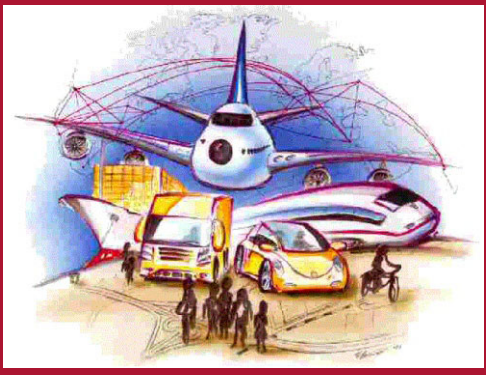
Carlos Paternina-Arboleda
Stefan Voß (Eds.)

LNCS 11756

Computational Logistics

10th International Conference, ICCL 2019
Barranquilla, Colombia, September 30 – October 2, 2019
Proceedings



Springer

Lecture Notes in Computer Science 11756

A Greedy Heuristic for the Vehicle Routing Problem with Time Windows, Synchronization Constraints and Heterogeneous Fleet

Luísa Brandão Cavalcanti(✉) and André Bergsten Mendes

Departamento de Engenharia Naval e Oceânica, Universidade de São Paulo, Av Prof. Mello Moraes 2231, Sao Paulo, SP 05508-030, Brazil
luisa.cavalcanti@usp.br

Abstract. This paper focuses on a new variant of the Vehicle Routing Problem with Time Windows and Synchronization constraints (VRPTWSyn), inspired by a routing problem faced by the oil & gas industry. Here, a heterogeneous fleet of Anchor Handling Tug Supply vessels (AHTS) must be assigned to perform tasks at offshore platforms, and most tasks require the simultaneous action of more than two tugs. Instances are proposed, as well as a greedy randomized heuristic that generates feasible solutions to the problem. As shown by the computational experiments reported here, although the complexity of the AHTS-routing problem is higher than the VRPTWSyn, the proposed method could produce a diverse set of feasible solutions, which in the future may be improved by a meta-heuristic-based algorithm.

Keywords: Anchor Handling Tug Supply · Vehicle Routing Problem · Synchronization · Greedy heuristic · Oil & gas

1 Introduction

Recent literature has shed a light on the Vehicle Routing Problem with Time Windows and Synchronization constraints (VRPTWSyn), applied to various contexts [1–3]. This proven NP-hard problem [4] consists of scheduling the execution of tasks at geographically spread clients that may demand the simultaneous presence of two or more vehicles and that must be started within a given time window [5]. For instance, vehicles may represent caregivers performing health related activities at patients' homes, as in [2, 4–8], or employees that provide different types of services at their clients' locations, such as installing electronic equipment and furniture [1, 9, 10].

The VRPTWSyn may also be applied to create routes for Anchor Handling Tug Supply vessels (AHTS) that must execute activities at offshore oil platforms, as described in [11, 12]. These include towing and anchoring drilling rigs at production units, performing maintenance and installation of underwater equipment, pulling shuttle tankers during offloading operations in order to avoid collision, amongst others (for a general overview of offshore logistics problems, we refer to [13]). In such context, differences amongst the available tugs must be considered when assigning them to tasks, since some tasks may demand vessels of higher potency, whereas other

C. Paternina-Arboleda and S. Voß (Eds.): ICCL 2019, LNCS 11756, pp. 265–280, 2019.
https://doi.org/10.1007/978-3-030-31140-7_17

may be accomplished by simpler, cheaper tugs. Moreover, tasks usually must be executed simultaneously by more than two tugs, which has not been successfully covered in the literature, as instances of the VRPTWSyn found in previous works are restricted to at most two vehicles per client, and the percentage of clients with synchronization requirements is low (while most papers adopt 10% [1, 2, 4, 5, 8], [14] uses 20%, [7] uses 30% and [10] proposes instances with up to 50% synchronized visits). These intrinsic characteristics of the oil industry problem add an extra layer of complexity to the VRPTWSyn, resulting in a new variant of the mathematical model that, to the best of the authors' knowledge, has not been discussed in the literature yet, except for previous works of the authors themselves, as in [12].

In this paper, we address the hereby called Vehicle Routing Problem with Time Windows, Synchronization constraints and Heterogeneous fleet (VRPTWSyn-Het), by introducing its optimization model and presenting a greedy heuristic that produces feasible solutions to the problem. To evaluate our solution method's performance, the heuristic is applied to 3 sets of 12 instances with 10, 20 and 30 tasks, and its results are compared with the ones produced by the Gurobi optimization package. It is important to note that the main goal of the proposed procedure is to produce feasible solutions that may be used as a start to local search procedures and meta-heuristics, so methodologies for improving the solution quality are left for future work. Although this may sound like an easy target, our computational experiments show that Gurobi had major difficulties in finding feasible solutions for instances with 20 and 30 tasks, which justifies the need of building an efficient algorithm to find feasible tug schedules, as well as it illustrates how the VRPTWSyn-Het differs from the VRPTWSyn, which has had instances of up to 50 clients solved to optimality by [14].

The structure of this paper is as follows: in Sect. 2, the VRPTWSyn-Het is properly defined and its mathematical optimization model is introduced; then, a greedy heuristic for solving the VRPTWSyn-Het is documented in Sect. 3; and its performance is attested in Sect. 4. Lastly, in Sect. 5, conclusions and recommendations for future work are presented.

