Received: 25 May 2020 | Accepted: 26 May 2020

DOI: 10.1002/net.21962

SPECIAL ISSUE ARTICLE

WILEY

Exact and hyper-heuristic solutions for the distribution-installation problem from the VeRoLog 2019 challenge

Ahmed Kheiri[1,2] | **Leena Ahmed**[3] | **Burak Boyacı**[1,2] | **Joaquim Gromicho**[4,5] | **Christine Mumford**[3] | **Ender Özcan**[6] | **Ali Selim Dirikoç**[2]

[1]Centre for Transport and Logistics, Department of Management Science, Lancaster University, Lancaster, UK
[2]Department of Management Science, Lancaster University, Lancaster, UK
[3]School of Computer Science, Cardiff University, Cardiff, UK
[4]VU Amsterdam, Amsterdam, The Netherlands
[5]ORTEC, Zoetermeer, The Netherlands
[6]School of Computer Science, University of Nottingham, Nottingham, UK

Correspondence
Ahmed Kheiri, Centre for Transport and Logistics, Department of Management Science, Lancaster University, Lancaster LA1 4YX, UK.
Email: a.kheiri@lancaster.ac.uk

Funding information
This research was supported by the KTP Scheme, Grant/Award Number: KTP11692.

Abstract

This work tackles a rich vehicle routing problem (VRP) problem integrating a capacitated vehicle routing problem with time windows (CVRPTW), and a service technician routing and scheduling problem (STRSP) for delivering various equipment based on customers' requests, and the subsequent installation by a number of technicians. The main objective is to reduce the overall costs of hired resources, and the total transportation costs of trucks/technicians. The problem was the topic of the fourth edition of the VeRoLog Solver Challenge in cooperation with the ORTEC company. Our contribution to research is the development of a mathematical model for this problem and a novel hyper-heuristic algorithm to solve the problem based on a population of solutions. Experimental results on two datasets of small and real-world size revealed the success of the hyper-heuristic approach in finding optimal solutions in a shorter computational time, when compared to our exact model. The results of the large size dataset were also compared to the results of the eight finalists in the competition and were found to be competitive, proving the potential of our developed hyper-heuristic framework.

KEYWORDS

hyper-heuristics, metaheuristics, optimization, routing, transportation, VRP

1 | INTRODUCTION

VeRoLog, the Euro Working Group on Vehicle Routing and Logistics optimization, has been organizing challenges for the routing community, where each challenge aims to promote the design and development of an applicable and effective algorithm for a particular vehicle routing problem (VRP). The organization of the challenges is a collaborative effort with companies, such as PTV, the leading German company in developing software solutions and consultations in traffic, transportation, and logistics, who organized the first and second editions in 2014 and 2015. The third edition in 2017 was organized by ORTEC, one of the largest providers of advanced planning and optimization solutions and services. They provided a real-world VRP problem involving the pickup and delivery of tools to measure milk quality at a number of farms, for a cattle improvement company.

This paper addresses the problem introduced in the VeRoLog Solver Challenge 2019,[1] which was organized by VeRoLog in cooperation with ORTEC. The challenge presented a new and exciting variant of a vehicle routing problem (VRP), based on

[1]https://verolog2019.ortec.com/ - last access: February 28, 2020

a real-world problem of one of ORTEC's clients. This problem is about the delivery of equipment, such as vending machines, to satisfy customers' requests, and the scheduling of a number of technicians each with a certain set of skills, who are required to install the equipment at least a day after the delivery. More formally, the overall problem is an integrated version of the capacitated vehicle routing problem with time windows (CVRPTW) [41], and the service technician routing and scheduling problem (STRSP) [20]. The problem is highly complex, combining two interacting and co-dependent NP-hard routing problems into a single model, each problem having its own set of constraints, making it a unique and challenging topic for VRP researchers and practitioners. The key issue in this challenge is to find efficient and low cost routes for both the delivery and installation, while scheduling the appropriate technicians according to their skills and working days for a given instance. The main objective is related to the total costs, aiming to reduce the total number of dispatched trucks and hired technicians on a single day, and within the planning horizon, in addition to minimizing the penalties incurred due to equipment awaiting installation. Many companies operating in the delivery and equipment installation businesses would benefit significantly from an effective and efficient solution to the proposed problem.

For such real-world complex problems, exact approaches, such as, mathematical programming often fail to provide solutions as the size of the instances get larger, and so heuristic-based search methods are preferred. As we have investigated different solution methods for the integrated problem from the VeRoLog Solver Challenge, we had a similar observation. Hence, in this study, we provide a mathematical model for the problem and report the results on a number of small instances. Additionally, we also present a population-based hyper-heuristic approach for the large scale problem instances. The proposed solution methods indeed provide competitive results with respect to the highest ranked scores for each challenge instance.

The remainder of the paper is organized as follows: Section 2 reviews the previous literature on the VRP, focusing on the CVRPTW and the technician routing problem. Section 3 provides a description of the tackled problem and Section 4 formulates the mathematical model, defining the objectives and the problem constraints. Section 5 describes the applied hyper-heuristic framework. Finally Sections 6 and 7 present the results and conclusion, respectively.

2 | RELATED WORK

The problem we deal with from the VeRoLog Solver Challenge 2019 is a unique vehicle routing problem (VRP), that combines a pickup and delivery problem with time windows with the scheduling of technicians to install the delivered equipment. However, it has much in common with some well-known variants of the VRP, thus in the following subsection we provide a brief overview of relevant previous studies. Following this, we cover selection hyper-heuristics in relation to this study.

2.1 | An overview of history and variants of VRP

The VRP (vehicle routing problem) is a generic term for a class of combinatorial problems that are concerned with the design of efficient routes for a number of vehicles serving a set of customers' requests, originating and ending at a depot location. A solution to a VRP instance is a tour for every vehicle, such that all customers' locations are visited, and each vehicle finishes its tour at a depot. An optimal solution is the one in which the total distance of all tours is minimized along with the associated costs. The term VRP was first introduced in [23] as a truck dispatching problem, where they modeled the problem of how a homogeneous fleet of vehicles can serve the oil demand of a number of gas stations from a central hub, with a minimum traveling distance. This method was then generalized in [19] to a linear optimization problem: how to serve a number of customers located around a central depot, using a fleet of vehicles with varying capacities. This has been known since then as the VRP problem which is one of the most researched topics in the field of operations research and logistics. Over the past decades, the VRP has grown more popular in the literature, and other variations have been developed to model real-life scenarios.

The capacitated VRP (CVRP), which imposes a capacity constraint on the size of vehicles, is considered one of the elementary variants of VRP from which other variants originated. Among exact mathematical approaches to the CVRP, branch-and-cut [7,45] and algorithms based on the set partitioning formulation [9,30] are the most popular approaches. Many studies also applied heuristic methods [29] and genetic algorithms (GAs) have also been widely used, usually by combining them with local search techniques [46,49,50,55].

A generalization of the CVRP is the CVRP with Time Windows, which imposes a time interval ("time window") on the delivery of each customer's request. Exact methods for the CVRPTW have been successful for cases with up to 100 customers [39], and as a result heuristic and metaheuristic methods have been preferred for solving instances of larger scale. Examples of heuristic methods applied to the CVRPTW problem can be found in [54,57,59,60], and other studies that applied metaheuristic methods such as genetic algorithms, ant colony optimization, tabu search, and simulated annealing are in [10,18,24,62].

In our proposed problem, customers' deliveries are scheduled within a planning period, with each customer requiring one or more visits during this period, similar to the multiperiod VRP problem (MPVRP). In contrast to the MPVRP, where service

days are known and the frequency of customers visits is predetermined, the visits in our case are scheduled within predefined time windows. The MPVRP is a well-studied variant in the literature of VRP. In a paper by Rahimi-Vahed et al. [56], a path relinking algorithm is applied to the multidepot periodic vehicle routing problem. This is done by generating a reference set of elite solutions, and combining characteristics from those solutions to find even better solutions. The computational results show that this method produces good results in both run-time and solution quality. Archetti et al. [6] present three ways to formulate the multiperiod vehicle routing problem with time windows, then they solve the problem using a branch-and-cut algorithm. The algorithm was able to find good solutions for small problems with 10 customer orders, but was unable to find many good solutions for larger problems. Alonso et al. [2] present a tabu search algorithm for the periodic vehicle routing problem with multiple trips and accessibility restrictions such that not every vehicle can visit every customer. When tested on randomly generated test cases, it performed reasonably well with regards to solution quality. Furthermore the computation time was manageable for instances up to 1000 orders. We refer the reader to [15] for more literature on MPVRP.

The multicompartment VRP, and the multicommodity VRP concentrate on delivering different types of commodities to the customer either using a single vehicle, or by splitting them to several vehicles, thus requiring multiple visits to the same customer. Mirzaei and Wøhlk [47] conducted research on two variants of the MCVRP, one concentrates on split deliveries for different commodities, and the second focuses on delivering all commodities by a single vehicle. They proposed a branch-and-price method and compared the optimal costs of the two variants. The computational results were presented for instances with up to 100 customers, and the algorithm optimally solved instances with up to 50 customers and four commodities. Heuristic examples include [16] who proposed an iterated local search algorithm for solving the multicommodity multitrip VRP with the objective of minimizing the number of used vehicles. In [32] the authors addressed the commodity constrained split delivery VRP, where multiple commodities can be mixed in a single vehicle while satisfying the capacity constraint, and similar to our problem, each customer can be visited more than once, and each visit should deliver only one commodity type. They proposed a heuristic based on adaptive large neighborhood search (ALNS) and tested their approach on benchmark instances. Among the metaheuristic methods applied to the MCVRP, genetic algorithms are the most common so far. One example can be found in [69], which describes a VRP encountered in frozen food delivery. Similar to our model, they associated a penalty cost for late delivery based on the types of products. A GA was proposed for solving the model on instances with real data.

Our problem involves loading-unloading operations from the depot to a customer location, similar to the classical VRP. The difference in our case is that a single customer demand can be split into several requests, if the customer requires more than one machine type, thus several visits by different vehicles may be required. This draws a similarity to the class of VRP problems with split deliveries (SDVRP) [26]. In the SDVRP, the constraint that each customer is visited by only one vehicle is relaxed, and thus customers' demand can be split between several vehicles for delivery. The first heuristic approaches to solve the SDVRP were introduced by Dror and Trudeau [26,27]. After these studies, most of the subsequent work focused on metaheuristic or hybrid schemes. One example is the work of Archetti and Speranza [3] who applied a tabu search algorithm, and Boudia et al. [13] used a genetic algorithm combined with a local search procedure. Hybrid algorithms have since grown in popularity. Examples are found in [4,18]. Many exact models have also been proposed for this problem and one example is the study in [5]. For further literature on SDVRP we refer to [3].

Recently, attention has been paid to a class of VRPs that model multiple vehicle trips under the name "multiple trips vehicle routing problem" (MTVRP). A clear improvement can be obtained by allowing a single vehicle to perform multiple trips, especially when the vehicle capacity is limited and the replenishment of stock is required. The problem assumes that trucks can visit the depot more than once in the time horizon of the problem. Example of studies that modeled multiple depot visits are [61,63]. A survey paper on the literature of MTVRP for further details can be found in [16].

Finally, we mention the workforce routing and scheduling problem (WSRP), which is the focus of the second part of our model that involves the routing of the technicians for the installation of machines at the delivery points. There are some scenarios where personnel are required to perform tasks in certain locations, and hence require some sort of transportation to these locations. These scenarios are known as the workforce routing and scheduling problem (WRSP), which has become a widespread term used by many service providers.

A similar problem to the WRSP, is the service technician routing and scheduling problem (STRSP), which involves designing the least cost routes for vehicles carrying a number of service technicians. Each task demands the allocation of a technician with the required skills set, and this may also be associated with a time window. One of the pioneering publications in this field is the work of Cordeau et al. [20] who solved a real life technician scheduling problem for a large telecommunication company set as a competition by the French Operational Research Society in 2007. In this paper an adaptive large neighborhood search algorithm is implemented. In [68], the authors concentrated on a field technician scheduling problem in the telecommunications industry, and their purpose was to maximize the number of served requests as well as considering the request's priority and the technician's skill level. A local search algorithm, a Greedy randomized adaptive search procedure (GRASP) and a greedy heuristic algorithm were proposed to solve the problem. Kovacs et al. [40] studied the service technician routing and scheduling problem with the objective of minimizing the total routing and outsourcing costs. The authors used an adaptive large neighborhood search

algorithm for solving the problem on artificial and real-world instances. Pillac et al. [52] proposed a parallel matheuristic approach for solving a variant of the TRSP in which a number of technicians with a set of accompanying skills, tools and spare parts need to be scheduled and routed within given time windows. The study dealt with the availability of tools and spare parts for the technicians and routing them to the depot for the replenishment of tools. Xie et al. [67] used an iterated local search algorithm to solve the TRSP. They studied a variant where it was given which technicians can serve which orders. The algorithm was benchmarked on instances ranging from 25 to 100 orders and compared to an ALNS algorithm, where it was found that it performs significantly better on large instances with fast computational times.

In addition to technician routing, similar problems can also be found in other fields where scheduling is important, such as the home health care [11], and the scheduling of security personnel [48].

The problem studied in this paper is a unique version considering its set of constraints and the integration with the staff rostering and routing problem. To the best of our knowledge there is no similar version investigated in the literature, and the best matching study to our problem description is by Bae and Moon [8] where they extended the multidepot vehicle routing problem with time windows to a problem of service vehicles used for delivery and installation of electronics. They developed a mixed integer programming model, a heuristic and a genetic algorithm, and compared their performances. There are differences between this study and our problem, for example we consider a longer planning horizon, and deal with multiple types of machines. We also allow multiple visits to the customer by different vehicles, while their model only allows a single visit for both delivery and installation. They also assign a maximum period of time between delivery and installation that must not be exceeded, while in our model we restrict this time by imposing a penalty.

