Solution approaches for the vehicle routing problem with occasional drivers and time windows

Luigi Di Puglia Pugliese a, Daniele Ferone b, Paola Festa c, Francesca Guerriero b and Giusy Macrina b

aIstituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche, Rende, Italy; bDepartment of Mechanical, Energetic and Management Engineering, University of Calabria, Rende, Italy; cDepartment of Mathematics and Applications, University of Napoli FEDERICO II, Napoli, Italy

ABSTRACT

The efficient management of last-mile delivery is one of the main challenges faced by on-line retailers and logistic companies. The main aim is to offer personalized delivery services, that meet speed, flexibility, and control requirements and try to reduce environmental impacts as well. Crowd-sourced shipping is an emerging strategy that can be used to optimize the last-mile delivery process. The main idea is to deliver packages to customers with the aid of non-professional couriers, called occasional drivers. In this paper, we address the vehicle routing problem with occasional drivers, time window constraints and multiple deliveries. To handle this problem, we design some greedy randomized adaptive search procedures (GRASP). In order to assess the behaviour of the proposed algorithms, computational experiments are carried out on benchmark instances and new generated test sets. A comparison with previous published approaches, tailored for the problem at hand, is also provided. The numerical results are very encouraging and highlight the superiority, in terms of both efficiency and effectiveness, of the proposed GRASP algorithms.

ARTICLE HISTORY

Received 3 July 2020
Accepted 27 October 2021

KEYWORDS

Crowd-shipping; vehicle routing; metaheuristics; occasional drivers

1. Introduction

The continuous growth in on-line shopping has led to an intensification of the competition among the on-line retailers. The efficiency of the same-day and last-mile delivery is one of the major issues that on-line retailers and logistics companies, in general, have to deal with. On the one hand, e-commerce is pushing retailers to faster deliveries for gaining competitive advantages, on the other hand, customers have high expectations for fast deliveries. Thus, to strengthen their market position and attract further potential customers, on-line retailers have started to propose and develop innovative delivery models.

Crowd-shipping is a promising model that is gaining popularity among on-line retailers. In crowd-shipping, non-professional couriers (i.e. ordinary people) en route to their
destinations take items to other people, usually in less than one hour, for a small compensation. This new delivery policy is widely used in the prepared food delivery sector; some examples are Deliveroo (www.deliveroo.co.uk), DoorDash (www.doordash.com), Foodora (www.foodora.com), Justeat (www.just-eat.com) and Glovo (www.glovoapp.com). All these companies hire ordinary people to occasionally take prepared foods to customers’ doors. The only requirements concern a smart-phone for downloading an application, an own vehicle and some free time. As the couriers, the customers need the same application to upload their orders and indicate a time window of availability. Thus, the dedicated application becomes a meeting place of on-line retailer, on-line shoppers and carrier. Following this idea, some big on-line retailers have started to exploit a crowd-shipping strategy for last-mile delivery. Walmart in 2013 tested its idea of crowd-shipping by introducing the possibility for in-store customers to deliver items to on-line customers (see [8]). In the same year, DHL launched its crowd-shipping platform named MyWays, testing it in Sweden (see [36,53]). DHL engaged ordinary people on their way to provide last-mile delivery service from the retailer depot to the on-line shoppers’ house. However, one of the most successful experiment was Amazon Flex, the crowd-shipping platform of Amazon (see [9]). Nowadays, Amazon Flex is still used in more than 30 cities in the world and offers several delivery opportunities (see [41]). For a complete survey of transportation trends and a discussion on challenges and opportunities of city logistics, the reader is referred to [51]; for a complete review on crowd-logistics, the reader is referred to [11,50].

1.1. The vehicle routing with crowd-shipping: the state of the art

Due to its potential benefits, crowd-shipping has gained much attention also from the scientific research world. Following the idea of Walmart, Archetti et al. [5] introduced the vehicle routing problem with occasional drivers (VRPOD). In the VRPOD, the company makes deliveries by using its own fleet of conventional vehicles as well as a fleet of non-professional couriers, named occasional drivers (ODs), who decide to make a deviation from their ordinary routes, for delivering parcels to other people, for a small compensation. Archetti et al. [5] proposed a multi-start heuristic for the VRPOD, which combines variable neighbourhood search (VNS) and tabu search (TS). To assess the performances of their heuristic and evaluate the effect of using the ODs, Archetti et al. [5] carried out a comprehensive computational study on instances derived from Solomon’s benchmark set (see [54]) with up to 100 customers and 50 ODs. The authors highlighted the benefits, in terms of saving cost, gained using ODs in last-mile delivery, and studied the effect of varying the ODs’ compensation scheme on the solutions cost. In [5], each classical vehicle may perform multiple deliveries, i.e. may serve several customers in a trip, whereas each OD may serve only one customer. Macrina et al. [40] extended the work of [5] by introducing the time window constraints and the multi-delivery policy also for the ODs. The authors named this variant of the problem VRPOD with time windows and multiple delivery (VRPODTWmd). In addition, they addressed a variant of the VRPODTWmd with split and delivery for the ODs. Their computational study, carried out by solving to optimality their models with CPLEX on instances with up to 15 customers and 3 ODs, showed the benefits of using ODs and the interesting cost saving obtained by introducing the multi-delivery and the split and delivery policies. A VNS heuristic for addressing the VRPODTWmd has been proposed in [39]. This heuristic solves instances derived from
Solomon’s benchmark set (see [54]) with up to 100 customers and 30 ODs. Recently, Macrina et al. [41] extended the VRPODTWmd considering the scenario in which transshipment nodes are included in the service network. In particular, transshipment nodes act as intermediate depots for ODs, closer to the delivery area, and have to be served by the traditional vehicles. The authors formulated the problem as a particular instance of the two-echelon VRP, then they showed the benefits of using transshipment nodes as intermediate depots. They proposed a math-heuristic that combines VNS and mathematical programming. Dahle et al. [15] studied a new version of the VRPOD of [5] by introducing pickup and delivery policy and time windows. In particular, Dahle et al. [15] proposed a load and a flow formulation for the problem and focused on the design of compensation schemes for the ODs. The authors made a computational study on instances based on the benchmark set of [49], with up to 50 ODs and 70 requests. They concluded that the use of ODs can lead to a substantial cost saving, even when the company utilizes a suboptimal compensation scheme.

All the aforementioned scientific contributions assume to know in advance all the information related to the requests and ODs availability. However, several researchers focused on the dynamical nature of the problem. Arslan et al. [6] considered a same-day delivery setting in which both tasks and drivers dynamically arrive over time. Hence, the authors introduced a variant of the dynamic pickup and delivery VRP that uses conventional vehicles as well as ad-hoc drivers. They formulated the problem and proposed a solution approach, based on a rolling horizon and an exact algorithm, to repeatedly solve the various route planning problems. Then, they considered a peer-to-peer platform and tested its performance with a simulation study. They created ad-hoc benchmark instances to test their approach. Dahle et al. [14] addressed the case where the availability of ODs is uncertain and supposed to know some stochastic information about their appearance. The authors proposed a stochastic formulation and compared its performance with those obtained by using deterministic strategies with re-optimization. They generated ad-hoc instances with up to 20 customers and three ODs. Their results showed the benefit of using a stochastic formulation in terms of saving costs. In [16] a VRPOD with dynamic customers requests is studied. Two rolling horizon dispatching approaches are developed: a myopic one that ignores any information regarding future events and one that incorporates probabilistic information about future on-line order and ODs arrivals. The authors generated a set of instances with up to 50 locations (ODs and customers), then compared the two approaches. In [30] the ODs are represented as independent agents, which are free to reject assignments, and assigned to each pair (OD-request) a random probability. The authors proposed a bi-level stochastic model and a heuristic approach to solve the studied problem. Their computational study, carried out on simulated instances with up to 15 customers, highlighted the need to define a dynamic compensation scheme for ODs. The online VRPOD, in which customer requests are either known in advance, or could arrive online during the distribution process, is studied in [4]. The authors propose an insertion algorithm and a re-optimization approach to solve this variant. Then, they analysed the impact of dynamic information on the delivery planning, considering several degree of dynamism.

The green vehicle routing problem with occasional drivers (GVRPOD) is addressed in [42]. In this work, it is assumed that the customers’ demand is satisfied by a mixed fleet of vehicles, composed of internal combustion commercial vehicles and electric vehicles, and by also using some ODs. The authors formulated mathematically the GVRPOD
as an integer linear programming model, aimed at minimizing the sum of routing costs of conventional and electric vehicles, by including fuel consumption cost and energy consumption cost, and the occasional drivers’ compensation. The computational results collected on instances derived from Solomon’s benchmark set underlined that using ODs may lead not only to more convenient solutions, but also more eco-friendly transportation plans.

For other works addressing some specific issues related to crowd-shipping, the reader is referred to [1,2,42,52]. The analysis of the conditions that can impact positively on people’ willingness to act as crowdshippers is presented in [52], where only ‘specific segments of the population that are travelling by metro’ are taken into consideration. Allahviranloo and Baghestani [2] studied the impacts of crowd-shipping on the travel behaviour and demand distribution. Abu Al Hla et al. [1] addressed a specific version of VRPOD where the regular drivers and occasional drivers’ behaviours are considered and their impact on the routing costs is monitored and analysed.

