ is then optimally split into routes using procedure Split(.) by solving the shortest path problem in the following acyclic directed graph containing a vertex for each customer in $\mathcal{V}$ except the depot.

An arc exists between two vertices j_s and j_t if $s < t$ and $\sum_{h=1}^{t-s} \mu_{j_{s+h}} \leq Q$. Its weight is equal to the cost of route $r = (0, j_{s+1}, \ldots, j_{t-1}, j_t, 0)$. We then add the source node 0 representing the starting point at the depot and add an arc going from the depot to each vertex in the graph using the same method. We then solve the shortest path problem in this graph using Dijkstra's algorithm between the source node 0 and the sink node $j_{|\mathcal{V}|}$. The nodes at the end of each arc selected in the solution are the ones at which it is optimal to make a detour to the depot without changing the order of customer visits determined by τ.

Finally we further improve the routes returned by the splitting procedure by applying a local search using two types of moves, namely, relocate (procedure Relocate(.)) and 2-opt (procedure 2opt(.)). We adopt the cost of the solution $\mathcal{C}_s$ as the evaluation function. We begin with relocate moves. For each route of the solution and then for each customer in the route, we try to move it to a different route while respecting the capacity constraint (2). We consider each possible insertion and perform the first improvement move found and return the corresponding solution. If no improving move is found, we return the input solution. In both cases, it also returns a boolean indicating if an improving move was found or not. The order in which the routes are considered and the order in which we choose the customer to try to relocate next are random. When we do not find any improving relocate move, we switch to a 2-opt neighborhood. For each route, we consider each pair of customers it visits and reverse the portion of the route between these two customers. Performing the 2-opt move on the i_1^{th} and i_2^{th} customers of route

$$r = \{j_0^r, \ldots, j_{i_1-1}^r, j_{i_1}^r, j_{i_1+1}^r, \ldots, j_{i_2-1}^r, j_{i_2}^r, j_{i_2+1}^r, \ldots, j_{|r|}^r\}$$

then yields the route

$$r' = \{j_0^r, \ldots, j_{i_1-1}^r, j_{i_2}^r, j_{i_2-1}^r, \ldots, j_{i_1+1}^r, j_{i_1}^r, j_{i_2+1}^r, \ldots, j_{|r|}^r\}$$

We consider all possible moves but only perform the first improving one found, returning the resulting solution. As before, if no improving move was found, the input solution is returned and it also returns a boolean indicating if an improving move was found or not. The order in which the routes are considered is also random.

This process is repeated until the allocated time has expired. Four partial solutions are constructed during each run, one per construction heuristic. At the end, only the best of the partial solutions considered is returned.

Algorithm 1 provides the pseudo-code for our GRASP completion algorithm, denoting a solution by its set of routes. The algorithm initializes the best solution s^* and its corresponding best cost $\mathcal{C}^*$ to default values, and computes the customers to visit at lines 2 to 4. Line 5 checks for the stopping criterion (the function get_time(.) here returns the current CPU time). Then, using each construction heuristic $h \in \mathcal{H}$ iteratively, the algorithm generates a giant tour at line 7, which is then converted into a partial solution at line 8. Lines 9 to 16 apply the local search to the resulting routes. Next, lines 17 to 19 check for improvement and update the best partial solution found, if required. The best partial solution is then returned at the end at line 21.

6 Restricted master heuristic

We combine the mechanisms described in Sections 4 and 5 into our RMH. We generate columns using the column generation algorithm described in Section 4 for a given time. Then, we apply the GRASP

Algorithm 1 GRASP completion procedure

Require: r is the route to complete
Require: t is the allocated computation time
Require: $\mathcal{N}$ is the set of customers
Require: $\mathcal{V}_r$ is the set of customers visited by route r
Require: $\mathcal{H}$ the set of construction heuristics
Output: s^* is the best computed solution and C_{s^*} is its cost

1: **procedure** GRASP(r, t)
2: $s^* \leftarrow \{\,\}$
3: $\mathcal{C}^* \leftarrow \infty$
4: $\mathcal{V} \leftarrow \mathcal{N} \backslash \mathcal{V}_r$
5: **while** GET_TIME() $\leq t$ **do**
6: **for** $h \in \mathcal{H}$ **do**
7: $\tau \leftarrow$ GENERATE_TOUR$(\mathcal{V}, h)$
8: $s \leftarrow$ SPLIT(τ)
9: $s' \leftarrow$ RELOCATE(s)
10: **while** $\mathcal{C}_{s'} < \mathcal{C}_s$ **do**
11: $s \leftarrow s'$
12: $s' \leftarrow$ RELOCATE(s)
13: $s' \leftarrow$ 2OPT(s)
14: **while** $\mathcal{C}_{s'} < \mathcal{C}_s$ **do**
15: $s \leftarrow s'$
16: $s' \leftarrow$ 2OPT(s)
17: **if** $\mathcal{C}_{s'} < \mathcal{C}^*$ **then**
18: $\mathcal{C}^* \leftarrow \mathcal{C}_{s'}$
19: $s^* \leftarrow s'$
20: $s^* \leftarrow s^* \cup \{r\}$
21: **return** s^*, C_{s^*}

to complete each column of positive value and add all new columns generated this way in the RMP, that we solve as a MIP, warm-started with the best solution obtained. The pseudo-code for this procedure is provided in Algorithm 2. It takes four parameters as input: the initial pool $\mathcal{P}$ in the RMP, the maximal computation time t_C for the column generation phase, the allocated time t_G to spend on the GRASP completion phase and the maximal time t_M for the MIP resolution phase.

We initialize the best solution and its cost to default values at lines 2 and 3. We then perform the column generation phase from line 4 to 9. At each iteration, we check the timeout criterion at line 4 then solve the RMP at line 5 resulting in a solution s_L of the RMP and the dual variables π_L. We obtain a pool of routes $\mathcal{P}^*$ to add to the RMP by solving the pricing problem using the labeling algorithm of Gauvin et al. (2014) at line 6. The column generation phase ends if the pool is empty, which is checked for at line 7. If not, we update the RMP and iterate at line 9.