2.2 | Selection hyper-heuristics

Hyper-heuristics are general purpose search methodologies for solving difficult combinatorial problems. They operate at an abstract and higher level than heuristics, that is, over the low level heuristics space [25]. One of the earliest hyper-heuristic frameworks proposed requires that hyper-heuristics should not use any specific knowledge from the solution domain [21], a feature which makes them applicable to problem instances with different characteristics or even different problems, without a need for further algorithmic or parametric adjustments. This feature forms a principle concept of hyper-heuristics in past and modern research. Hyper-heuristics are defined as "heuristics to choose heuristics" in [21], although the first attempt to design hyper-heuristics dates to as early as 1963 [28]. Burke et al. [14] identifies two categories based on the nature of the heuristic search space: *selection* and *generation* hyper-heuristics. In the former class, a heuristic is selected from an existing repository of heuristics to try to discover the behavior of these heuristics in order to enable/disable some of them during the search process; while in the latter, new heuristics are built by discovering the characteristics of the input heuristics. The approach in this study uses the former type of hyper-heuristics, which are based on a single-point based search framework with two consecutive operations that work iteratively to improve a single initial solution through *heuristic selection* and *move acceptance*. With the existence of a defined number of low level heuristics, the selection method chooses one of these heuristics and applies it to a solution in hand, generating a new solution. The move acceptance decides on the acceptance of the new solution based on the fitness/objective evaluation. Heuristic selection can be carried out using simple methods such as random selection or by selecting from a predefined ordering of the low level heuristics, or it can incorporate learning by defining some probability measures for the performance of low level heuristics. For the most recent advances and classification of selection hyper-heuristics we refer to [25].

There are different criteria to classify selection hyper-heuristics, and one of them is the solution nature, where selection hyper-heuristics are classified based on this measure into single point or multiple point. Multiple point (population) selection hyper-heuristics utilize multiple current solutions during the search, while single point (single solution) hyper-heuristics are based on a single current solution that is iteratively improved during the search. The majority of the previous studies on selection hyper-heuristics present approaches based on single-point-based search, and only a few used a population of solutions or a mixed approach alternating between using single and multiple solutions for the search. Moreover, those previously proposed population based approaches are mostly a hybrid between a selection hyper-heuristic and an evolutionary algorithm framework.

Cowling et al. [22] investigated a genetic algorithm based on hyper-heuristics for the personnel scheduling problem. A GA is implemented and applied as a high level selector, and a set of low level heuristics are used at each generation to locally improve the quality of each individual, where the low level heuristics are applied in any sequence. Sabar and Kendall [58] proposed a Monte Carlo tree search hyper-heuristic framework that tries to identify good sequences of heuristics using the Monte Carlo search tree. A memory mechanism containing a population of solutions is utilized, and at each iteration a solution from the population is selected, and the population is subsequently updated using several updating rules. Lei et al. [43] proposed a memetic algorithm based on hyper-heuristics to solve an examination timetabling problem. Their approach constructs several heuristic lists based on graph coloring heuristics and applies evolutionary operators to generate new lists. A local search method is used to further optimize the solutions. Hsiao et al. [33] implemented a hyper-heuristic based on variable neighborhood search

(VNS) iterating in two stages, first using a population of solutions, and the second stage uses only a single solution. Their approach consists of two main steps, *shaking* and *local search*. The shaking phase improves the exploration of the search space, and the local search step looks for the local optima. A population of solutions is used in the shaking stage, where the authors argued that the diversity of solutions is important in the first stages of the search to explore the right search path, and after a period of time the best solution is picked from the population. Tournament selection is used to filter unfit solutions from the population. Lehrbaum and Musliu [42] introduced a hyper-heuristic that alternates between working on a single solution and a population of solutions. Their algorithm starts by scoring the available local search heuristics, and a serial phase working with single solutions starts by applying the heuristics sequentially according to their quality scores. A parallel phase uses a population of solutions, and a heuristic is applied to each individual in the population. The algorithm switches back to the serial phase whenever a global improvement is found (i.e., better than the best found solution so far) .

We also discuss here the successful application of hyper-heuristics in different variants of VRP problems, since the VRP is considered an active research field in hyper-heuristics. In [53] the adaptive very large neighborhood search hyper-heuristic was successful in finding the state-of-the-art results for many benchmark instances of several variants of VRP. Also the CVRP with time windows is one of HyFlex (hyper-heuristic flexible framework) problem domains, for which newly developed approaches in hyper-heuristics are tested [65]. Other VRP problems solved using selection hyper-heuristics include the periodic VRP [17], dial a ride [64], urban transit routing [1], and inventory routing [34].

The previous VeRoLog challenge 2016-2017 tackled a rich VRP problem related to a cattle improvement company that regularly measures the milk quality at a number of farms using specialized tools. These tools have to be delivered to a number of farms (customers) on request and picked up again a few days after delivery. The key challenge is how to schedule the deliveries to satisfy the requests, while at the same time design efficient routes for the pick-ups and deliveries. The second place winner on this challenge used a hyper-heuristic approach based on an online selection method [38].

3 | DESCRIPTION OF THE PROBLEM

The real-world problem from VeRoLog Solver Challenge 2019 can formally be stated as follows (see Figure 1). There are a number of customers $Cr = \{cr_1, cr_2, \ldots, cr_{|Cr|}\}$ geographically spread at different locations, and a depot located at H_0. The distance between any two locations i and j is given by $d_{i,j}$. The purpose is to respond to customers' requests by delivering machines and getting them installed by a technician within a defined time horizon T given as a consecutive number of days.

An unlimited number of identical trucks (i.e., vehicles) $K = \{k_1, k_2, \ldots\}$ can be hired to transport the machines to the customers. They are located at the depot each with a maximum capacity C. Also, a number of machines $M = \{m_1, m_2, \ldots, m_{|M|}\}$ are available to be delivered to the customers at their request, and there are different types of machines with different sizes expressed in the same size unit as the truck capacity. The machines are all located in the depot, with enough machines to satisfy all the demand. A set of customer requests $R = \{r_1, r_2, \ldots, r_{|R|}\}$ should be satisfied. The requests are known at the start of the planning period. A single request $r_i = \{cr_i, w_i, m_i, n_i\}$ asks for one machine type $m_i \in M$, of quantity n_i, for exactly one customer cr_i, and w_i is the associated time window to deliver the request, where w_i is specified by the earliest day e_i and the latest day l_i to deliver request r_i. A request of the same type of machines cannot be split and should be delivered by the same truck, and if a customer requires another machine type, a separate request is made. Each truck journey on a day should start and end at the depot location H_0 and can carry different types of machines to satisfy several customers' requests, and each request occupies c_i of the truck capacity, and should not exceed its maximum capacity C. The truck can return back to the depot location multiple times during the day to pick up more machines. Also, there is a limit D on the maximum distance a single truck can travel per day. It does not take any time to load a machine at the depot or to unload a machine at a customer.

There is a fixed number of technicians $S = \{s_1, s_2, \ldots, s_{|S|}\}$ who are responsible for installing the delivered machines, at the customer location, at least 1 day after the delivery. Each technician $s_i \in S$ is located at a certain home location H_{s_i}. A technician's daily route starts and ends at his/her home location, and like trucks, there is a maximum distance D_{s_i} the technician can travel per day. In addition, there is a maximum limit N_{s_i} on the number of requests a technician can carry each day, where carrying out a request means installing all the machines for that request. The technician can maximally work for five consecutive days, and must have 2 days off if he/she has worked for five consecutive days.

Each technician has a skill set for installing certain types of machines. a_{sm} refers to technician $s \in S$ installing machine $m \in M$, and is equal 1 if the technician is eligible to install this machine, and zero otherwise. Installing a machine does not take any time. A technician is described with the following entry $s = \{H_s, D_s, N_s, \{a_{sm_1}, a_{sm_2}, \ldots, a_{sm_{|M|}}\}\}$ referring respectively to the technician home location, maximum travel distance per day, maximum number of installations per day, and which machines they have the skill to install.

The last point to mention, is that the technician should install a delivered machine as soon as possible after the delivery, and for each delayed installation of request r_i, a penalty Cl_i is added to the cost, and each machine type has a different penalty value.

The main objective is to reduce the overall costs associated with trucks, technicians and idle machines costs. The trucks/technicians total cost is constituted of the following parts: the cost of hiring a truck/technician per day, the cost of hiring a truck/technician within the planning horizon T, and the cost per unit of distance for the traveling of truck/technician. The distance between coordinates (x_1, y_1) and (x_2, y_2) is defined as the ceiling of the Euclidean distance, $\lceil \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \rceil$. In addition, there is the cost for penalizing idle machines that remain without installation for more than 1 day. This penalty cost is dependent on which machine it is and the number of days it was idle. The objective function is described by Equation (1) in the following section.

A solution gives, for each day in the planning horizon, the routes followed by each truck/technician (see Figure 2). Assuming that {1,6,7,0,1,2} is the route of a single truck in one of the planning days. The first element in the route "1" is the truck ID, followed by the requests ID's that this truck served. The ID "0" refers to the depot, and it means that truck 1 visited the depot after serving requests "6" and "7" and was loaded to serve requests "1" and "2." Each series of requests before the truck goes back to the depot is named "tour". In this route, there are two tours given as {6, 7} and {1, 2}. The start and end of the truck journey at the depot is not explicitly written in this route format.

The technician routes are very similar, starting with the technician ID, followed by the ID's of the requests that this technician served. Also, the start and end of the technician journey at their home location is not explicitly mentioned in the solution.

3.1 | Problem instances

We have used two datasets of instances, one of them has been developed for this work and one, referred to as the *hidden* dataset, was used in the competition to evaluate the participants' algorithms in the VeRoLog Solver Challenge. Each of the instances provides different types of information such as the weights of the objective function components, the maximum truck capacity, the number of days in the planning horizon, and the maximum travel distance allowed by each truck. The details of the requests, locations given as x, y coordinates (i.e., depot, technicians homes, customers) and technicians are also given. The characteristics of these datasets are provided in Table 1.

The *small* dataset, which includes instances of sizes varying between 6 and 16 requests, is developed specifically for this work. The reason for generating this dataset is to provide an ideal size of instances for testing the mathematical model which can only be applied on instances of such sizes. It is also essential to test our developed hyper-heuristic approach on instances with different characteristics and scales and to compare its performance to the exact model by its ability of finding optimal solutions in a short duration of time.

The hidden dataset was used to assess the performance of the competitors algorithms and rank the finalists in the restricted resource challenge.[2] This dataset contains instances of large sizes up to 900 requests. The number of different types of machines vary between 3 and 7 in each instance, and the number of technicians range from 25 to 125. The highest variation can be found in the costs of using trucks and technicians, distance costs, and the costs per day that trucks and technicians are used. These values range from 10 to 100 000. We refer the reader to [31] for a comprehensive description of the problem and the formal challenge rules.[3]

4 | MATHEMATICAL FORMULATION OF THE CVRP FOR DELIVERY AND INSTALLATION OF MACHINES

4.1 | Sets and indices

R: requests ($i, j \in R$).
R_0: requests and the depot ($R_0 = H_0 \cup R$).
K: vehicles ($k \in K, |K| = \max_t\{M_t\}$).
S: technicians ($s \in S$).
R_s: requests that technician s can install and the home location of technician s ($R_s = \{s, i : a_{si} = 1, \forall i \in R\} \cup H_s$).

4.2 | Parameters

T: number of days in the entire planning horizon.
H_0: location of the depot.
D: maximum distance a vehicle can travel per day.
M_t: upper bound on the number of visits a vehicle can do to depot on day t.