2 Problem Definition and Model Formulation

The VRPTWSyn-Het consists of routing a heterogeneous fleet of non-capacitated vehicles (AHTS) to perform activities at geographically spread locations. The fleet is subdivided into classes according to their potency, such that the vehicles from a class c have lower power levels than vehicles of class $c + 1$. The higher the potency, the higher the fuel consumption of the tug, so variable transportation costs are also higher. Each activity demands a minimum number of vehicles of each class and must be started within a time window, defined by a lower and an upper time limitation. Vehicles of the same class are interchangeable between one another and vehicles of any given class can execute all the tasks that can be performed by vehicles of a lower class. In other words, a vehicle of class c that can execute a task i may be replaced by any vehicle of class $c + 1$, even though the resulting variable costs will probably be higher. Tasks may start and end at different locations, as they include transportation of equipment from one platform to another. A task will only be started after all demanded vehicles arrive at its

starting point and, once it is started, all vehicles assigned will be released to perform other tasks just after its conclusion. Each tug r is associated with a class c_r and a release date s^r.

The following mathematical formulation of the VRPTWSyn-Het is based on previous works that addressed the VRPTWSyn, with some changes to accommodate the specificities of the AHTS routing problem. The VRPTWSyn-Het is formulated as a directed graph $G = (V, A)$ composed by a set V of vertices and a set A of arcs. Each task j is represented by a unique node, being the set of task nodes denoted by J. There are two vertices in V for each tug $r \in M$ to represent the start and end of its route, O_r^+ and O_r^-, but the former represents an actual location, whereas the latter is a dummy node used to avoid sub cycles. The set of arcs A comprises all pairwise connections between nodes of J, plus arcs connecting each O_r^+ to all nodes of $J \cup O_r^-$, and all the nodes of J to the nodes O_r^-. Each arc $(i, j) \in A$ represents the movement of a tug from a task/origin node i to a task/origin node j, so a cost c_{ij}^r for vehicle r to transverse arc (i, j) is also defined for each pair of tug and arc. To execute all demanded tasks, a fleet of m tugs, subdivided into l classes, is available. The set of all tugs is denoted by M, the subset of tugs of class c is $M^c \subset M$ and the set of all classes is C.

Each task j must be started at any time within its time window $[a_j, b_j]$, *i.e.* no earlier than a date a_j and no later than b_j. However, if it is initiated after a given $\hat{b}_j$ $(a_j \leq \hat{b}_j \leq b_j)$, which is the actual desired day for starting the task, a daily financial penalty w_j applies. The duration p_j of task j includes the eventual transportation time from its start to its end location. Finally, the basic mode for executing j is represented by a demand vector $D_j = \{d_j^1, d_j^2 \ldots d_j^1\}$.

Finally, a solution S to the VRPTWSyn-Het must assign a value to each of the following decision variables: x_{ij}^r, binary that equals one if tug r is assigned to execute task j right after finishing task i; s_j^r, continuous variable that equals the date of arrival of tug r at the starting location of task j; $\hat{s}_j$, another continuous variable that indicates the starting time of task j, when all assigned tugs are available at their starting location; and T_j, the resulting delay of task j, measured from its desired starting time $\hat{b}_j$ and within its time window. The objective function $z(S)$ is:

$$\min z(S) = \sum_{j \in J} T_j w_j + \sum_{i \in N} \sum_{j|(i,j) \in A} \sum_{r \in M} x_{ij}^r c_{ij}^r \tag{1}$$

Subject to:

$$\sum_{j \in J \cup O_k^-} x_{O_r^+ j}^r = 1 \quad \forall r \in M \tag{2}$$

$$\sum_{j \in J \cup O_r^+} x_{j O_r^-}^r = 1 \quad \forall r \in M \tag{3}$$

$$\sum_{r \in \mathrm{M} | c_r \geq c} \sum_{(i,j) \in A} x_{ij}^r \geq d_j^c \quad \forall j \in J, \ \forall c \in C \tag{4}$$

$$\sum_{r \in M} \sum_{(i,j) \in A} x_{ij}^r \geq \sum_{c \in C} d_j^c \quad \forall j \in J \tag{5}$$

$$\sum_{(i,j)\in A} x_{ij}^{r} = \sum_{(j,k)\in A} x_{jk}^{r} \quad \forall r \in M, j \in J \tag{6}$$

$$s_j^r \geq s^r + t_{O_r^+ j}^r - \left(1 - x_{O_r^+ j}^r\right)K \quad \forall r \in M,\ \forall\left(O_r^+, j\right) \in \mathrm{A} \tag{7}$$

$$s_j^r \geq \hat{s}_i + p_i + t_{ij}^r - \left(1 - x_{ij}^r\right)K \quad \forall r \in M,\ \forall (i,j) | \mathrm{i} \in \mathrm{J}, \mathrm{j} \in \mathrm{J}, \mathrm{i} \neq \mathrm{j} \tag{8}$$

$$\hat{s}_j \geq s_j^{\mathrm{r}} \quad \forall j \in J,\ \forall r \in M \tag{9}$$

$$\hat{s}_j \geq a_{\mathrm{j}} \quad \forall j \in J \tag{10}$$

$$\hat{s}_j \leq b_j \quad \forall j \in J \tag{11}$$

$$T_j \geq \hat{s}_j - \hat{b}_j \quad \forall j \in J \tag{12}$$

$$x_{ij}^r \in \{0,1\} \quad \forall (i,j) \in A, r \in M \tag{13}$$

$$\hat{s}_j, T_j \in R | \hat{s}_j, T_j \geq 0 \quad \forall j \in J \tag{14}$$

$$\mathrm{s}_{\mathrm{j}}^{\mathrm{r}} \in R | \mathrm{s}_{\mathrm{j}}^{\mathrm{r}} \geq 0 \quad \forall j \in J, r \in M \tag{15}$$