We perform the GRASP completion phase at lines 10 to 16. We start by equally dividing the time dedicated to the GRASP completion phase between all columns of positive value at line 10. We iterate over each column of positive value in the RMP at line 11. We apply the GRASP completion algorithm to the current column at line 12, resulting in a solution s. We update the best solution, cost, and the RMP at lines 13 to 16. Finally, we solve the RMP as a MIP with procedure SOLVE_MIP(.) at line 17 using a commercial solver, warm-started with the best solution s^* found during the GRASP completion and a timeout of t_M.

7 Computational results

Since the instances used in Gauvin et al. (2014) are almost all optimally solved in less than 20 minutes, we aim to evaluate our method on harder instances. We derive 40 instances featuring 34 to 60 customers from those considered by Gauvin et al. (2014): the demands and distances between customers are the same. However, we reduce the fleet size and proportionally increase the capacity, keeping the KQ product constant. The resulting instances feature longer possible routes for which the dynamic programming algorithm struggles to solve the pricing problem efficiently. The method by Gauvin

Algorithm 2 Restricted master heuristic with GRASP completion.

Require: $\mathcal{P}$ an initial pool of columns in the RMP
Require: t_C the time spent in column generation
Require: t_G the time spent for the GRASP completion
Require: t_M the time spent in the MIP resolution phase
Output: s^* is the best computed solution

procedure RMH($\mathcal{P}, t_C, t_G, t_M$)
  $s^* \leftarrow \{\ \}$
  $\mathcal{C}^* \leftarrow \infty$
  while GET_TIME() $\leq t_C$ **do**
    $(s_L, \pi_L) \leftarrow$ SOLVE_RMP($\mathcal{P}$)
    $\mathcal{P}^* \leftarrow$ SOLVE_PRICING(π_L)
    if $|\mathcal{P}^*| = 0$ **then**
      break
    $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}^*$
  $t \leftarrow \frac{t_G}{|s_L|}$
  for $r \in s_L$ **do**
    $s, \mathcal{C}_s \leftarrow$ GRASP(r, t)
    if $\mathcal{C}_s \leq \mathcal{C}^*$ **then**
      $s^* \leftarrow s$
      $\mathcal{C}^* \leftarrow \mathcal{C}_s$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{r_G | r_G \in s \backslash \{r\}\}$
  $s^* \leftarrow$ SOLVE_MIP($\mathcal{P}, s^*, t_M$)
  return s^*

et al. (2014) does not reach optimality for these instances after one hour. These instances are available online.[1] Regarding the parameter L_h of each heuristic, we tried several values (between 3 and 15) without getting a significant difference in computation time or solution quality. We thus decided to use the same values as Mendoza et al. (2016), namely $L_{RNN} = 3, L_{RNI} = L_{RFI} = L_{RBI} = 6$. One could also question the relevance of using all four heuristics. The same tests showed that even when considering various values for parameters L_h, each heuristic manages to outperform the others for some instances.

We perform the column generation phase using Gencol 4.5. All tests were conducted on an Intel(R) Core(TM) $i7-8700$ 3.20GHz CPU.

In this section, we compare seven solution methods:

cg+grasp+mip (our contribution) : The column generation method of Gauvin et al. (2014) with our GRASP completion phase and final MIP resolution, as described in Section 6.

cg+mip (Gauvin et al.) : The column generation method of Gauvin et al. (2014) using a dynamic programming-based pricing with final MIP resolution. We let the algorithm generate columns for a predefined time and then solve the RMP as a MIP with a predefined timeout.

cg+mip (Florio et al.): The column generation method implemented by Florio et al. (2023) in its configuration using the capacity constraint, with an additional final MIP resolution. It is important to note that this method does not rely on Gencol to work, which causes a significant decrease in performance compared to method CG+MIP (Gauvin et al.).

cg+grasp : The column generation method of Gauvin et al. (2014) with our GRASP completion phase. We generate columns for a given amount of time. The time dedicated to the GRASP completion is equally divided between all columns of positive value in the RMP solution.

grasp: The GRASP described in Section 5 applied to the whole problem (i.e. $r = \emptyset$).

grasp+mip: Same as method GRASP but with an additional MIP resolution considering all routes generated with the GRASP, warm-started with the best solution computed during the GRASP phase. Since it is a reimplementation of the one described by Mendoza et al. (2016), we tested it

[1] https://github.com/GaelReynal/RMH_DP_GRASP.git

on the same set of instances of the VRPSD as them to ensure the quality of our implementation and achieved the same results.

cg+mip_1h : Same as method CG+MIP (Gauvin et al.) but with a 1-hour column generation phase and an unlimited MIP resolution. We also use the best lower bound found during the column generation phase as a reference to compute our optimality gaps.

We test the methods using four timeout criteria. The computation times allocated to each phase, for each method, and each timeout are given in Table 1.

Table 1: Computation times allocated for each phase and method.

Method	Phase	Total time			
		2 minutes	5 minutes	10 minutes	15 minutes
CG+GRASP+MIP	CG	60s	180s	300s	420s
	GRASP	50s	90s	240s	390s
	MIP	10s	30s	60s	90s
CG+MIP (Gauvin et al.)	CG	110s	270s	540s	810s
	MIP	10s	30s	60s	90s
CG+MIP (Florio et al.)	CG	110s	270s	540s	810s
	MIP	10s	30s	60s	90s
CG+GRASP	CG	60s	180s	300s	420s
	GRASP	60s	120s	300s	480s
GRASP	GRASP	120s	300s	600s	900s
GRASP+MIP	GRASP	110s	270s	540s	810s
	MIP	10s	30s	60s	90s

Concerning the MIP resolution, due to the variance in computation time between the instances, the difficulty in predicting how long the MIP will take to solve, and the fact that its resolution always comes last in the process, we selected its timeout, taking two considerations into account. The first one is that the MIP resolution must be long enough to provide relevant solutions. On the other hand, if too much time is left for the MIP and the resolution only takes a small part of that time, it creates a difference between the methods regarding computation time. Our timeouts result in a majority of instances having their corresponding MIPs solved to optimality while keeping the differences in computation time in the case of a fast MIP resolution small enough to be considered negligible (at most 10% of the total execution time).

