






# A Q-Learning Based Hyper-Heuristic for Generating Efficient UAV Swarming Behaviours

Gabriel Duflo<sup>1</sup>(✉), Grégoire Danoy<sup>1,2</sup>, El-Ghazali Talbi<sup>3</sup>,  
and Pascal Bouvry<sup>1,2</sup>

<sup>1</sup> SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg  
{gabriel.duflo,gregoire.danoy,pascal.bouvry}@uni.lu

<sup>2</sup> FSTM/DCS, University of Luxembourg, Esch-sur-Alzette, Luxembourg

<sup>3</sup> University of Lille, CNRS/CRISTAL, Inria Lille, Lille, France  
el-ghazali.talbi@univ-lille.fr

**Abstract.** The usage of Unmanned Aerial Vehicles (UAVs) is gradually gaining momentum for commercial applications. These however often rely on a single UAV, which comes with constraints such as its range of capacity or the number of sensors it can carry. Using several UAVs as a swarm makes it possible to overcome these limitations. Many meta-heuristics have been designed to optimise the behaviour of UAV swarms. Manually designing such algorithms can however be very time-consuming and error prone since swarming relies on an emergent behaviour which can be hard to predict from local interactions. As a solution, this work proposes to automate the design of UAV swarming behaviours thanks to a Q-learning based hyper heuristic. Experimental results demonstrate that it is possible to obtain efficient swarming heuristics independently of the problem size, thus allowing a fast training on small instances.

**Keywords:** Hyper-heuristic · UAV swarming · Reinforcement learning

## 1 Introduction

In the past years the Unmanned Aerial Vehicles (UAVs) have found their way into an increasing number of civilian applications, such as delivery, infrastructure inspection or urban traffic monitoring. However, most of these rely on a single UAV, either remotely operated or autonomous, which comes with limitations due to the battery and payload capacity. A promising way to overcome these limitations while relying on current technology is to use multiple UAVs simultaneously as a swarm.

This work specifically considers the problem of area coverage by a swarm of UAVs. The latter finds applications in surveillance, search and rescue and smart agriculture to name a few. More precisely we tackle the *Coverage of a Connected-UAV Swarm* (CCUS) problem [8,9], which models the surveillance

problem as a bi-objective one, where both the area coverage and the swarm network connectivity [4] have to be maximised.

Many metaheuristics have been manually designed to address the problem of area coverage by a swarm of UAVs, for instance using ant colony methods [1]. This process can however be error-prone and time-consuming since it consists in predicting an emergent behaviour by designing local interaction.

Instead, this work proposes to automate the design of swarming behaviours, which remains an open research problem [3]. This principle of searching into a space of heuristics is also referred to as a hyper-heuristic. We here consider a Q-learning based hyper-heuristic (QLHH) to generate efficient distributed heuristic for the CCUS problem.

The remainder of this article is organised as follows. Section 2 first presents the related work on hyper-heuristics and their usage for swarming applications. Then, in Sect. 3 the proposed CCUS model is presented, followed by the designed QLHH in Sect. 4. The empirical performance of QLHH is then provided in Sect. 5 using coverage and connectivity metrics from the literature. Finally Sect. 6 provides our conclusions and perspectives.

## 2 Related Work

Hyper-heuristics were first mentioned as “heuristics to choose heuristics” [7]. More generally, they refer to high-level algorithms performing a searching process in a space of low-level heuristics. The purpose is thus to search a heuristic for a given problem, unlike metaheuristics which aim at searching a solution for a given instance.

Burke et al. made a classification of hyper-heuristics [5]. The latter is divided in 2 parts: the nature of the search space, and the nature of the feedback for the learning process. At the lowest level, the heuristics from the search space can be constructive or perturbative. A constructive heuristic modifies a non-feasible solution until it becomes feasible. At a higher level, two approaches can be used by the algorithm performing the searching process: selective and generative approaches. For the generative approach, the high-level algorithm is given a set of predefined “building-blocks” containing information relative to the problem. Its goal is thus to combine these blocks in order to make a new heuristic. The idea is to extract a dynamic part common to several known heuristics and to learn it. Finally, the last distinction is whether the feedback uses online or offline learning. For an online learning feedback, the learning process occurs while the model is executed on instances. This work proposes a hyper-heuristic for generating constructive heuristics using an online learning feedback.