In Eq. (1), the first part of *z(s)* is the total penalty costs of delayed tasks, whereas the second part is the total transportation costs of all arcs traveled by the tug fleet. Constraints (2) state that for each tug, a single arc leaving from its origin O_r^+ must be selected, similarly to constraints (3), which restrict the selection of a single arc ending at its final destination O_r^- (note that this arc may be from O_r^+ to O_r^- for both cases).

Inequalities set by (4) and (5) guarantee the fulfillment of each task's demand vector, by assuring that for each class c and task j, the number of designated tugs of this class and above is greater or equal to the demanded d_j^c tugs; and that for each task the total number of designated tugs is equal to the total amount of demanded tugs, regardless of their classes. Constraints (6) state that all the vessels that travel to a task node j must leave it. The inequalities (7)–(9) force the starting time of each task to be greater or equal to the arrival time of each of the vessels assigned to perform it, whereas constraints (10) and (11) guarantee tasks will be started within their time windows. Constraints (12) calculate the penalized delay of all tasks, and the domain of all decision variables is stablished by expressions (13), (14) and (15).

3 Solution Procedure

To generate feasible solutions to the VRPTWSyn-Het, a greedy heuristic with randomized steps was developed. Before presenting the algorithm of the Constructive Heuristic for the VRPTWSyn-Het (VRPTWSyn-Het_CH), it is important to define a couple of terms that are used throughout this paper.

As any task with synchronization constraints that requires low class tugs may be executed by tugs of higher classes, when one lists all subsets of tugs that are compatible with such tasks, the number of vessels of each class will not be the same in all subsets (although the total amount of tugs must be the same). For instance, if task *i* requires two class-1 tugs and one of class 2, subsets of compatible tugs may have: (*i*) 2 class-1 tugs and 1 class-2; (ii) 1 of class 1 and 2 of class 2; (iii) none of class 1 and 3 class-2 tugs. To distinguish subsets with different numbers of tugs of each class, we borrow the term *mode* from the scheduling literature, more specifically from the multi-processor scheduling problems [15, 16]. Here, a *mode* is a combination of the number of tugs from each available class, such that, in the previous example, there are three modes for executing task *i*. We also name the *basic mode* as the combination that uses the least higher-class tugs, which in the previous example is the first enumerated mode (2,1).

Another useful term we borrow from scheduling literature is the *slack time* of a task. In this problem, however, we define the slack time of a task as the amount of time it may be postponed (from its current starting time) without breaking any of the routed tasks' time windows, including its own. The use of slack times is crucial to the viability of the proposed heuristic, because it spares the method from having to check time windows of all tasks that may be affected by the insertion of a new task every time an insertion option is evaluated. As stated elsewhere [14], synchronizations constraints generate complex interconnections between tasks, making routing problems extremely difficult to solve, so to find a way of smoothing calculations related to route changes is an important gap to bridge in this field.

Algorithm 1. VRPTWSyn-Het_CH algorithm

```
VRPTWSyn-Het_CH(J, M, nExp):
  Initialize Routes and it ← 1
  While no solution has been found AND it ≤ nExp:
    S ≔ sorted tasks of set J
    While S is not empty:
      job ← select_task_from(S, jPos)
      S ← S \{job}
      Modes ≔ all_modes_compatible_with(job)
      Options ≔ ∅
      For each mode in Modes:
        M̃_mode ← set will all subsets of tugs type mode
        For each tug subset m̃ ⊂ M̃_mode:
          For combination of insertion positions in m̃:
            Routes' ≔ insert(job, m̃, positions in m̃, Routes)
            If Routes' feasible: Options ← Options ∪ Routes'
          End loop that exhaustively checks m̃
        End loop that exhaustively checks mode
        If Options is not empty, exit loop
      End loop that iterates through Modes
      If Options is empty: exit while loop
      Else: Routes ←select_option_from(Options, oPos)
    End loop that removes task by task from S
    it ← it + 1
  End while loop
  Return Routes
End
```

The VRPTWSyn-Het_CH algorithm, presented by Algorithm 1, starts by initializing a set *Routes* of m vectors $route_r$, with r varying from 1 to m, and each $route_r$ is initialized with two elements: the initial and final depot nodes. Then, it performs the following procedure until a solution has been found or the maximum number of restarts is reached.

At the beginning of an iteration, demanded tasks are sorted according to relevant characteristics of each task, and stored in set $\boldsymbol{S}$. As later described, we have tested 8 different sorting rules and selected the one with the best performance. Next, the algorithm removes a task from $\boldsymbol{S}$, by randomly picking one of the first $\boldsymbol{jPos}$ of $\boldsymbol{S}$ (method *select_task_from*), and then checks possibilities of inserting the task into the routes of vessels that are compatible with the task. However, instead of exhaustively testing all the insertion possibilities in all compatible subsets of tugs, the algorithm starts with the basic mode for fulfilling the task and, if there is at least one feasible partial solution that can be generated using such mode, it will not look any further. In case none of the tug subsets for the basic mode may have the task added to their routes, the algorithm will exhaustively check the insertion possibilities for the next mode (second lowest amount of high class vessels used), and so on, until a feasible partial solution can be generated or all modes have been tested.