We provide detailed results concerning the performance of each method in Tables 2 to 6. Table 2 gives information regarding the number of best solutions obtained by each method, and the number of instances under a given optimality gap for each computation time. Tables 3 to 6 provide the exact value of the solution obtained by each method, each table corresponding to a different computation time. In all tables, bold numbers allow for quick identification of the best method (or methods if tied) in each line.

We also compare our methods using performance profiles. To obtain these figures, we first consider a set $\mathcal{M}$ of methods to compare on a set $\mathcal{I}$ of instances. We then compute the ratio $r_{i,m}$ for each method $m \in \mathcal{M}$ on each instance $i \in \mathcal{I}$. Let $\mathcal{C}_{s_{i,m}} > 0$ be the cost of the solution $s_{i,m}$ obtained by method m on instance i, then :

$$r_{i,m} = \frac{\mathcal{C}_{s_{i,m}}}{\min\limits_{m' \in \mathcal{M}} \mathcal{C}_{s_{i,m'}}} \quad \forall m \in \mathcal{M},\ i \in \mathcal{I}$$

$r_{i,m}$ is the ratio between the performance of method m on instance i and the performance of the best method among those considered on the same instance (as a consequence, $r_{i,m} \geq 1$). We now construct one curve for each method with the performance ratio on the x-axis and the proportion of instances

on the y-axis: a point (r, p) is part of the curve representing a method m if given method achieves a ratio $r_{i,m}$ of at most r for a proportion p of the instances in the dataset. Therefore, the more a curve is in the top left corner, the better the method it represents performs compared to the others. In our case, it means the method manages to find solutions of lower cost than the other methods represented in the figure.

In the rest of this section, we perform an ablation study on the proposed RMH to estimate the impact of each of our components in Section 7.1 before comparing this RMH to the other solution methods in Section 7.2.

Table 2: Average optimality gap and number of instances solved under certain gaps for all methods and computation times.

	Instance	CG +GRASP +MIP	CG +GRASP	CG+MIP (Gauvin et al.)	GRASP	GRASP +MIP	CG+MIP (Florio et al.)	CG +MIP _1H
2 minutes	Avg optimality gap	2.35%	2.50%	19.46%	2.34%	**1.94%**	43.73%	5.52%
	# best solutions	**29**	24	3	11	19	0	13
	# < 1% optimality gap	**13**	11	3	10	12	0	10
	# < 5% optimality gap	36	35	9	39	**40**	2	25
5 minutes	Avg optimality gap	**1.76%**	1.91%	10.25%	2.14%	1.81%	30.03%	5.52%
	# best solutions	**36**	27	5	17	28	0	13
	# < 1% optimality gap	14	13	4	12	**15**	0	10
	# < 5% optimality gap	**40**	**40**	16	39	**40**	5	25
10 minutes	Avg optimality gap	**1.71%**	1.78%	7.38%	1.84%	1.75%	27.47%	5.52%
	# best solutions	**35**	29	8	18	30	0	12
	# < 1% optimality gap	15	15	6	13	**16**	0	10
	# < 5% optimality gap	**40**	**40**	20	**40**	**40**	5	25
15 minutes	Avg optimality gap	1.71%	1.74%	6.90%	1.76%	**1.69%**	23.16%	5.52%
	# best solutions	32	29	8	21	**37**	0	11
	# < 1% optimality gap	15	15	7	**16**	**16**	0	10
	# < 5% optimality gap	**40**	**40**	22	**40**	**40**	6	25

7.1 Ablation study

In this section, we perform an ablation study of our core method to stress the impact of each component. We compare CG+GRASP+MIP to CG+GRASP, GRASP+MIP, and GRASP. Since the ablation of the GRASP component from our method CG+GRASP+MIP yields the method CG+MIP (Gauvin et al.) which is a standard metaheuristic, the comparison between these two methods is provided in Section 7.2. The performance profiles related to these experiences are provided in Figure 2. The performance ratio is computed using the cost of the solutions obtained by the four methods considered.

We can first observe that our hybrid approach reaches the best solution of the four methods much more often than the GRASP+MIP method (10 more for the 2 minutes case, 8 more for the 5 minutes case), which makes CG+GRASP+MIP the best choice, especially with low computation times, if one