[2]The solvers of the finalists were run on the hidden dataset for a limited computational times determined by the challenge rules
[3]A detailed description of the challenge and the datasets is also provided here: https://verolog2019.ortec.com/

Table 1 Characteristics of the small and hidden instances

Instance	Days	Truck capacity	Truck max distance	Truck distance cost	Truck day cost	Truck Cost	Technician distance cost	Technician day cost	Technician cost	Machines	Locations	Requests	Technicians
Small_01	4	18	1090	10 000	1000	1000	10 000	10 000	10	3	5	6	5
Small_02	5	18	905	100	10 000	10	1000	100	100	4	8	8	10
Small_03	6	18	2000	10 000	100	100	100	1000	1000	5	10	10	15
Small_04	7	18	2000	10 000	10	1000	10 000	10 000	10 000	6	9	12	20
Small_05	4	18	2000	1000	10 000	10 000	100 000	100	100 000	7	10	14	25
Small_06	5	18	2000	10	1000	100 000	10	1000	10	3	10	16	5
Small_07	6	18	2000	100 000	100	10 000	10 000	100	100	4	6	6	10
Small_08	7	18	1045	1000	10	1000	100 000	10	1000	5	8	8	15
Small_09	4	18	970	10	100	100 000	1000	100	10 000	6	8	10	20
Small_10	5	18	2000	100	100 000	10	100 000	10	100 000	7	10	12	25
Small_11	6	18	1195	100 000	1000	100 000	100 000	100 000	10	3	9	14	5
Small_12	7	18	980	1000	10 000	1000	10 000	1000	100	4	9	16	10
Small_13	4	18	2000	100 000	10 000	10 000	100	10	1000	5	4	6	15
Small_14	5	18	465	100	1000	100	10	10	10 000	6	8	8	20
Small_15	6	18	620	10	1000	10	1000	100 000	100 000	7	8	10	25
Small_16	7	18	775	100 000	100 000	100	10	100 000	10	3	7	12	5
Small_17	4	18	2000	10	10 000	10	1000	100 000	100	4	9	14	10
Small_18	5	18	1135	100 000	100 000	100 000	100	1000	1000	5	13	16	15
Small_19	6	18	830	100	100 000	100 000	100	1000	10 000	6	8	6	20
Small_20	7	18	2000	1000	100	10 000	100	10 000	100 000	7	9	8	25
Small_21	4	18	2000	100	100 000	10 000	10	100 000	10	3	8	10	5
Small_22	5	18	980	1000	10	100	1000	10 000	100	4	11	12	10
Small_23	6	18	2000	10 000	10	100	10	100	1000	5	10	14	15
Small_24	7	18	960	10 000	10	1000	100 000	10 000	10 000	6	12	16	20
Small_25	4	18	2000	10	100	10	10 000	10	100 000	7	10	6	25
Hidden_01	15	18	1620	1000	1000	100	100	10	10	3	54	150	25
Hidden_02	25	18	1350	10	100 000	1000	100 000	1000	100	4	112	300	50
Hidden_03	35	18	1060	10 000	1000	100 000	100	1000	1000	5	163	450	75
Hidden_04	45	18	1040	10	10 000	100	10	10 000	10 000	6	217	600	100
Hidden_05	55	18	2000	1000	100 000	10 000	100 000	100 000	100 000	7	270	750	125
Hidden_06	15	18	1205	100	10	10	10	10	10	3	306	900	25
Hidden_07	25	18	980	1000	100 000	1000	10 000	1000	100	4	59	150	50
Hidden_08	35	18	1030	10 000	100	100 000	1000	100	1000	5	116	300	75
Hidden_09	45	18	2000	1000	100	1000	100 000	100	10 000	6	167	450	100
Hidden_10	55	18	950	10	10 000	100 000	1000	10 000	100 000	7	220	600	125
Hidden_11	15	18	2000	10 000	10 000	10	100 000	100	10	3	254	750	25

Table 1 (continued)

Instance	Days	Truck capacity	Truck max distance	Truck distance cost	Truck day cost	Truck Cost	Technician distance cost	Technician day cost	Technician cost	Machines	Locations	Requests	Technicians
Hidden_12	25	18	1405	10 000	100	10 000	1000	100 000	100	4	310	900	50
Hidden_13	35	18	2000	100 000	100	1000	10 000	100 000	1000	5	68	150	75
Hidden_14	45	18	1430	10 000	1000	100 000	100	100 000	10 000	6	117	300	100
Hidden_15	55	18	1350	1000	100	10	10	10	100 000	7	173	450	125
Hidden_16	15	18	1170	100	100 000	100	1000	10 000	10	3	205	600	25
Hidden_17	25	18	2000	100 000	10 000	100	10 000	100	100	4	260	750	50
Hidden_18	35	18	1435	100	100 000	1000	10	10	1000	5	313	900	75
Hidden_19	45	18	1010	100 000	10	10	10 000	100 000	10 000	6	60	150	100
Hidden_20	55	18	1205	10	10	100	10 000	1000	100 000	7	125	300	125
Hidden_21	15	18	1230	100	1000	10 000	1000	100	10	3	154	450	25
Hidden_22	25	18	1500	10	10	10 000	100	1000	100	4	206	600	50
Hidden_23	35	18	1100	100 000	1000	100 000	100	10 000	1000	5	266	750	75
Hidden_24	45	18	1290	100 000	10	10	10	10	10 000	6	317	900	100
Hidden_25	55	18	1160	100	10 000	10 000	100 000	10 000	100 000	7	68	150	125

d_{ij}: distance between request/depot/home i and j.
e_i: earliest (first) day that request i can be delivered.
l_i: latest (last) day that request i can be delivered.
C: vehicle capacity.
c_i: capacity needed to deliver request i.
a_{si}: 1, if technician s is eligible to satisfy request i; 0, otherwise.
H_s: home location of technician s.
D_s: maximum distance that technician s can travel per day.
N_s: maximum number of installations that technician s can do per day.
CI_i: cost of delaying the installation of request i per day.
CV: cost of using a vehicle any day during the planning horizon.
CT: cost of using a technician any day during the planning horizon.
CVU: cost of using a vehicle per day.
CTU: cost of using a technician per day.
CVT: cost of traveling unit distance by a vehicle.
CTT: cost of traveling unit distance by a technician.

4.3 | Decision variables

x^t_{ijk}: 1, if vehicle k visits {request j}/depot right after {request i}/depot on day t; 0, otherwise.
z^t_{ijs}: 1 if technician s visits {request j}/home right after {request i}/home on day t; 0, otherwise.
m_k: 1 if vehicle k is used during the planning horizon; 0, otherwise.
r_s: 1 if technician s is used during the planning horizon; 0, otherwise.
v^t_k: 1 if vehicle k is used during day t; 0, otherwise.
p^t_s: 1 if technician s is used during day t; 0, otherwise.
w^t_i: 1 if request i is delivered on day t; 0, otherwise.
y^t_i: 1 if request i is installed on day t; 0, otherwise.
q_{ik}: upper bound on the weight of the machines on vehicle k right after leaving {request i}/depot.
g_{is}: number of visits done by technician s before visiting {request i}/home.
b_i: number of days installation of request j is delayed after its delivery.

4.4 | Mathematical modeling

$$\min \overbrace{\sum_{k\in K}\mathrm{CV}m_k + \sum_{t=1}^{T}\sum_{k\in K}\mathrm{CVU}v^t_k + \sum_{t=1}^{T}\sum_{\substack{k\in K,\\ i,j\in R_0, i\neq j}}\mathrm{CVT}d_{ij}x^t_{ijk}}^{\text{vehicle cost}}$$
$$+\underbrace{\sum_{s\in S}\mathrm{CT}r_s + \sum_{t=1}^{T}\sum_{s\in S}\mathrm{CTU}p^t_s + \sum_{t=1}^{T}\sum_{\substack{s\in S,\\ i,j\in R_s, i\neq j}}\mathrm{CTT}d_{ij}z^t_{ijs}}_{\text{technician cost}} + \underbrace{\sum_{i\in R}\mathrm{CI}_i b_i}_{\text{idling cost}} \tag{1}$$

subject to

$$\sum_{j\in R_0, i\neq j} x^t_{ijk} = \sum_{j\in R_0, i\neq j} x^t_{jik} \quad \forall i\in R_0, k\in K, t\in[1,T] \tag{2}$$

$$\sum_{i\in R} x^t_{H_0ik} = \sum_{i\in R} x^t_{iH_0k} \leq M_t v^t_k \quad \forall k\in K, t\in[1,T] \tag{3}$$

$$\sum_{i,j\in R_0, i\neq j} d_{ij}x^t_{ijk} \leq D \quad \forall k\in K, t\in[1,T] \tag{4}$$

$$v^t_k \leq m_k \quad \forall k\in K, t\in[1,T] \tag{5}$$

$$x_{ijk}^t \le v_k^t \quad \forall i,j \in R_0, k \in K, t \in [1,T] \tag{6}$$

$$\sum_{k\in K, j\in R_0, i\neq j} x_{ijk}^t = w_i^t \quad \forall i \in R, t \in [e_i, l_i] \tag{7}$$

$$\sum_{t=e_i}^{l_i} w_i^t = 1 \quad \forall i \in R \tag{8}$$

$$q_{jk} \le q_{ik} - x_{ijk}^t(C + c_j) + C \quad \forall i \in R_0, j \in R, i \neq j, k \in K, t \in [1,T] \tag{9}$$

$$q_{H_0 k} = C \quad \forall k \in K \tag{10}$$

$$\sum_{j\in R_s, i\neq j} z_{ijs}^t = \sum_{j\in R_s, i\neq j} z_{jis}^t \quad \forall i \in R_s, s \in S, t \in [1,T] \tag{11}$$

$$\sum_{i\in R} z_{H_s is}^t = \sum_{i\in R} z_{iH_s s}^t = p_s^t \quad \forall s \in S, t \in [1,T] \tag{12}$$

$$\sum_{i,j\in R_s, i\neq j} d_{ij} z_{ijs}^t \le D_s \quad \forall s \in S, t \in [1,T] \tag{13}$$

$$\sum_{i\in R_s, j\in R, i\neq j} z_{ijs}^t \le N_s \quad \forall s \in S, t \in [1,T] \tag{14}$$

$$p_s^t \le r_s \quad \forall s \in S, t \in [1,T] \tag{15}$$

$$z_{ijs}^t \le p_s^t \quad \forall i,j \in R_s, s \in S, t \in [1,T] \tag{16}$$

$$\sum_{s\in S, j\in R_s, i\neq j} z_{ijs}^t = y_i^t \quad \forall i \in R, t \in [e_i + 1, T] \tag{17}$$

$$\sum_{t=e_i+1}^{T} y_i^t = 1 \quad \forall i \in R \tag{18}$$

$$g_{js} \le g_{is} - z_{ijs}^t(1 + N_s) + N_s \quad \forall i \in R_s, j \in R, i \neq j, s \in S, t \in [1,T] \tag{19}$$

$$g_{H_0 s} = N_s \quad \forall s \in S \tag{20}$$

$$\sum_{u=t}^{t+4} p_s^u \le 5 - p_s^{t+5} \quad \forall s \in S, t \in [1, T-5] \tag{21}$$

$$\sum_{u=t}^{t+4} p_s^u \le 5 - p_s^{t+6} \quad \forall s \in S, t \in [1, T-6] \tag{22}$$

$$\sum_{u=T-5}^{T-1} p_s^u \le 5 - p_s^T \quad \forall s \in S \tag{23}$$

$$\sum_{t=e_i+1}^{T} t y_i^t - \sum_{t=e_i}^{l_i} t w_i^t - 1 = b_i \quad \forall i \in R \tag{24}$$

$$q_{ik} \in \mathcal{Z}_{\ge 0}; \quad x_{ijk}^t, v_k^t, m_k \in \{0,1\} \quad \forall i,j \in R_0, k \in K, t \in [1,T], i \neq j \tag{25}$$

$$g_{is} \in \mathcal{Z}_{\geq 0}; \quad z^t_{ijs}, p^t_s, r_s \in \{0,1\} \quad \forall s \in S, i,j \in R_s, t \in [1,T], i \neq j \tag{26}$$

$$b_i \in \mathcal{Z}_{\geq 0}; \quad w^t_i, y^t_i \in \{0,1\} \quad \forall i \in R, t \in [1,T] \tag{27}$$

The exact model for the given problem is formulated by Equations (1)–(27). The objective function (1) is composed of three types of costs: the vehicle (first three summations), technician (next three summations), and idling cost (last summation). The sum of the vehicle hiring cost, the vehicle cost per day, and the vehicle cost per distance is equal to the total vehicle cost. Similarly the sum of the personnel hiring cost, the personnel cost per day, and the personnel cost per distance is equal to the personnel cost. The idling cost is calculated by multiplying the cost of idling all the machines at each request by the difference between the delivery and installation days.

In the mathematical model, constraints (2)–(10) are vehicle, constraints (11)–(23) are technician, and constraints (24) are idle time related constraints. Constraints (25)–(27) define the domains of the variables.

Constraints (2) and (11) are balance equations for the vehicles and technicians respectively. They ensure that in any day, in any vehicle and personnel route, the number of arcs entering to a location of a request, depot or home should be equal to the number of arcs exiting from the same location.

Constraints (3) and (12) ensure that both the vehicles and technicians start/end their routes from/at the depot and home respectively. For the technicians, the problem dictates upper bounds for the number of installations (N_s) and travel distance (D_s) for any given day. Therefore, in an optimal solution, if a technician is used on any given day, they should leave and return their home only once. On the other hand, for the vehicles, in addition to the total travel distance on any given day (D), there is an upper bound on the weight of the machines that are being carried by the vehicle, defined as vehicle capacity (C), at any given time of the day. This makes it possible for a given vehicle on any given day to deliver more than its capacity by making multiple visits to the depot. Because of the difference of the restrictions on the vehicles and technicians, constraints (12) are satisfied with an equality to p^t_s whereas constraints (3) are satisfied with an inequality to v^t_k. The former constraints force technicians to leave and return to their home only once if they are working on day t, whereas the latter constraints force vehicles to have an equal number of trips that leave from and return to the depot, and these trips can only occur if the truck is operating on day t. M_t on the RHS of the constraints (3) is calculated by counting the number of orders that can receive a delivery on day t. Note that, the maximum value that M_t can take on different days also sets the upper bound for the maximum number of hired vehicles in the planning horizon.

Constraints (4) and (13) restrict the total travel distance for each day of the vehicles and the technicians, respectively. In addition, constraints (14) set the maximum number of installations for a technician on a single day. Similarly, in the problem definition, there is a limit set on the total weight of machines a truck can carry. This is ensured by constraints (9) and (10). The variable q_{H_0k}, that represents the weight of the machines right after the vehicle is leaving the depot, is set to be C for any truck by constraints (10). Since constraints (9) ensure that the weight on the truck decreases at every request stop by the weight of the delivery of the same request, and the q_{ik}'s are defined as nonnegative integer variables, no truck can carry more than its capacity.

Constraints (5) and (15) ensure that in order to use a vehicle or a technician respectively in any day of the planning horizon, we need to hire them first. Similarly, constraints (6) and (16) ensure that if a vehicle or a technician travel between two requests on any given day, they are already hired for the day.

Constraints (7) and (17) establish the relationship between the routing (x^t_{ijk} and z^t_{ijs}) and service, that is, delivery (w^t_i) and installation (y^t_i) variables for the vehicles and technicians respectively. According to constraints (7), if the location of a request is visited by a vehicle, then the machines ordered by this request are delivered between the first (e_i) and the last (l_i) days the delivery can be done. Similarly, according to constraints (17), if the location of a request is visited by a technician, then the installation that is ordered by this request is done at least 1 day after the earliest day that the request can be delivered.