Hence, when testing a mode, all the vessel subsets of such mode are considered and, for each subset, all the combinations of insertion positions from each route are also checked. To expedite this exhaustive procedure, we recalculate the slack times of all routed tasks every time the set of routes is updated (i.e. have a task added to one or more vessels' routes), so when trying to add a task i before j at a tug's route r, we calculate the new starting time of task j after such insertion and only accept the insertion if the difference from j's current starting time is equal or smaller than j's current slack time. Not only does this guarantee that task j will have its time window respected, but also that all other impacted tasks will have their time windows preserved. One should also note that, since the starting time of any task depends on the arrival of all assigned tugs, it is possible that the insertion of i into r will not impact any of the following tasks, in case tug r was originally waiting for the arrival of another tug to start task j and had enough idle time to accommodate i.

Every time an insertion check leads to a feasible solution, the partial routes generated by such insertion are added to a set *Options*, sorted according to the cost increment caused by the insertion. Once a mode with feasible solutions has been found and exhaustively examined, the algorithm randomly chooses amongst the first *oPos* partial solutions stored in *Options* and updates *Routes* accordingly. Otherwise, if a task may not be inserted in any of the compatible tug subsets (considering all possible modes), the loop that removes task by task from S is finished earlier and the whole procedure is restarted, provided that the maximum number of restarts has not been reached yet.

The constructive heuristic is finished when a complete solution is found (all tasks have been successfully added to the routes of compatible tugs and their time windows are respected) or the maximum number of restarts has been reached. However, as the purpose of the heuristic is to find a feasible solution to all instances, the maximum number of restarts was only used for tuning experiments, with which the values for *jPos* and *oPos* were determined, as explained in the next section.

It is worth mentioning that in previous versions of the heuristic, at the step of checking the insertion possibilities for a task, the heuristic looked for all modes and picked the cheapest option. However, this final version, which restricts the search to the first viable mode, outperformed the original algorithm in terms of quality and computational time.

4 Computational Experiments

Computational experiments conducted on the VRPTWSyn-Het_CH included the definition of sorting rule, parameter-tuning tests, efficiency evaluation and assertion of solution quality and diversity. More specifically:

- Definition of sorting rule: eight sorting rules, based on classical scheduling approaches for minimizing total tardiness (see [17] for more details), were tested using the VRPTWSyn-Het_CH with both *jPos* and *oPos* equal to one (which is equivalent to a non-randomized version of the heuristic, as the first job of the sorted test is always selected as the next job to insert into routes of the partial solution, as well as the insertion positions yielding the cheapest partial solution).

- Parameter-tuning tests: to calibrate *jPos*, the value of *oPos* was fixed as one and the heuristic was tested with *jPos* set from one to ten. With *nExp* reduced to one hundred restarts, the heuristic was executed ten times for each value of *jPos*, then medium performances were compared with each other. Similarly, the heuristic was executed ten times to calibrate the value of *oPos*, which varied from one to twenty-one whereas *jPos* and *nExp* were fixed at one and one hundred, respectively. Such value ranges were obtained with preliminary tests on randomly picked instances.
- Efficiency evaluation: once *jPos* and *oPos* were tuned, the resulting algorithm was run with *nExp* equal to ten thousand restarts, to enable finding a solution to all proposed instances. Again, ten executions were performed for each instance, to evaluate the robustness of the method. The computational time spent to generate a feasible solution was compared to the time Gurobi needed to find a single solution.
- Assertion of solution quality and diversity: even though the main goal of the constructive heuristic was to find feasible solutions to difficult instances of the VRPTWSyn-Het, we also tested it as a way of producing good solutions to the problem. In these tests, the algorithm was applied with just one stop criterium, the maximum number of restarts. Here, two thousand restarts were allowed and, although the best solution found was retrieved by the method, all feasible solutions were also stored, which allowed us to check how diverse the set of produced solutions is. As we intend to apply GRASP to the VRPTWSyn-Het, having a diverse set of initial solutions may be useful to future developments of this research.

All tests reported here were conducted on a PC notebook with Intel Core i7-6500U CPU, 2.50–2.59 GHz and 8 GB RAM, using Microsoft Visual Studio 2015 and Gurobi version 8.1.0. Next, a description of the instances is provided, along with the test results.

4.1 Instances for the VRPTWSyn-Het

Three sets of 12 instances each are proposed for the VRPTWSyn-Het, with 10, 20 and 30 tasks. These instances were derived from [11], with a reduction of the fleet's size and homogenization of the tug characteristics within each tug class. All instances have two classes of tugs and their planning horizon is a month (starting dates of the tasks are within 30 days).

Table 1 shows the main characteristics that describe an instance: number of demanded tasks; number of tasks with synchronization needs; average time length of the tasks; fleet's size and distribution across classes; number of vehicles required per task (on average and the maximum value). Dashed lines mark the transition between the three sets of instances, according to their sizes (total number of demanded tasks).