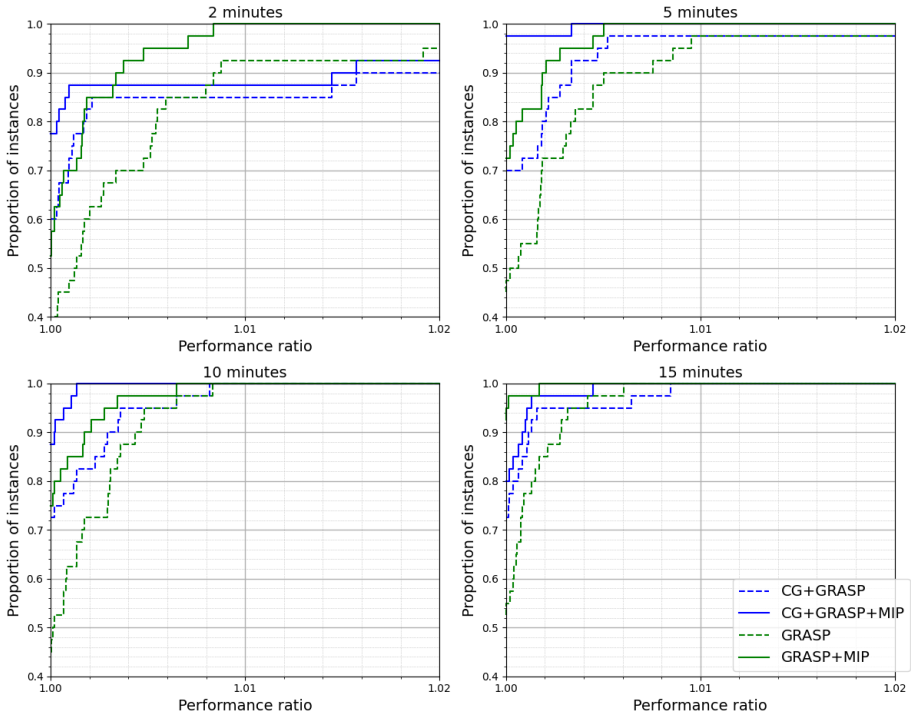

Figure 2: Performance profiles on cost for methods cg+grasp+mip, cg+grasp, grasp and grasp+mip.

wishes to achieve a high-quality feasible solution quickly. However, the 2-minute case also shows that for 10% of the instances (the hardest in our set), the column generation phase is too short to provide enough good columns to guide the GRASP completion, resulting in an optimality gap over 5%. In contrast, the GRASP+MIP method reaches better results for these specific instances. The 15-minute criterion stresses a major issue of column generation: its difficulty in improving the solution after the dual variables have converged. Compared to the 10-minute case, the 2 additional minutes of column generation do not result in a significant improvement in terms of solution quality: we observe no change in the average optimality gap and in more than 75% of the cases, the same solution is obtained for the 10-minute and the 15-minute computation time. On the other hand, the extra time for methods GRASP and GRASP+MIP leads to an improvement of these methods with more than a 0.5% reduction of the optimality gap between the 10-minute and the 15-minute computation time.

The MIP phase always results in a significant improvement of the methods, especially for the cases with 5 and 10 minutes: the solutions generated during the GRASP phase and the columns in the RMP can be combined for further improvement, even with a relatively short MIP phase, as in our case. However, it is important to stress that with more computation time allocated to the column generation and/or the GRASP, the impact of the MIP phase becomes marginal: for the 15-minute case, the methods featuring a MIP resolution do not manage to get any additional instances under a 1% optimality gap and reduce the average optimality gap by less than 0.1% compared to their versions without the MIP resolution.

7.2 Comparison to classical restricted master heuristics

In this section, we compare our RMH CG+GRASP+MIP to three classical RMHs: CG+MIP (Gauvin et al.), CG+MIP (Florio et al.) and CG+MIP_1H. The performance profiles related to these experiments are provided in Figure 3. The performance ratio is computed using the cost of the solutions obtained by the four methods considered.

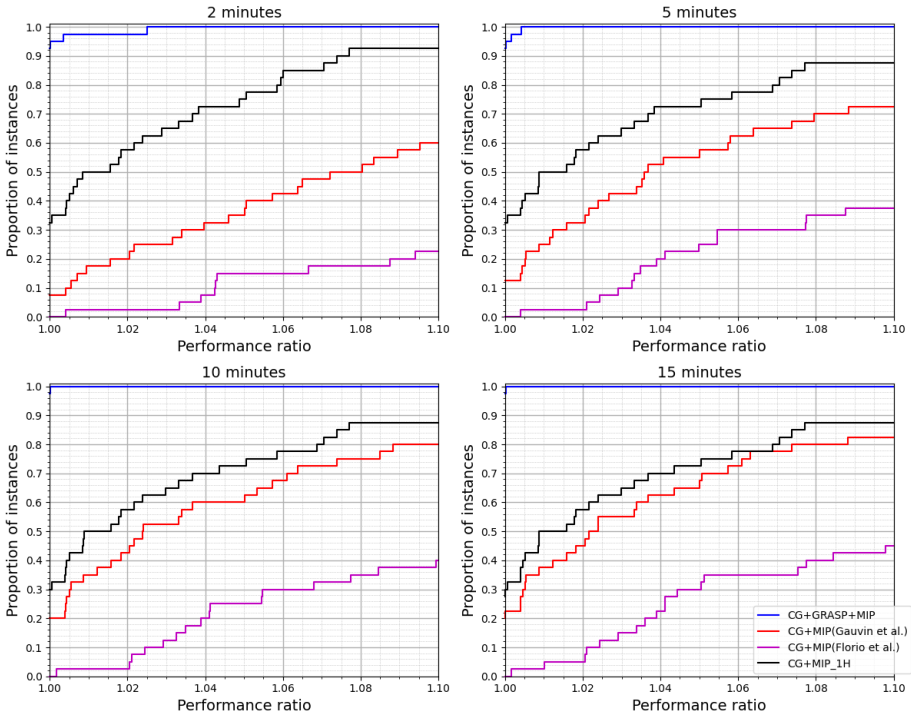

Figure 3: Performance profiles on cost for methods cg+grasp+mip, cg+mip (Gauvin et al.), cg+mip (Florio et al.), and cg+mip_1h.

As can be seen on the performance profiles, methods CG+MIP (Gauvin et al.) and CG+MIP_1H consistently struggle to produce feasible solutions, primarily because they are not designed to generate mutually complementary columns, which are essential for feasibility. Consequently, applying a RMH that solves the RMP as a MIP becomes inefficient, as noted by Sadykov et al. (2019). In contrast, the GRASP rapidly yields feasible and high-quality solutions, thanks to its tour construction heuristics and the splitting procedure, which ensure early feasibility and a powerful local search that substantially enhances solution quality. The final MIP resolution stage further enhances these solutions by combining both high-quality columns dispatched in several solutions obtained during the GRASP and columns in the RMP that were overlooked by the column generation and GRASP phase because they did not contribute to the solution of the RMP.