Constraints (8) and (18) ensure that all the deliveries and installations are done within their predefined times respectively. Constraints (8) ensure that each request is delivered by one of the vehicles. These constraints restrict each request to be delivered between their first (e_i) and the last (l_i) days the delivery can be made. Similarly, constraints (18) ensure that each request is installed by one of the technicians. Since the earliest installation day is 1 day after the delivery of the machines, the constraints only consider the days after the first day that the request can be delivered.

Constraints (9) and (10) prevent subtours in vehicle routes. Constraints (10) assign the maximum weight (C) a vehicle can carry when leaving the depot to variables q_{ik} on any given day. Constraints (9) subtract the weight of the machines that are being delivered from q_{ik} every time a vehicle visits a request. Keeping track of each q_{ik} and forcing q_{jk} less than or equal to q_{ik} if request j is visited immediately after request i or depot by vehicle k (i.e., $x^t_{ijk} = 1$) prevent the formation of subtours. Note that, since a vehicle can make multiple visits to the depot on any given day, these constraints are not being forced for the trips to the depot.

Constraints (19) and (20) prevent subtours in technician routes. Constraints (20) assign the maximum number of visits a technician can make (N_s) to variables g_{is} on any given day. Note that, since the maximum number of installations that a technician

can make is an upper bound for the maximum number of visits in a day, we use this constant in our model. Constraints (19) subtract 1 from g_{is} every time a technician visits a node. Similar to vehicle subtour elimination constraints, since g_{js} is forced to be less than or equal to g_{is} if request j is visited immediately after request i or home H_s by technician s (i.e., $z^t_{ijs} = 1$), subtours never form in technician routes.

Constraints (21) and (22) ensure that the solution complies with the working day restrictions for the technicians. According to constraints (21) and (22), if a technician works four or fewer consecutive days, that is, LHS is less than or equal to 4, he/she can work either the next day or the day after the next day unless he/she is working more than five consecutive days. If a technician works five consecutive days, that is, LHS is equal to 5, the technician cannot work the next two consecutive days. Constraints (23) ensure that this restriction still holds at the end of the planning horizon and prevents any technician from working more than 5 days in the last 6 days.

Constraints (24) calculate the idling time, the difference between the delivery day and the installation day, for all requests. Since b_i is defined as a nonnegative integer, these constraints also ensure that machines are installed at least 1 day after the delivery for every request.

5 | HYPER-HEURISTICS METHODOLOGY OF CVRP FOR DELIVERY AND INSTALLATION OF MACHINES

In this section we describe the hyper-heuristic framework applied to this problem and discuss solution initialization and representation and the low level heuristics set.

5.1 | Population-based hyper-heuristic framework

Following the description of the selection hyper-heuristic framework in Section 2.2, most of the previously proposed solution methodologies in selection hyper-heuristics utilize a single solution during the search process and iteratively improve it, while some other methodologies adopt the idea of using a population of solutions during the search as a whole, or during some part of it. Our proposed framework is based on a population of solutions from which one of them will be selected and applied to a selection hyper-heuristic at each step in the search. We are motivated in this work to use a population of solutions as we believe that this provides diversity in the search and allows better exploration of new areas in the search space. The process starts by initializing a number of solutions using a generation method to create an initial population $pop = \{sol_1, sol_2, \dots, sol_{pop_{size}}\}$. A number of selection hyper-heuristics $HH = \{hh_1, hh_2, \dots, hh_n\}$ combining different selection and move acceptance methods are implemented.

A solution sol_i and a selection hyper-heuristic hh_j are randomly selected from pop and HH respectively, where sol_i will serve as $S_{current}$ to hh_j. The selection hyper-heuristic hh_j selects a heuristic (or sequence of heuristics) and applies it to $S_{current}$ to create new solution S_{new}, which is checked for feasibility, and rejected if it is not feasible (i.e., violates at least one of the problem constraints). If S_{new} is feasible it will be evaluated and the decision of its acceptance is made by the move acceptance component of hh_j. The best found solution S_{best} is replaced by S_{new} if it is better. The iteration between the selection and acceptance components continues until the termination criteria are satisfied, that is until the global time limit is exceeded or when there is no improvement on the best obtained solution for a certain number of iterations.

After hh_j terminates, S_{best} is checked against the best found global solution S_{global} and replaces it if it is better. Afterwards, S_{best} is shuffled by randomly selecting and applying a series of low level heuristics. The number of steps to shuffle a solution is tuned by the user, and is constant during the search. This shuffling is necessary in order to avoid the possibility of early convergence and to refresh the population. Next, a solution from the population and a selection hyper-heuristic are randomly selected and the same steps mentioned above are repeated. This iterates until the specified time for running an instance passes. Algorithm 1 outlines our applied framework (Figures 1 and 2).

For this framework we have tested a total of eight selection hyper-heuristics combining the selection methods: simple random (SR), sequence-based selection hyper-heuristic (SS), and the move acceptance methods: record-to-record (RR), Naïve acceptance (Naïve), great deluge (GD), and simulated annealing (SA).

SR is the most basic heuristic selection method. It randomly selects a low level heuristic at each step according to a uniform probability distribution. SS on the other hand applies sequences of heuristics to the solution, instead of single applications. This selection method learns and identifies the sequences of heuristics most likely to improve the current solution. More details of this method can be found in [35,36].

The move acceptance methods applied accept nonworsening solutions, and worsening solutions are accepted using different criteria.

Algorithm 1: Algorithm of the population based hyper-heuristic

Let $S_{current}, S_{new}, S_{best}, S_{global}$ be current, new, best and global solutions respectively;
Let $HH = [hh_1, hh_2, \ldots, hh_n]$ be the combinations of selection hyper-heuristics;
Let $pop = [sol_1, sol_2, \ldots, sol_{pop_{size}}]$ be the solutions in the population;
Let $LLH = [llh_1, llh_2, \ldots, llh_{|LLH|}]$ be the set of low level heuristics;
$pop \leftarrow$ `InitialGeneration()`;
repeat
  $sol_i \leftarrow$ `SelectRandomly(`pop`)`;
  $S_{current} \leftarrow sol_i$;
  $S_{best} \leftarrow S_{current}$;
  $hh_j \leftarrow$ `SelectRandomly(`HH`)`;
  repeat
    $llh \leftarrow Select(hh_j, LLH)$;
    $S_{new} \leftarrow$ `ApplyLLH(`$llh, S_{current}$`)` ;
    if $Accept(hh_j, S_{new}, S_{current})$ **then**
      $S_{current} = S_{new}$;
    if $S_{current}$ $isBetterThan$ S_{best} **then**
      $S_{best} \leftarrow S_{current}$;
  until $TerminationCriteria$;
  if S_{best} $isBetterThan$ S_{global} **then**
    $S_{global} \leftarrow S_{best}$;
  $sol_i \leftarrow S_{best}$;
  `Shuffle(`sol_i`)`;
until $timeLimit$;
return S_{global};

In Naïve and SA, the worsening solutions are accepted with a certain probability. This probability is fixed for Naïve which is predefined by the user, while in SA, the probability varies in time and it is calculated using the following formula:

$$p_t = e^{-\frac{\Delta f}{\Delta F\left(1-\frac{t_{current}}{t_{limit}}\right)}}, \tag{28}$$

where Δf is the change in the cost at time $t_{current}$, ΔF is the expected maximum change in the cost, and t_{limit} is the time limit. GD is a threshold move acceptance method which allows worsening solutions if their cost (objective) value is less than or equal

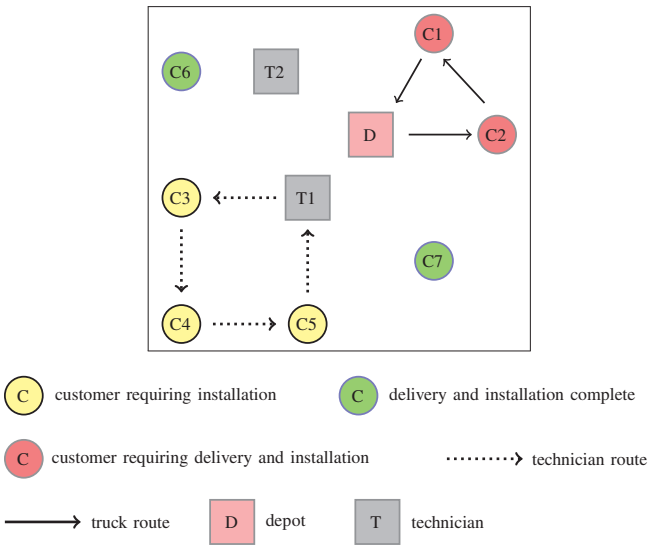

Figure 1 Problem description [Color figure can be viewed at wileyonlinelibrary.com]

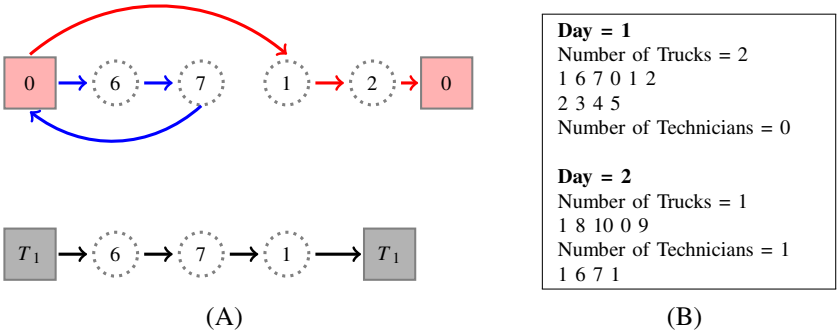

Figure 2 Solution example, and the routes of truck 1 in day 1, and technician 1 in day 2. The red and blue arcs represent two different tours. The pink and gray colored boxes represent the depot and technician T_i respectively [Color figure can be viewed at wileyonlinelibrary.com]

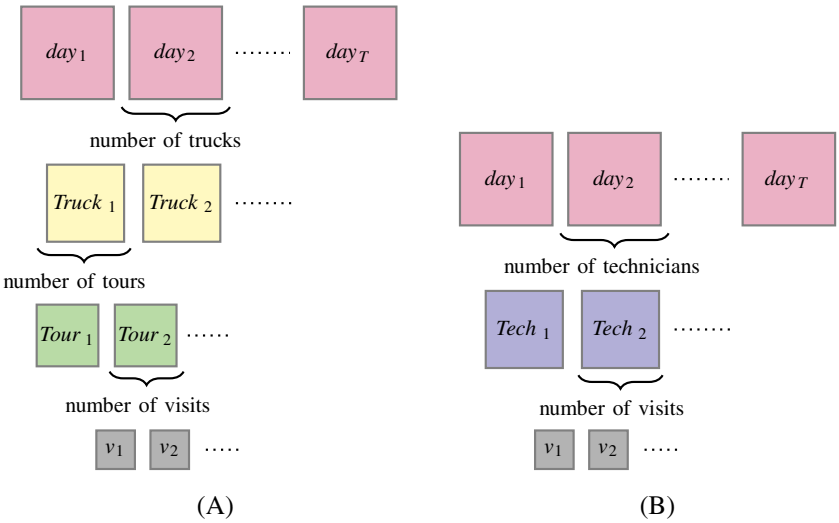

Figure 3 Solution representation [Color figure can be viewed at wileyonlinelibrary.com]

to a threshold value referred to as "level" (τ_t) which gets updated at each time step (t) during the search. The level is initially set to the initial cost. The update formula for the level is as follows:

$$\tau_t = f + \Delta F \times \left(1 - \frac{t_{current}}{t_{limit}}\right), \tag{29}$$

where f is the final expected cost value, ΔF is the maximum expected change in the cost value, and $t_{current}$, t_{limit} is the time at the current step, and the time limit respectively. RR is a variant of GD which accepts worsening solutions that are not much worse than the best solution in hand to an extent based on the following formula:

$$obj(S_{new}) \leq obj(s_{best}) + fr \times obj(S_{best}), \tag{30}$$

where fr is a factor that is updated during the search, starting with a large value and gradually decreasing.

5.2 | Solution representation scheme

The described hyper-heuristic framework requires initializing a number of solutions to build a population, and this is achieved using an initial generation method. The structure of each of the initialized solutions is demonstrated in Figure 3. Each solution is composed of two main components: truck visits, and technician visits. The truck visits component corresponds to the schedule of trucks during the planning horizon which can be modeled as four levels: days, trucks dispatched on each day, tours performed by each truck, and the requests to deliver on each tour. Similarly, the technician visits correspond to the technicians' schedule composed of three levels: days, technicians scheduled on each day, and visits performed by each technician. The initial generation method randomly produces these schedules, while ensuring the final constructed solution is feasible. The main focus of the initial generation method is the feasibility of the solution and not its quality.