Most generative hyper-heuristics use offline-learning, and more specifically genetic programming (GP). For instance Lin et al. evolve a task sequence for the multi-skill resource constrained project scheduling problem (MS-RCPSP) [13]. In [10], Dufflo et al. evaluate the unvisited nodes at each iteration in the travelling salesman problem (TSP). Finally, Kieffer et al. sort the bundles in the bi-level cloud pricing optimisation problem (BCPOP) in [12].

When it comes to online learning, reinforcement learning (RL) is the most widely used technique for hyper-heuristics. It is however mainly used for selective approaches, while few generative ones have considered RL. For instance, Khalil et al. design in [11] a template of RL-based hyper-heuristic for several combinatorial optimisation problems.

Hyper-heuristics have also been used in the context of swarming. However, since the automation of swarming behaviours is a recent and still open research area, as mentioned in [3], the selective approach is prominently used in the literature. For instance Babic et al. design in [2] a hyper-heuristic where robots can choose at each iteration the best heuristic among a pool of predefined low-level heuristics to perform a certain number of predefined actions. To date, only Duflo et al. use a generative approach based on RL for defining the movement of UAVs in a swarm [8].

In the context of a swarm of UAV/robot, multiple agents are interacting. RL has then been applied on more general Multi-Agent Systems (MAS). Tuyls and Stone [14] provide a classification of paradigms, including “Online RL towards social welfare”. In this paradigm, every agent of the MAS share the same policy, which is the context of this work.

Finally the CCUS problem is a bi-objective one, which implies that the generated heuristics must tackle bi-objective problems. However, for online approaches, Multi-Objective Reinforcement Learning (MORL) remains an open area [6]. One popular approach is to transform a multiple policy RL into a single one using a scalarisation function [15].

The work proposed in this article thus goes beyond the state-of-the-art as it proposes a generative hyper-heuristic, and more precisely a Q-learning hyper-heuristic (QLHH) to generate efficient UAV swarming behaviours. This QLHH is applied to tackle a problem introduced hereinafter for optimising the Coverage of a Connected UAV Swarm, so called CCUS. Since CCUS is a multi-objective optimisation problem (as described in the following section), one of the challenges with this work is to apply MORL techniques to the context of hyper-heuristics.

### 3 CCUS Model

This section provides a formal definition of the CCUS model. It considers the usage of a swarm of UAVs equipped with wireless communication interfaces, also referred to as flying ad hoc network (FANET), for covering an area. In that scenario, each UAV starts from a given point, executes its tour and returns to its starting point, which can be referred to as its base. CCUS aims at optimising both the coverage speed and the connectivity of the swarm.

#### 3.1 Formal Expression

The CCUS model contains two main components: an environment graph  $G_e = (V, E_e)$  and a communication graph  $G_c = (U, E_c)$ . Both are represented in Fig. 1.  $G_e$  is composed of a set  $V$  of vertices on which the UAVs can move. This graph

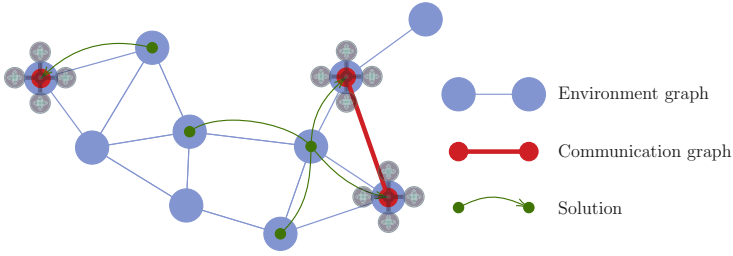


Fig. 1. CCUS solution at a certain time step

is weighted, non-directed and must be strongly connected so that every UAV can move to any other vertex. Otherwise, another constraint stating that there should be at least one UAV per connected component should be required. The weight corresponds here to the distance between two adjacent vertices. The function  $dist : V^2 \rightarrow \mathbb{R}$  then returns the length of the shortest path between any two given nodes.  $G_c$  is composed of a set  $U$  of UAVs, which are linked if they are close enough for communicating (in a fixed communication range  $D_{com}$  which is specific to the FANET setup). This graph is not necessarily connected and especially dynamic since positions of UAVs evolve during the coverage. The function  $pos : U \rightarrow V$  returns the current position of a given UAV.