The combination of the high-quality columns generated via column generation and the complementary columns introduced by the completion phase results in CG+GRASP+MIP delivering the best solutions among all considered RMHs for 37 out of the 40 instances within a 2 and 5-minute timeout and all 40 at the 10 and 15-minute mark. Even in the rare cases where it does not achieve the best-known solution, the gap remains below 1% except for a single instance in the 2-minute case. An overview of the numerical results are available in Table 2. Full results can be found in Tables 3 to 6.

Moreover, when analyzing the average optimality gap, the benefit of the GRASP completion phase becomes clear. Within just 2 minutes, it reduces the gap from 19.46% obtained by a classical RMH to only 2.35%, and from 10.25% to 1.76% in the 5-minute case. In contrast, even method CG+MIP_1H, which runs for a full hour, only narrows the gap to 5.52%. This underscores the critical importance of the completion phase as an effective mechanism for addressing the column complementarity issue inherent in RMHs.

Table 3: cost of the solution obtained by the methods with a 2-minute computation time.

Instance	CG +GRASP +MIP	CG +GRASP	CG+MIP (Gauvin et al.)	GRASP	GRASP +MIP	CG+MIP (Florio et al.)	CG +MIP _1H
A-n39-k2-300	**578.49**	**578.49**	**578.49**	**578.49**	**578.49**	750.96	**578.49**
A-n44-k2-300	658.72	658.72	848.34	**658.43**	**658.43**	979.96	683.95
A-n44-k4-150	**843.33**	**843.33**	851.32	847.37	847.37	879.59	848.42
A-n44-k5-120	**931.97**	**931.97**	935.74	949.84	932.51	935.75	935.74
A-n45-k2-300	**636.63**	**636.63**	678.0	**636.63**	**636.63**	891.95	**636.63**
A-n45-k3-200	**711.11**	**711.11**	**711.11**	771.93	**711.11**	808.96	**711.11**
A-n48-k2-350	655.42	655.42	1027.19	631.73	**628.02**	1306.95	822.73
A-n48-k4-175	831.81	832.73	884.78	831.81	831.81	909.96	**831.67**
A-n53-k2-350	651.97	651.97	894.79	641.88	641.88	1845.95	**636.01**
A-n53-k4-175	**815.02**	**815.02**	840.71	817.74	817.74	1235.95	818.58
A-n53-k5-140	918.13	918.49	1001.96	925.44	924.62	1066.95	**914.84**
A-n55-k2-450	**587.68**	615.83	775.97	**587.68**	**587.68**	1243.95	**587.68**
A-n55-k3-300	**653.92**	**653.92**	687.04	657.45	654.35	945.95	**653.92**
A-n60-k3-300	750.35	750.35	1126.03	739.9	**739.65**	1733.94	786.97
B-n34-k2-250	464.85	464.85	715.57	464.85	**464.42**	816.96	518.51
B-n34-k4-125	**746.25**	746.93	789.03	762.55	746.6	795.91	783.95
B-n34-k5-100	**888.4**	890.27	918.49	892.97	891.73	922.99	895.92
B-n35-k2-250	**565.09**	**565.09**	652.49	**565.09**	**565.09**	781.95	**565.09**
B-n39-k2-250	**366.59**	367.27	368.57	369.66	367.27	569.97	366.8
B-n41-k3-200	**608.7**	**608.7**	699.67	609.71	609.71	693.96	653.64
B-n43-k3-200	**511.38**	**511.38**	579.64	**511.38**	**511.38**	674.97	530.16
B-n43-k4-150	**604.47**	**604.47**	648.07	605.52	604.48	692.53	**604.47**
B-n43-k6-100	**816.21**	**816.21**	**816.21**	818.33	817.63	843.39	**816.21**
B-n44-k4-175	669.3	669.95	854.92	669.06	**668.81**	825.96	740.72
B-n44-k5-140	**780.47**	**780.47**	915.08	781.52	781.52	813.59	799.14
B-n44-k7-100	**1069.74**	1070.99	1159.01	1075.32	1073.14	1115.45	1100.65
B-n45-k2-300	414.49	414.49	420.93	**414.37**	**414.37**	688.97	420.93
B-n45-k3-200	**503.31**	**503.31**	513.68	504.31	503.32	715.97	512.27
B-n45-k4-125	**671.59**	**671.59**	751.08	**671.59**	**671.59**	785.97	**671.59**
B-n45-k4-150	**569.65**	**569.65**	581.96	**569.65**	**569.65**	737.97	581.96
B-n45-k5-120	**673.02**	**673.02**	733.31	678.91	678.65	731.94	720.51
B-n50-k4-175	**549.63**	**549.63**	577.16	**549.63**	**549.63**	781.97	**549.63**
B-n50-k5-140	**612.81**	**612.81**	617.12	612.92	612.92	781.98	617.12
B-n51-k2-350	612.59	612.59	898.27	565.84	**565.15**	1202.96	649.32
B-n51-k4-175	**792.87**	**792.87**	959.94	794.11	794.11	926.96	854.03
B-n51-k5-140	**897.6**	**897.6**	969.71	900.03	899.03	1024.07	913.98
B-n51-k7-100	**1162.94**	**1162.94**	1216.39	1163.01	**1162.94**	1288.83	1168.92
B-n52-k2-350	433.01	434.01	920.0	**415.21**	**415.21**	848.97	458.79
B-n52-k4-175	**577.3**	**577.3**	697.43	577.4	577.4	820.97	611.01
B-n52-k5-140	**643.39**	**643.39**	668.98	646.93	**643.39**	818.97	664.73

8 Conclusion

This work introduces a new RMH designed to efficiently compute high-quality integer solutions within short computation times by focusing on improving column complementarity. The method rapidly identifies promising columns by leveraging dual information through column generation, and finds the complementary columns required to achieve a good feasible integer solution with a GRASP. An additional MIP phase allows for further combination of the generated routes thus refining the solution even more.