The feasibility of the solution must also be maintained during the hyper-heuristic operation. A feasibility test is implemented to ensure that the constraints described in Section 3 still hold after each application of low level heuristic(s). A single violation

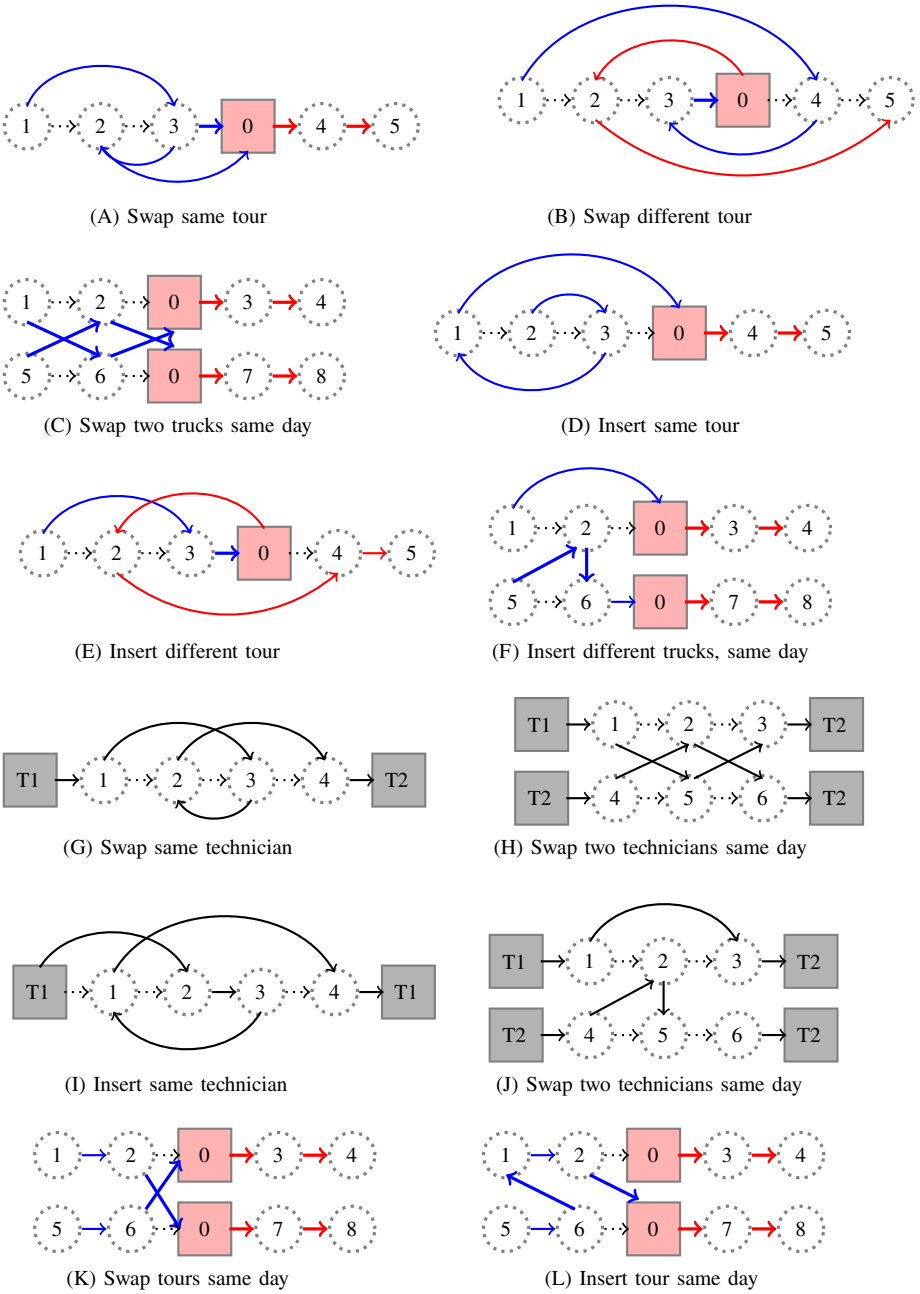

Figure 4 Some selected low level heuristic descriptions. Blue and red arrows represent two different tours. Dashed arrows are edges removed by the application of the low level heuristic [Color figure can be viewed at wileyonlinelibrary.com]

of any of these constraints results in rejecting the solution. This test prevents the evaluation of too many infeasible solutions which can consume valuable search time.

5.3 | Low level heuristics

The hyper-heuristic controls a total of 25 low level heuristics to improve the quality of a given initial solution. These low level heuristics perform swap and insert operations for requests in truck and technician routes (see Figure 4). Low level heuristics are restricted, as needed, to only produce routes that respect some of the constraints. For example, some low level heuristics perform operations between different days in the planning period; in this case if the operation involves delivery requests, the time windows of these requests must be respected and any installations that as a consequence violate the time windows constraints must be rescheduled. If it involves installation requests, the delivery of these requests must be ensured at least the day before.

- **LLH0**: selects a random day, a random truck route and a random tour, and swaps any two randomly selected requests on this tour.

- **LLH1**: selects a random day, a random truck route, and two different random tours, and swaps any two randomly selected requests on each tour.
- **LLH2**: selects a random day, two different random truck routes, two random tours from each route, and swaps two randomly selected requests from each tour.
- **LLH3**: selects two different random days, two random truck routes from each day, and two random tours from each route, and swaps two randomly selected requests from each tour.
- **LLH4**: selects a random day, a random technician scheduled on this day, and swaps two randomly selected requests of this technician.
- **LLH5**: selects a random day, two different random technicians scheduled on this day, and swaps two randomly selected requests of these technicians.
- **LLH6**: selects two different random days, and two random technicians, and swaps two randomly selected requests of these technicians.
- **LLH7**: selects a random day, a random truck route, a random tour, and two random positions on this tour. The request on the first position is inserted into the second position.
- **LLH8**: selects a random day, a random truck route, two different random tours on the selected route, and a random position on each tour. The request on the first position of the first tour, is inserted into the second position of the second tour.
- **LLH9**: selects a random day, two different random truck routes, a random tour on each route, and a random position on each tour. The request on the first position is inserted into the second position.
- **LLH10**: selects two different random days, a random truck route on each day, a random tour on each route, and a random position on each tour. The request on the first position is inserted into the second position.
- **LLH11**: selects a random day, a random technician scheduled on this day, and two random positions on the technician route. The request on the first position is inserted into the second position.
- **LLH12**: selects a random day, two different random technicians, and a random position on each technician route, and inserts the request on the first position into the second position.
- **LLH13**: selects two different random days, a random technician on each day, and a random position on each technician route. The request on the first position is inserted into the second position.
- **LLH14**: selects a random day, two different random truck routes, and a random tour on each route, and swaps the two selected tours.
- **LLH15**: selects two different random days, a random truck route on each day, and a random tour on each route, and swaps the two selected tours.
- **LLH16**: selects a random day, two different random truck routes, and a random position on each route. The tour on the first position is inserted into the second position.
- **LLH17**: selects two different random days, a random truck route on each day, and a random position on each route. The tour on the first position is inserted into the second position.
- **LLH18**: selects a random day, two different random truck routes, and two random positions. A block of consecutive requests starting at the first position is swapped with another block of requests starting at the second position. The size of the block is randomly selected.
- **LLH19**: selects two different random days, a random truck route on each day, and two random positions on each route. A block of visits starting at each of the positions are swapped with each other.
- **LLH20**: selects a random day and two different random technicians, and swaps two blocks of requests for these technicians.
- **LLH21**: selects two random different days and two random technicians from each day, and swaps two blocks of requests of these technicians.
- **LLH22**: selects a random day, and two different random truck routes. A block of requests is moved from the first route to the second route at randomly selected positions.
- **LLH23**: selects two different random days and two random truck routes. A block of requests is moved from the first route to the second route at a randomly selected positions.
- **LLH24**: selects a random day and two different random technicians, and moves a block of requests from the first technician to the second technician.
- **LLH25**: selects two different random days and two random technicians, and moves a block of requests from the first technician to the second technician.

Table 2 The algorithm parameters and the chosen values

Parameter	Tuning	irace
Population size (pop_{size})	2–5	3
Limit on iterations without improvement	10^5	63 144
Naïve acceptance probability	0.1	0.1
RR factor (fr) starting value	10	10.64
Number of iterations to shuffle solutions	10	7

6 | EXPERIMENTAL RESULTS

The experiments were performed on several machines. CPLEX 12.10 and Gurobi 9.0 together with C# and Python were used for the exact technique and Microsoft Visual Studio 2017 C++ for the heuristic method. The experiments for the heuristic method on the hidden dataset were performed on a device with the specifications: Intel Core i5 at 2.3 GHz with memory of 8 GB. On both datasets, the experiments were designed according to the competition rules, which required nine runs per instance with nine different random seeds also determined by the competition rules. The run time for each instance in both datasets was also calculated according to the competition rules, where it has been specified that each instance is run for a limited time on the user machine calculated with the formula: $T_{limit} = fb \times (10+|R|)$, where T_{limit} is the time limit for running an instance according to the user local machine, fb is a factor calculated by a benchmark tool provided by the competition to estimate the equivalent time on any machine compared to the organizers core machine, and $|R|$ is the number of delivery requests in the instance. To tune these parameters we have followed two approaches: a manual approach where a series of extensive experiments were performed to fine tune the design parameters. We arrived at a combination of parameter values that resulted in a relatively better performance across a subset of public instances. The second approach is using the irace package [44] to automatically tune the parameters on the five instances with the highest variance in the small dataset. Irace performed a maximum of 8000 experiments and ran for about 9 hours to find the top four configurations. The best configuration in the top four was selected to perform another round of experiments on the small dataset with the same experimental design (i.e., nine runs per instance with the random seed values set by the competition rules). The parameter settings of the hyper-heuristic using the two approaches are shown in Table 2. The results of the small dataset reported in the next section are the ones found using the manual tuning. For convenience, the developed approach is denoted as POHH.

6.1 | Results on the small dataset

Table 3 provides the results of the small instances dataset for the exact and the hyper-heuristic method. For the exact model, the upper and lower bounds are provided for each instance. The lower bound indicates that an optimal solution was not found for a particular instance. The results of the hyper-heuristic experiments are reported in terms of the minimum and maximum objective values achieved in the nine runs, the average of the nine runs and the SD. The time in seconds is the time that was required to find the reported results by the exact model, and the time limit for each run of POHH. The time was normalized to its equivalent in the standard machine using the calibration tool provided by the competition. We also note that the execution time of the exact model on any instance was limited to up to 30 minutes.

From Table 3 we can directly compare the performances of the two models in terms of finding optimal solutions and the time required to find them. The exact model was able to find optimal solutions for 12 instances out of the 25 with CPLEX and Gurobi, and no feasible solution with CPLEX or Gurobi was found for the instance Small_09. For the rest of the instances the same upper bounds were found by CPLEX and Gurobi, while we notice that for some instances Gurobi was able to find better lower bounds. Comparing the results of the exact model to the minimum value in the nine runs, POHH was able to find the optimal solutions in all 12 instances where the exact model found optimal solutions. In the other cases where no optimal solution was proved by the exact model, the POHH algorithm either found the same upper bound or, in the case of seven of the instances, a better upper bound was discovered. Also, a feasible solution for Small_09 was found by POHH. Although the POHH was able to find the same value for the upper bound as Gurobi and CPLEX in many instances, we cannot yet argue that this upper bound is the optimal solution to these instances. In terms of run time, POHH achieved improved run times in most of the cases. The exact model in many instances required more than 3000 seconds to find a solution, while POHH required less than 30 seconds on the same instances.

We also compared the results on the small dataset using the manually tuned parameters with the best configuration found by the irace package. The results of the two approaches were very similar using the minimum value in the nine runs as comparison

Table 3 Summary of the results

	CPLEX				Gurobi				POHH				
Instance	**Upper Bound**	**Lower Bound**	**Deviation**	**Time**	**Upper Bound**	**Lower Bound**	**Deviation**	**Time**	**Min**	**Max**	**Avg**	**Std**	**Time**
Small_01	36 016 020			5	36 016 020			0	36 016 020	36 016 020	36 016 020	0	16
Small_02	1 786 430			10	1 786 430			2	1 786 430	1 786 520	1 786 440	30	18
Small_03	32 085 900	14 442 700	55	1800	32 085 800	21 461 179	33	1800	32 085 800	32 086 000	32 085 922	83	20
Small_04	18 816 347	14 825 629	21	1807	18 816 347			1797	18 816 347	18 816 555	18 816 393	92	22
Small_05	128 025 356	112 613 264	12	1800	128 124 700	112 490 056	12	1800	128 014 700	128 015 356	128 015 034	237	24
Small_06	180 524	154 910	14	1805	180 524	171 931	5	1800	180 524	180 524	180 524	0	26
Small_07	239 350 700			92	239 350 700			27	239 350 700	239 350 700	239 350 700	0	16
Small_08	66 877 075	34 378 156	49	1800	66 877 075			408	66 877 075	66 878 040	66 877 826	426	18
Small_09	Infeasible Solution			1800	Infeasible Solution			1800	1 073 410	1 073 510	1 073 421	33	20
Small_10	133 887 780	133 873 627	0	1800	133 887 780			270	133 887 780	133 887 790	133 887 789	3	22
Small_11	537 212 505	341 632 540	36	1800	537 613 730	363 037 030	32	1800	537 212 505	537 213 820	537 212 866	577	24
Small_12	16 209 060	6 570 854	59	1800	16 205 455	6 528 676	60	1800	16 203 435	16 205 395	16 204 949	581	26
Small_13	221 514 230			202	221 514 230			367	221 514 230	221 514 230	221 514 230	0	16
Small_14	211 980	101 866	52	1802	212 005	113 930	46	1800	211 980	212 080	212 003	35	18
Small_15	2 148 165			122	2 148 165			33	2 148 165	2 151 980	2 149 312	1442	20
Small_16	333 416 760	138 661 467	58	1800	333 416 760	158 241 503	53	1800	333 416 760	333 422 700	333 418 473	2203	22
Small_17	758 620	615 961	19	1810	758 620	660 245	13	1800	758 620	758 620	758 620	0	24
Small_18	480 118 510	194 470 467	59	1842	479 818 890	219 958 625	54	1800	479 817 740	479 821 390	479 819 453	969	26
Small_19	877 960			7	877 960			0	877 960	878 320	878 080	180	16
Small_20	1 763 630	1 413 949	20	1802	1 763 630			528	1 763 630	1 766 945	1 764 367	1462	18
Small_21	1 074 748	893 161	17	1802	1 074 748			1255	1 074 748	1 074 748	1 074 748	0	20
Small_22	8 326 840	6 880 399	17	1802	8 326 840	7 542 245	9	1800	8 326 840	8 326 840	8 326 840	0	22
Small_23	35 499 570	23 517 476	34	1802	35 499 570	27 966 620	21	1800	35 499 539	35 499 580	35 499 563	19	24
Small_24	182 073 280	130 142 469	29	1805	182 073 280	132 738 156	27	1800	182 072 650	182 099 930	182 076 697	8778	26
Small_25	3 499 190			785	3 499 190			10	3 499 190	3 499 190	3 499 190	0	16

Note: The table shows upper bound (or optimum), lower bound (if not optimum), percentage deviation, solution time (in second).

base. Both found the same minimum in all the instances, except for Small_18, in which the irace configuration found a new best result of "479 817 740", and Small_23, in which the manual configuration was better.