The VRPOD is quite similar to the VRP with a private fleet and common carrier (VRPPC) [7,29]: in both problems, the company has a private fleet to perform deliveries. It outsources some delivery requests when either the capacity of the own fleet is not enough to guarantee a high quality of service or it gains advantages in terms of cost. The substantial difference between the VRPOD and the VRPPC is that the deliveries are outsourced to a not professional carrier for the former. This aspect prevents saving opportunities in subcontracting deliveries to a professional external carrier. It follows that the management of the compensation for the ODs cannot be the same as that for a professional carrier. In addition, in the VRPPC, the external carriers start and end their route at the depot of the company. The decision of which kind of vehicle, either trucks from the own fleet or from the external carrier (professional or OD), is common to both the VRPPC and the VRPOD. Gahma et al. [29] and Goeke et al. [31] studied the VRPPC and proposed several approaches to solve it. In particular, Gahma et al. [29] proposed a mixed integer model and three heuristics based on VNS for the VRPPC, whereas, Goeke et al. [31] designed an exact solution approach. In this work, we address the VRPOD Twin described in [40], and compare several approaches to solve this specific version of the VRPOD.

1.2. Aims and organization of the paper

The aim of this work is to provide novel solution approaches for the VRPOD Twin. In particular, we design Greedy Randomized Adaptive Search Procedures (GRASP) [19] and propose a hybrid approach where a VNS is applied as local search. The hybrid version allows to explore large portions of the solution space. We use the VNS scheme proposed by Macrina et al. [39]. We also carried out a comparison with a modified version of the approach proposed by Archetti et al. [5], defined so that it handles the specific features of the VRPOD variant studied in this paper, which are not taken into account in [5] (i.e. time window constraints and multiple deliveries for the ODs).

The rest of the paper is organized as follows. The mathematical formulation of the VRPOD Twin is given in Section 2. The proposed solution approaches are described in Section 3. The analysis of the computational performance of the developed algorithms is reported in Section 4. Conclusions are summarized in Section 5.
2. Problem definition

In this section, we report the mathematical formulation of the VRPODTWmd introduced by Macrina et al. [40].

2.1. Sets and parameters

Let $C$ be the set of customers, $s$ and $t$ the origin and destination node for the traditional vehicles, respectively. Let $K$ be the set of ODs; since the ODs have the same origin but different destinations, let $V$ be the set of $v_k$ destinations associated with the ODs. We define the node set as $N = C \cup \{s, t\} \cup V$. Let $A$ be the set of arcs. We model the VRPODTWmd on a complete directed graph $G = (N, A)$. Each arc $(i, j) \in A$ has a cost $c_{ij}$ and a time $t_{ij}$ associated with it. Note that both $c_{ij}$ and $t_{ij}$ satisfy the triangle inequality. Each arc $(i, j) \in A$ can be traversed by either a classical vehicle or an OD only once. Each node $i \in C \cup V$ has a time window $[e_i, l_i]$ and each customer $i \in C$ has a demand $d_i$. Let $Q$ be the capacity of the traditional vehicles, $P$ the number of available traditional vehicles, and $Q_k$ the capacity of OD $k \in K$.

2.2. Variables

Let $x_{ij}$ be a binary variable equal to 1 if a traditional vehicle traverses the arc $(i, j)$, and 0 otherwise. For each node $i \in C \cup \{s, t\}$, let $y_i$ be the current load after visiting the node $i$, while $s_i$ the instant time in which the traditional vehicle starts the service at node $i$. Let $r^k_{ij}$ be a binary variable equal to 1 if the OD $k$ traverses the arc $(i, j)$, 0 otherwise. Let $f^K_i$ indicate the instant time in which customer $i$ is served by OD $k$ and $w^K_i$ is the current load of OD vehicle $k$ after visiting the customer $i$. The variable $f^k_{vk}$ indicates the arrival time at destination $v_k$ of OD $k$.

2.3. Formulation

The VRPODTWmd can be mathematically modelled as follows:

Minimize \[ \sum_{i \in C \cup \{t\}} \sum_{j \in C \cup \{s\}} c_{ij} x_{ij} + \sum_{k \in K} \sum_{i \in C \cup \{s\}} \sum_{j \in C} \rho c_{ij} r^k_{ij} - \sum_{k \in K} \sum_{j \in C} c_{sv_k} r^k_{sj} \] (1)

s.t. \[ \sum_{j \in C \cup \{t\}} x_{ij} - \sum_{j \in C \cup \{s\}} x_{ji} = 0 \quad i \in C \] (2)

\[ \sum_{j \in C} x_{sj} - \sum_{j \in C} x_{jt} = 0 \] (3)

\[ y_j \geq y_i + d_j x_{ij} - Q(1 - x_{ij}) \quad j \in C \cup \{t\}, \quad i \in C \cup \{s\} \] (4)

\[ y_s \leq Q \] (5)

\[ s_j \geq s_i + t_{ij} x_{ij} - M(1 - x_{ij}) \quad i \in C, \quad j \in C \] (6)

\[ e_i \leq s_i \leq l_i \quad i \in C \] (7)
The objective function (1) minimizes the total costs. The first term of (1) is the transportation cost associated with traditional vehicles. The second and the third terms define the compensation scheme for the ODs. In particular, the first one is the compensation cost of OD $k$ for the delivery service, with the compensation factor $\rho \geq 0$. The third term represents the cost of OD $k$ when it does not perform the delivery. In fact, the OD compensation is based on the evaluation of the ‘deviation’ that the OD makes from the origin-destination trip assigned. Hence, an OD will be paid only for the additional distance travelled, given by the distance of the total tour minus the origin-destination distance (see [5]).
2.4. Traditional vehicles constraints (2)–(8)

Constraints (2) and (3) are the flow conservation constraints. Constraints (4) and (5) guarantee that the maximum capacity is not exceeded. Constraints (6) compute the initial service time at node \( j \), where \( M \) is a very large number, and (7) are the time window constraints. Constraint (8) imposes a limit on the maximum number of available traditional vehicles.

2.5. ODs constraints (9)–(19)

Constraints (9) and (10) are the flow conservation constraints. Conditions (11) and (12) impose a limit on the number of available ODs and on the number of departures from the depot, respectively. Constraints (13) and (14) are the capacity constraints. Constraints (15) allow to determine the initial service time at node \( j \). Constraints (16) and (17) are the time window constraints and also define the time at which the ODs are available for delivering, while constraints (18) compute the arrival time of OD \( k \) at the own destination node \( v_k \). Conditions (19) guarantee that each customer is visited within its time window.

Constraints (20) ensure that each customer is visited at most once, either by a traditional vehicle or an OD. Constraints (21) to (25) define the domains of variables.

3. Solution approaches

In this section, we describe the solution strategies considered to handle the VRPODTWmd. In particular, to make a comparison of our proposal with previous published approaches, we present a modified version of the method presented in [5], named MATHOD, defined in such a way to handle the specific features of the VRPOD variant studied in this paper, that are time window constraints and multiple deliveries for the ODs, which are not taken into account in [5]. Then, we also consider a VNS, developed by starting from the scheme and the local search procedures defined in [39] for the VRPODTWmd. Finally, we provide the description of the proposed GRASP.

3.1. MATHOD heuristic

MATHOD algorithm is the multi-start approach proposed in [5] to solve the VRPOD. Its pseudo-code is depicted in Algorithm 1. MATHOD generates five initial solutions characterized by a set \( S \subset C \) of customers to be served by regular drivers (line 2). A greedy algorithm generates routes to serve \( S \) inserting customers in the active route in non-decreasing order of their distance to the depot. When all customers in \( S \) have been inserted, each route is improved using the 2-Opt neighbourhood. Four sets \( S \) are generated by solving four different mathematical models (we refer to the original paper for more details), and the last one is picked as \( S = C \).

A Tabu Search (line 7) and a perturbation procedure (line 8) are repetitively applied to the initial solutions. For a given solution \( s \), let \( C' \) denote the set of customers served by the occasional drivers and let \( r_i \), for \( i \in C \setminus C' \), denote the route of the regular driver serving customer \( i \). The Tabu Search performs the following moves: \( i\)-move, a customer \( i \in C \setminus C' \) is moved from \( r_i \) to a route \( r \neq r_i \); swap-move, customers \( i, j \in C \setminus C' \), with \( r_i \neq r_j \), are
Algorithm 1: Pseudo-code of the MATHOD algorithm (reported from [5]).

```
Algorithm MATHOD \( (k_{\text{max}}) \)
I ← ConstructInitialSolutions();
foreach \( i \in I \) do
    s ← i;
    k ← 1;
    while \( k < k_{\text{max}} \) do
        s ← InternalTabuSearch(s);
        s ← Perturb\( (k, s) \);
        if s is better than s_{\text{best}} then
            s_{\text{best}} ← TwoOpt(s);
            s ← s_{\text{best}};
            k ← 1;
        else
            k ← k + 1;
return s_{\text{best}};
```

exchanged; in-move, a customer \( i \in C' \) is inserted into a route \( r \); out-move, a customer \( i \in C \setminus C' \) is served by an occasional driver.

The tabu status forbids customers to be inserted in routes from which they have been recently removed, or move them back to \( C' \) if they have been recently moved to \( C \). All non-tabu moves are evaluated, and the best one is chosen. Each time a new best solution is found, each route is improved with a local search procedure using the 2-Opt neighbourhood. The perturbation procedure moves some customers served by occasional drivers to some randomly selected routes. Afterward, a mathematical model is solved to assign some customers to occasional drivers to obtain a solution that is considerably different from the initial one.