It should be noted that for each instance a task requires, on average, the simultaneous action of at least 2.4 vessels, but for some of them this figure can go up to 2.8. At most, five tugs are required per task, but most instances have no tasks demanding more than four tugs. The mean percentage of tasks with synchronized needs is 86% across all instances, but in four of them, all tasks have synchronized requirements. Instance 11 has the smallest number of synchronized tasks, with only seven of them (or 70% of the total) requiring more than one tug.

Table 1. Main characteristics of the proposed instances

ID	# tasks (n)	# sync. tasks	Avg. p_j [days]	Fleet's size			Tugs per task	
				Class-1 tugs	Class-2 tugs	Total (m)	Avg #tugs/task	Max. #tugs/task
1	10	8	9.9	8	4	12	2.4	4
2	10	10	9.9	8	4	12	2.6	4
3	10	9	10.1	9	4	13	2.8	4
4	10	9	11.9	8	5	13	2.4	4
5	10	9	12.1	8	4	12	2.6	4
6	10	9	12.0	11	5	16	2.8	4
7	10	10	10.1	8	3	11	2.4	4
8	10	8	9.9	8	4	12	2.6	4
9	10	9	10.1	9	4	13	2.8	4
10	10	8	12.0	8	4	12	2.4	4
11	10	7	12.0	10	4	14	2.6	4
12	10	10	12.0	10	5	15	2.8	4
13	20	15	9.9	16	6	22	2.4	5
14	20	17	9.9	11	9	20	2.6	4
15	20	19	9.9	14	9	23	2.8	4
16	20	15	12.0	20	7	27	2.4	4
17	20	17	12.0	22	7	29	2.6	4
18	20	18	12.1	20	7	27	2.8	4
19	20	16	10.0	11	6	17	2.4	4
20	20	18	9.5	11	5	16	2.6	5
21	20	20	9.1	20	7	27	2.8	4
22	20	15	12.0	14	7	21	2.4	4
23	20	17	12.1	16	6	22	2.6	4
24	20	19	12.0	18	8	26	2.8	4
25	30	24	9.9	14	6	20	2.4	5
26	30	25	10.1	14	7	21	2.6	4
27	30	27	10.1	15	8	23	2.8	4
28	30	26	12.0	15	9	24	2.4	4
29	30	26	12.0	16	9	25	2.6	4
30	30	28	12.1	17	9	26	2.8	4
31	30	24	10.0	14	7	21	2.4	4
32	30	24	10.0	16	6	22	2.6	4
33	30	25	10.0	16	8	24	2.8	4
34	30	24	12.1	16	8	24	2.4	4
35	30	26	12.0	19	9	28	2.6	4
36	30	27	12.0	19	10	29	2.8	4

4.2 Tuning Experiments

The following sorting rules were tested on instances with 10 and 20 tasks, using a non-randomized version of the VRPTWSyn-Het_CH:

1. Non-decreasingly by a_j (lower limit of the task's time window), or by b_j (its upper limit) whenever there is a tie;
2. Non-increasingly by total amount of vessels demanded, or non-decreasingly by the demand of upper-class vessels in case of a tie;
3. Non-decreasingly by b_j, or non-increasingly by p_j (task's duration) for ties;
4. Non-decreasingly by b_j, or non-decreasingly by the demand of upper-class vessels for ties;
5. Non-decreasingly by *bj*, or non-increasingly by total amount of vessels demanded;
6. Non-increasingly by total demanded (in number of tugs required) multiplied by p_j, untied by b_j (non-decreasingly)
7. Non-increasingly by compatibility degree, untie by duration of time window $(b_j - a_j)$;
8. Non-increasingly by compatibility degree, untie by total demand;

Rules 7 and 8 use the concept of *compatibility degree* of a task, defined next. As a way of measuring how difficult it is to fit a given task into a tug's route, we checked if such task is able to share a tug with each one of the remaining $n - 1$ tasks of J, considering their time windows and the travel time needed, using the speed of the fastest tug available. The compatibility degree of a task is then equal to the number of tasks it can share a route with.

Table 2 shows the average number of backlogged tasks for each instance set and sorting rule, as well as the number of solved instances by each rule. It can be noted that the 7th rule outperformed the others and, therefore, was chosen for the next tests.

Table 2. Results of experiments to define the sorting rule

Result	Instance set (# of tasks)	Sorting rule							
		1	2	3	4	5	6	7	8
Solved instances	n = 10	10	6	9	9	11	10	8	9
	n = 20	5	2	6	6	7	4	6	7
Avg. backlogged tasks	n = 10	1.0	1.2	1.0	1.0	1.0	1.0	1.0	1.0
	n = 20	2.1	2.5	1.8	2.5	1.4	1.8	1.7	2.0

The main results of the parameter tuning experiments are presented by Tables 3 and 4. Table 3 presents the number of solved instances for each value of *jPos*, showing how it varied on each execution performed, whereas Table 4 shows the heuristic's performance for each value of *oPos*. Based on these results, the values of *jPos* and *oPos* were set to 3 and 16, respectively, as these values yielded the biggest number of solved instances, on average (for the ten executions).