An instance  $I = (G_e, G_c)$  is then defined by an environment graph and a communication graph giving the initial positions of UAVs. Different initial positions would make a different instance. Finally, a solution for CCUS is a set of  $|U|$  paths in  $G_e$  (one path per UAV). Each path must start from the initial vertex of the corresponding UAV (i.e., its base). Such a solution is feasible if and only if these paths are cycles and their union covers the  $G_e$ , so that the whole environment is covered and the UAVs have returned to their base.

### 3.2 Representation of Solutions

A solution  $S$  can be defined as a set of paths  $\{S_u\}_{u \in U}$ , where  $S_u = (v_1, v_2, \dots, v_{|S_u|})$  with  $v_i \in V$ .  $\bar{S}$  then refers to the unvisited vertices from  $G_e$ .

$$\bar{S} = \{v \in V | \nexists u \in U, v \in S_u\}$$

If the coverage is paused at any moment, a solution will be obtained (not necessarily a feasible solution). Since CCUS is bi-objective, any solution  $S$  can be represented as a 2-dimensional vector  $O(S) = (O^{cov}(S), O^{con}(S))^T$ , containing both objective values defined earlier. The two components  $O^{cov}(S)$  and  $O^{con}(S)$  respectively correspond to the biggest path at the current time step (see Sect. 3.3) and the average number of connected components in the communication graph until the current time step (see Sect. 3.4).

### 3.3 Coverage Objective

In the CCUS model, UAVs are considered to fly at a constant speed. The speed is hence equivalent to the distance. Minimising the coverage speed then corresponds to minimising the longest path (cycle) of the (feasible) solution. This also prevents some UAVs to do short tours leaving more cells to cover for the others. This means that every cycle will have approximately the same length.

Let  $L_u$  be the length of the cycle of the UAV  $u$ , then the coverage objective for CCUS is defined as:

$$\text{Minimise } \left\{ \frac{|V|}{|V| - |\bar{S}|} \cdot \max_{u \in U} L_u \right\}$$

This objective value is slightly different from the one presented in [9]. The factor  $|V|/(|V| - |\bar{S}|)$  has indeed been added. Since  $(|V| - |\bar{S}|)/|V|$  corresponds to the rate of visited vertices, the length of the longest path is divided by this rate in order to penalise a non-feasible solution with vertices visited several times. This factor does not affect a feasible solution since it equals 1 in that case.

### 3.4 Connectivity Objective

Two UAVs can communicate if they are in a certain communication range  $D_{com}$ . They can exchange their local information. It is therefore possible for a UAV to access the local information of every UAV in its connected component in  $G_c$ . Maximising the connectivity then consists in minimising the global number of connected components in  $G_c$  over time.

Let  $C_t$  be the number of connected components in  $G_c$  at the time step  $t$ , then the connectivity objective for CCUS is defined as:

$$\text{Minimise } \left\{ \sum_{t \in T} C_t \right\}$$

## 4 QLHH Algorithm

This section describes in detail QLHH, a Q-learning based algorithm for generating heuristics for the CCUS problem.

### 4.1 Structure

As a low-level heuristic, each UAV will asynchronously move to a new vertex until the whole environment graph is covered. Algorithm 1 gives an overview of the process. The choice of a new vertex is done thanks to a fitness function  $f$  which evaluates each possible destination. The UAVs then move to the vertex maximising  $f$  (line 4). The fitness function  $f$  is specific to the heuristic. Algorithm 1 then provides a template of low-level heuristics, where  $f$  is the dynamic part. The goal of the high-level algorithm, here Q-learning, is thus to find the

best possible definition for  $f$ . Such a low-level heuristic runs in a quadratic time. Each UAV visits  $\mathcal{O}(|V|)$  vertices during its tour. At each step, every unvisited vertex is evaluated, so there are  $\mathcal{O}(|V|)$  operations. This gives a final time complexity of  $\mathcal{O}(|V|^2)$  per UAV. Since the heuristic is distributed, the number of UAVs is not added to the complexity.