We implemented MATHOD following the description of [5] as faithful as possible. Nevertheless, our problem differs from the original version of [5] in two aspects: the presence of the time windows and the possibility for the ODs to perform multiple deliveries.

Therefore, we adapted all the moves to take into account time windows feasibility. Thus, a move is performed only if the resulting solution is feasible with respect to all the time window constraints. For example, a 2-Opt move in the same route is always feasible when time window constraints are not considered. On the contrary, we take into account a 2-Opt exchange only if all the customers’ time windows are respected after the move application. Moreover, the out-move can assign a customer to an occasional driver even if the occasional driver already has to perform other deliveries.

### 3.2. Variable neighbourhood search

In the following, we give a description of VNS proposed in [39]. First, we present the general VNS framework, then we describe the single components defined in [39]. The basic idea of VNS is to define a set of neighbourhood structures \( N_h, h = 1, \ldots, h_{\text{max}} \) to explore
Algorithm 2: Pseudo-code of a generic VNS.

The solution space. Let $N_h(s)$ be the set of solutions in the $h$th neighbourhood of a solution $s$. Algorithm 2 describes a pseudo-code of a general basic VNS.

VNS is characterized by the execution of three main phases: construction, perturbing, and local search.

1. **Construction**: this phase finds an initial feasible solution $s$.
2. **Perturbing**: the main aim of this step is to perturb the current solution through local search operators, which allow worsening solutions, obtaining a solution $s'$. The solution $s'$ is the starting point for the local search procedure.
3. **Local search**: the purpose of this phase is to improve the solution $s'$ generated after the perturbation, by exploring several neighbourhoods. At the end of this phase, we obtain the solution $s''$. In particular, for each $h = 1, \ldots, h_{\text{max}}$, the $h$th local search is applied to $s'$ obtaining a solution $s''$.

If $s''$ is more effective than $s$, then it replaces $s$. The VNS iterates between perturbing and local search phases until a predefined stopping criterion is met.

In what follows, we give a detailed description of the construction algorithm, the local search moves and the perturbing procedure used by VNS proposed in [39].

### 3.2.1. Solution construction

The construction phase is an insertion heuristic adapted for the VRPOD TWmd, mixed with a Clark & Wright savings (CWS) [13] scheme for the traditional vehicles. At first, it tries to serve the farthest customers with the available ODs. The customers are sorted in decreasing order with respect to the distance from the depot. For each customer, starting from the farthest one, the algorithm verifies if an OD can perform the service with respect to the time window and capacity constraints. Therefore, it inserts the customer in the best feasible position, i.e. the one with the lowest increase of the objective function.

Then, the unserved customers are served by the traditional vehicles, following the CWS algorithm. In particular, for each unserved customer, the CWS initializes a round-trip route
from the depot to the customer. Then, the routes are merged with respect to the feasibility constraints and the savings are calculated. The merging with the best saving is then selected. The construction phase ends when all the feasible merges have been exploited.

### 3.2.2. Local search

We designed six different neighbourhoods based on several local search moves. In this section, if not otherwise specified, we refer to routes $r$ and $r'$ both as traditional routes and OD paths.

**2-Opt.** This operator has been widely used in several solution approaches for different variants of the traveling salesman and the vehicle routing problems. The main idea is to remove two arcs $(i, j)$ and $(u, v)$ in the same route $r$ or in two distinct routes $r$ and $r'$, and to reconnect the path(s) created using arcs $(i, u)$ and $(j, v)$. Every time the best feasible swap is performed.

**Move Node.** Each customer $i$ is removed from its route $r$ and is inserted in the position that leads to the solution with the minimum cost. The route $r'$ where $i$ has been inserted can be equal to $r$.

**Swap Inter-Route.** This operator removes one node $i$ from a route $r$ and one node $j$ from another route $r'$, and inserts $i$ into $r'$ and $j$ into $r$ in the best feasible positions. Every time the best feasible swap is performed.

**Swap Intra-Route.** This operator swaps the position of two nodes $i$ and $j$ in a given route $r$. All the routes are scanned, and for each of them the best feasible swap is performed.

**New Route best.** A new traditional route $r'$ is initialized only if this choice leads to a less expensive solution. A node $i$ belonging to a given route $r$ is removed from $r$ and inserted in $r'$.

**New Route.** Similar to the previous one, but the move is performed also if it leads to a worse solution. In both cases, all the customers are analyzed and the selected node $i$ is the one that leads to the best solution in terms of objective function.

A Variable Neighbourhood Descent (VND) is used as local search. The main operations executed by the developed VND are described in Algorithm 3.

The initial solution $s'$ for the VND is that obtained after the perturbing phase. For each $h = 1, \ldots, h_{\text{max}}$, the $h$th local search is applied to $s'$ obtaining a solution $s''$. If the cost of $s''$ is lower than the current one, it replaces $s'$ and the procedure restarts from the first neighbourhood structure, otherwise it uses the next neighbourhood structure.

The local search stops when all the neighbourhoods have been explored without improving the solution.

It is worth observing that considering a random neighbourhood ordering for the VND could improve the performance of the VNS (see, e.g. [3,55]). We decide to consider the VNS proposed in [39] without modifying the strategy used to implement the VND. Therefore, the neighbourhoods are ordered as presented in this Section.

### 3.2.3. Perturbing procedure

The aim of the perturbing phase is to modify the solution $s$ and to provide a new starting point for the local search. This strategy helps to escape from local minima and hence ensures a better exploration of the solution space. Therefore, at each perturbation, we
applied two different local search operators, where the moves are designed so that a worsening in the solution is allowed, i.e. the moves are applied even if the resulting solution is of a lower quality than the starting one. The local search operators are selected through a semi-random procedure, that takes into account their effectiveness. At the beginning of the algorithm, we assign a score equal to $1$ to each operator. At the end of each VNS iteration, if there is an improvement in solution cost, we increment the scores of the perturbing moves, otherwise we decrease them, maintaining the minimum score for each move equal to $1$. For each neighbourhood $N_h$, the probability $P(N_h)$ of being selected is directly proportional to its score($N_h$). More specifically, $P(N_h)$ is calculated as follows:

$$P(N_h) = \frac{\text{score}(N_h)}{\sum_{q=1,...,h_{\text{max}}} \text{score}(N_q)}.$$ 

### 3.3. Greedy randomized adaptive search procedure

GRASP has been introduced in [19] as an iterative multi-start metaheuristic for difficult combinatorial optimization problems. It is a very flexible approach, widely applied to a large set of problems, including facility location [22], scheduling [32], satisfiability [18], constrained shortest path tour [20,21], max-cut [24], single-container loading [34] and graph drawing [44]. For an extensive survey, the reader is referred to [25–27].

GRASP has been also successfully applied to routing problems. One of the first attempt to use GRASP for solving the VRP with time windows is represented by the work of [12]. Li et al. [37] proposed a GRASP improved by Tabu Search for the VRP with time windows, and Nguyen et al. [45] solved a two-echelon location routing problem with a hybridized GRASP. Haddadene et al. [33] hybridized it with ILS to solve a VRP with time windows in the field of home health care. Moreover, Radojičić et al. [46] proposed a Fuzzy GRASP for a Risk-constrained Cash-in-Transit VRP.
There are several similarities between our approach and GRASPs proposed in the scientific literature. For example, many of them use a variant of CWS algorithm in the construction phase. Moreover, some local search moves (e.g. 2-opt, move, swap) have been extensively used in many VRP variants, and they are present also in other GRASP implementations.

Nevertheless, to the best of our knowledge, GRASP has never used in the presence of ODs, and we designed a tailored construction phase taking advantage of their availability. Moreover, we use a biased-randomized construction phase that is less applied than the classical restricted candidate list (RCL) method, but it showed good results in similar contexts. Finally, in [39], VNS showed a good potential in solving VRPODTWmd. Therefore, we chose to embed the VNS in a GRASP framework.

Each GRASP iteration is characterized by two main phases: construction and local search.

(1) **Construction**: starting with an empty solution, the construction phase iteratively adds an element, one at a time, until a complete solution is obtained. The elements are sorted according to some greedy functions. These functions measure the benefit of selecting each element, then construct the RCL with the best elements. At the end, an element is randomly selected from the RCL.

(2) **Local search**: the aim of this procedure is to improve the solution generated by the construction phase; it returns a locally optimal solution with respect to some neighborhood structures. The best locally optimal solution found is returned. Algorithm 4 depicts the pseudo-code of a generic GRASP heuristic for a minimization problem.

```plaintext
Algorithm GRASP(MaxIterations)
1   x* ← Nil;
2   cost(x*) ← +∞;
while a stop criterion is not met do
3       Build a greedy randomized solution x;
4       x ← LocalSearch(x);
5       if cost(x) < cost(x*) then
6           x* ← x;
7   return x*;
```

**Algorithm 4**: Pseudo-code of a generic GRASP.

Referring to the construction phase, we designed two different strategies: the classical RCL and a Biased Randomized version. This choice is motivated by the work of [47], that presents a survey of the advances and the extensions of GRASP. The proposed procedures can be viewed as a modified versions of the algorithm presented in Section 3.2.1. Meanwhile, the original scheme always takes the greediest choice both in the insertion of customers in ODs paths and in the merging of routes, the following two approaches are randomized with two different schemes.
For the local search procedure, we consider three different variants by combining the features related to the VNS, described in Section 3.2.