Table 3. Calibration of *jPos*

jPos	Avg. unsolved instances	Avg. Restarts needed for solved instances	Avg. Restarts needed considering all instances
1	10.0	0.0	35.6
2	2.8	1.9	11.8
3	**2.3**	**2.6**	**10.7**
4	2.4	2.9	11.3
5	2.6	3.9	12.8
6	3.0	4.1	14.4
7	2.7	5.0	14.1
8	3.3	4.6	15.8
9	3.5	5.3	17.0
10	3.7	5.0	17.4
11	3.9	6.3	19.2

Table 4. Calibration of *oPos*

oPos	Avg. unsolved instances	Avg. Restarts needed for solved instances	Avg. Restarts needed considering all instances
1	13.0	0.0	35.8
2	5.1	2.0	15.8
3	5.3	2.1	16.3
4	5.7	1.9	17.3
5	4.7	4.1	16.6
6	4.1	3.0	13.9
7	3.6	3.7	13.3
8	3.9	3.8	14.1
9	3.8	2.5	12.7
10	3.7	2.4	12.3
11	3.5	3.3	12.6
12	3.9	2.5	12.9
13	3.6	3.0	12.6
14	3.5	2.9	12.3
15	3.7	2.9	12.8
16	**3.4**	**3.0**	**12.1**
17	3.5	2.7	12.0
18	3.6	2.7	12.3
19	3.9	2.4	12.8

4.3 VRPTWSyn-Het_CH Efficiency Tests

Once the VRPTWSyn-Het_CH was tuned, it was executed with the aim of solving 36 proposed instances. As the heuristic has two randomized steps, each instance was

solved 10 times, with the aim of checking the method's robustness. Also, to grasp the difficulty level of these instances, Gurobi was run with a time limitation of 24 h and a solution count limit of one, enabling the measurement of how long it took to find a single feasible solution.

Table 5 shows the main results obtained by the constructive heuristic (denoted as CH), in terms of solution quality and execution time, along with Gurobi's results. For each instance, bold results within the Gurobi column indicate that the optimization package outperformed the heuristic; whereas within CH's column indicate otherwise.

These results indicate that the heuristic produces feasible solutions much faster than Gurobi for instances with 20 to 30 tasks, and the methods performed similarly on small instances of 10 tasks. Also, the VRPTHSyn-Het_HC successfully solved all instances, whereas there were two cases for which Gurobi could not provide a solution within 24 h, one of which was solved by the heuristic in less than 8 min and the other in about 9 s (on average). The worst performance obtained by the VRPTHSyn-Het_HC, considering its ten executions on all 36 instances, was 17.5 min. On the other hand, the average solution quality delivered by the non-optimizing method was worse than the first solution provided by Gurobi, as expected. Also, comparing its performance over ten executions, one can conclude that the solution's quality varies significantly from one execution to the other, which is reflected both on the standard deviation of $z(s)$ and on the difference between best and average solutions' costs.

4.4 Capability of Generating a Diverse Set of Feasible Solutions

These final experiments on the VRPTWSyn-Het_CH aimed at checking if the heuristic was able to find a heterogeneous set of feasible solutions. To this end, *nExp* was fixed at 2,000 restarts and the method was not stopped when a feasible solution was found. Instead, all generated solutions were stored, so the results' cost range could be evaluated.

Table 6 shows the number of solutions generated after 2,000 restarts of the procedure and the iteration that yielded the best solution. The minimum and maximum objective function values amongst the produced solutions are presented, as well as the total time spent and the average time per generated solution. Bold figures indicate better performance as compared to Gurobi's first solution.

These results show that the method could generate a heterogeneous solution set, as the total cost of the schedules varied greatly amongst them. The iteration yielding the best solution varied from one instance to the other and was close to the stop criterium in several cases, which could mean the method would deliver even better solutions if *nExp* was higher. Even though the aim of the CH was to get a feasible solution in short time, it created relatively good solutions (as compared to Gurobi's results) faster than Gurobi, at least for instances of 20 and 30 tasks. However, to prove or discard these hypotheses, more experiments are necessary.

Table 5. Results obtained by the heuristic and comparison with Gurobi's first solution

