Noname manuscript No.
(will be inserted by the editor)

A Multi-Objective Metaheuristic with learning capabilities for a Green UAV Grid Routing Problem

Elias L. Marques Jr. · Vitor N. Coelho · Igor M. Coelho · Bruno N. Coelho · Luiz S. Ochi · Nelson Maculan

Received: date / Accepted: date

Abstract This paper deals with Unmanned Aerial Vehicle (UAV) routing in dynamic grid scenarios with limited battery autonomy and multiple charging stations. Inspired by a multi-criteria view of real systems, we consider different objective functions introduced in the literature, while respecting the navigation over forbidden areas and also a real-time flight autonomy. A multi-objective variant of Variable Neighborhood Search is considered for finding sets of non-dominated solutions. Twelve neighborhood structures were developed in order to explore the solution space, including learning techniques. The latter stores known routes in order to speed up the search. A case of study was developed where UAVs have to serve clients spread throughout a grid, representing a map. Each UAV starts in a given grid point with a given battery charge, where the grid is composed by four different kinds of points: a regular one and three special (prohibited, recharge and client). Any update can happen on the routes on real-time, so the metaheuristic should handle real-time information and update the plan and GUI (Graphics User Interface) accordingly. Any sequence of valid adjacent points forms a route, but since this yields a huge number of combinations, a preprocessing technique is proposed to pre-compute

Elias L. Marques Jr. · Igor M. Coelho · Luiz S. Ochi
Institute of Computer Science, Universidade Federal Fluminense, Niterói, Brazil
E-mail: eliaslawrence.jr@gmail.com, imcoelho@ic.uff.br, satoru@ic.uff.br

Vitor N. Coelho
OptBlocks, Av. João Pinheiro, 274 Room 201, 30130-186, Belo Horizonte - MG, Brazil
E-mail: vncoelho@gmail.com

Bruno N. Coelho
REDEMAT, Universidade Federal de São João del Rey, Brazil
E-mail: brunonazario@gmail.com

Nelson Maculan
Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil
E-mail: maculan@cos.ufrj.br

distances in a given dynamic scenario. Computational results demonstrate the performance of different variants of the proposed algorithms.

Keywords Unmanned Aerial Vehicle · Microgrids · Multi-objective Problem · Variable Neighborhood Search · Biased Random Key Genetic Algorithm

1 Introduction

Recent technological advances have pushed towards the development and practical adoption of novel aerial transportation methods, under the topic of Unmanned Aerial Vehicles (UAV). In this sense, the work of Coelho et al. [4] proposes a mathematical programming model for the multi-objective UAV routing problem, seeking to minimize travelled distance while respecting UAV battery constraints.

The problem stems from the Traveling Salesman Problem with Drones (TSPD) [2] and the Vehicle Routing Problem with Drones (VRPD) [23]. Although both problems address the visit of clients by drones used as complementary vehicles to a main one (such as a truck). In addition, these problems are represented by graphs, which differs from the grid representation addressed in this work.

Since the VRPD is an extension of the VRP when working with drones concurrently with ground vehicles on routes; it is also a NP-hard problem. Consequently, the exclusive use of a MILP formulation to obtain optimum results in a reasonable period of time is only possible in small instances. Therefore, to address large-scale instances of VRPD, some studies propose a metaheuristic based on Variable Neighborhood Search (VNS), as in Schermer et al. [21].

The main contribution of this current work is to complement the linear mathematical model of Coelho et al. [4] by developing a metaheuristic algorithm for a time-dependent UAV routing problem. In particular, it respects UAVs operational requirements; tackling the micro-airspace considering a scenario of points inspection, and avoiding prohibited points (docking constraints) [5]. It also integrates UAVs into the new concepts of mini/microgrid systems, in which vehicles can be charged at different points of smart cities; dynamic routes considering in-route drones: initial battery different than 100%, random origin point and a number of clients already visited.

Specially, this work can be seen as an extension of [4], taking into account the many applications of drones, not limited to deliver activities, as infrastructure inspection ([16], [15], [13], [1], [6]), surveillance ([18], [12]), area mapping ([11]) and others. Focusing, then, on realistic assumptions required for fast re-optimization (via heuristics) and taking into account many requirements like restricted area. We delve further into the striking differences between this work and [4] in Section 2.

This paper is organized as follows. Section 2 describes the proposed model of the Multi-Objective Green Routing Drone Grid Problem (MOGRDGP), while Section 3 contains the methodology employed to solve the problem. In Section 4, one can find the computational experiments comparing the different

implementations, instances, variables and results. Finally, Section 5 concludes the work and presents future research directions.

2 Problem Description

The proposed MOGRDGP consists of an airspace divided into horizontal and vertical bands, organized as a grid of points in two-dimensional space. Each UAV can move following the Chebyshev distance, where the distances between any adjacent points are the same. This distance metric was chosen to simplify the calculations, without loss of generality, and without prejudice to the construction of the routes.

Power stations are scattered around the routing area and accessed by the drone to recharge its batteries. To represent prohibited areas, the grid is also composed of prohibited points that the UAV cannot access, otherwise, this invalidates the route. Figure 4 illustrates a solution for a PMORVD instance.

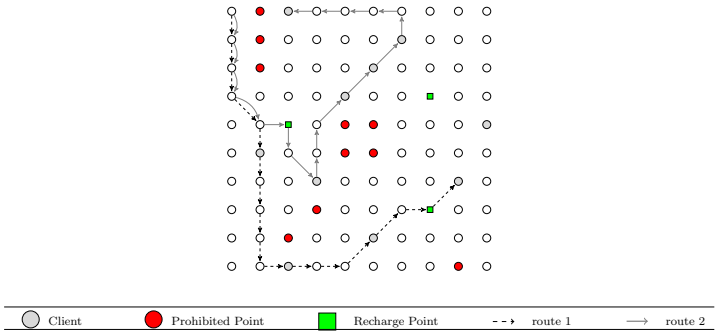

Fig. 1: Instance and solution example to MOGRDGP

As a routing problem, the vehicle must serve/visit clients that are spread across the grid. For this, the point corresponding to the client must be part of the final route. This means that the clients' x and y coordinates must be part of the matrix that represents the solution.