A detailed description of the construction phase is provided in Section 3.3.1, whereas, the description of the local search procedures is reported in Section 3.3.2.

### 3.3.1. Construction phase

In what follows, we give a detailed description of the operations executed by the two strategies used to build an initial solution.

**RCL method.** The first method designed is a ‘classical’ RCL scheme. Let suppose that \( c \in C \) is the currently selected customer, and let \( CL = \{p_1, \ldots, p_l\} \) be the set of feasible positions in which the customer can be inserted, i.e. the positions in the paths followed by the ODs, where a customer can be inserted without violating any constraint. In this phase, all ODs are considered, therefore in the case of a previously unused OD \( k \), the only possible insertion position is between the depot \( s \) and the final destination \( v_k \) of the OD. Moreover, let \( \text{cost}(p_i) \) be the increasing in the objective function when the customer \( c \) is inserted in the position \( p_i \).

The RCL is constructed as follows:

\[
RCL = \{p_i, i = 1, \ldots, l \mid \text{cost}(p_i) \leq \text{cost}_{\text{min}} + \alpha (\text{cost}_{\text{max}} - \text{cost}_{\text{min}})\},
\]

where \( \text{cost}_{\text{min}} = \min_{i=1, \ldots, l} \text{cost}(p_i) \), and \( \text{cost}_{\text{max}} = \max_{i=1, \ldots, l} \text{cost}(p_i) \). Once the RCL is constructed, a position is selected with uniform probability from the RCL. It is worthy of note that the parameter \( \alpha \) controls the randomization level of the algorithm. By setting \( \alpha = 0 \), we obtain a fully greedy algorithm. On the contrary, with \( \alpha = 1 \), a completely random choice is performed. In our implementation, \( \alpha \) is an input parameter (see Section 4.3).

Conversely, the Clark & Wright scheme, used to merge the routes, prefers the routes merging that maximizes the savings. More formally, let \( M = \{m_1, \ldots, m_p\} \) be the set of possible merges, and let \( s(m_i) \) be the savings of the merge \( m_i \). The RCL is defined as follows:

\[
RCL = \{m_i, i = 1, \ldots, p \mid s(m_i) \geq s_{\text{min}} + (1 - \alpha) (s_{\text{max}} - s_{\text{min}})\},
\]

where \( s_{\text{min}} = \min_{i=1, \ldots, p} s(m_i) \), and \( s_{\text{max}} = \max_{i=1, \ldots, p} s(m_i) \). Using \( 1 - \alpha \), the algorithm remains fully greedy with \( \alpha = 0 \) and totally random with \( \alpha = 1 \).

**Biased randomized method.** The biased randomized method (BR) was firstly introduced in [10], and successfully used in [23,28]. The construction with a biased selection function does not rely on the construction of the RCL, but it randomly selects from all the elements using a (skewed) probability distribution. Depending on the distribution, it is possible to obtain an algorithm that is more greedy or more random. For example, using a uniform probability distribution the algorithm is totally random. Using a geometric distribution with different parameters, it is possible to obtain a more greedy algorithm. Therefore, both the selections of the insertion position and the route merge do not use the RCL, and the elements are sorted with respect to \( \text{cost}() \) and \( s() \) and the selection follows a probability distribution. In our case, we use a geometric distribution with the parameter \( \beta \), which intuitively corresponds to the probability of selection of the first element in the candidate list. Thus, the probabilities are biased towards the most promising solution elements.

Figure 1 depicts a visual representation of the difference between the two approaches.
3.3.2. Local search

For the local search procedure, we consider three different variants with incremental intensification of solution space searching. A first version of the GRASP uses the move node local search described in Section 3.2.2. The second one uses the VND procedure as local search, presented in Section 3.2.2 and described in Algorithm 3. The last version is a hybrid metaheuristic, which combines the GRASP and the VNS where the latter is applied as local search. At each iteration, the VNS is performed to obtain a better solution than that provided by the construction phase of the GRASP.

4. Experimental results

In this section, we describe the experimental settings used to validate the considered solution approaches. All the algorithms have been implemented in C++ with gcc 8.3.0 using the flags -std=c++17 -O3. We carried out all the experiments on an Intel Core i7-4510U CPU @ 2.00GHz × 4 with 8GB of RAM under Linux (Ubuntu 18.04). To compare our results, we also solved the mathematical model using the commercial solver CPLEX 12.10 with default options (a thread for each core is used) and a time limit of 1800 seconds.

4.1. Generation of instances

We conducted several numerical experiments on different instances size. We have used the VRPOD instances, based on the classical Solomon VRPTW instances [54], introduced in [39].

Solomon’s original set is composed of instances grouped in six main classes, i.e. C1, C2, R1, R2, RC1 and RC2. In particular, sets C1 and C2 contain clustered customers, in sets R1 and R2 the customers location are randomly generated over a square and sets RC1 and...
RC2 have a combination of randomly and clustered customers. Sets of type 1 (i.e. C1, R1 and RC1) have narrow time windows, instead sets of type 2 (i.e. C2, R2 and RC2) have large time windows.

The authors in [39] derived 36 small instances with 5, 10 and 15 customers, considering all the six main classes of Solomon’s networks. In addition, they derived 30 medium instances with 25 and 50 customers and 15 large instances with 100 customers. For medium and large size instances, only the three Solomon’s classes C1, R1 and RC1 have been considered in [39]. Hence, following their rules, we have generated additional instances of type 2, maintaining the original trucks capacity. In particular, we have generated 30 medium size instances of class C2, R2, RC2, with 25 and 50 customers and 15 large size instances of class C2, R2, RC2 with 100 customers.

We have also considered other 30 large instances with 200 customers and 30 very large instances with 400 customers, starting from the set introduced in [35].

Following the idea of [39], we have identified the customers locations by the coordinates \((x_i, y_i)\), then we randomly generated the destinations for the ODs, in the square with lower left-hand corner \((\min_i{x_i}, \min_i{y_i})\) and upper right-hand corner \((\max_i{x_i}, \max_i{y_i})\). For each OD \(k\), we randomly generated a reasonably and feasible time window \([e_k, l_k]\) where \(e_k\) is in the range \([0, (T_{\text{max}} - t_{sk} - t')]\) and \(l_k\) is chosen in the range \([e_k + t_{sk} + t', T_{\text{max}}]\). In particular, \(T_{\text{max}}\) is the maximum duration of the tour, \(t_{sk}\) is the time necessary to travel from the origin \(s\) to the destination assigned to OD itself, while \(t'\) is a random number in the range \([\min_{i \in C}(t_{si} + t_{ii}), \max_{i \in C}(t_{si} + t_{ii})]\), where \(t_{ij}\) for each \((i, j) \in A\) is the time necessary to travel from \(i\) to \(j\).

In addition to the test problems described above, we have generated additional instances starting from those used in [5] for the VRPOD. We added the time windows for the ODs following the same procedure described above. Since the instances in [5] are derived from Solomon VRPTW set [54], we maintained the original time windows for the customers. Hence, we obtained 6 additional medium size instances with 25 customers and six large size instances with 100 customers.

To summarize, we used for our computational study 36 small size instances, 66 medium size instances, 66 large size instances and 30 very large size instances. Table 1 provides a summary of the classes of instances considered in the testing phase.1

| Size       | Number of customers | Benchmark [39] | Modified Benchmark [54] | Modified Benchmark [5] | Modified Benchmark [35] |
|------------|---------------------|----------------|--------------------------|------------------------|-------------------------|
|            |                     | type 1         | type 2                   | type 1                 | type 2                  | type 1 | type 2 |
| Small      | 5                   | x              | x                        |                        |                         |
|            | 10                  | x              | x                        |                        |                         |
|            | 15                  | x              | x                        |                        |                         |
| Medium     | 25                  | x              |                          | x                      |                        | x      |
|            | 50                  | x              |                          | x                      |                        |        |
| Large      | 100                 | x              | x                        | x                      |                        |        |
|            | 200                 |                |                          |                        |                         |        |
| Very large | 400                 |                |                          |                        |                         |        |

Table 1. Summary of the classes of instances used for the computational study.
4.2. Implemented codes

We implemented and compared the following algorithms:

- ‘MATHOD’: the multi-start heuristic described in Section 3.1;
- ‘VNS’: the VNS algorithm described in Section 3.2;
- ‘GRASP’: this version of GRASP uses the move node strategy, described in Section 3.2.2, as local search;
- ‘GRASP+VND’: this version uses the VND described in Section 3.3.2 as local search;
- ‘GRASP+VNS’: this version is a hybrid metaheuristic, which combines the GRASP and the VNS. In particular, it uses the VNS described in Section 3.2 as local search.

All the GRASP variants have been implemented using either BR or RCL in the construction phase as described in Section 3.3.1, but in the experiments we only used the variants with BR due to the results of the tuning process (see Section 4.3).

All the implementations exploit the advantage of parallel execution to reduce the computational times. In particular, in MATHOD after the initial construction, each thread analyzes a solution. In other words, OpenMP has been used to parallelize the loop of Line 3 of Algorithm 1. The parallelization in VNS is similar, the initial solution is built by a single thread. Therefore, four different threads perturb and perform local search independently. Moreover, since GRASP is a multi-start approach, the while loop of Line 4 of Algorithm 4 is parallelized; i.e. each thread builds a new solution and performs local search. Obviously, for all the methods the incumbent solution is shared among all the threads.