We focus on the VRPSD, a variant of the routing problem that has been tackled using either column generation or metaheuristics, and consider instances featuring longer feasible routes, a situation in which the dynamic programming classically used during the pricing phase faces a challenge. Compared to more standard metaheuristics like the GRASP+MIP by Mendoza et al. (2011), the number of best solutions returned by our method is significantly higher, especially with short computation times. It also offers more robustness along with a fast convergence and, more importantly, substantially outperforms classical RMHs, reducing the optimality gap from 10.25% to 1.76% in 5 minutes. This highlights column incompatibility as a critical bottleneck in column generation-based methods and reinforces the importance of solution completion mechanisms.

Table 4: cost of the solution obtained by the methods with a 5-minute computation time

Instance	CG +GRASP +MIP	CG +GRASP	CG+MIP (Gauvin et al.)	GRASP	GRASP +MIP	CG+MIP (Florio et al.)	CG +MIP _1H
A-n39-k2-300	**578.49**	**578.49**	**578.49**	**578.49**	**578.49**	716.95	**578.49**
A-n44-k2-300	658.72	658.72	711.07	658.43	**656.51**	864.96	683.95
A-n44-k4-150	**841.01**	843.33	851.32	847.37	843.33	861.56	848.42
A-n44-k5-120	**931.97**	**931.97**	935.74	934.86	**931.97**	935.75	935.74
A-n45-k2-300	**636.63**	**636.63**	**636.63**	**636.63**	**636.63**	759.95	**636.63**
A-n45-k3-200	**711.11**	**711.11**	**711.11**	771.93	**711.11**	749.96	**711.11**
A-n48-k2-350	**623.4**	644.61	893.09	**623.4**	**623.4**	980.95	822.73
A-n48-k4-175	831.81	831.81	884.78	831.81	831.81	896.09	**831.67**
A-n53-k2-350	638.68	641.68	760.59	641.88	641.88	1507.95	**636.01**
A-n53-k4-175	**815.02**	**815.02**	818.58	817.74	**815.02**	1093.95	818.58
A-n53-k5-140	916.35	918.02	925.3	925.09	918.02	985.52	**914.84**
A-n55-k2-450	**587.68**	590.76	**587.68**	**587.68**	**587.68**	895.94	**587.68**
A-n55-k3-300	**653.92**	**653.92**	669.65	654.35	654.16	853.95	**653.92**
A-n60-k3-300	**736.34**	738.83	1082.88	739.65	739.65	1350.94	786.97
B-n34-k2-250	**464.42**	**464.42**	642.54	**464.42**	**464.42**	762.95	518.51
B-n34-k4-125	**746.25**	**746.25**	789.03	747.59	**746.25**	783.43	783.95
B-n34-k5-100	**888.4**	889.16	918.49	892.37	889.15	922.99	895.92
B-n35-k2-250	**565.09**	**565.09**	639.25	**565.09**	**565.09**	775.96	**565.09**
B-n39-k2-250	**366.59**	367.27	368.57	367.27	367.27	552.97	366.8
B-n41-k3-200	**608.7**	**608.7**	653.64	609.71	**608.7**	693.96	653.64
B-n43-k3-200	**511.38**	**511.38**	530.16	**511.38**	**511.38**	648.97	530.16
B-n43-k4-150	**604.47**	**604.47**	626.08	**604.47**	**604.47**	637.45	**604.47**
B-n43-k6-100	**816.21**	**816.21**	**816.21**	817.63	816.63	843.39	**816.21**
B-n44-k4-175	**668.12**	669.58	773.2	**668.12**	**668.12**	814.96	740.72
B-n44-k5-140	**780.47**	**780.47**	812.28	781.07	**780.47**	812.56	799.14
B-n44-k7-100	**1068.77**	1070.99	1130.75	1072.59	1070.99	1091.23	1100.65
B-n45-k2-300	**414.37**	**414.37**	420.93	**414.37**	**414.37**	650.97	420.93
B-n45-k3-200	**503.31**	**503.31**	513.68	503.32	**503.31**	628.97	512.27
B-n45-k4-125	**671.59**	**671.59**	748.99	**671.59**	**671.59**	746.68	**671.59**
B-n45-k4-150	**569.65**	**569.65**	581.96	**569.65**	**569.65**	717.97	581.96
B-n45-k5-120	**673.02**	**673.02**	732.47	678.79	**673.02**	731.94	720.51
B-n50-k4-175	**549.63**	**549.63**	577.16	**549.63**	**549.63**	649.96	**549.63**
B-n50-k5-140	**611.81**	612.81	617.12	612.92	612.92	631.74	617.12
B-n51-k2-350	**564.24**	**564.24**	649.32	565.15	**564.24**	952.96	649.32
B-n51-k4-175	**792.87**	**792.87**	955.97	**792.87**	**792.87**	816.02	854.03
B-n51-k5-140	**897.6**	**897.6**	921.51	899.03	**897.6**	928.93	913.98
B-n51-k7-100	**1162.94**	**1162.94**	1168.92	**1162.94**	**1162.94**	1288.83	1168.92
B-n52-k2-350	**415.21**	**415.21**	458.79	**415.21**	**415.21**	813.96	458.79
B-n52-k4-175	**577.29**	**577.29**	678.57	577.4	577.4	687.96	611.01
B-n52-k5-140	**643.39**	**643.39**	666.09	**643.39**	**643.39**	796.01	664.73

Table 5: cost of the solution obtained by the methods with a 10-minute computation time