In Section 3, we discuss an alternative to preprocess shorter distances and store them in an auxiliary data structure, so that only the points of origin and destination need to be considered between client and recharge points.

In order to model the problem computationally, we mapped three main objectives that satisfactorily summarize how the real system should behave. It is desirable to end the route with the maximum possible charge rate (*final charge* - O1), ensuring that the drone is prepared for a future route, since the current work has a dynamic nature where the instances already consider the initial location and battery charge. This means that a drone can start a new route at the end of an old one. The total route must be completed in the

shortest possible *time* (O2). Also, it is desirable that during the route, the vehicle consumes (*consumption* - O3) as little battery / fuel as possible.

If we compare the objective functions of this work with the ones explored in [4], it can be related O1 with *toFull*, O2 with *time* and O3 is combination of *time*, *distance* and *maxvel* in a certain way. The number of drones was not seen as an objective function that should be minimized but as a parameter of the instances to test the methods developed.

Since our focus was to elaborate a model so it could fit to different applications involving drones, constraints and objectives exclusively related to delivery problems - such as *makespanC*, *makespanD*, weight of the products, capacity of the UAVs - were not taken into consideration in this work.

The algorithm proposed in this article focuses on finding a balance between solutions, since one element can affect another. The shorter the time, the greater the speed, the greater the consumption. Higher final charge means more time spent on recharging/refueling, which results in longer times.

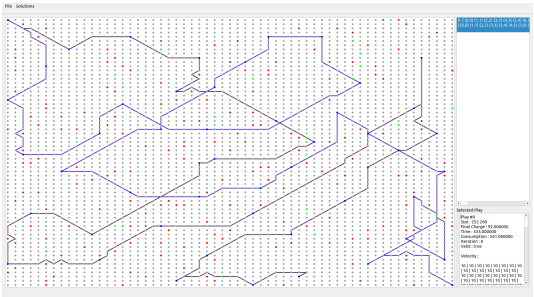

Fig. 2: Developed Graphic Interface

A graphical interface, shown in Fig. 2, was implemented in order to facilitate the visualization of the solutions found. On the right side, we have two panels, the upper one presents a list of solutions, representing the pool generated by the current execution. The solution selected in this panel is demonstrated on the main canvas. The bottom panel presents the main information of the selected solution, such as the value of the objective functions, the speed of the drones in each stretch and the rate of recharge of the drone in each recharge station.

Every valid route must respect some requirements:

Consumption: the fuel/battery level of the vehicle should not reach below zero in any part of the path, that would mean that the UAV would be out of fuel/energy in the middle of the route. However, if it reaches zero and the route is over or the drone reached one energy point, this does not affect the validity of the solution.

Prohibited area: in real life, there are areas where drones are not allowed to access or cross. This situation was represented by special points scattered across the grid. If the route contains these points, the solution is invalid.

This forbidden-point strategy is proposed as a docking constraint mechanism [5], typically used in vehicle routing problems, that disallows certain UAV to visit points in grid. Note that, in this work, we have dealt with this in a homogeneous way (all drones have the same limitations / capabilities).

There are two variables that affect the final result of the objective functions: velocity and charging time.

Velocity: the vehicle speed through the whole route impacts not just the total time of the route, but also the consumption, once higher the velocity (v), greater the consumption. The final fuel/battery level (f) is a result of the initial fuel/battery level (f_0) decreased by the fixed consumption (c_f) and the velocity multiplied by the coefficient of variable consumption (c_v), as shown at Eq. (1).

$$f = f_0 - v \times c_v - c_f \tag{1}$$

Time at energy station: the time spent at the energy station (r) is added to the total time spent at the route. However, if the vehicle spends more time at it, it can accumulate more fuel/energy to its battery. The end fuel/battery level is a result of the fuel/battery level at the beginning of the stretch increased by the quantity of fuel/energy recharged (f_r) as shown at Eq. (2).

$$f = f_0 + f_r \tag{2}$$

The time at the end of the stretch (t) is the result of the begin time (t_0) increased by the quantity of fuel/energy recharged multiplied by the coefficient of time per fuel/energy (t_f), as shown in Eq. (3).

$$t = t_0 + f_r \times t_f \tag{3}$$

3 Methodology

In this work, three metaheuristics were proposed. A Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic with a multi-objective local Variable Neighborhood Descent (VND) search (MOVND), called G-MOVND, a mono-objective variant (G-VND)[1] and a Biased Random Key Genetic Algorithm (BRKGA)[2].

3.1 A* Algorithm

When working with grid routing problems, instead of graph based ones, we must generate the path between each two nodes in order to have a complete solution. For our metaheuristic, we use the A* algorithm.

[1] The G-MOVND and G-VND code can be currently accessed at https://github.com/eliaslawrence/uvrp-vnd.

[2] BRKGA implementation is available at https://github.com/eliaslawrence/uvrp-vnd.

In MOGRDGP, as in other problems where you want to search for minimal routes on a map, $h(n)$ must represent the straight line distance (Euclidean distance) from n to the objective, since this is the shortest possible distance between any two points.

3.2 G-VND

The G-VND metaheuristic can be seen as a multi-start metaheuristic for combinatorial optimization problems, in which each iteration basically consists of two phases: construction and local search.

3.2.1 Construction

The GRASP, proposed by [20], was used to find an initial solution to the problem addressed in this work. In a greedy construction of this problem, we iteratively choose the client closest to the current position.