4.3. Parameters tuning

In what follows we briefly describe the parameters of the developed algorithms and we provide details of the applied tuning procedure. Following the ‘less is more’ approach proposed in [43], we tried to reduce to the minimum the number of parameters to be selected.

All the proposed solution approaches are characterized by a stopping criterion that could be related to the number of iterations, the execution time, and the maximum allowable number of iterations without an improvement. Details about the stopping criterion selected and the rationale for each choice are provided in the experimental results sections.

In what follows, we describe the tuning phase for the parameters related to the construction of the initial solutions for the GRASP framework. The algorithms based on the GRASP are characterized by the random selection of the construction method, i.e. RCL or BR (see Section 3.3.1.1), and by the parameters $\alpha$ and $\beta$ that guide RCL and BR, respectively, in the generation of the initial solutions. Both $\alpha$ and $\beta$ belong to the range $[0, 1]$.

To determine the values of the aforementioned parameters, we construct a training set, by randomly selecting the 30% of the instances for each size of the set of customers $C$. Once the set has been constructed, we performed an iterated racing procedure, implemented in irace package (see [38]), to automatically find the best configuration of the considered parameters. On the basis of the experiments carried out by using the irace package, the best performance are obtained by applying BR and setting $\beta = 0.85$. Thus this configuration is used in the testing phase.
4.4. Experimental results on the small size instances

In order to assess the performance of the proposed metaheuristics in terms of solution’s quality, we conducted a first experimental study on small size instances and we solved them to optimality by using CPLEX.

The stopping criterion for the tested solution approaches has been selected in such a way to ensure a good compromise between accuracy and efficiency. On the basis of preliminary computational experiments, we set the maximum number of iterations equal to 100 for the GRASP variants and VNS, whereas we set to 50 the maximum allowable number of iterations without an improvement for MATHOD.

Tables S1, S2 and S3, available at [17], show the related computational results in terms of objective function values and execution time.

Moreover, for MATHOD, VNS, and our metaheuristics we calculate the optimality gap as \( \left( \frac{\text{Obj}_{MH} - \text{Obj}_{CPLEX}}{\text{Obj}_{CPLEX}} \right) \times 100 \), where \( \text{Obj}_{MH} \) and \( \text{Obj}_{CPLEX} \) are the values of the objective function found by the metaheuristic and CPLEX, respectively. All the results (except for CPLEX) are the average of five different runs. All the times refer to wall clock. Looking at Tables S1 – S3 (see [17]), it is evident that all the metaheuristics are very efficient; in fact, they find a feasible solution for all instances in less than one second. All the metaheuristics are also very effective; they find the optimal solutions for all the instances with five customers (see Table S1).

We can observe a similar trend for instances with 10 customers, summarized in Table S2. All GRASP variants find the optimal solutions for almost all the instances, as long as MATHOD and VNS provides the optimal solutions for six instances and the optimality gap is 0.70% and 1.76% on average, respectively.

The results reported in Table S3 are more interesting than the previous ones, as they show the efficiency of the metaheuristics.

Indeed, the time spent by all the metaheuristics is of the order of milliseconds, while CPLEX required on average 72.16 seconds.

The proposed algorithms outperform MATHOD and VNS in terms of effectiveness, their optimality gap on average spans between 0.42% of GRASP+VNS and 0.56% of GRASP, whereas the gap for VNS is 1.74% and 3.70% for MATHOD.

In order to understand how the characteristics of the instances may influence the performance of the metaheuristics, we summarize in Table 2 the average results grouped by typology, i.e. C, R and RC. Looking at Table 2 it is evident that GRASP variants give the best results on instances R in terms of effectiveness, whereas the VNS performs worst on R type and the lowest gap is for instances C, and MATHOD performs the best on RC and the worst on C. The efficiency remains almost the same for all the typologies.

To sum up, the computational results collected on small size instances underline that all the methods outperform CPLEX in terms of efficiency and this behaviour is not affected by the instances’ characteristics (see Table 2).

In addition, our approaches are more effective than MATHOD and VNS: their average gap value is less than about 0.23%, whereas the average gap of MATHOD and VNS is 1.56% and 1.16%, respectively. The best results in terms of solution quality are obtained on the type 2 instances.

On the basis of the results collected on small size instances, we can conclude that all GRASP variants are very performing for the VRPODTWmd.
4.5. Experimental results on the medium size instances

In order to assess the effectiveness of the proposed algorithms on the medium size instances, we solve also this set of problems with CPLEX. Since preliminary computational results highlighted that CPLEX does not solve all instances in a reasonable amount of time, we impose a time limit of 1800 seconds to the solver for each instance. Within the imposed time limit, CPLEX finds the optimal solution of 13 and 4 instances with 25 and 50 customers, respectively; whereas, for 15 instances with 25 customers and 18 with 50, it returns feasible solutions. It does not determine solutions within the imposed time limit for 2 and 8 instances with 25 and 50 customers, respectively.

To make a fair comparison among the results obtained with the algorithms and those determined by CPLEX, we use as stopping criteria for the metaheuristics the execution time, considering also in this case a time limit of 1800 seconds.

The analysis of the collected results is divided into two parts. At first, we show the effectiveness of the algorithms comparing the solutions obtained with those determined by CPLEX. For this study, we consider the instances solved to optimality by CPLEX and those for which CPLEX provides a feasible solution, separately. At second, we discuss the performance of the algorithms considering all the instances.

### 4.5.1. Comparison with CPLEX

Table S4 available at [17] summarizes the average gap with respect to the solutions obtained by CPLEX. In particular, we consider gap$^1$ that represents the average gap over all instances solved to optimality by CPLEX and gap$^2$ that is the average gap over the instances for which CPLEX provides a feasible solution.

Focusing on the average results for the instances solved to optimality by CPLEX, gap$^1$ is lower than 3% for all the algorithms considering all the medium size instances, except for MATHOD. In particular, all the algorithms based on GRASP provide almost the same quality solution for all the medium size instances. However, they are more effective on instances with 50 customers, in fact the gap$^1$ is lower than 2%. Focusing on VNS, it provides a value for gap$^1$ of about 2.42% for both the instances with 25 and 50 customers, whereas MATHOD is the less effective, with a gap$^1$ equal to 6.79% and 35.89% for the instances with 25 and 50 customers, respectively.

Referring to the gap$^2$, the numerical results collected in Table S4 highlight that all the algorithms, except MATHOD, are more effective than CPLEX. In addition, all the algorithms based on GRASP produce almost the same quality solutions and outperform VNS.
The numerical results clearly highlight that MATHOD is not effective for the considered instances as CPLEX outperforms MATHOD considering both the optimal and the feasible solutions. For these reasons, we do not consider MATHOD for the subsequent analysis.

4.5.2. Algorithms comparison

The first part of the computational study was aimed at making a comparison among the results obtained with the proposed algorithms and those found by CPLEX; thus, as underlined in the previous section, we chose as stopping criterion a time limit of 1800 seconds, determined on the basis of the results collected by CPLEX. In this second part of our computational study, the aim is to analyse the behaviour of the algorithms, when different stopping criteria are applied. Thus, we decided to focus on the best solution found at given iterations; discussing the variation of the average costs and the percentage decreasing cost at varying the number of iterations. In particular, Figure 2 plots the average costs obtained by the algorithms at varying the number of iterations. We show these results for the first 100,000 iterations. Looking at Figure 2 we may observe that all the GRASP versions provide lower costs than that obtained by VNS. Focusing on the GRASP versions, GRASP+VNS behaves the best, followed by GRASP+VND. This is an expected trend, because each algorithm works on the same initial solution at each iteration. Hence, VND and VNS embedded as local search procedures in GRASP+VND and GRASP+VNS, respectively, allow to improve the initial solution, obtaining a cost lower than that provided by the local search included in GRASP. This behavior is observed for all the instances belonging to the medium size class. In addition, Figure 2 clearly shows that the decreasing in cost is more evident in the first iterations. We may observe that after a given number of iterations, that we call break-iteration-point (bip), the decreasing is less impressive. We calculate the bip by considering, at each iteration, the percentage decreasing in cost with respect to that obtained at the previous iteration. The iteration in which we obtain a decreasing in cost less than or equal to a given threshold corresponds to the bip. We analyze these results considering the first 1000 iterations, with a step size of 50 iterations. The related results are represented graphically in Figure 3.

For our study, we use three values of the threshold, i.e. 0.25%, 0.05% and 0.01%. For each value of the threshold, we consider the bip, the execution time needed to perform bip iterations, the average cost of the best solution obtained within bip iterations, and gap 3, that indicates the percentage increase of the cost associated with the best solution found within the bip iterations, with respect to the cost of the solution obtained within 1800 seconds of computation. This information is given in Table S5 available at [17].

The results reported in Table S5 show that all the GRASP versions have almost the same bip. An exception is observed for the instances with 25 customers and a threshold equal to 0.01%. Overall, the GRASP versions determine almost the same solution quality. GRASP+VNS provides the best solution for each value of the threshold and for both the instances with 25 and 50 customers. However, the distance on cost between GRASP+VNS and the other two GRASP versions decreases at decreasing value of the threshold. This trend is clearly shown for the instances with 25 customers.