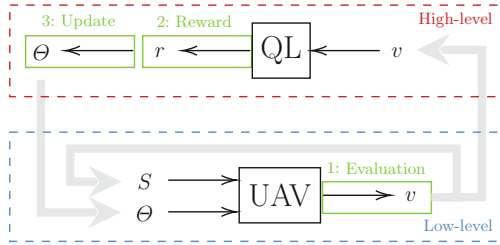


Fig. 2. Overview of the proposed QLHH algorithm

---

**Algorithm 1:** Low-level-heuristic template

---

```

input   : Instance  $I = (G_e, G_c)$ 
output : Solution  $S = \bigcup_{u \in U} S_u$ 

1 foreach UAV  $u \in U$  do                                     // asynchronously
2    $S_u \leftarrow (pos(u))$ 
3   while  $\bar{S} \neq \emptyset$  do
4      $next \leftarrow \arg \max_{v \in V} f(v)$ 
5     //  $u$  flies to  $next$ 
6      $S_u \leftarrow S_u + shortest\_path(pos(u), next)$ 
7   end
8   //  $u$  flies back to its initial vertex
9    $S_u \leftarrow S_u + shortest\_path(pos(u), S_u(0))$ 
10 end
11 return  $S$ 

```

---

For that purpose, it is needed to adapt Q-learning components, i.e. actions, states and policy, to the CCUS model. A state is a solution  $S$  at the time when the UAVs choose a new vertex. The latter corresponds to the action of UAVs. An action is thus represented by a vertex  $v \in V$  on which the UAVs can move. The policy finally returns the node which maximises the fitness function  $f$ . Figure 2 shows an overview of the whole process. At each step, each UAV moves to a vertex  $v$  thanks to an evaluation function depending on the current solution  $S$  and a set of variables  $\Theta$ . The latter defines a heuristic based on the template of low-level heuristics. It means that  $\Theta$  is a parameter of the evaluation function. When every UAV has finished its tour, i.e. has returned to its initial node, a reward is given for each evaluation which has been made. This set of rewards is used to modify the value of  $\Theta$ , and therefore to update the current low-level heuristic.

### 4.2 Detailed Steps

This section will detail the three steps shown in Fig. 2: firstly, the evaluation of vertices done by UAVs in order to choose their next destination; secondly, the reward given for such a choice made by a UAV; thirdly, the update of  $\Theta$  according to the set of obtained rewards.

**Evaluation.** This section refers to point 1 in Fig. 2. In Q-learning, the  $Q$  function evaluates the choice of an action from a certain state. In this context, a  $Q_u$  function is thus defined for the UAV  $u$  to evaluate its choice of going to a vertex  $v$  from a solution  $S$ .

$$Q_u(S, v; \Theta) = \text{scal} (Q_u^{cov}(S, v; \Theta), Q_u^{con}(S, v; \Theta))$$

$Q_u$  is a function of  $Q_u^{cov}$  and  $Q_u^{con}$  evaluating an action according to the coverage and the connectivity objectives respectively.  $\Theta$  is also divided into  $\Theta^{cov}$  and  $\Theta^{con}$ , which are both a collection of eight parameters  $\Theta^o = \{\theta_j^o\}_{j=1}^8$ ,  $\forall o \in \{cov, con\}$ . Since both evaluations are not of the same order of magnitude, they are normalised in order to have a balance between both objectives. We here considered two possible scalarisation functions **scal**, i.e. linear and Chebyshev as introduced in [15].

With the linear function,  $Q_u$  is a linear combination of  $Q_u^{cov}$  and  $Q_u^{con}$ .

$$Q_u(S, v; \Theta) = \sum_{o \in \{cov, con\}} Q_u^o(S, v; \Theta^o)$$

A UAV  $u$  at the solution  $S^{(t)}$  will thus move to the vertex  $v^{(t+1)}$  which maximises  $Q_u$ .

$$v^{(t+1)} = \arg \max_{v \in V} \left\{ Q_u \left( S^{(t)}, v; \Theta \right) \right\}$$

With the Chebyshev function,  $Q_u$  is the distance from an utopian point  $z$  according to the  $L_\infty$  metric (also called Chebyshev metric).  $z^{cov}$  and  $z^{con}$  are constantly adjusted during the learning process to represent the best evaluations for the coverage and the connectivity respectively.

$$Q_u(S, v; \Theta) = \max_{o \in \{cov, con\}} |Q_u^o(S, v; \Theta) - z^o|$$

A UAV  $u$  at the solution  $S^{(t)}$  will thus move to the vertex  $v^{(t+1)}$  which minimises  $Q_u$ , i.e. the distance from the best point  $z$  found so far.

$$v^{(t+1)} = \arg \min_{v \in V} \left\{ Q_u \left( S^{(t)}, v; \Theta \right) \right\}$$