The construction phase is a greedy procedure with random and adaptive components, which means that instead of always choosing the closest one, we select from a set of k clients closest to the current position. After, we choose a client randomly from this set (restricted candidate list) to be inserted in the initial solution. Speed and recharge rate on all routes are set to maximum.

3.2.2 Local Search

The inter-routes neighborhoods used are *swap (1,1)* and *shift (1,0)*. The set of the intra-route ones is composed of 10 NS (neighborhoods search), which are *exchange*, *remove recharge*, *closest recharge*, *remove repeated*, *speed section increase*, *speed section decrease*, *speed random increase*, *speed random decrease*, *recharge random increase* and *recharge random decrease*.

As *exchange* is the most costly intra-route neighborhood, other versions were implemented, so that it was possible to compare the one that provides better results during computational experiments. The difference between the versions was that, instead of exchanging a client on the route with each other (E1), the exchange only occurs with the k nearest clients (E2), more distant (E3) or random (E4).

Remove Recharge neighborhood is responsible for remove possible recharge points in each subpath (path between clients) and verify if it improves the current solution. After we remove unnecessary recharge points, we try to add other ones that could improve the current solution with the *closest recharge* neighborhood.

The idea of the *remove repeated* neighborhood is to remove repeated clients on the route, trying to reduce the size of the route and, in consequence, reduce the consumption and/or time.

Velocity Increase/Decrease increases/decreases velocity by 1 unit in each entire section of the route that links two points (clients and/or recharge point).

The *velocity/recharge random increase/decrease* increases/decreases speed/recharge rate by 1 unit in each segment of a subpath chosen randomly.

3.2.3 Acceptance Criterion

After generating a neighbor of the current solution, the current route is evaluated and compared to the routes in the pool of solutions. It will be inserted if it is not dominated by any other one present in the pool. If the new route dominates any other, the latter is removed from the set of solutions.

In this case study, we limit the number of non-dominated solutions, thereby, limiting the number of operations (local search). If the set is full, the new route will only be inserted if at least one solution from the current pool is strongly dominated. All strongly dominated solutions are removed from the pool.

A mono-objective variant has also been implemented. The difference consists of the fixed size of the *pool* in a solution and the comparison criteria, where the dominance criteria is not applied, but an objective function, explained in Eq. (4), in which $t(x)$ represents the longest time among the vehicles that constitute the solution; $c(x)$, the consumption of all vehicles and $cf(x)$, the lowest charge among the vehicles at the end of the route.

$$f(x) = \alpha * t(x) + \beta * c(x) + \gamma * cf(x) \tag{4}$$

Since we want to minimize time and consumption and, on the other hand, maximize the value of the final charge, α and β must have a sign opposite to γ. In addition, since the values of $cf(x)$ vary between 0 (minimum charge) and 100 (maximum charge), while $t(x)$ and $c(x)$ reach much higher values (there are no superior constraints to time and consumption), the γ module must be greater than the other coefficients. Empirically, we arrive at the values of 1, 1 and -5 respectively for α, β and γ. The fitness function is then shown in Eq. (5).

$$f(x) = t(x) + c(x) - 5 * cf(x) \tag{5}$$

3.2.4 MOVND

MOVND [7], a multi-purpose variant of VND, with a pool of solutions, performs a loop for each solution and within a loop for each neighborhood. The Algorithm 1 demonstrates how this local search is implemented.

The algorithm receives as input a pool (*localPool*) with the initial solution generated by the constructor, in addition to a array with the neighborhood structures. *LocalPool* is used as a temporary pool and when it is empty, the execution is interrupted and *globalPool* is returned. The insertion method is implemented as described in the previous section which discusses the acceptance criteria for the solutions.

The activity of calculating routes can be really costly when performed many times through the algorithm. Therefore, a pre-processing was implemented to

Algorithm 1: MOVND

Input: $localPool$, $Neighborhood$

```
while localPool ≠ ∅ do
    S ← pop(localPool);
    insert(globalPool, S);
    forall k ∈ Neighborhood do
        S' ← neighbor(S, k);
        if dominates(S, S') then
            insert(localPool, S');
            k ← 0;
        end
    end
end
return globalPool;
```

make comparisons if it brings gains to the results. The method consists of pre-calculating the best routes between important points on the map (clients, prohibited area and recharge points). The routes are saved and then read as part of the problem input.

3.3 BRKGA

To solve the MOGRDGP, another heuristic algorithm proposed to solve the problem in this work, was the BRKGA [10], metaheuristic based on genetic algorithms as can be inferred from its name. The concept of genetic algorithms and random keys was introduced by Bean [3] for sequential problems.

A random key is a random real number in the continuous range [0.1). Solutions to optimization problems can be encoded by random keys. An array of size equal to the size of the individuals is generated and at each position, a random key is generated.

A decoder is a deterministic algorithm that receives an array of random keys as input and returns a solution to the optimization problem. Bean [3] proposed decoders that sort the array of random keys to produce a sequence.

The metaheuristic takes as input population size (n), chromosome (c), elite population (e), mutant population (m), maximum number of generations (G) and ρ, which refers to the factor of choice of the elite individual.

The initial population consists of n vectors with c random keys each. The $rank$ method uses the fitness value of each individual to do the sorting. A loop is performed until the maximum number of generations is reached. With each iteration, the first e individuals are kept in the population, as well as other random m individuals (mutation).

The crossover is the cross between a solution of the elite population (factor ρ) with another solution of the population to generate a child. Parents are chosen at random. $(N-e-m)$ crosses are performed and the children generated are inserted in the remaining positions of the population.

At the end of each iteration, the population is ranked. The first comparison criterion is the number of times the vehicle has a battery charge less than or equal to zero. The second comparison criterion is a fitness function itself, explained in the Eq. 4. Same function used in G-VND, facilitating later comparison of the methods.