Instance	CG +GRASP +MIP	CG +GRASP	CG+MIP (Gauvin et al.)	GRASP	GRASP +MIP	CG+MIP (Florio et al.)	CG +MIP _1H
A-n39-k2-300	**578.49**	**578.49**	**578.49**	**578.49**	**578.49**	716.95	**578.49**
A-n44-k2-300	**655.38**	**655.38**	711.07	658.43	656.51	860.95	683.95
A-n44-k4-150	**841.01**	843.33	851.32	844.67	843.33	861.56	848.42
A-n44-k5-120	**931.97**	**931.97**	935.74	934.74	**931.97**	933.57	935.74
A-n45-k2-300	**636.63**	**636.63**	**636.63**	**636.63**	**636.63**	759.95	**636.63**
A-n45-k3-200	**711.11**	**711.11**	**711.11**	**711.11**	**711.11**	749.96	**711.11**
A-n48-k2-350	**623.4**	**623.4**	822.73	**623.4**	**623.4**	955.95	822.73
A-n48-k4-175	831.81	831.81	884.78	**831.67**	**831.67**	896.09	**831.67**
A-n53-k2-350	**633.5**	638.68	636.03	635.68	635.68	1507.95	636.01
A-n53-k4-175	**815.02**	**815.02**	818.58	817.49	**815.02**	1093.95	818.58
A-n53-k5-140	**914.84**	918.02	**914.84**	919.23	916.35	976.95	**914.84**
A-n55-k2-450	**587.68**	**587.68**	**587.68**	**587.68**	**587.68**	895.94	**587.68**
A-n55-k3-300	**653.92**	**653.92**	669.65	654.35	**653.92**	853.95	**653.92**
A-n60-k3-300	736.34	738.83	953.04	**736.18**	**736.18**	1237.93	786.97
B-n34-k2-250	**461.44**	464.42	522.68	464.42	464.42	680.96	518.51
B-n34-k4-125	**746.25**	**746.25**	789.03	747.24	**746.25**	776.98	783.95
B-n34-k5-100	**888.4**	**888.4**	918.49	891.12	889.15	922.99	895.92
B-n35-k2-250	**565.09**	**565.09**	639.25	**565.09**	**565.09**	775.96	**565.09**
B-n39-k2-250	366.59	367.27	368.57	367.27	**366.2**	521.97	366.8
B-n41-k3-200	**608.7**	**608.7**	653.64	609.11	**608.7**	669.24	653.64
B-n43-k3-200	511.38	511.38	530.16	**511.05**	**511.05**	583.97	530.16
B-n43-k4-150	**604.47**	**604.47**	**604.47**	**604.47**	**604.47**	637.45	**604.47**
B-n43-k6-100	**816.21**	**816.21**	**816.21**	817.63	816.63	832.99	**816.21**
B-n44-k4-175	**668.05**	669.58	741.5	668.12	668.12	771.03	740.72
B-n44-k5-140	**780.47**	**780.47**	799.14	781.07	**780.47**	812.56	799.14
B-n44-k7-100	**1068.77**	1070.02	1125.71	1072.59	1070.99	1091.23	1100.65
B-n45-k2-300	414.37	414.37	420.93	414.37	**413.82**	650.97	420.93
B-n45-k3-200	**503.31**	**503.31**	513.68	503.32	**503.31**	592.96	512.27
B-n45-k4-125	**671.59**	**671.59**	**671.59**	**671.59**	**671.59**	746.68	**671.59**
B-n45-k4-150	**569.65**	**569.65**	581.96	**569.65**	**569.65**	717.97	581.96
B-n45-k5-120	**673.02**	**673.02**	732.47	678.64	**673.02**	729.89	720.51
B-n50-k4-175	**549.63**	**549.63**	577.16	**549.63**	**549.63**	622.97	**549.63**
B-n50-k5-140	**611.81**	**611.81**	617.12	**611.81**	**611.81**	631.74	617.12
B-n51-k2-350	**564.24**	**564.24**	649.32	565.15	**564.24**	905.95	649.32
B-n51-k4-175	**792.87**	**792.87**	936.01	**792.87**	**792.87**	816.02	854.03
B-n51-k5-140	**897.6**	**897.6**	913.98	898.33	**897.6**	928.93	913.98
B-n51-k7-100	**1162.94**	**1162.94**	1168.92	**1162.94**	**1162.94**	1288.83	1168.92
B-n52-k2-350	**415.21**	**415.21**	458.79	**415.21**	**415.21**	813.96	458.79
B-n52-k4-175	**577.29**	**577.29**	612.48	577.4	577.4	665.97	611.01
B-n52-k5-140	**643.39**	**643.39**	664.73	**643.39**	**643.39**	775.4	664.73

Future work could focus on enhancing the pricing phase. Although dynamic programming-based pricing has demonstrated strong performance in this study, its computational cost limits scalability to instances with longer routes. Exploring alternative pricing heuristics may yield further improvements, enabling even faster solution times or enhanced solution quality.

Table 6: cost of the solution obtained by the methods with a 15-minute computation time