6.2 | Results on the hidden dataset

As mentioned previously, the hidden dataset was used to assess the competitors' algorithms in the restricted resources challenge, and according to the results of this challenge, eight teams were selected as finalists. We have followed the same competition rules as the other finalists team, with this set of experiments to fairly compare and justify our results.

Table 4 summarizes the results of the top six teams, including our hyper-heuristic approach, for each instance ranked based on their best found solution from best to worst. For each instance, the results are reported for the best six teams out of nine using the average of the nine runs and the minimum objective value.

Considering the minimum and average objective values obtained over nine runs for each hidden instance, POHH is relatively competitive between the finalists. The POHH achieved a position in the top six in 15 instances out of 25. For 11 out of 25 instances, including: Hidden_02, Hidden_07, Hidden_09, Hidden_11, Hidden_13, Hidden_16, Hidden_17, Hidden_19, Hidden_21, Hidden_22, Hidden_25, POHH performs better than *at least* half of the finalist approaches in terms of average objective value. Except the instances Hidden_13 and Hidden_17, the same phenomena is observed with those instances with respect to the minimum objective values. The best achieved results are found on instances Hidden_07, Hidden_11, Hidden_16, and Hidden_21, where POHH is ranked the fourth based on both the average and the minimum objective values. These instances are all of different sizes: 150, 450, 600, and 750 requests, reflecting the ability of POHH to perform well on instances with varying characteristics and complexities (Figure 4).

We have also ranked our approach among the eight finalists using the same method used in the competition. A ranking score is calculated per instance for each submitted solver by removing the two best and worst solutions found by this solver. We then take the average objective value of these five solutions as score for the algorithm, and rank all methods accordingly. The average of all ranking scores for the instances represents the final mean rank of the solver, which was used to order the competitors from the first to the last. Figure 5 displays the ranking of the POHH algorithm among the eights finalists based on this method. It is clearly seen that our method was able to produce results competitive with the finalists by achieving a better final mean rank than three teams, and an insignificant difference from the ranks of the third, fourth, and fifth teams.

Although the proposed algorithm did not succeed in improving any current best known solutions on hidden instances, it performed well (see Section 6.1) on small instances with few requests, and the results on the hidden instances are considered reasonably good.

6.3 | Performance analysis of POHH

Figure 6 visualizes the six sample instances, including Small_01, 03, 06 and Hidden_01, 03, 06 that we used for the analysis purposes, reflecting the varying characteristics of each instance. These instances were arbitrarily chosen to represent varying sizes. The visualization was obtained with the tools provided by the challenge organizers [51]. The large light blue circle indicates the depot, and this may or may not have technicians on the premises at any given time. Yellow locations, indicated by medium circles, have technicians, and they may or may not have requests. Green locations have requests, but no technicians. The large diameter is used when a location (of any color) is open, and the small diameter indicates that a location is not open for delivery. The depot (blue) is open throughout the planning horizon, and each request location is open on the days specified. The figure shows only the beginning of day 1 for each instance. Figure 6 also shows semi-transparent green circles centered on the locations with technicians. The radius of these circles is equal to the half of the maximum daily distance of the corresponding technician. Because these circles are semitransparent, overlap leads to color intensification, making visually clear which customer locations are within reach of few or many technicians. One may notice from the clustering, for example, as for Hidden_06, that the locations are making the shape of the Netherlands. This implies that these are indeed based on real-world data offered by ORTEC.

Although Figure 6 gives a rough picture of those instances (e.g., some instances are more limiting in terms of number and action radius of technicians), we must emphasize that it does not describe a given problem instance fully. For example, the importance of violating a given constraint is not depicted and, as we mentioned before, penalties for violating the different constraints can differ substantially as a part of the cost function. Table 1 is particularly useful in this case.

The pie charts in Figure 7 depict the utilization rates of the different selection hyper-heuristics applied in our framework. The utilization is calculated in terms of the ratio between the number of times a selection hyper-heuristic was successful in finding an improved solution over the best global solution to the total number of improvements made by all the selection hyper-heuristics in the duration of run time.

Table 4 The performance comparison of the finalists and POHH, based on the average, and minimum of the objective values over nine runs for each instance

Instance	Team	Min	Avg	Instance	Team	Min	Avg	Instance	Team	Min	Avg
Hidden_01	UOS	**67 303 070**	**67 347 510.0**	Hidden_10	UOS	**31 425 420**	**31 615 172.8**	Hidden_19	UOS	**4 379 062 260**	**4 379 503 211.1**
	MJG	67 539 160	67 768 841.7		MJG	32 134 970	32 406 070		MJG	4 379 599 620	4 384 265 171
	CokaCoders	67 936 410	68 239 128.3		TCSExplorer	35 602 350	36 296 800		CokaCoders	4 385 514 720	4 416 094 113
	AAVK	68 049 230	68 497 476.7		CokaCoders	41 951 125	44 037 471.7		orlab	4 390 908 575	4 397 116 298
	orlab	68 059 355	68 531 090.6		orlab	42 557 550	43 110 900.6		POHH	4 400 633 465	4 433 562 786
	TCSExplorer	68 067 705	68 669 280		POHH	44 757 390	46 971 126.1		justFall	4 415 787 735	4 460 775 575
Hidden_02	UOS	**866 780 485**	**884 328 838.3**	Hidden_11	UOS	**4 052 063 633**	**4 143 464 703.2**	Hidden_20	UOS	**126 020 890**	**127 117 758.9**
	MJG	878 698 335	887 583 730		MJG	4 104 065 729	4 150 825 522		MJG	128 220 285	130 004 861.7
	TCSExplorer	940 755 820	966 644 283.9		TCSExplorer	4 490 010 535	4 573 692 086		TCSExplorer	138 750 815	141 489 971.1
	CokaCoders	978 877 270	992 970 729.3		POHH	4 792 051 226	4 947 603 379		CokaCoders	143 190 180	146 078 606.1
	POHH	999 703 125	1 070 465 586		justFall	4 933 454 022	5 124 240 633		orlab	164 537 160	167 708 532.2
	justFall	1 178 709 770	1 289 905 508		AAVK	5 050 880 606	5 176 963 402		POHH	171 503 820	177 538 151.1
Hidden_03	UOS	**1 353 070 685**	**1 356 206 683.3**	Hidden_12	UOS	**2 985 079 895**	**2 988 919 325.0**	Hidden_21	UOS	**33 041 255**	**33 217 240.6**
	MJG	1 358 875 910	1 363 989 363		MJG	2 985 608 315	2 989 299 754		MJG	33 313 460	33 871 058.9
	CokaCoders	1 362 273 745	1 374 383 989		CokaCoders	2 998 137 785	3 062 709 553		TCSExplorer	35 525 950	36 048 057.8
	orlab	1 365 243 450	1 373 092 383		orlab	3 020 441 765	3 030 703 193		AAVK	38 119 685	38 845 431.7
	AAVK	1 388 162 220	1 394 947 955		wanderer	3 056 711 365	3 059 281 502		POHH	38 347 205	38 839 638.3
	justFall	1 389 700 390	1 407 156 949		justFall	3 072 299 875	3 086 846 959		justFall	40 036 785	41 366 946.1
Hidden_04	MJG	**5 354 045**	**5 382 928.3**	Hidden_13	UOS	**5 237 955 246**	**5 239 090 235.3**	Hidden_22	MJG	**6 642 535**	**6 677 298.3**
	UOS	5 407 020	5 470 823.3		MJG	5 243 444 173	5 270 924 784		UOS	6 700 250	6 750 354.4
	TCSExplorer	5 782 480	5 973 581.1		CokaCoders	5 251 322 598	5 267 097 465		TCSExplorer	6 986 000	7 020 108.9
	AAVK	6 053 945	6 122 060		orlab	5 273 032 383	5 300 095 233		AAVK	7 473 115	7 575 487.8
	orlab	6 258 020	6 409 908.3		justFall	5 284 251 162	5 322 430 964		POHH	7 671 390	7 831 162.8
	POHH	6 750 245	6 812 897.2		POHH	5 289 271 553	5 312 240 754		orlab	7 777 350	7 896 450.6
Hidden_05	UOS	**2 420 668 237**	**2 438 943 861.4**	Hidden_14	UOS	**1 378 863 050**	**1 380 531 068.9**	Hidden_23	CokaCoders	**22 341 696 590**	22 374 478 803
	MJG	2 461 219 726	2 472 452 759		orlab	1 385 370 725	1 392 172 054		UOS	22 342 274 615	**22 368 503 380.6**
	CokaCoders	2 793 126 594	2 824 641 910		CokaCoders	1 385 514 390	1 387 323 213		MJG	22 473 152 460	22 489 158 018
	TCSExplorer	2 873 811 043	2 900 849 794		MJG	1 386 071 905	1 389 987 112		orlab	22 564 706 605	22 644 343 334
	wanderer	3 463 496 082	3 463 496 082		AAVK	1 394 884 865	1 403 511 603		justFall	22 803 650 615	22 946 341 903
	orlab	3 489 050 112	3 582 195 247		justFall	1 399 908 710	1 414 358 900		AAVK	22 946 775 355	23 021 877 311
Hidden_06	MJG	**32 919 590**	**32 950 587.2**	Hidden_15	UOS	**163 646 890**	**163 861 015.0**	Hidden_24	UOS	**31 350 467 425**	**31 373 781 566.1**
	UOS	33 035 255	33 122 941.7		orlab	164 986 370	165 539 965.6		CokaCoders	31 386 137 725	31 441 905 918
	orlab	33 243 100	33 310 187.2		MJG	165 442 970	165 746 732.2		orlab	31 457 461 925	31 554 867 014
	AAVK	33 537 475	33 642 695		AAVK	166 559 140	167 109 473.3		MJG	31 502 515 080	31 579 529 472
	TCSExplorer	33 730 430	33 798 883.1		wanderer	167 271 785	167 271 785		justFall	31 915 258 830	32 102 001 066
	wanderer	33 799 140	33 799 140		TCSExplorer	168 523 765	168 889 354.4		wanderer	31 945 701 195	31 947 545 553

Table 4 (Continued)

Instance	Team	Min	Avg	Instance	Team	Min	Avg	Instance	Team	Min	Avg
Hidden_07	UOS	**102 098 250**	**102 289 614.4**	Hidden_16	MJG	**52 730 075**	**52 860 556.7**	Hidden_25	UOS	**549 505 255**	**549 854 756.1**
	MJG	102 375 745	103 005 780.6		UOS	53 232 615	53 561 470.6		MJG	552 735 110	563 054 914.4
	CokaCoders	107 548 420	110 818 258.9		TCSExplorer	58 750 440	59 286 328.3		CokaCoders	586 771 940	605 819 273.9
	POHH	108 198 340	109 634 304.4		POHH	59 861 645	60 499 887.8		TCSExplorer	611 102 855	621 490 488.9
	TCSExplorer	108 308 895	110 359 084.4		justFall	61 176 575	63 036 133.3		POHH	625 293 110	652 332 916.1
	justFall	121 495 535	132 836 498.3		AAVK	61 626 085	63 624 953.9		orlab	659 488 500	678 896 693.9
Hidden_08	UOS	**728 784 055**	**729 462 820.6**	Hidden_17	UOS	**27 301 086 592**	**27 322 051 463.0**				
	MJG	729 588 325	732 733 357.2		MJG	27 387 159 376	27 426 339 743				
	CokaCoders	734 907 040	737 740 395		CokaCoders	27 396 284 273	27 996 989 611				
	orlab	735 203 675	737 068 948.3		orlab	27 486 127 713	27 525 238 097				
	AAVK	743 077 790	750 319 370.6		justFall	27 717 856 052	27 822 517 277				
	justFall	746 651 030	757 387 192.2		POHH	27 867 305 050	27 914 747 560				
Hidden_09	UOS	**1 692 627 537**	**1 713 909 634.1**	Hidden_18	MJG	**52 602 450**	**52 658 013.3**				
	MJG	1 732 570 744	1 746 683 152		UOS	52 921 995	53 040 623.3				
	CokaCoders	1 884 282 890	1 939 897 718		AAVK	54 778 115	55 313 427.2				
	TCSExplorer	1 964 621 385	2 033 485 372		TCSExplorer	54 906 655	55 136 475				
	POHH	2 375 666 512	2 439 427 249		justFall	55 108 940	58 268 698.3				
	justFall	2 509 944 603	2 639 808 947		POHH	55 585 100	56 052 686.1				

Note: The top six methods per each instance are reported and best values are highlighted in bold.