3.3.1 Implementation

The algorithm was divided into two stages. In the first one, the metaheuristic observed the problem as a problem of routing in simple graphs (where only the order of visitation matters), so it was not necessary to calculate the path between each node, using only the Euclidean distance for this. The idea of this initial stage is to accelerate the convergence of metaheuristics, since working with grid routing is more costly.

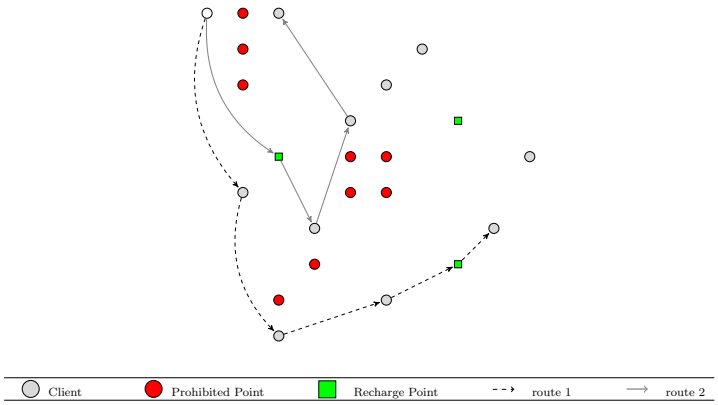

Fig. 3: Example of first stage of BRKGA

In the second stage, we begin to observe the original problem (in a grid). This phase is responsible for refining the solution generated in the previous stage. What changes, in the metaheuristic, in practice, is only the decoding, that is integrated into the A* method to generate a valid solution.

Each individual is represented by three arrays of random keys: visitation order, speed and vehicle recharge rate. The arrays have a dimension equal to the number of clients plus number of recharge stations in the instance. The end of the route is determined by the position of the last client represented in the first array. The speed between two special points (client/client, client/recharge point) A and B is constant and equal to the value represented in the second array, in the same position as point A, in the visitation order. The recharge rate at point C of the route follows the same pattern, where it will be determined by the third array at the same position as point C, in the order of visitation.

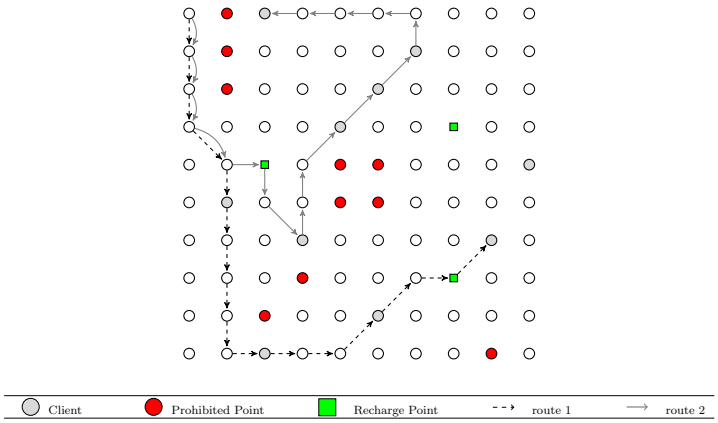

Fig. 4: Example of second stage of BRKGA

The random keys representing the visitation order follow the basic principle of RKGA applied to routing. In this way, encoding and decoding is performed by sorting the keys. After decoding, we get the visited client order.

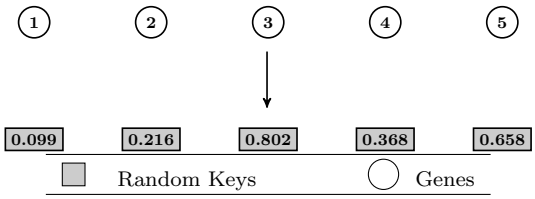

Fig. 5: Coding with random keys

The speed and recharge rate arrays follow the same idea. Decoding works by multiplying the value of the random key by the maximum value of the variable. Thus, at the end of decoding, we have an array of speeds and recharge rates for each segment of the route.

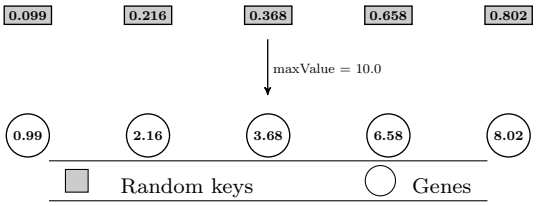

Fig. 6: Decoding of random keys representing speed and recharge rate

4 Computational experiments

Our experiments were performed with a virtual machine with 2 GB of virtual RAM running a 64-bit version of Ubuntu 18.04 on the VirtualBox 5.0.10 hypervisor with Windows 10 as the host operating system. The host hardware configuration consists of an Intel Core i5-6400 CPU with 16 GB of RAM.

To the best of our knowledge, as there is no well-established library of instances for the problem addressed in this work, three well-known TSP instances in 2D Euclidean format were used as the basis for the tests: Christofides/Eilon eil51, eil101 and rat195. These instances have 51, 101 and 195 clients, respectively. From these, arrays were generated representing the area that comprises all clients. After the array is generated, points in it are chosen at random and defined as recharge points and prohibited points.

For both the G-MOVND and the G-VND methods, the different implementations of the intra-route exchange method were taken into account for comparative criteria as described in the section 3.2.2. In the case of experiments, we use the value of $k = 3$. The E2-E3 method means that both the E2 and E3 methods were used as neighborhood structures. The same applies to E2-E4.

Each sample was run 3 times, each one for a standard time according to the size of the instance. The set of non-dominated solutions of all executions was considered for results. To summarize the different implementations, a set of 92 instances were generated, considering different parameters.