Instance	CG +GRASP +MIP	CG +GRASP	CG+MIP (Gauvin et al.)	GRASP	GRASP +MIP	CG+MIP (Florio et al.)	CG +MIP _1H
A-n39-k2-300	**578.49**	**578.49**	**578.49**	**578.49**	**578.49**	669.95	**578.49**
A-n44-k2-300	**655.38**	**655.38**	683.95	656.51	656.51	795.94	683.95
A-n44-k4-150	**841.01**	**841.01**	851.32	843.33	**841.01**	861.56	848.42
A-n44-k5-120	**931.97**	**931.97**	935.74	932.51	**931.97**	933.57	935.74
A-n45-k2-300	**636.63**	**636.63**	**636.63**	**636.63**	**636.63**	711.95	**636.63**
A-n45-k3-200	**711.11**	**711.11**	**711.11**	**711.11**	**711.11**	746.96	**711.11**
A-n48-k2-350	**623.4**	**623.4**	822.73	**623.4**	**623.4**	932.95	822.73
A-n48-k4-175	831.81	831.81	884.16	**831.67**	**831.67**	896.09	**831.67**
A-n53-k2-350	633.5	636.01	636.01	634.5	**630.68**	1266.95	636.01
A-n53-k4-175	**814.91**	815.02	818.58	817.49	815.02	1040.95	818.58
A-n53-k5-140	**914.83**	**914.83**	914.84	918.67	**914.83**	947.73	914.84
A-n55-k2-450	**587.68**	**587.68**	**587.68**	**587.68**	**587.68**	836.94	**587.68**
A-n55-k3-300	**653.92**	**653.92**	669.65	654.16	**653.92**	806.93	**653.92**
A-n60-k3-300	736.34	736.78	943.19	736.18	**735.62**	1193.93	786.97
B-n34-k2-250	**461.44**	464.42	522.68	**461.44**	**461.44**	680.96	518.51
B-n34-k4-125	**746.25**	**746.25**	789.03	746.93	**746.25**	776.98	783.95
B-n34-k5-100	**888.4**	**888.4**	918.49	889.16	**888.4**	922.99	895.92
B-n35-k2-250	**565.09**	**565.09**	**565.09**	**565.09**	**565.09**	775.96	**565.09**
B-n39-k2-250	366.59	366.59	368.57	**366.2**	**366.2**	516.97	366.8
B-n41-k3-200	608.7	608.7	653.64	609.11	**608.19**	668.24	653.64
B-n43-k3-200	511.38	511.38	530.16	**511.05**	**511.05**	569.12	530.16
B-n43-k4-150	**604.47**	**604.47**	**604.47**	**604.47**	**604.47**	635.45	**604.47**
B-n43-k6-100	**816.21**	**816.21**	**816.21**	816.63	**816.21**	832.99	**816.21**
B-n44-k4-175	668.05	668.05	740.72	668.06	**667.8**	736.45	740.72
B-n44-k5-140	**780.47**	**780.47**	799.14	781.07	**780.47**	812.56	799.14
B-n44-k7-100	**1068.77**	1070.02	1122.89	1071.84	1068.8	1091.23	1100.65
B-n45-k2-300	414.37	414.37	420.93	414.37	**413.82**	599.97	420.93
B-n45-k3-200	**503.31**	**503.31**	513.68	503.32	**503.31**	579.96	512.27
B-n45-k4-125	**671.59**	**671.59**	**671.59**	**671.59**	**671.59**	742.24	**671.59**
B-n45-k4-150	**569.65**	**569.65**	581.96	**569.65**	**569.65**	673.77	581.96
B-n45-k5-120	**673.02**	**673.02**	732.36	674.45	**673.02**	729.89	720.51
B-n50-k4-175	**549.63**	**549.63**	577.16	**549.63**	**549.63**	591.01	**549.63**
B-n50-k5-140	**611.81**	**611.81**	617.12	**611.81**	**611.81**	629.61	617.12
B-n51-k2-350	**564.24**	**564.24**	649.32	**564.24**	**564.24**	860.95	649.32
B-n51-k4-175	**792.87**	**792.87**	935.78	**792.87**	**792.87**	800.94	854.03
B-n51-k5-140	**897.6**	**897.6**	913.98	**897.6**	**897.6**	927.93	913.98
B-n51-k7-100	**1162.94**	**1162.94**	1168.92	**1162.94**	**1162.94**	1288.83	1168.92
B-n52-k2-350	**415.21**	**415.21**	458.79	**415.21**	**415.21**	800.96	458.79
B-n52-k4-175	**577.29**	**577.29**	612.48	577.4	**577.29**	665.97	611.01
B-n52-k5-140	**643.39**	**643.39**	664.73	**643.39**	**643.39**	671.87	664.73

References

Baldacci, R., Mingozzi, A., Roberti, R., 2011. New route relaxation and pricing strategies for the vehicle routing problem. Operations Research 59, 1269–1283.

Bianchessi, N., Corberán, Á., Plana, I., Reula, M., Sanchis, J.M., 2022. The min-max close-enough arc routing problem. European Journal of Operational Research 300, 837–851.

Christiansen, C.H., Lysgaard, J., 2007. A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. Operations Research Letters 35, 773–781.

Costa, L., Contardo, C., Desaulniers, G., 2019. Exact branch-price-and-cut algorithms for vehicle routing. Transportation Science 53, 946–985.

Florio, A.M., Feillet, D., Poggi, M., Vidal, T., 2022. Vehicle routing with stochastic demands and partial reoptimization. Transportation Science 56, 1393–1408.

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. New exact algorithm for the vehicle routing problem with stochastic demands. Transportation Science 54, 1073–1090.

Gauvin, C., Desaulniers, G., Gendreau, M., 2014. A branch-cut-and-price algorithm for the vehicle routing problem with stochastic demands. Computers & Operations Research 50, 141–153.

He, Y., Ma, H.L., Park, W.Y., Liu, S.Q., Chung, S.H., 2023. Maximizing robustness of aircraft routing with heterogeneous maintenance tasks. Transportation Research Part E: Logistics and Transportation Review 177, 103237.

Hoogendoorn, Y., Spliet, R., 2025. An evaluation of common modeling choices for the vehicle routing problem with stochastic demands. European Journal of Operational Research 321, 107–122.

Mendoza, J.E., Castanier, B., Guéret, C., Medaglia, A.L., Velasco, N., 2011. Constructive heuristics for the multicompartment vehicle routing problem with stochastic demands. Transportation Science 45, 346–363.

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.

Mor, A., Speranza, M.G., 2022. Vehicle routing problems over time: a survey. Annals of Operations Research 314, 255–275.

Petris, M., Archetti, C., Cattaruzza, D., Ogier, M., Semet, F., 2024. A heuristic with a performance guarantee for the commodity constrained split delivery vehicle routing problem. Networks 84, 446–464.

Sadykov, R., Vanderbeck, F., Pessoa, A., Tahiri, I., Uchoa, E., 2019. Primal heuristics for branch and price: The assets of diving methods. INFORMS Journal on Computing 31, 251–267.

Tillman, F.A., 1969. The multiple terminal delivery problem with probabilistic demands. Transportation Science 3, 192–204.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., Subramanian, A., 2017. New benchmark instances for the capacitated vehicle routing problem. European Journal of Operational Research 257, 845–858.

Yee, J.R., Golden, B.L., 1980. A note on determining operating strategies for probabilistic vehicle routing. Naval Research Logistics Quarterly 27, 159–163.