ID	Solution quality (z(s)) [$]				Computational time [s]		
	CH			*Gurobi*	CH		*Gurobi*
	Best $z(s)$	Mean *z(s)*	Std.dev.	$z(s^G)$	Avg.	Std.dev.	*Exec.time*
1	1,018,849	1,153,235	75,692	***397,100***	**0.26**	0.10	*0.29*
2	**634,117**	1,320,615	471,689	*789,297*	**0.17**	0.06	*0.45*
3	**293,853**	1,383,961	936,821	*1,071,380*	0.20	0.09	***0.16***
4	**941,325**	**1,589,366**	432,722	*2,273,560*	0.28	0.14	***0.16***
5	**1,655,450**	**1,965,072**	279,732	*2,506,470*	0.52	0.29	***0.16***
6	**835,080**	**2,167,923**	717,704	*2,604,400*	0.27	0.15	***0.12***
7	**2,894,217**	4,004,370	942,723	*3,381,290*	**0.13**	0.04	*2.49*
8	**2,262,165**	3,915,317	1,017,668	*2,608,060*	**0.25**	0.13	*3.03*
9	**1,247,058**	**2,496,697**	952,264	*3,366,270*	**0.16**	0.06	*1.49*
10	**1,595,904**	5,356,688	2,504,757	*2,091,850*	**0.15**	0.05	*0.79*
11	**1,067,153**	**3,995,589**	1,441,217	*5,615,650*	0.28	0.10	***0.23***
12	**3,935,391**	5,161,747	874,394	*4,685,390*	0.27	0.11	***0.16***
13	3,735,207	5,303,278	1,028,693	***1,953,910***	**0.84**	0.44	*159.28*
14	**3,887,022**	4,874,806	467,879	*4,199,490*	**2.47**	1.06	*422.24*
15	**4,295,783**	5,142,315	764,492	*5,003,350*	**10.71**	3.79	*143.68*
16	**3,464,147**	4,736,351	755,393	*4,319,430*	**0.85**	0.39	*1.75*
17	**2,268,634**	**3,582,896**	772,067	*5,484,430*	**1.05**	0.48	*7.66*
18	**2,515,237**	3,581,634	711,066	*3,467,740*	**5.02**	3.18	*115.11*
19	8,074,180	8,964,541	767,770	***6,403,480***	**0.96**	0.59	*74,306.70*
20	**6,599,834**	**7,446,873**	476,774	**–**	**440.16**	257.03	*86,400.00*
21	**914,391**	**2,001,079**	604,243	*3,257,070*	**4.90**	2.30	*517.16*
22	6,581,820	8,349,962	1,311,610	***6,508,570***	**1.46**	0.84	*648.08*
23	**5,541,485**	**6,984,207**	983,566	**–**	**9.03**	3.26	*86,400.00*
24	3,723,538	5,262,822	1,048,945	***2,468,210***	**1.20**	0.60	*330.05*
25	**3,682,847**	4,882,373	850,877	*4,845,650*	442.52	302.44	***211.34***
26	**5,704,137**	6,740,694	860,348	*6,316,010*	**337.23**	191.34	*633.43*
27	**3,397,119**	6,426,697	1,536,478	*6,217,920*	**5.59**	2.67	*1,812.00*
28	**6,021,856**	**7,285,937**	719,566	*8,996,560*	**4.20**	2.54	*1,579.26*
29	**2,927,804**	4,883,168	808,302	*3,112,020*	**3.09**	1.52	*2,652.72*
30	5,760,690	7,205,681	901,976	***5,273,030***	**2.89**	1.22	*232.31*
31	8,687,666	10,597,010	1,405,808	***7,780,720***	**0.62**	0.26	*311.05*
32	**6,247,951**	9,029,322	1,724,166	*6,969,420*	**1.58**	0.79	*3,804.42*
33	**5,726,122**	8,349,493	1,176,210	*6,029,180*	**2.14**	1.40	*3,183.62*
34	4,266,441	8,318,461	1,611,908	***3,174,490***	**0.73**	0.32	*680.31*
35	**8,283,730**	11,103,458	1,518,805	*8,631,800*	**2.40**	1.32	*1,032.58*
36	**9,498,005**	12,633,478	1,929,291	*10,591,500*	**2.11**	1.08	*145.13*

Table 6. Diversity analysis of the solutions produced by the VRPTWSyn-Het_CH

ID	Solution count	Iteration of best solution	Min. z(s)	Max. z(s)	Total time [s]	Time per solution [s]
1	1,326	1,456	**227,380**	2,968,402	44.98	0.03
2	1,468	53	**468,712**	4,098,323	38.78	0.03
3	1,513	1,832	**238,643**	4,365,893	44.31	0.03
4	1,534	636	**435,742**	2,311,310	39.19	0.03
5	232	1,457	**1,129,223**	2,591,399	39.32	0.17
6	1,843	151	**211,671**	4,036,078	60.64	0.03
7	1,632	862	**1,317,689**	7,345,716	38.25	0.02
8	1,683	1,573	**1,252,225**	8,278,343	52.30	0.03
9	1,926	817	**324,914**	9,820,846	52.39	0.03
10	1,935	301	**815,982**	10,198,051	38.46	0.02
11	1,117	543	**1,065,496**	8,650,128	59.93	0.05
12	1,876	1,612	**1,748,199**	9,891,738	77.95	0.04
13	1,429	665	2,986,270	8,297,097	289.81	0.20
14	264	98	**3,122,831**	6,752,361	**136.99**	0.52
15	82	1,989	**3,067,867**	7,311,398	177.27	2.16
16	1,901	1,041	**2,627,635**	6,636,653	442.46	0.23
17	1,943	429	**2,130,372**	7,981,841	646.17	0.33
18	308	1,150	**2,337,985**	5,849,192	372.65	1.21
19	485	740	**6,066,304**	13,588,533	**89.68**	0.18
20	1	1,494	6,921,631	6,921,631	**325.55**	325.55
21	2,000	933	**684,117**	5,913,543	2,910.14	1.46
22	401	46	**5,770,438**	11,792,608	**131.70**	0.33
23	147	507	4,746,371	9,277,908	**224.87**	1.53
24	1,930	1,704	2,948,047	9,883,758	378.72	0.20
25	4	464	5,044,479	6,764,789	**155.19**	38.80
26	2	658	7,246,379	7,758,663	**69.49**	34.74
27	221	755	**3,688,258**	11,625,163	**380.06**	1.72
28	151	858	**5,889,585**	10,606,733	**275.24**	1.82
29	486	237	**2,683,374**	7,156,007	**346.72**	0.71
30	745	628	**4,137,186**	10,754,740	476.56	0.64
31	1,854	660	**5,716,261**	16,678,259	**309.06**	0.17
32	1,121	1,123	**4,663,924**	15,986,095	**555.77**	0.50
33	1,466	1,382	**3,611,944**	13,158,217	**965.32**	0.66
34	1,974	709	4,433,396	15,840,437	**341.10**	0.17
35	1,637	1,389	**6,615,046**	17,090,015	**818.78**	0.50
36	1,811	1,290	**7,086,348**	19,640,345	1,330.34	0.73