Focusing on the instances with 25 customers, GRASP+VNS is the most effective; as gap 3 is below 0.74% for each value of the threshold. Whereas, gap 3 for GRASP is greater
than 0.88% for each value of the threshold. As expected, the fastest algorithm is GRASP. For all the algorithms, the execution time is very limited for each value of the threshold. Hence, 1800 seconds is an overestimated time limit. VNS behaves the worst, since

**Figure 2.** Average cost at varying the number of iterations. (a) Average cost for the instances with 25 customers. (b) Average cost for the instances with 50 customers.
Figure 3. Average percentage of cost decreasing at varying the number of iterations. (a) Average percentage of cost decreasing for the instances with 25 customers. (b) Average percentage of cost decreasing for the instances with 50 customers.

...it presents a higher $bip$ than that observed for all the GRASP versions with a value of gap$^3$ equal to 3.59%, 2.76%, and 1.68% for values of threshold equal to 0.25%, 0.05% and 0.01%, respectively.
For the instances with 50 customers, the lowest value of gap$^3$ is reached by GRASP+VNS. This behaviour is observed for each value of the threshold. Overall, gap$^3$ is lower than 4% for all the GRASP versions and it decreases to 1.91% when the threshold is 0.01%. VNS behaves the worst, with a value of gap$^3$ that ranges from 5.63% to 3.57%. Focusing on the efficiency, all the algorithms are very fast, but GRASP+VNS is the slowest one. However, the execution time is lower than 13 seconds for each value of the threshold. GRASP is the fastest with 2.62 seconds in the worst case, i.e. threshold equal to 0.01%.

We highlight that all the algorithms have room for improving gap$^3$. The execution time is below 1 and 13 seconds for all the algorithms on the instances with 25 and 50 customers, respectively.

4.6. Experimental results on the large and the very large size instances

Since CPLEX is not able to determine feasible solutions for the large and the very large instances, we solve these instances with the metaheuristics. Then, we conduct the same analysis carried out for the medium size instances. Hence, we considered as stopping criterion the execution time and we imposed a time limit of 1800 seconds.

4.6.1. Analysis on large size instances

To analyse the behaviour of the heuristics, firstly, we depict in Figure 4 the average costs obtained by the algorithms at varying the number of iterations for the instances with 100 and 200 customers.

Focusing on the instances with 100 customers (see Figure 4(a)), we may observe that GRASP and GRASP+VND have similar average costs for each iteration. An impressive reduction in cost is observed for GRASP+VNS compared with both GRASP and GRASP+VND, for each iteration. VNS performs the worst.

Whereas, looking at Figure 4(b), we may observe that for the instances with 200 customers, GRASP+VNS performs the lowest number of iterations within the imposed time limit, followed by GRASP+VND. This is an expected results, as GRASP has a lower computational cost per iteration than both GRASP+VND and GRASP+VNS. It is worth noting that, even though GRASP performs the highest number of iterations within 1800 seconds, the costs are always greater than those obtained by both GRASP+VND and GRASP+VNS. This result suggests that a more sophisticated local search embedded in a GRASP framework allows to obtain better results on large size instances than those obtained with simple local search. As for the instances with 100 customers, VNS behaves the worst.

Secondly, we focus on the average numerical results summarized in Table S6 available at [17].

Looking at Table S6, it is clear that GRASP+VNS provides solutions with the lowest cost for the instances with both 100 and 200 customers and for any value of the threshold; confirming the trend graphically observed in the Figure 4.

Focusing on the instances with 100 customers, GRASP+VNS reaches a value of gap$^3$ lower than 3% with a threshold equal to 0.01%. This value is greater than 4% for all others algorithms and for each value of the threshold. The execution time required by the GRASP+VNS for obtaining such solutions doubled with respect to the GRASP. However, the highest average execution time is 30 seconds with a gap$^3 = 2.87\%$ against 10 seconds required by GRASP for obtaining solutions with gap$^3 = 5.29\%$. 
Considering the instances with 200 customers, the difference in terms of costs among GRASP, GRASP+VND and GRASP+VNS is limited. However, GRASP+VNS shows the best performance for any value of the threshold, except for 0.01%. In this case, the
best solutions are obtained by GRASP+VND with gap$^3 = 0.55\%$ against a gap$^3 = 0.66\%$ obtained by GRASP+VNS. The execution times of the two versions are almost the same.

4.6.2. Analysis on very large size instances

Focusing on Figure 5, that depicts the average costs at varying the number of iterations for the instances with 400 customers, we may note that GRASP+VNS provides the lowest average cost for each iteration. Hence, GRASP+VNS is clearly the most effective for this class. In addition, we may observe that, even though GRASP+VNS performs a lower number of iterations within 1800 seconds than that performed by the other algorithms, the latter ones do not find more effective solutions.

The good behavior of GRASP+VNS is highlighted in Table S7 available at [17]. GRASP+VNS finds the solutions with the lowest cost. The value of gap$^3$ is lower than 0.40\% for each value of threshold; whereas it is higher than 4\% for the other algorithms. Focusing on the efficiency (see Table S7 available at [17]), the execution time of GRASP+VNS is slightly higher than that required by GRASP. In the worst case, i.e. with threshold equal to 0.01\%, GRASP+VNS requires 884 seconds against 788 seconds needed to GRASP for obtaining solutions with gap$^3 = 4.55\%$. In addition, the value of gap$^3$ for GRASP+VNS with threshold equal to 0.01\% is 0\%.

4.7. Comparison of the algorithms based on GRASP

In this section, we study the performance of GRASP+VND and GRASP+VNS compared with GRASP on medium, large, and very large size instances. For this analysis, we consider two parameters, i.e. gap and speed up. The gap is computed as $(Obj_A -
Obj\(_{\text{GRASP}}\)/Obj\(_{\text{GRASP}}\) × 100 where Obj\(_A\) is the best average solution cost obtained with algorithm \(A\), i.e. \(\text{GRASP}+\text{VND}\) and \(\text{GRASP}+\text{VNS}\), and Obj\(_{\text{GRASP}}\) that obtained with GRASP within the imposed time limit of 1800 seconds. The speed up is computed as \(t(A)/t(\text{GRASP})\) where \(t(\cdot)\) is the execution time to obtain the best average solution cost within 1800 seconds.

Figure 6 shows that both \(\text{GRASP}+\text{VND}\) and \(\text{GRASP}+\text{VNS}\) are more effective than GRASP when the number of customers increases (see Figure 6(a)). In particular, the value of gap for \(\text{GRASP}+\text{VNS}\) is 0.04% for \(|C|=25\) and it decreases to −3% for \(|C|=400\). Hence, for medium size instances \(\text{GRASP}+\text{VNS}\) finds solutions slightly worse than those determined by GRASP, however, the former finds solutions with average cost 3% lower than that obtained by GRASP for the very large size instances.

In addition, analyzing the efficiency ratio, i.e. the speed up plotted in Figure 6(b), it is less than 1.15 for each instance dimension. For the very large instances, the speed up of \(\text{GRASP}+\text{VNS}\) is about 1.05. This extra effort in terms of execution time required by \(\text{GRASP}+\text{VNS}\) with respect to GRASP, largely suffices the good quality solution obtained.

### 4.8. Time-to-target analysis

To study the converge speed of the algorithms, we performed a time-to-target analysis [48]. The test shows the probability that an algorithm will find a solution at least as good as a given target value for a given problem instance, within a given running time.

Let consider \(n\) instances \(\mathcal{I}_j\) and the corresponding targets \(\text{look}_j\). For \(j=1,\ldots,n\), let \(X_j \geq 0\) be a random variable representing the time taken by the heuristic \(H\) to find a solution as good as the target value (e.g. \(\text{look}_j\)), for instance \(\mathcal{I}_j\).

In addition, let \(F_{X_j}(x) = P(X_j \leq x)\) and \(f_{X_j}(x)\) be the cumulative distribution function and the probability density function of \(X_j\), respectively. Then, our goal is to generate a sample of the cumulative distribution function \(F_{X_1+\ldots+X_n}(x) = P(X_1 + \cdots + X_n \leq x)\).

We used the results of the testing considered in Sections 4.5 and 4.6 to select the target values. Given a generic instance \(\mathcal{I}_j\), let \(\text{Obj}_{MHx}(\mathcal{I}_j, q)\) be the objective function of the best solution found by method \(MHx\) in the \(q\)th run (each one ran five times), and let \(\text{Obj}_{MHx}(\mathcal{I}_j) = \max_{q=1,\ldots,5}\text{Obj}_{MHx}(\mathcal{I}_j, q)\) be the largest of the best solution values reached by all the runs of the algorithm.

Therefore, we define

\[
\text{look}_j = \max_{x \in \mathcal{M}} \text{Obj}_{MHx}(\mathcal{I}_j),
\]

where \(\mathcal{M} = \{\text{GRASP}, \text{GRASP}+\text{VND}, \text{GRASP}+\text{VNS}, \text{VNS}\}\).

In other words, \(\text{look}_j\) is the maximum objective function value of the best solution reached by all the methods in all the runs.

The results are shown in Figure 7. The time (in seconds) is represented on the abscissa axis, while the ordinate axis shows the probability that the algorithm reaches solutions as good as the targets in that time (summed over all instances).