Three sizes of instances were generated (51, 101 and 195). For each size, two different formations obeying the percentages of 5% of prohibited points and 1% of recharge points (ex: eil51a and eil51b). For each formation, two different and random drone origin points were chosen (ex: eil51a1 and eil51a2). The number of drones (1 or 2, ex: eil51a1_1d or eil51a1_2d), pre-processing (with or without, ex: eil51a1_pp or eil51a1) were other parameters taken into consideration. As well as, the variable consumption (c_v) where higher the speed, greater the consumption, but shorter the duration of the route. Two values were used in the experiments (0.05 and 0.1, ex: eil51a1_pp_2d_005 or eil51a1_pp_2d_010). The last parameter was the metaheuristic execution timeout:

- Instances of 51 clients: 5s, 10s, 30s, 60s, 120s and 300s (default)
- 101 clients: 10s, 30s, 60s, 120s, 300s and 600s (default)
- 195 clients: 900s (default) and 1800s

4.1 Heuristics Comparisons

In order to compare the results obtained during the experiments, two solution quality indicators were used: hypervolume and coverage.

According to [22], hypervolume is an indicator associated with an approximation given by the volume of the portion of the objective space that is weakly dominated by a set. This indicator needs the specification of a Z reference point that denotes an upper limit on all objectives.

In this problem, the normalized objective function was used in Eq. (6). The hypervolume calculation code is provided by [8] and as default it deals with minimization models, then *minPoints* is represented by the vector [−100, 0, 0], the best values possible for final charge (multiplied by −1), time and consumption respectively. The vector *refPoints* is composed by the worst values in the current pool of solutions.

$$z[o] = \frac{z[o] + minPoints[o]}{refPoints[o] + minPoints[o]} \tag{6}$$

The other measure refers to the number of solutions in the Pareto reference ($PRef$) generated by a specific method. Coverage, represents the number of solutions in the weakly dominated set of pareto divided by the total number of solutions in the set of pareto. The Pareto reference is generated from the solutions of all executions of all methods for each instance. Non-dominated solutions, then, integrate this set.

On Figure 7, it is easy to notice that although G-VND generates the best solutions, it takes time to generate valid solutions. The opposite happens with BRKGA, which can generate valid solutions much faster than other methods, but it takes time to arrive at solutions as good as other methods.

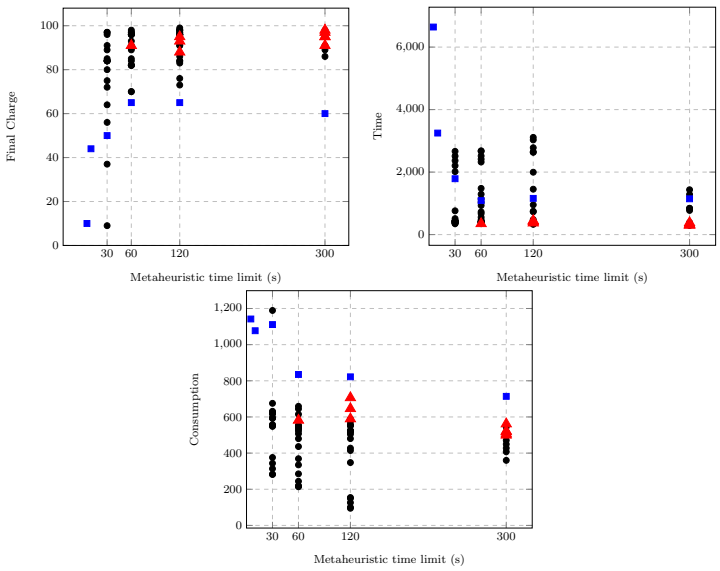

Fig. 7: Variation of objective values for eil51b2 instance due to the duration of the metaheuristic: BRKGA (blue), GMOVND (black) and GVND (red)

With the basic instances, the multiobjective method, as it contains a pool of solutions, manages to generate good solutions that attend each objective

separately, which can be seen in the table, because when it finds valid solutions, it manages to win over the other methods in O1 (final charge), O3 (consumption) and get closer to the results of G-VND in O2 (time).

This mono-objective version, on the other hand, works on just one solution and therefore finds a little more difficulty in the variability of its solution, losing in O1 and O3 in most of the instances and in O2 in the most difficult instances like Table 1. However, precisely because it works with only one solution, it manages to be better (in terms of dominance) than each one, individually, generated by the multiobjective method.

BRKGA, although not outstanding for the quality of its solutions, has merits for generating valid solutions much faster than the other two methods. This statement is validated by the most complex instances, such as those based on rat195 and in instances with c_v equal to 0.1.

In order to compare the efficiency of the different implementations of the swap method, the tables 2 and 3 summarize the hypervolume values in all instances for both the G-VND and the multiobjective version.

It is interesting to note that for the simplest instances, the less costly method S4 (random) generates the best solutions, while for the more complex instances the method that generates the best solutions is E1 (full exchange), which is more costly.

4.2 Two-phase procedure

According to Michie et al. [17], it would be useful if computers could learn from past experiences and thus automatically improve the efficiency of their own programs during execution. Inspired by this concept of machine learning and regarding the results of the computational experiments, we developed techniques where the program could benefit from each method.

The core mechanism of our technique is to first run the BRKGA, then save the routes generated and use them as initial solutions (construction phase) for the other methods (VND and MOVND) to speed-up their optimization. This way, the program would take advantage of the speed in finding valid solutions of the BRKGA and use the other methods to improve the solutions generated. We called these new methods B-VND and B-MOVND. As shown by [19] and [14], this 2-stage approach can be very efficient if the methods start from a population of good quality, in the place of using only one random solution as starting solution. Similar hybrid approaches have been recently explored on machine learning field as *ablation studies*, specially for neural networks [9].

In terms of experiments, instances considered difficult were used to verify if the hybrid technique would provide better results. The Table 4 refer to the instances with 101 clients, 2 drones and c_v equals to 0.1, with guiding function according to Eq. (5). Without this, G-VND and G-MOVND can't find valid solutions on less or equals to 60 seconds.