5 Conclusions

This paper discusses a variation of the VRPTWSyn, named VRPTWSyn-Het, in which the available fleet is heterogeneous and there are several modes for fulfilling a task's demand. A constructive heuristic is presented and tested on three groups of instances proposed for the problem, with the aim of generating a set of initial solutions for future application of a meta-heuristic. Computational experiments using Gurobi illustrate the complexity of the VRPTWSyn-Het, as the optimization package was not able to find a feasible solution within reasonable time. On the other hand, the proposed solution method could rapidly generate a feasible solution to all instances, and our tests indicate it is able to get a heterogeneous solution set.

For future work, we intend to apply a meta-heuristic-based method for improving the initial solution produced by this greedy algorithm. Moreover, Gurobi or another optimization package should be used to find optimal values of the instances, or at least better lower bounds. Finally, the proposed methodology should be tested on instances of the VRPTWSyn found in the literature.

Acknowledgments. This study was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES)* – Finance Code 001.

References

1. Parragh, S.N., Doerner, K.F.: Solving routing problems with pairwise synchronization constraints. Cent. Eur. J. Oper. Res. **26**, 443–464 (2018). https://doi.org/10.1007/s10100-018-0520-4
2. Bredström, D., Rönnqvist, M.: Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. Eur. J. Oper. Res. **191**, 19–31 (2008). https://doi.org/10.1016/j.ejor.2007.07.033
3. Drexl, M.: Synchronization in vehicle routing–a survey of VRPs with multiple synchronization constraints. Transp. Sci. **46**, 297–316 (2012). https://doi.org/10.1287/trsc.1110.0400
4. Afifi, S., Dang, D.C., Moukrim, A.: Heuristic solutions for the vehicle routing problem with time windows and synchronized visits. Optim. Lett. **10**, 511–525 (2016). https://doi.org/10.1007/s11590-015-0878-3
5. Afifi, S., Dang, D.-C., Moukrim, A.: A simulated annealing algorithm for the vehicle routing problem with time windows and synchronization constraints. In: Nicosia, G., Pardalos, P. (eds.) LION 2013. LNCS, vol. 7997, pp. 259–265. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-44973-4_27
6. Eveborn, P., Flisberg, P., Rönnqvist, M.: Laps Care-an operational system for staff planning of home care. Eur. J. Oper. Res. **171**, 962–976 (2006). https://doi.org/10.1016/j.ejor.2005.01.011
7. Mankowska, D.S., Meisel, F., Bierwirth, C.: The home health care routing and scheduling problem with interdependent services. Health Care Manag. Sci. **17**, 15–30 (2014). https://doi.org/10.1007/s10729-013-9243-1
8. Ait Haddadene, S.R., Labadie, N., Prodhon, C.: A GRASP × ILS for the vehicle routing problem with time windows, synchronization and precedence constraints. Expert Syst. Appl. **66**, 274–294 (2016). https://doi.org/10.1016/j.eswa.2016.09.002

9. Dohn, A., Kolind, E., Clausen, J.: The manpower allocation problem with time windows and job-teaming constraints: a branch-and-price approach. Comput. Oper. Res. **36**, 1145–1157 (2009). https://doi.org/10.1016/j.cor.2007.12.011
10. Hojabri, H., Gendreau, M., Potvin, J.Y., Rousseau, L.M.: Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints. Comput. Oper. Res. **92**, 87–97 (2018). https://doi.org/10.1016/j.cor.2017.11.011
11. Mendes, A.B.: Scheduling offshore support fleet under the requirement of multiple vessels per task. PhD, University of Sao Paulo (2007). (in Portuguese). http://www.teses.usp.br/teses/disponiveis/3/3135/tde-14012008-171216/en.php
12. Shyshou, A., Gribkovskaia, I., Barceló, J.: A simulation study of the fleet sizing problem arising in offshore anchor handling operations. Eur. J. Oper. Res. **203**, 230–240 (2010). https://doi.org/10.1016/j.ejor.2009.07.012
13. Seixas, M.P., et al.: A heuristic approach to stowing general cargo into platform supply vessels. J. Oper. Res. Soc. **67**, 148–158 (2016). https://doi.org/10.1057/jors.2015.62
14. Liu, R., Tao, Y., Xie, X.: An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits. Comput. Oper. Res. **101**, 250–262 (2019). https://doi.org/10.1016/j.cor.2018.08.002
15. Artigues, C., Roubellat, F.: A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. Eur. J. Oper. Res. **127**, 297–316 (2000). https://doi.org/10.1016/S0377-2217(99)00496-8
16. Edis, E.B., Oguz, C., Ozkarahan, I.: Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. Eur. J. Oper. Res. **230**, 449–463 (2013). https://doi.org/10.1016/j.ejor.2013.02.042
17. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. Springer, New York (2012). https://doi.org/10.1007/978-1-4614-2361-4