We can observe the following patterns: \(\text{GRASP}+\text{VNS}\) has the highest converge rate, followed by \(\text{GRASP}+\text{VND}\) and \(\text{VNS}\). Finally, \(\text{GRASP}\) is the method with the lowest converge speed. \(\text{GRASP}+\text{VNS}\) and \(\text{GRASP}+\text{VND}\) variants require more time when run for the same number of iterations, but they are faster to converge to a good solution, since the single iteration is more effective.
Therefore, given a target objective function value, GRASP+VNS has the highest probability to reach the target in a given computational time. In particular, after 20,000 seconds of cumulative time, GRASP+VNS reached the target on all the instances for all the runs.
and GRASP+VND has a probability higher than 0.9 to reach the target for all the cases. Meanwhile, VNS and GRASP present a 0 probability to reach all the targets.

5. Conclusions

In this work, we have proposed three versions of one metaheuristic, i.e. the greedy randomized adaptive search procedure (GRASP), for the vehicle routing problem with occasional drivers, time windows, and multiple deliveries (VRPODTWmd). In addition, we have implemented a variable neighbourhood search (VNS), based on that one proposed by Macrina et al. [39] to solve the VRPODTWmd and an adapted version of the MATHOD heuristic proposed by Archetti et al. [5] to solve the VRPOD. We have proposed a basic GRASP approach, then we have embedded the VNS framework into the GRASP as well as a variable neighbourhood descend. In order to evaluate the performance of the proposed approaches, we have solved a set of small size instances with the commercial solver CPLEX, then we have carried out a comparative analysis, considering the two state of the art solution approaches adapted for our problem, i.e. VNS and MATHOD. The results have shown that our metaheuristics are highly performing in terms of effectiveness and efficiency. Overall, the heuristics are less time consuming than CPLEX, showing a low optimality gap that is less than 3%. Moreover, the metaheuristics rapidly solve bigger instances and they are more efficient and effective than the MATHOD and VNS approaches. In particular, GRASP-VND and GRASP-VNS behave the best for each instance dimension and topology. Both versions present a good trade-off between execution time and solution quality. However, the higher the instance dimension, the better GRASP-VNS than GRASP-VND. Accordingly to these results, the proposed metaheuristics seem to be very promising. As
future work, it is worth to investigate different local search strategies, i.e. new moves, different order, random choice of the neighbourhood, and so on. In addition, it is worth to investigate the effectiveness of the proposed solution approaches in solving a variant of the problem at hand proposed in [40] where split delivery strategy is considered.

**Note**

1. All the instances are available at the following link: https://bit.ly/3cE0ZlV.

**Disclosure statement**

No potential conflict of interest was reported by the author(s).

**Funding**

This work was supported by PRIN 2015.

**Notes on contributors**

Luigi Di Puglia Pugliese received the PhD degree in operations research from the University of Calabria, Italy, with a thesis entitled “Models and Methods for the Constrained Shortest Path Problem and Its Variants”. He is a researcher with the Institute for High Performance Computing and Networking ICAR, Italian National Research Council, Italy. His main research interests include the field of network optimization. Other areas of interests include logistics, combinatorial optimization, and project scheduling.

Daniele Ferone obtained his Ph.D. in Mathematics and Computer Science at the University of Naples. He spent some years as PostDoc researcher at the Universitat Oberta de Catalunya, University of Milano Bicocca, and University of Calabria. Currently, he is a researcher at the University of Naples and his main research interests are meta-heuristics, shortest path problems, and logistic optimization.

Paola Festa obtained her Ph.D. degree in Operations Research in 2000 and is Full Professor in Operations Research at the University of Napoli FEDERICO II. Since 2000, she has been teaching Operations Research and Combinatorial Optimization to undergraduate and master students of several courses, including Mathematics, Data Science, and Computer Science. Paola Festa is in the scientific committee of the Ph.D. Program in Mathematics and Computer Science (University of Napoli FEDERICO II). She has been in the scientific committee of the Ph.D. Program in Computational Biology and Bioinformatics (University of Napoli FEDERICO II) and of the Ph.D. Program in Operations Research (University of Calabria). Since 1999, she has been frequently and regularly Research Scholar at several national and international research institutions, including CNRs, MIT Lab. for Information and Decision Systems (USA), AT&T Labs Research (USA), and Department of Industrial and Systems Engineering of the University of Florida (USA). Paola Festa is Associate Editor of several international journals, including Journal of Global Optimization, Optimization Letters, ACM Journal on Experimental Algorithmics, and Journal of Biomedical Data Mining. She has been chair or member of the scientific committee of more than 60 international conferences and workshops and the organizer of more than 10 international scientific workshops/conferences. She is member of the Steering Committee of the International Workshops on Hybrid Metaheuristics. She is author/co-author of more than 90 publications appeared in international journals, books, and peer-reviewed conference proceedings.

Francesca Guerriero graduated with honours in Management Engineering at the University of Calabria (UNICAL), Italy, obtained the PhD in System Engineering and Computer Science at the same
University. She was visiting research fellow at the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, MA, USA. Currently, she is Full Professor of Operation Research at the Dept. of Mechanical, Energy and Management Engineering, University of Calabria. She is the Chair of the Dept. of Mechanical, Energy and Management Engineering, University of Calabria and the Vice-President of AIRO, the Italian Operations Research Society. Her main research interests are in the area of network optimization, logistics and distribution, revenue management, project management, optimization and big data. She is co-author of more than 140 papers published in prestigious journal in the Operations Research field. She has been and is member of the scientific committee of several International Conferences. She is member of the editorial board of several scientific journals and supervises master and PhD research projects.

**Giusy Macrina** graduated cum laude in Management Engineering in 2013 at the University of Calabria (UNICAL). She was visiting student at the Centre interuniversitaire de recherche sur les reseaux d’entreprise (CIRRELT) Montréal, and then, she obtained her PhD degree in Mathematics and Computer Science in 2018 at UNICAL. She is currently a research fellow at the Department of Mechanical Energy and Management Engineering at UNICAL. She is a member of the board of AIRO young, the young chapter of the Italian Operations Research Society. Her main area of scientific activity is Operations Research, in particular she focuses on logistics and distribution problems.

**ORCID**

Luigi Di Puglia Pugliese [http://orcid.org/0000-0002-6895-1457](http://orcid.org/0000-0002-6895-1457)

Daniele Ferone [http://orcid.org/0000-0003-4696-7826](http://orcid.org/0000-0003-4696-7826)

Paola Festa [http://orcid.org/0000-0002-2243-6054](http://orcid.org/0000-0002-2243-6054)

Francesca Guerriero [http://orcid.org/0000-0002-3887-1317](http://orcid.org/0000-0002-3887-1317)

Giusy Macrina [http://orcid.org/0000-0001-6762-3622](http://orcid.org/0000-0001-6762-3622)

**References**

[1] Y. Abu Al Hla, M. Othman, and Y. Saleh, *Optimising an eco-friendly vehicle routing problem model using regular and occasional drivers integrated with driver behaviour control*, J. Clean. Prod. 234 (2019), pp. 984–1001.

[2] M. Allahviranloo and A. Baghestani, *A dynamic crowdshipping model and daily travel behavior*, Transp. Res. Part E: Logistics Transp. Rev. 128 (2019), pp. 175–190.

[3] A. Alvarez, P. Munari, and R. Morabito, *Iterated local search and simulated annealing algorithms for the inventory routing problem*, Int. Trans. Oper. Res. 25 (2018), pp. 1785–1809.

[4] C. Archetti, F. Guerriero, and G. Macrina, *The online vehicle routing problem with occasional drivers*, Comput. Oper. Res. 127 (2020), p. 105144.

[5] C. Archetti, M. Savelsbergh, and M.G. Speranza, *The vehicle routing problem with occasional drivers*, Eur. J. Oper. Res. 254 (2016), pp. 472–480.

[6] A.M. Arslan, N. Agatz, L. Kroon, and R. Zuidwijk, *Crowdsourced delivery: a dynamic pickup and delivery problem with ad-hoc drivers*, Tech. Rep., ERIM, Report Series Reference, 2016.

[7] M. Ball, B. Golden, A. Assad, and L. Bodin, *Planning for truck fleet size in the presence of a common-carrier option*, Decision Sci. 14 (1983), pp. 103–120.

[8] A. Barr and J. Wohl, *Exclusive: Wal-Mart may get customers to deliver packages to online buyers*, 2013. REUTERS–Business.

[9] G. Bensinger, *Amazon’s next delivery drone: You*, 2015. Wall Street Journal.

[10] J.L. Bresina, *Heuristic-Biased Stochastic Sampling*, Proceedings of the Thirteenth National Conference on Artificial Intelligence, Portland, OR, 1996, pp. 271–278.