Table 1: Comparison of the values of the objective functions in the eil51 instances with 2 drones and c_v equals to 0,1 (60s, 30s, 10s and 5s). Lower the time and consumption and higher the final charge, the better. Wins/draws represent the number of best results of each method.

	Final Charge			Time			Consumption		
Instance	BRKGA	G-VND	G-MOVND	BRKGA	G-VND	G-MOVND	BRKGA	G-VND	G-MOVND
eil51a1_pp_2d_010_60	74	**99**	95	2466	339	**338**	526	366	**288**
eil51a2_pp_2d_010_60	73	**92**	**92**	3703	375	**313**	639	590	**201**
eil51b1_pp_2d_010_60	48	97	**99**	3071	344	**319**	549	556	**263**
eil51b2_pp_2d_010_60	66	95	**98**	1087	**349**	370	835	558	**214**
eil51a1_pp_2d_010_30	70	-	**96**	171691	-	**344**	574	-	**245**
eil51a2_pp_2d_010_30	**82**	-	76	3254	-	**312**	869	-	**525**
eil51b1_pp_2d_010_30	67	-	**97**	1925	-	**359**	841	-	**385**
eil51b2_pp_2d_010_30	50	-	**97**	1788	-	**347**	1111	-	**282**
eil51a1_pp_2d_010_10	**83**	-	-	**3694**	-	-	**960**	-	-
eil51a2_pp_2d_010_10	**61**	-	-	**46142**	-	-	**889**	-	-
eil51b1_pp_2d_010_10	**42**	-	-	**3104**	-	-	**1145**	-	-
eil51b2_pp_2d_010_10	**44**	-	-	**3249**	-	-	**1077**	-	-
eil51a1_pp_2d_010_5	**86**	-	-	**5612**	-	-	**1055**	-	-
eil51a2_pp_2d_010_5	**72**	-	-	**12997**	-	-	**1178**	-	-
eil51b1_pp_2d_010_5	**18**	-	-	**6813**	-	-	**1110**	-	-
eil51b2_pp_2d_010_5	**10**	-	-	**6642**	-	-	**1142**	-	-
eil101a1_pp_2d_010_600	**75**	-	-	**4275**	-	-	**1279**	-	-
eil101a2_pp_2d_010_600	**77**	-	-	**3327**	-	-	**13882**	-	-
eil101b1_pp_2d_010_600	**94**	-	-	**2844**	-	-	**1298**	-	-
eil101b2_pp_2d_010_600	**88**	-	-	**3382**	-	-	**1485**	-	-
eil101a1_pp_2d_010_300	**70**	-	-	**3415**	-	-	**1538**	-	-
eil101a2_pp_2d_010_300	**70**	-	-	**5681**	-	-	**1591**	-	-
eil101b1_pp_2d_010_300	**80**	-	-	**2602**	-	-	**1826**	-	-
eil101b2_pp_2d_010_300	**83**	-	-	**5456**	-	-	**1599**	-	-
eil101a1_pp_2d_010_120	**94**	-	-	**5734**	-	-	**1751**	-	-
eil101a2_pp_2d_010_120	**59**	-	-	**9791**	-	-	**1928**	-	-
eil101b1_pp_2d_010_120	**63**	-	-	**5469**	-	-	**2894**	-	-
eil101b2_pp_2d_010_120	**76**	-	-	**3663**	-	-	**2480**	-	-
eil101a1_pp_2d_010_60	**95**	-	-	**6853**	-	-	**1973**	-	-
eil101a2_pp_2d_010_60	**40**	-	-	**16252**	-	-	**1859**	-	-
eil101b1_pp_2d_010_60	**97**	-	-	**11621**	-	-	**2070**	-	-
eil101b2_pp_2d_010_60	**58**	-	-	**6563**	-	-	**2245**	-	-
eil101a1_pp_2d_010_30	**29**	-	-	**12176**	-	-	**2336**	-	-
eil101a2_pp_2d_010_30	**7**	-	-	**19280**	-	-	**2166**	-	-
eil101b1_pp_2d_010_30	**63**	-	-	**9231**	-	-	**2543**	-	-
eil101b2_pp_2d_010_30	**46**	-	-	**7188**	-	-	**2699**	-	-
eil101a1_pp_2d_010_10	**46**	-	-	**23792**	-	-	**2945**	-	-
eil101a2_pp_2d_010_10	**2**	-	-	**76780**	-	-	**2855**	-	-
eil101b1_pp_2d_010_10	**70**	-	-	**231696**	-	-	**2961**	-	-
eil101b2_pp_2d_010_10	**56**	-	-	**11341**	-	-	**3398**	-	-
rat195a1_pp_2d_005_900	**44**	-	-	**73438**	-	-	**11938**	-	-
rat195a2_pp_2d_005_900	-	-	-	-	-	-	-	-	-
rat195b1_pp_2d_005_900	**63**	-	-	**134721**	-	-	**13887**	-	-
rat195b2_pp_2d_005_900	**34**			**108718**			**14632**		
Wins/Draws	**36**	2	6	**35**	1	7	**35**	0	8

Table 2: Comparison of the hypervolume values in the standard instances.