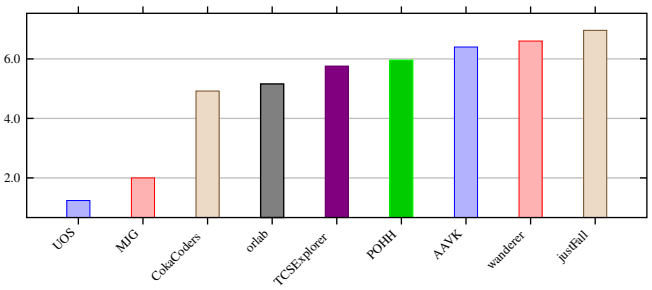

Figure 5 Mean ranks of the finalists teams and the POHH algorithm computed according to the competition rules [Color figure can be viewed at wileyonlinelibrary.com]

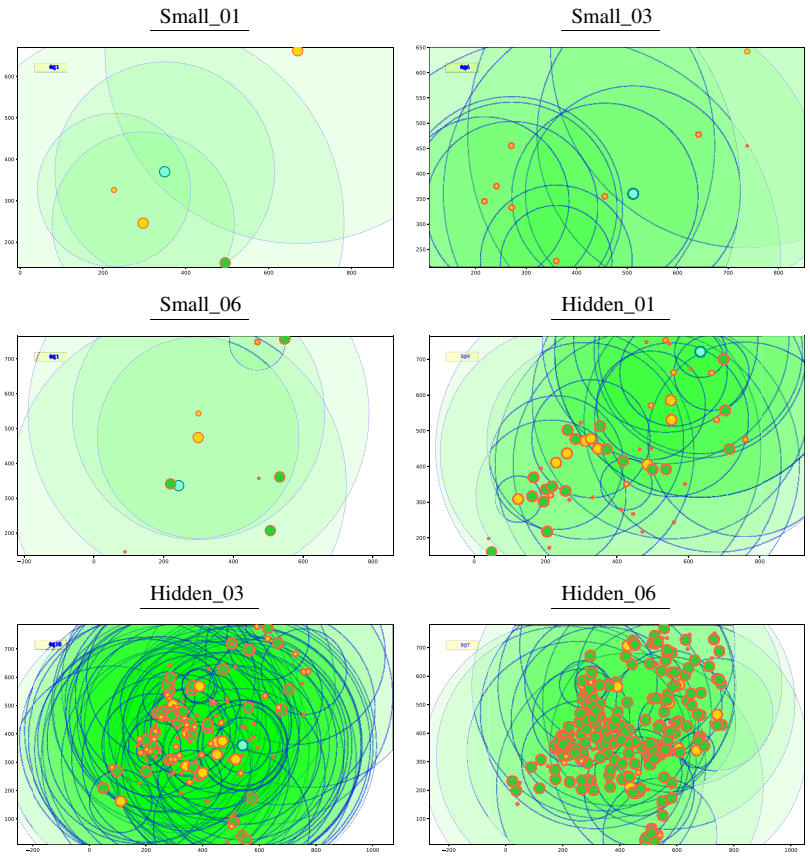

Figure 6 Visualization of sample instances [Color figure can be viewed at wileyonlinelibrary.com]

There are particular selection hyper-heuristic methods that clearly performed better than the rest in making improvements to the solutions during the search process. The incorporation of the RR-based selection hyper-heuristics in the POHH appears to play a key role of solving the problem in a relatively effective manner, in particular SS-RR which performed equally well in the small and hidden datasets. SR-GD was very successful in the larger size instances, where 50% of the improvement rate was achieved by SR-GD in Hidden_06 that has 900 requests. The least successful selection hyper-heuristics are the ones combined with the simulated annealing acceptance. The utilization of SR-SA was down to 0% in all instances, except for an insignificant improvement rate of 5% in Small_03. Also, SS-SA did make much contribution in terms of improvement for the hidden set. The naïve acceptance is more successful for the small instances than the larger ones, but only when it is combined with the sequence based selection method.

There seems to be a variation in the performance between the selection hyper-heuristics in instances with different complexities, and we cannot generalize that a certain combination of a selection and a move acceptance methods would be successful in every instance in this problem. An interesting idea would be to embed a high-level intelligent control mechanism that can observe these variations, and apply the components of selection hyper-heuristics accordingly during the search time, similar to the online selection methods. The random selection criteria that we apply currently in our framework was able to find "reasonable" results as reported, and we expect that the suggested improvements in the selection mechanism could yield even better results.

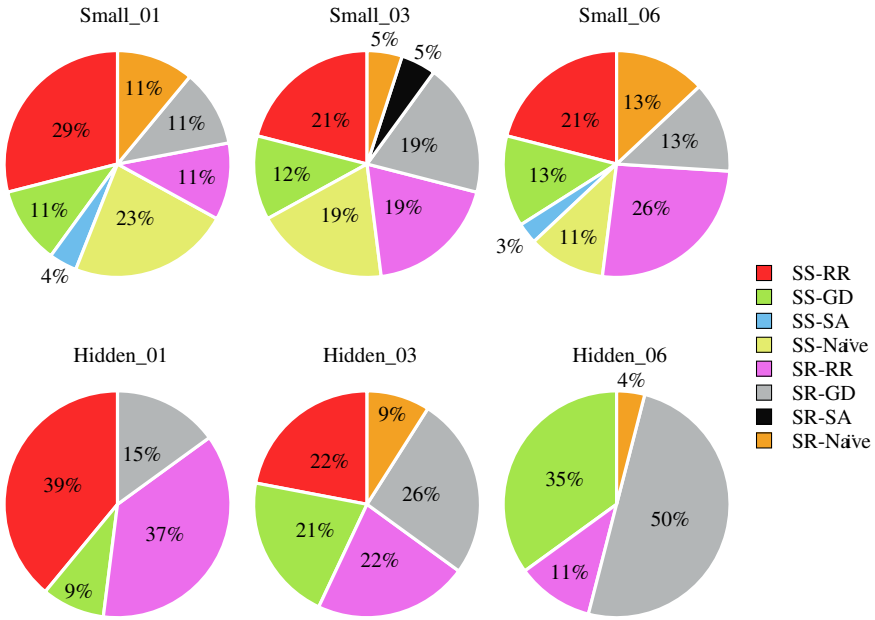

Figure 7 Average utilization rate [Color figure can be viewed at wileyonlinelibrary.com]

6.3.1 | Performance comparison to the constituent hyper-heuristics

Another round of experiments have been conducted by applying the eight selection hyper-heuristics employed in our framework independently on the instances displayed in Figure 6. Each selection hyper-heuristic was run for nine times using the same rules to calculate the run time of an instance described in this section. The results are displayed in Table 5 using the best and average objective values from the nine runs, along with the associated SD. The Mann-Whitney-Wilcoxon test is performed with a 95% confidence level in order to compare pairwise performance variations of two given algorithms statistically. The following notations are used: (a) "+" denotes that our algorithm (POHH) is better and this performance variance is statistically significant, (b) "−" denotes that the performance of POHH is worse and this performance variance is statistically significant, and (c) "=" indicates that there is no statistical significant between the two methods.

The POHH algorithm performed statistically better than each of the constituent hyper-heuristic for all instances, except Hidden_03, where the two methods SR-RR and SS-RR found slightly better averages and performed statistically better. Other than those two cases, the POHH algorithm found the best averages and minimum values in all instances, performing exceptionally better in particular on the largest instance of Hidden_06. This provides evidence for the success of two proposals: (a) utilizing multiple solutions allows better exploration and more possibilities for further improvement in new areas of the search space, instead of focusing on a single solution; (b) applying a sequence of selection hyper-heuristics to a solution might be useful in utilizing the varying performances of these selection hyper-heuristics. This can lead the search to different directions that can yield further improvement.

Overall, POHH creates a synergy among the eight selection hyper-heuristics resulting in an improved performance for almost all instances. Although POHH does not utilize learning, various learning mechanisms embedded into each one of the low level selection hyper-heuristics potentially contributes to the overall success of the proposed approach when compared to the performance of each constituent selection hyper-heuristic.

7 | CONCLUSION

In this study, we tackled a complex VRP problem which was the subject of the fourth edition of the VeRoLog solver challenge (2019). The challenge consisted of a novel VRP problem comprising two interdependent stages: a capacitated VRP problem with time windows (CVRPTW) for delivering various equipment to customers on their requests, and a service technician routing and scheduling problem (STRSP) for the installation of the delivered equipment. We propose two approaches, and apply them to the set of instances supplied by the competition organizers, and also to another small set of generated test instances. The first approach is an exact mathematical approach, which is the first attempt to implement an exact model for such version of VRP problem. We show that even with small sized instances, the method requires large amounts of computing time to solve the problem. Moreover we prove that the exact model cannot solve instances of large sizes. In light of this, and due to the large number of constraints and the wide range of differing instances which make it difficult to create problem-specific algorithms,

Table 5 The performance comparison of POHH, SS-RR, SS-GD, SS-SA, SS-Naïve, SR-RR, SR-GD, SR-SA and SR-Naïve based on the average (Avg), associated standard deviation (Std), minimum (Min) of the objective values over nine trials and the pairwise average performance comparison of POHH vs (SS-RR, SS-GD, SS-SA, SS-Naïve, SR-RR, SR-GD, SR-SA and SR-Naïve) based on Mann-Whitney-Wilcoxon for each instance produced by each approach

	Small_01				Small_03				Small_06			
Method	**vs**	**Avg**	**Std**	**Min**	**vs**	**Avg**	**Std**	**Min**	**vs**	**Avg**	**Std**	**Min**
POHH	=	**36 016 020**	0	**36 016 020**	=	**32 085 922**	83	**32 085 800**	=	**180 524**	0	**180 524**
SS-RR	=	**36 016 020**	0	**36 016 020**	+	32 087 251	858	**32 085 800**	+	247 213	49 513	**180 524**
SS-GD	=	**36 016 020**	0	**36 016 020**	+	32 094 142	3088	32 089 068	+	200 911	2055	198 515
SS-SA	=	36 016 433	620	**36 016 020**	+	32 114 432	9290	32 098 239	+	211 280	3317	206 996
SS-Naïve	=	**36 016 020**	0	**36 016 020**	+	32 087 868	1110	32 086 696	+	183 922	649	182 888
SR-RR	+	36 016 847	620	**36 016 020**	+	32 086 500	533	**32 085 800**	+	280 221	0	280 221
SR-GD	=	36 016 219	395	**36 016 020**	+	32 092 212	1004	32 090 760	+	199 464	2466	196 666
SR-SA	+	36 016 732	554	**36 016 020**	+	32 109 815	5690	32 104 880	+	210 099	2883	205 800
SR-Naïve	=	**36 016 020**	0	**36 016 020**	+	32 086 480	462	32 085 900	+	183 029	1056	182 161
	Hidden_01				**Hidden_03**				**Hidden_06**			
Method	**vs**	**Avg**	**Std**	**Min**	**vs**	**Avg**	**Std**	**Min**	**vs**	**Avg**	**Std**	**Min**
POHH	=	**68 683 339**	290 545	**68 151 265**	=	1 410 411 798	5 353 659	1 399 418 890	=	**34 157 264**	106 666	**34 008 705**
SS-RR	+	69 197 055	311 273	68 917 615	−	1 389 617 007	3 467 539	**1 381 431 185**	+	36 409 477	75 941	36 313 420
SS-GD	+	85 782 696	778 892	84 218 865	+	1 815 339 912	13 783 473	1 791 026 650	+	41 742 650	251 435	41 235 155
SS-SA	+	86 554 467	657 440	85 333 250	+	1 802 310 946	18 417 576	1 773 264 280	+	42 003 434	158 875	41 697 785
SS-Naïve	+	73 793 282	565 531	72 767 395	+	1 537 797 289	6 427 024	1 529 124 090	+	36 526 647	154 146	36 315 960
SR-RR	+	69 209 684	153 049	69 060 100	−	**1 387 767 927**	2 077 527	1 385 115 295	+	36 291 840	47 266	36 224 495
SR-GD	+	86 004 059	466 746	85 462 365	+	1 811 403 039	8 342 226	1 800 385 550	+	41 963 413	197 981	41 518 920
SR-SA	+	86 552 397	462 097	85 483 775	+	1 812 705 973	4 911 482	1 805 213 785	+	42 044 408	139 089	41 796 340
SR-Naïve	+	74 076 952	437 930	73 144 995	+	1 527 445 028	5 356 630	1 515 517 920	+	36 719 875	63 682	36 626 675

Note: The hyper-heuristic producing the best value for Avg and Min per each instance are highlighted in bold.

we proposed a problem-independent population-based hyper-heuristic algorithm (POHH). The algorithm maintains a number of solutions during the search, and a sequence of constituent selection hyper-heuristics together with a large set of low level heuristics which are applied to one solution at a time. The constituent hyper-heuristics have been proven to tackle a wide range of problem domains [12,37,66]. Our analysis shows the efficiency of the proposed hyper-heuristic algorithm compared to the results of the exact model, insofar as optimal solutions where found in shorter computational times. The hyper-heuristic results also compared well to the results of the eight finalists of the competition. The approach also performed better than the constituent hyper-heuristics performed on their own, for most of the instances. There is scope for further research into several aspects of POHH. One example is how to decide on which hyper-heuristic strategies to include into our population-based algorithm. Additionally, the algorithm does not have the ability to learn from history which strategy performs well. Thus, it may be beneficial to exclude certain strategies altogether to speed-up the algorithm.

ACKNOWLEDGMENTS

We express our gratitude to the VeRoLog board as well as the organizing committee for the VeRoLog Conference that was held in Seville, Spain, June 2019. We would like to thank the organizing team of the VeRoLog Solver Challenge 2019 and for developing the visualizer presented in the results section. KTP Scheme, Grant/Award Number: KTP11692.

ORCID

Ahmed Kheiri https://orcid.org/0000-0002-6716-2130

REFERENCES

[1] L. Ahmed, C. Mumford, and A. Kheiri, *Solving urban transit route design problem using selection hyper-heuristics*, Eur. J. Oper. Res. **274** (2019), 545–559.
[2] F. Alonso, M.J. Alvarez, and J.E. Beasley, *A tabu search algorithm for the periodic vehicle routing problem with multiple vehicle trips and accessibility restrictions*, J. Oper. Res.Soc. **59** (2008), 963–976.
[3] C. Archetti and M.G. Speranza, *Vehicle routing problems with split deliveries*, Int. Trans. Oper. Res. **19** (2012), 3–22.