[11] H. Buldeo Rai, S. Verlinde, J. Merckx, and C. Macharis, *Crowd logistics: an opportunity for more sustainable urban freight transport?* Eur. Transp. Res. Rev. 9 (2017), p. 39.
[12] W. Chaovalitwongse, D. Kim, and P.M. Pardalos, GRASP with a new local search scheme for vehicle routing problems with time windows, J. Comb. Optim. 7 (2003), pp. 179–207.
[13] G. Clarke and J.W. Wright, Scheduling of vehicles from a central depot to a number of delivery points, Oper. Res. 12 (1964), pp. 568–581.
[14] L. Dahle, H. Andersson, and M. Christiansen, The vehicle routing problem with dynamic occasional drivers, in Computational Logistics, T. Bektaş, S. Coniglio, A. Martínez-Sykora, and S. Voß, eds., Springer International Publishing, Cham, 2017, pp. 49–63.
[15] L. Dahle, H. Andersson, M. Christiansen, and M.G. Speranza, The pickup and delivery problem with time windows and occasional drivers, Comput. Oper. Res. 109 (2019), pp. 122–133.
[16] I. Dayarian and M. Savelsbergh, Crowdsourcing and same-day delivery: employing in-store customers to deliver online orders, Production Oper. Manage. 29 (2020), pp. 2153–2174.
[17] L. Di Puglia Pugliese, D. Ferone, P. Fest, F. Guerriero, and G. Macrina, Results, 2021. Available at https://figshare.com/s/c4542e461d706c0df60a.
[18] G. Felici, D. Ferone, P. Festa, A. Napoletano, and T. Pastore, A grasp for the minimum cost sat problem, in Learning and Intelligent Optimization. LION 2017, R. Battiti, D.E. Kvasov, and Y.D. Sergeyev, eds., Lecture Notes in Computer Science Vol. 10556, Springer, Cham, 2017, pp. 64–78.
[19] T.A. Feo and M.G. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Oper. Res. Lett. 8 (1989), pp. 67–71.
[20] D. Ferone, P. Festa, and F. Guerriero, An efficient exact approach for the constrained shortest path tour problem, Optim. Methods Softw. 35 (2020), pp. 1–20.
[21] D. Ferone, P. Festa, F. Guerriero, and D. Laganà, The constrained shortest path tour problem, Comput. Oper. Res. 74 (2016), pp. 64–77.
[22] D. Ferone, P. Festa, A. Napoletano, and M.G.C. Resende, A new local search for the p-center problem based on the critical vertex concept, in Learning and Intelligent Optimization. LION 2017, R. Battiti, D.E. Kvasov, and Y.D. Sergeyev, eds., Lecture Notes in Computer Science Vol. 10556, Springer, Cham, 2017, pp. 79–92.
[23] D. Ferone, A. Gruler, P. Festa, and A.A. Juan, Enhancing and extending the classical grasp framework with biased randomization and simulation, J. Oper. Res. Soc. 70 (2019), pp. 1362–1375.
[24] P. Festa, P. Pardalos, M. Resende, and C. Ribeiro, Randomized heuristics for the max-cut problem, Optim. Methods Softw. 17 (2002), pp. 1033–1058.
[25] P. Festa and M.G.C. Resende, GRASP: an annotated bibliography, in Essays and Surveys in Metaheuristics, Springer, Boston, MA, 2002, pp. 325–367.
[26] P. Festa and M.G.C. Resende, An annotated bibliography of GRASP–part I: algorithms, Int. Trans. Oper. Res. 16 (2009), pp. 1–24.
[27] P. Festa and M.G.C. Resende, An annotated bibliography of GRASP–part II: applications, Int. Trans. Oper. Res. 16 (2009), pp. 131–172.
[28] P. Festa, T. Pastore, D. Ferone, A.A. Juan, and C. Bayliss, Integrating biased-randomized GRASP with Monte Carlo simulation for solving the vehicle routing problem with stochastic demands, in 2018 Winter Simulation Conference (WSC). 2018, pp. 2989–3000. IEEE Press.
[29] C. Gahma, C. Brabänder, and A. Tuma, Vehicle routing with private fleet, multiple common carriers offering volume discounts, and rental options, Transp. Res. Part E 97 (2017), pp. 192–216.
[30] K. Gdowska, A. Viana, and J. Pedroso, Stochastic last-mile delivery with crowdsourcing, Transp. Res. Procedia 30 (2018), pp. 90–100.
[31] D. Goeke, T. Gschwind, and M. Schneider, Upper and lower bounds for the vehicle-routing problem with private fleet and common carrier, Discrete Appl. Math. 264 (2019), pp. 43–61.
[32] E.M. González-Neira, D. Ferone, S. Hatami, and A.A. Juan, A biased-randomized simheuristic for the distributed assembly permutation flowshop problem with stochastic processing times, Simul. Model. Pract. Theory. 79 (2017), pp. 23–36.
[33] S.R.A. Haddadene, N. Labadie, and C. Prudhon, A GRASP × ILS for the vehicle routing problem with time windows, synchronization and precedence constraints, Expert. Syst. Appl. 66 (2016), pp. 274–294.
[34] K. He and W. Huang, Solving the single-container loading problem by a fast heuristic method, Optim. Methods Softw. 25 (2010), pp. 263–277.
[35] J. Homberger and H. Gehring, A two-phase hybrid metaheuristic for the vehicle routing problem with time windows, Eur. J. Oper. Res. 162 (2015), pp. 220–238.

[36] R. Landa, Thinking Creatively in the Digital Age, 1st ed., Nimble, Blue Ash, 2015.

[37] Z. Li, S. Guo, F. Wang, and A. Lim, Improved GRASP with Tabu search for vehicle routing with both time window and limited number of vehicles, in Innovations in Applied Artificial Intelligence, B. Orchard, C. Yang, and M. Ali, eds., Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 552–561.

[38] M. López-Ibáñez, J. Dubois-Lacoste, L.P. Cáceres, M. Birattari, and T. Stutzle, The irace package: iterated racing for automatic algorithm configuration, Oper. Res. Perspect. 3 (2016), pp. 43–58.

[39] G. Macrina, L. Di Puglia Pugliese, and F. Guerriero, A Variable Neighborhood Search for the Vehicle Routing Problem with Occasional Drivers and Time Windows, Proceedings of the 9th International Conference on Operations Research and Enterprise Systems–Volume 1: ICORES. INSTICC. SciTePress, Valletta, Malta, 2020, pp. 270–277.

[40] G. Macrina, L. Di Puglia Pugliese, F. Guerriero, and D. Laganà, The vehicle routing problem with occasional drivers and time windows, in Optimization and Decision Science: Methodologies and Applications, A. Sforza and C. Sterle, eds., Springer Proceedings in Mathematics & Statistics Vol. 217, Springer, Cham, Sorrento, 2017, pp. 577–587.

[41] G. Macrina, L. Di Puglia Pugliese, F. Guerriero, and G. Laporte, Crowd-shipping with time windows and transshipment nodes, Comput. Oper. Res. 113 (2020), p. 104806.

[42] G. Macrina and F. Guerriero, The green vehicle routing problem with occasional drivers, in New Trends in Emerging Complex Real Life Problems, P. Daniele and L. Scrimali, eds. Springer International Publishing, Springer New York LLC, 2018.

[43] N. Mladenović, R. Todosijević, and D. Urošević, Less is more: basic variable neighborhood search for minimum differential dispersion problem, Inf. Sci. (NY)326 (2016), pp. 160–171.

[44] A. Napoletano, A. Martinez-Gavara, P. Festa, T. Pastore, and R. Martí, Heuristics for the constrained incremental graph drawing problem, Eur. J. Oper. Res. 274 (2019), pp. 710–729.

[45] V.P. Nguyen, C. Prins, and C. Prodhon, Solving the two-echelon location routing problem by a grasp reinforced by a learning process and path relinking, Eur. J. Oper. Res. 216 (2012), pp. 113–126.

[46] N. Radiojiči, A. Djenić, and M. Marić, Fuzzy grasp with path relinking for the risk-constrained cash-in-transit vehicle routing problem, Appl. Soft. Comput.72 (2018), pp. 486–497.

[47] M.G. Resende and C.C. Ribeiro, Greedy Randomized Adaptive Search Procedures: Advances and Extensions, Vol. 272, Boston, MA, 2019.

[48] A. Reyes and C.C. Ribeiro, Extending time-to-target plots to multiple instances, Int. Trans. Oper. Res. 25 (2018), pp. 1515–1536.

[49] S. Ropke and J. Cordeau, Branch and cut and price for the pickup and delivery problem with time windows, Transp. Sci. 43 (2009), pp. 267–286.

[50] A. Sampaio, M. Savelsbergh, L. Veeletrurf, and T. Van Woensel, Crowd-based city logistics, in Sustainable Transportation and Smart Logistics, J. Faulin, S.E. Grasman, A.A. Juan, and P. Hirsch, eds., chap. 15, Oxford University Press, Oxford, 2019, pp. 381–400.

[51] M.W.P. Savelsbergh and T. Van Woensel, City logistics: challenges and opportunities, Transp. Sci. 50 (2016), pp. 579–590.

[52] S. Serafini, M. Nigro, V. Gatta, and E. Marcucci, Sustainable crowdsourcing using public transport: a case study evaluation in rome, Transp. Res. Procedia 30 (2018), pp. 101–110.

[53] M. Slabinac, Innovative Solutions for a 'Last-Mile' Delivery–A European Experience, 15th International Scientific Conference 'Business Logistics in Modern Management'. Josip Juraj Strossmayer University of Osijek, Croatia, 2015, p. 111–129.

[54] M.M. Solomon, Algorithms for the vehicle routing and scheduling problems with time window constraints, Oper. Res. 35 (1987), pp. 254–265.

[55] A. Subramanian, L. Drummond, C. Bentes, L. Ochi, and R. Farias, A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery, Comput. Oper. Res. 37 (2010), pp. 1899–1911.