	G-VND					G-MOVND				
Instância	S1	S2	S2-S3	S2-S4	S4	S1	S2	S2-S3	S2-S4	S4
eil51a1_pp_1d_005_300	0.020795	2e-06	0.030861	**0.023091**	0.016475	0.134317	0.061298	0.091998	0.15867	**0.211702**
eil51a2_pp_1d_005_300	0.010786	0.000325	0.005431	**0.057932**	0.013954	0.149194	0.020133	0.101524	0.077179	**0.18835**
eil51b1_pp_1d_005_300	0.025811	0.004155	0.019824	0.010168	**0.06474**	0.108522	0.049915	0.107278	0.167067	**0.220386**
eil51b2_pp_1d_005_300	0.025261	0.001832	0.007746	**0.078744**	0.058592	0.239336	0.133163	0.115436	0.169022	**0.29454**
eil101a1_pp_1d_005_600	0	0.154562	0.128263	**0.174743**	0.159414	0.079605	**0.441809**	0.172888	0.255015	0.2897
eil101a2_pp_1d_005_600	1e-06	0.057373	0.009536	0.081314	**0.105031**	0.03035	0.20937	0.136948	0.332207	**0.372734**
eil101b1_pp_1d_005_600	0.004005	0.059317	0.025549	0.175163	**0.196375**	0.002264	0.190141	0.209588	**0.388912**	0.380655
eil101b2_pp_1d_005_600	8e-06	0.132503	0.20232	0.190885	**0.239937**	0.002315	0.350836	0.298897	0.403424	**0.40505**
Wins/Draws	0	0	0	**4**	**4**	0	1	0	1	**6**

Table 3: Comparison of the hypervolume values in the instances eil51 with 2 drones and c_v equals to 0,1 (300s, 120s, 60s, 30s, 10s and 5s).

	G-VND					G-MOVND				
Instância	S1	S2	S2-S3	S2-S4	S4	S1	S2	S2-S3	S2-S4	S4
eil51a1_pp_2d_010_300	**0.069485**	0.021716	0.015454	0.062854	0.000548	0.131346	**0.281926**	0.149923	0.199639	0.110301
eil51a2_pp_2d_010_300	**0.099269**	0.06019	0.079028	0.030441	0.000709	0.025053	0.124247	0.132469	**0.245771**	0.213089
eil51b1_pp_2d_010_300	0.065252	**0.081112**	0.039958	0.020731	0.044003	0.268067	0.290394	0.223338	0.284001	**0.299283**
eil51b2_pp_2d_010_300	**0.06844**	0.065438	0.06374	0.054796	0.026421	**0.253256**	0.095927	0.158712	0.154838	0.230625
eil51a1_pp_2d_010_120	**0.057691**	0.039874	0.028219	0.004429	0.011841	0.249306	0.097837	**0.387598**	0.304051	0.257775
eil51a2_pp_2d_010_120	0	0.003552	0.000195	**0.014564**	0.020997	**0.502308**	0.330587	0.161815	0.50082	0.400457
eil51b1_pp_2d_010_120	0.022257	0.007968	0.010326	**0.038498**	1e-06	**0.255684**	0.24811	0.18098	0.184642	0.241851
eil51b2_pp_2d_010_120	**0.069421**	0.015079	0.020671	0.004019	1e-06	**0.488216**	0.231857	0.137772	0.37673	0.244011
eil51a1_pp_2d_010_60	0.200294	**0.24779**	0.122142	0.150754	0.136076	0.141028	**0.213526**	0.162016	0.15151	0.167547
eil51a2_pp_2d_010_60	**0.234699**	0.170401	0.105089	0.003862	0	**0.44086**	0.431543	0.37044	0.128351	0.355009
eil51b1_pp_2d_010_60	0.005943	0.052966	0.051464	**0.059508**	0.001448	**0.313255**	0.111483	0.17727	0.038938	0.15267
eil51b2_pp_2d_010_60	0.083241	**0.122473**	2e-06	0.118052	0.000142	0.020816	0.054042	0.144838	0.159713	**0.311948**
eil51a1_pp_2d_010_30	-	-	-	-	-	0	**0.092966**	0.028953	0	0.073083
eil51a2_pp_2d_010_30	-	-	-	-	-	**0.049724**	0.027657	0.023998	0.018424	0.0079
eil51b1_pp_2d_010_30	-	-	-	-	-	0	0.03002	**0.061249**	0.036549	0.000435
eil51b2_pp_2d_010_30	-	-	-	-	-	0.37616	0.365801	0.369478	**0.398487**	0.159026
Vitórias/Empates	**6**	3	0	3	0	**7**	3	2	2	2

The values of the objective functions $(O1, O2, O3)$ generated by the BRKGA at the first stage were, respectively, $(74, 13474, 2765)$ for $a1$ and $(15, 14838, 29118)$ for $b1$. It couldn't find valid solutions for the instances *a2* and *b2*.

Table 4: Comparing the simple metaheuristics (BRKGA, G-VND and G-MOVND) with the 2-stage ones (B-VND and B-MOVND). Number indicates the objective values achieved by the best solution according to guiding function

		Instances ($eil101 - ... - pp\_2d\_010 - ...$)					
Objectives	Methods	$a1\_60$	$b1\_60$	$a1\_30$	$b1\_30$	$a1\_10$	$b1\_10$
Final Charge	BRKGA	95	97	29	63	46	70
	VND	-	-	-	-	-	-
	MOVND	-	-	-	-	-	-
	B-VND	92	89	92	96	74	94
	B-MOVND	79	62	75	53	74	53
Time	BRKGA	6853	11621	12176	9231	23792	231696
	VND	-	-	-	-	-	-
	MOVND	-	-	-	-	-	-
	B-VND	11463	13877	11463	14801	11463	14801
	B-MOVND	11463	13384	11463	14801	11463	14801
Consump.	BRKGA	1973	2070	2336	2543	2945	2961
	VND	-	-	-	-	-	-
	MOVND	-	-	-	-	-	-
	B-VND	2873	2960	2981	2928	2948	2956
	B-MOVND	2613	2669	2677	2719	2682	2755

We could verify that even with smaller timeouts, VND and MOVND were able to improve the solutions generated by the BRKGA. On the other hand, when the solutions generated at the first stage are not valid (due to timeout

constraints), they also can't generate valid solutions from invalid ones. For this reason, B-VND and B-MOVND couldn't find solutions for both instance packs *a2* and *b2*.