For each objective  $o$ , the evaluation  $Q_u^o(S, v; \Theta)$  is a matrix computation. The state  $S$  and the action  $v$  are represented in the formula by  $p$ -dimensional vectors, respectively  $\mu$  and  $\mu_v$ . Their computation is detailed in the paragraph

*Embedding structure* below. The definition of  $Q_u^o(S, v; \Theta)$  and the embedding structure differ from [9]. This new definition has shown that it is more stable.

$$Q_u^o(S, v; \Theta^o) = \theta_2^{o\top} \mathbf{relu}([\theta_3^o \mu, \theta_4^o \mu_v] + \theta_5^o \delta_{uv}^o)$$

where  $\theta_2^o, \theta_5^o \in \mathbb{R}^{2p}$ ,  $\theta_3^o, \theta_4^o \in \mathbb{R}^{p \times p}$ ,  $[\cdot, \cdot]$  is the concatenation operator and  $\mathbf{relu}$  is the rectified linear unit, i.e. for any vector  $X = (x_i)_i$ ,  $\mathbf{relu}(X) = (\max(0, x_i))_i$ . The term  $\delta_{uv}^o$  is useful for making each UAV  $u$  have a different evaluation of a vertex  $v$ . It is different according to the objective  $o$ .  $\delta_{uv}^{cov}$  is the distance between the evaluating vertex and the UAV evaluating it, while  $\delta_{uv}^{con}$  is the sum of distances between the evaluated vertex and the other UAVs.

$$\delta_{uv}^{cov} = \text{dist}(\text{pos}(u), v) \qquad \delta_{uv}^{con} = \sum_{u' \in U \setminus \{u\}} \text{dist}(\text{pos}(u'), v)$$

*Embedding Structure.* Each node  $v \in V$  is represented by a  $p$ -dimensional feature  $\mu_v$ . The latter is recursively computed according to the structure of the environment graph.  $\forall v \in V$

$$\mu_v = \mathbf{relu}(\theta_1^o \cdot x_v^o)$$

where  $\theta_1^o \in \mathbb{R}^p$ . The state variable  $x_v$  is different between the coverage and the connectivity.  $x_v^{cov}$  is a binary variable determining whether  $v$  has been visited or not, while  $x_v^{con}$  corresponds to the number of UAVs currently on  $v$ .

$$x_v^{cov} = \begin{cases} 1 & \text{if } \exists u \in U, v \in S_u \\ 0 & \text{otherwise} \end{cases} \qquad x_v^{con} = |\{u \in U \mid \text{pos}(u) = v\}|$$

Regarding the Q-learning aspect, any action is therefore represented as a  $p$ -dimensional vector  $\mu_v$ . Similarly, any state is written as a  $p$ -dimensional vector  $\mu = \sum_{v \in V} \mu_v$ , by summing the embedding structure of every vertex in the environment graph. Thanks to this embedding structure, it is then possible to evaluate any action from any state of any instance.

**Reward.** This section details point 2 in Fig. 2. The asynchronous process performed by each UAV is shown in Algorithm 2. When a UAV  $u$  moves to a vertex  $v^{(t+1)}$  from the solution  $S^{(t)}$ , the latter is added to the list  $S_u$  and a new solution  $S^{(t+1)}$  is obtained. A reward  $r$  must then be given in order to value this choice according to the coverage and the connectivity objectives.

$$r(S^{(t)}, v^{(t+1)}) = O(S^{(t)}) - O(S^{(t+1)})$$

The reward corresponds to the difference between the objective values of the solutions before and after the movement. Since these values must be minimised, the lower is the new one compared to the old one, the better is the action, and thus the greater must be the reward.



With such a reward, the objective value of a solution  $S^{(t)}$  is equivalent to the cumulative reward  $R_{0,t}$  since  $O(S^{(0)}) = (0, 0)^\top$ . The cumulative reward  $R_{i,j}$  defines the sum of every reward obtained by a UAV from the solution  $S^{(i)}$  to reach the solution  $S^{(j)}$ .

$$R_{i,j} = \sum_{t=i}^{j-1} r(S^{(t)}, v^{(t+1)}) = O(S^{(i)}) - O(S^{(j)})$$

When every UAV has finished its tour, a reward is given for each sliding window of  $\tau$  movements made during the coverage. The action made by a UAV at time step  $t$  will be rewarded by the cumulative reward  $R_{t,t+\tau}$ .