[4] C. Archetti, M.G. Speranza, and M.W. Savelsbergh, *An optimization-based heuristic for the split delivery vehicle routing problem*, Transp. Sci. **42** (2008), 22–31.

[5] C. Archetti, N. Bianchessi, and M.G. Speranza, *A column generation approach for the split delivery vehicle routing problem*, Networks **58** (2011), 241–254.

[6] C. Archetti, O. Jabali, and M.G. Speranza, *Multi-period vehicle routing problem with due dates*, Comput. Oper. Res. **61** (2015), 122–134.

[7] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberán, D. Naddef, and G. Rinaldi, *Computational Results with a Branch and Cut Code for the Capacitated Vehicle Routing Problem*, France: IMAG. 1995.

[8] H. Bae and I. Moon, *Multi-depot vehicle routing problem with time windows considering delivery and installation vehicles*, Appl. Math. Model **40** (2016), 6536–6549.

[9] M.L. Balinski and R.E. Quandt, *On an integer program for a delivery problem*, Oper. Res. **12** (1964), 300–304.

[10] S. Belhaiza, P. Hansen, and G. Laporte, *A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows*, Comput. Oper. Res. **52** (2014), 269–281.

[11] S. Bertels and T. Fahle, *A hybrid setup for a hybrid scenario: Combining heuristics for the home health care problem*, Comput. Oper. Res. **33** (2006), 2866–2890.

[12] B. Bilgin, E. Özcan, and E.E. Korkmaz, "*An experimental study on hyper-heuristics and exam timetabling*," *International Conference on the Practice and Theory of Automated Timetabling*Springer, Berlin, 2006, pp. 394–412.

[13] M. Boudia, C. Prins, and M. Reghioui, "*An effective memetic algorithm with population management for the split delivery vehicle routing problem*," *International Workshop on Hybrid Metaheuristics*Springer, In, Berlin, 2007, pp. 16–30.

[14] E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J.R. Woodward, "*A classification of hyper-heuristic approaches: Revisited*," *Handbook of Metaheuristics*, vol. **272**Springer, Berlin, 2019, pp. 453–477.

[15] A.M. Campbell and J.H. Wilson, *Forty years of periodic vehicle routing*, Networks **63** (2014), 2–15.

[16] D. Cattaruzza, N. Absi, D. Feillet, and D. Vigo, *An iterated local search for the multi-commodity multi-trip vehicle routing problem with time windows*, Comput. Oper. Res. **51** (2014), 257–267.

[17] Y. Chen, P. Mourdjis, F. Polack, P. Cowling, and S. Remde, "*Evaluating hyperheuristics and local search operators for periodic routing problems*," *Evolutionary Computation in Combinatorial Optimization*Springer, In, Berlin, 2016, pp. 104–120.

[18] C.B. Cheng and K.P. Wang, *Solving a vehicle routing problem with time windows by a decomposition technique and a genetic algorithm*, Expert Syst. Appl. **36** (2009), 7758–7763.

[19] G. Clarke and J.W. Wright, *Scheduling of vehicles from a central depot to a number of delivery points*, Oper. Res. **12** (1964), 568–581.

[20] J.F. Cordeau, G. Laporte, F. Pasin, and S. Ropke, *Scheduling technicians and tasks in a telecommunications company*, J. Sched. **13** (2010), 393–409.

[21] P. Cowling, G. Kendall, and E. Soubeiga, "*A hyperheuristic approach to scheduling a sales summit*," *International Conference on the Practice and Theory of Automated Timetabling*Springer, In, Berlin, 2000, pp. 176–190.

[22] P. Cowling, G. Kendall, and L. Han, "*An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem*," *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, vol. **2**IEEE, 2002, pp. 1185–1190. https://ieeexplore.ieee.org/document/1004411

[23] G.B. Dantzig and J.H. Ramser, *The truck dispatching problem*, Manag. Sci. **6** (1959), 80–91.

[24] Q. Ding, X. Hu, L. Sun, and Y. Wang, *An improved ant colony optimization and its application to vehicle routing problem with time windows*, Neurocomputing **98** (2012), 101–107.

[25] J.H. Drake, A. Kheiri, E. Özcan, and E.K. Burke, *Recent advances in selection hyper-heuristics*, Eur. J. Oper. Res. **285** (2020), 405–428.

[26] M. Dror and P. Trudeau, *Savings by split delivery routing*, Transp. Sci. **23** (1989), 141–145.

[27] M. Dror and P. Trudeau, *Split delivery routing*, Naval Res. Logist. **37** (1990), 383–402.

[28] H. Fisher, *Probabilistic learning combinations of local job-shop scheduling rules*, Ind. Schedul. **3**(2) (1963), 225–251.

[29] M.L. Fisher and R. Jaikumar, *A generalized assignment heuristic for vehicle routing*, Networks **11** (1981), 109–124.

[30] R. Fukasawa, H. Longo, J. Lysgaard, M.P. de Aragão, M. Reis, E. Uchoa, and R.F. Werneck, *Robust branch-and-cut-and-price for the capacitated vehicle routing problem*, Math. Programming **106** (2006), 491–511.

[31] J. Gromicho, P. van't Hof, and D. Vigo, *The verolog solver challenge 2019*, J. Vehicle Routing Algor. **2** (2019), 109–111.

[32] W. Gu, D. Cattaruzza, M. Ogier, and F. Semet, *Adaptive large neighborhood search for the commodity constrained split delivery VRP*, Comput. Oper. Res. **112** (2019), 104761.

[33] P.C. Hsiao, T.C. Chiang, and L.C. Fu, "*A VNS-based hyper-heuristic with adaptive computational budget of local search*," *2012 IEEE Congress on Evolutionary Computation*IEEE, New York, 2012, pp. 1–8.

[34] A. Kheiri, *Heuristic sequence selection for inventory routing problem*, Transp. Sci. **54** (2020), 302–312.

[35] A. Kheiri and E. Keedwell, "*A sequence-based selection hyper-heuristic utilising a hidden Markov model*," *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference, GECCO '15*, New York: ACM; 2015, pp. 417–424.

[36] A. Kheiri and E. Keedwell, *A hidden Markov model approach to the problem of heuristic selection in hyper-heuristics with a case study in high school timetabling problems*, Evol. Comput. **25** (2017), 473–501.

[37] A. Kheiri, E. Keedwell, M.J. Gibson, and D. Savic, *Sequence analysis-based hyper-heuristics for water distribution network optimisation*, Procedia Eng. **119** (2015), 1269–1277.

[38] A. Kheiri, A.G. Dragomir, D. Mueller, J. Gromicho, C. Jagtenberg, and J.J. van Hoorn, *Tackling a vrp challenge to redistribute scarce equipment within time windows using metaheuristic algorithms*, EURO J. Transp. Logistics **8** (2019), 561–595.

[39] A.W. Kolen, A. Rinnooy Kan, and H.W. Trienekens, *Vehicle routing with time windows*, Oper. Res. **35**(2) (1987), 266–273.

[40] A.A. Kovacs, S.N. Parragh, K.F. Doerner, and R.F. Hartl, *Adaptive large neighborhood search for service technician routing and scheduling problems*, J. Sched. **15** (2012), 579–600.

[41] G. Laporte, *Fifty years of vehicle routing*, Transp. Sci. **43** (2009), 408–416.

[42] A. Lehrbaum and N. Musliu, "*A New Hyperheuristic Algorithm for Cross-Domain Search Problems*," *International Conference on Learning and Intelligent Optimization*Springer, Berlin, 2012, pp. 437–442.

[43] Y. Lei, M. Gong, L. Jiao, and Y. Zuo, *A memetic algorithm based on hyper-heuristics for examination timetabling problems*, Int. J. Intell. Comput. Cybern. **8** (2015), 139–151.

[44] M. López-Ibáñez, J. Dubois-Lacoste, L.P. Cáceres, M. Birattari, and T. Stützle, *The irace package: Iterated racing for automatic algorithm configuration*, Oper. Res. Persp. **3** (2016), 43–58.

[45] J. Lysgaard, A.N. Letchford, and R.W. Eglese, *A new branch-and-cut algorithm for the capacitated vehicle routing problem*, Math. Programming **100** (2004), 423–445.

[46] D. Mester and O. Bräysy, *Active-guided evolution strategies for large-scale capacitated vehicle routing problems*, Comput. Oper. Res. **34** (2007), 2964–2975.
[47] S. Mirzaei and S. Wøhlk, *Erratum to: A branch-and-price algorithm for two multi-compartment vehicle routing problems*, EURO J. Transp. Logist. **6** (2017), 185–218.
[48] M. Misir, P. Smet, K. Verbeeck, and G. Vanden Berghe, "*Security personnel routing and rostering: A hyper-heuristic approach*," *Proceedings of the 3rd International Conference on Applied Operational Research*, vol. 3, Canada: Tadbir; 2011, pp. 193–205.
[49] Y. Nagata, "*Edge assembly crossover for the capacitated vehicle routing problem*," *European Conference on Evolutionary Computation in Combinatorial Optimization*Springer, Berlin, 2007, pp. 142–153.
[50] Y. Nagata and O. Bräysy, *Edge assembly-based memetic algorithm for the capacitated vehicle routing problem*, Netw. Int. J. **54** (2009), 205–215.
[51] ORTEC, VeRoLog. "Manual of the tool for analysing instances and solutions"; 2019. https://verolog2019.ortec.com/zips/VSC2019VizAnaTools.zip
[52] V. Pillac, C. Gueret, and A.L. Medaglia, *A parallel matheuristic for the technician routing and scheduling problem*, Optim. Lett. **7** (2013), 1525–1535.
[53] D. Pisinger and S. Ropke, *A general heuristic for vehicle routing problems*, Comput. Oper. Res. **34** (2007), 2403–2435.
[54] J.Y. Potvin and J.M. Rousseau, *A parallel route building algorithm for the vehicle routing and scheduling problem with time windows*, Eur. J. Oper. Res. **66** (1993), 331–340.
[55] C. Prins, *A simple and effective evolutionary algorithm for the vehicle routing problem*, Comput. Oper. Res. **31** (2004), 1985–2002.
[56] A. Rahimi-Vahed, T.G. Crainic, M. Gendreau, and W. Rei, *A path relinking algorithm for a multi-depot periodic vehicle routing problem*, J. Heuristics **19**(3) (2013), 497–524.
[57] R.A. Russell, *Hybrid heuristics for the vehicle routing problem with time windows*, Transp. Sci. **29**(2) (1995), 156–166.
[58] N.R. Sabar and G. Kendall, *Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems*, Inform. Sci. **314** (2015), 225–239.
[59] M.M. Solomon, *Algorithms for the vehicle routing and scheduling problems with time window constraints*, Oper. Res. **35** (1987), 254–265.
[60] H. Sontrop, P. Van Der Horn, and M. Uetz, "*Fast ejection chain algorithms for vehicle routing with time windows*," *International Workshop on Hybrid Metaheuristics*Springer, Berlin, 2005, pp. 78–89.
[61] A. Tatarakis and I. Minis, *Stochastic single vehicle routing with a predefined customer sequence and multiple depot returns*, Eur. J. Oper. Res. **197**(2) (2009), 557–571.
[62] R. Tavakkoli-Moghaddam, M. Gazanfari, M. Alinaghian, A. Salamatbakhsh, and N. Norouzi, *A new mathematical model for a competitive vehicle routing problem with time windows solved by simulated annealing*, J. Manuf. Syst. **30** (2011), 83–92.
[63] P. Tsirimpas, A. Tatarakis, I. Minis, and E. Kyriakidis, *Single vehicle routing with a predefined customer sequence and multiple depot returns*, Eur. J. Oper. Res. **187** (2008), 483–495.
[64] E. Urra, C. Cubillos, and D. Cabrera-Paniagua, *A hyperheuristic for the dial-a-ride problem with time windows*, Math. Probl. Eng. **2015** (2015), 1–12. https://doi.org/10.1155/2015/707056.
[65] J.D. Walker, G. Ochoa, M. Gendreau, and E.K. Burke, "*Vehicle routing and adaptive iterated local search within the Hyflex hyper-heuristic framework*," *International Conference on Learning and Intelligent Optimization* Springer, Berlin, 2012, pp. 265–276.
[66] D. Wilson, S. Rodrigues, C. Segura, I. Loshchilov, F. Hutter, G.L. Buenfil, A. Kheiri, E. Keedwell, M. Ocampo-Pineda, E. Özcan, S.I.V. Peña, B. Goldman, S.B. Rionda, A. Hernández-Aguirre, K. Veeramachaneni, and S. Cussat-Blanc, *Evolutionary computation for wind farm layout optimization*, Renew Energy **126** (2018), 681–691.
[67] F. Xie, C.N. Potts, and T. Bektaş, *Iterated local search for workforce scheduling and routing problems*, J. Heuristics **23** (2017), 471–500.
[68] J. Xu and S.Y. Chiu, *Effective heuristic procedures for a field technician scheduling problem*, J. Heuristics **7** (2001), 495–509.
[69] Y. Zhang and X. Chen, *An optimization model for the vehicle routing problem in multi-product frozen food delivery*, J. Appl. Res. Technol. **12** (2014), 239–250.

How to cite this article: Kheiri A, Ahmed L, Boyacı B, et al. Exact and hyper-heuristic solutions for the distribution-installation problem from the VeRoLog 2019 challenge. *Networks.* 2020;1–26. https://doi.org/10.1002/net.21962