5 Conclusions

In this work, we approach the MOGRDGP, considering a novel and rich range of constraints, in addition to using a model with several objective functions. The multi-objective, grid, restrictions of prohibited areas (docking constraint), the concern with consumption (Green Computing) and the dynamism of this problem shows a practical approach for real applications. The instances considered arbitrary drone initial positions and also variable initial battery charge, so it is possible to integrate this tool in an online solver that solves a series of instances considering changes in the dynamic characteristics of the scenario (due to wind conditions, logistics and other operational restrictions).

To solve the problem addressed, we propose three algorithms (BRKGA, G-VND and G-MOVND) and through the computational experiments carried out, we can conclude that for PMORVDG, BRKGA, despite generating viable solutions very quickly, their generated solutions lose in quality compared to the other methods. In the comparison between G-VND (mono-objective) and G-MOVND (multi-objective), we can see that G-VND generates the best solutions but it takes time to find valid solutions. In this way, G-MOVND, transits between the other two methods in terms of advantages and disadvantages, being faster than the mono-objective method to find valid solutions and generating better solutions than BRKGA.

We visualize future work taking into account more vertical layers in the grid, heterogeneous drones and prohibited temporary points. Next steps for this work could also require a study of exact methods for this problem , consisting of mathematical formulations and hybrid methods combining metaheuristics with exact methods (such as columns generation) for the case of multiple drones. Problem solving methods based on neural networks are also in our plans with sensors and IoT equipment for guiding them on real-time operation.

References

1. Adabo, G.J.: Long range unmanned aircraft system for power line inspection of brazilian electrical system. Journal of Energy and Power Engineering **8**(2) (2014)
2. Agatz, N., Bouman, P., Schmidt, M.: Optimization approaches for the traveling salesman problem with drone. Transportation Science **52**(4), 965–981 (2018)
3. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA journal on computing **6**(2), 154–160 (1994)
4. Coelho, B.N., Coelho, V.N., Coelho, I.M., Ochi, L.S., Zuidema, D., Lima, M.S., da Costa, A.R., et al.: A multi-objective green uav routing problem. Computers & Operations Research **88**, 306–315 (2017)

5. Coelho, V.N., Grasas, A., Ramalhinho, H., Coelho, I.M., Souza, M.J., Cruz, R.C.: An ils-based algorithm to solve a large-scale real heterogeneous fleet vrp with multi-trips and docking constraints. European Journal of Operational Research **250**(2), 367–376 (2016)
6. Deng, C., Wang, S., Huang, Z., Tan, Z., Liu, J.: Unmanned aerial vehicles for power line inspection: A cooperative way in platforms and communications. J. Commun **9**(9), 687–692 (2014)
7. Duarte, A., Pantrigo, J.J., Pardo, E.G., Mladenovic, N.: Multi-objective variable neighborhood search: an application to combinatorial optimization problems. Journal of Global Optimization **63**(3), 515–536 (2015). DOI 10.1007/s10898-014-0213-z
8. Fonseca, C.M., Paquete, L., López-Ibánez, M.: An improved dimension-sweep algorithm for the hypervolume indicator. In: 2006 IEEE international conference on evolutionary computation, pp. 1157–1163. IEEE (2006)
9. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 580–587 (2014)
10. Gonçalves, J.F., Resende, M.G.: Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics **17**(5), 487–525 (2011)
11. Haala, N., Cramer, M., Weimer, F., Trittler, M.: Performance test on uav-based photogrammetric data collection. Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences **38**(1/C22), 7–12 (2011)
12. Harris, A., Sluss, J.J., Refai, H.H., LoPresti, P.G.: Alignment and tracking of a free-space optical communications link to a uav. In: Digital Avionics Systems Conference, 2005. DASC 2005. The 24th, vol. 1, pp. 1–C. IEEE (2005)
13. Irizarry, J., Gheisari, M., Walker, B.N.: Usability assessment of drone technology as safety inspection tools. Journal of Information Technology in Construction (ITcon) **17**(12), 194–212 (2012)
14. Lust, T., Teghem, J.: Two-phase pareto local search for the biobjective traveling salesman problem. Journal of Heuristics **16**(3), 475–510 (2010)
15. Máthé, K., Buşoniu, L.: Vision and control for uavs: A survey of general methods and of inexpensive platforms for infrastructure inspection. Sensors **15**(7), 14887–14916 (2015)
16. Metni, N., Hamel, T.: A uav for bridge inspection: Visual servoing control law with orientation limits. Automation in construction **17**(1), 3–10 (2007)
17. Michie, D., Spiegelhalter, D.J., Taylor, C., et al.: Machine learning. Neural and Statistical Classification **13**(1994), 1–298 (1994)
18. Nigam, N., Kroo, I.: Persistent surveillance using multiple unmanned air vehicles. In: Aerospace Conference, 2008 IEEE, pp. 1–14. IEEE (2008)
19. Paquete, L., Chiarandini, M., Stützle, T.: Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In: Metaheuristics for multiobjective optimisation, pp. 177–199. Springer (2004)
20. Resende, M.G., Ribeiro, C.C.: Grasp: Greedy randomized adaptive search procedures. In: Search methodologies, pp. 287–312. Springer (2014)
21. Schermer, D., Moeini, M., Wendt, O.: A variable neighborhood search algorithm for solving the vehicle routing problem with drones. Tech. rep., Technical Report Technische Universität Kaiserslautern (2018)
22. Talbi, E.G.: Metaheuristics: from design to implementation, vol. 74. John Wiley & Sons (2009)
23. Wang, X., Poikonen, S., Golden, B.: The vehicle routing problem with drones: several worst-case results. Optimization Letters **11**(4), 679–697 (2017)