**Update.** This section corresponds to point 3 in Fig. 2. The reward is therefore used for improving the future choices of action (which vertex to go). For that purpose,  $\Theta^{cov}$  and  $\Theta^{con}$  are updated when an instance has been processed, by performing a stochastic gradient descent (SGD) step to minimise the squared loss for every UAV  $u$ .

$$\left(y - Q_u^o(S^{(t)}, v^{(t+1)}; \Theta^o)\right)^2$$

with  $y = \gamma \max_v \{Q_u^o(S^{(t+\tau)}, v; \Theta^o)\} + R_{t,t+\tau}$ .

$\gamma$  is the discount factor. Its value is between 0 and 1 and represents the importance of the future reward depicted by  $\max_v \{Q_u^o(S^{(t+\tau)}, v; \Theta^o)\}$ .  $R_{t,t+\tau}$  is the cumulative reward obtained during the frame of  $\tau$  movements. For each evaluation which has been made, a  $y$  is associated and can be assimilated to the rectified evaluation.

### 4.3 General Pseudo-code

Algorithm 2 describes the whole process of the proposed QLHH. The operation that each UAV executes until it comes back to its starting point appears between lines 7 and 25. If there are still unvisited vertices in  $G_e$ , every node is evaluated with  $Q_u$  according to the current solution  $S$  and  $\Theta$  (line 13). The vertex maximising this evaluation is chosen. If every vertex has been visited, the UAVs will return to their initial position (line 15). The shortest path between the current position and destination of a UAV is then added to its path (line 18). The reward for such a movement is computed (line 19) and  $S^{(t+1)}$  is the new current solution (line 20). Finally, the solution  $\tau$  iterations ago, the current solution and the cumulative reward between both is registered in the memory  $\mathcal{M}$  (line 23). This memory is then used to process the SGD step to modify  $\Theta$  when every UAV has finished its tour (line 27).

## 5 Experiments

This section presents the experimental results of QLHH on the CCUS problem which have been conducted on the High Performance Computing (HPC) platform of the University of Luxembourg [17].

## 5.1 Comparison Heuristic

In order to evaluate the performance of the heuristic generated by QLHH, the results have been compared to those obtained with the manually-designed heuristic described in this section. This heuristic, so called *Weighted Objective* heuristic (WO), belongs to the space of low-level heuristic of QLHH. It means that it respects the template defined in Algorithm 1. At each step, every UAV evaluate every vertex in order to choose the next destination. In this case, this evaluation of the UAV  $u$  will be the sum of 2 values,  $e^{cov}(u, v)$  and  $e^{con}(u, v)$  evaluating the vertex  $v$  according to the coverage and the connectivity respectively.

$$e^{cov}(u, v) = \begin{cases} W - W \cdot dist(pos(u), v) & \text{if } x_v = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$e^{con}(u, v) = \begin{cases} W & \text{if } D(u, v) \leq D_{com} \\ 2W - \frac{W}{D_{com}} D(u, v) & \text{otherwise} \end{cases}$$

where  $W$  is a given weight, representing the maximal value for both objectives, and  $D(u, v) = \min_{u' \in U \setminus \{u\}} dist(pos(u'), v)$ .

---

### Algorithm 2: Algorithm of the proposed QLHH

---

```

input   : Distribution  $\mathbb{D}$  of instances
output  : Updated  $\Theta$ 
1 Randomly generate  $\Theta$ 
2  $I \leftarrow$  collection of instances  $i \hookrightarrow \mathbb{D}$ 
3  $\mathcal{M} \leftarrow \emptyset$  // initialisation of the memory
4 foreach epoch  $e$  do
5   foreach instance  $i \in I$  do
6      $S \leftarrow \emptyset$  // initialisation of the solution
7     foreach UAV  $u \in U$  do // asynchronously
8        $t \leftarrow 0$ 
9        $S^{(0)} \leftarrow S$ 
10       $S_u \leftarrow (pos(u))$ 
11      repeat
12        if  $\bar{S} \neq \emptyset$  then
13           $v^{(t+1)} \leftarrow \arg \max_{v \in V} Q_u(S, v; \Theta)$ 
14        else
15           $v^{(t+1)} \leftarrow S_u(0)$  // every vertex has been covered
16        end
17        //  $u$  flies to  $v^{(t+1)}$ 
18         $S_u \leftarrow S_u + \text{shortest\_path}(pos(u), v^{(t+1)})$ 
19        Compute  $r(S^{(t)}, v^{(t+1)})$ 
20         $S^{(t+1)} \leftarrow S$ 
21         $t \leftarrow t + 1$ 
22        if  $t \geq \tau$  then
23           $\mathcal{M} \leftarrow \mathcal{M} \cup \{(S^{(t-\tau)}, R_{t-\tau, t}, S^{(t)})\}$ 
24        end
25      until  $v^{(t+1)} = S_u(0)$ 
26    end
27    Update  $\Theta$  with a SGD step for  $\mathcal{M}$ 
28  end
29 end
30 return  $\Theta$ 

```

---

## 5.2 Performance Metrics

**QLHH Metrics.** Two state-of-the-art metrics [4] have been used for assessing the performance of the heuristics generated with QLHH: one coverage metric (coverage speed) and one connectivity metric (number of connected components). Coverage speed expresses how fast the UAVs cover a certain rate  $r$  of vertices of the grid. It corresponds to the lowest time step  $t$  for which the rate of visited vertices exceeds  $r$ . For the experiments,  $r = 0.95$ , i.e. the coverage speed represents the time needed by the UAVs to cover 95% of the grid. The number of connected components gives the average number of connected components of the connectivity graph (represented in Fig. 1) at each time  $t \in T$ .

**Table 1.** Parameters used for the training executions

Parameter	Notation	Value
<i>Problem-specific parameters</i>		
Embedding dimension	$p$	8
Experience duration	$\tau$	10
Maximum communication distance	$D_{com}$	4
<i>Q-learning parameters</i>		
Learning rate	$\alpha$	0.01
Discount factor	$\gamma$	0.9

**MO Metrics.** Three metrics have been used to compare Pareto fronts in a bi-objective space, defined by both objective metrics explained earlier: Hyper-Volume (HV), Spread ( $\Delta$ ) and Inverted Generational Distance (IGD). HV represents the volume in the objective space covered by non-dominated solutions [18].  $\Delta$  defines how well the non-dominated solutions are spread in the front. It is the average of gaps between the distance between two adjacent solutions in the front and the mean of these distances. Finally IGD measures the average distance between the approximated front and the optimal one [16].

## 5.3 Experimental Setup

For the experiments, grid graphs have been used as environment graphs. Moreover, instances have been split into classes defined by their grid dimension and their number of UAVs. Within a class, instances thus differ in terms of the initial position of UAVs. A class is then written in the following format: (dim\_grid/nb\_uavs). QLHH has been trained only on the smallest instance class (smallest number of vertices and UAVs). This not only permits to reduce the training time which can be considerably long on large instances, but it will also permit to demonstrate that the generated heuristics also perform well on larger problem instances. After training QLHH on the class (5 × 5/3), the obtained

heuristic will be executed 30 times on each instance class. Table 1 presents the parameterisation used for training QLHH. The first set of parameters depends on the problem, i.e. they are used in the context of CCUS. The second set of parameters is problem independent and is only used in the Q-learning process.

### 5.4 Experimental Results

After executing the generated heuristic (GH) and WO heuristic on 30 instances of each class, both Pareto fronts are compared according to the three MO metrics defined in Sect. 5.2. Results are presented in Table 2. For each class, bold values correspond to the best ones and relative differences are relative to highest values. The IGD metric shows that the Pareto front obtained with GH features a better convergence on all instances. On the contrary, the spread of solutions in the

**Table 2.** Comparison between WO heuristic and the heuristic generated with QLHH

Metrics	Instances		Heuristics		Relative Differences	
	# Vertices	# UAVs	QLHH	WO		
HV	5 × 5	3	9.385e+1	<b>9.495e+1</b>	1.15%	
		10 × 10	3	4.189e+2	<b>4.484e+2</b>	6.57%
			5	5.814e+2	<b>6.345e+2</b>	8.37%
			10	1.112e+3	<b>1.128e+3</b>	1.44%
			15	1.591e+3	<b>1.624e+3</b>	2.00%
	15 × 15	3	6.982e+2	<b>1.006e+3</b>	30.58%	
		5	<b>1.493e+3</b>	1.342e+03	10.09%	
		10	<b>2.581e+3</b>	2.386e+03	7.56%	
	Δ	5 × 5	3	<b>0.000e+0</b>	<b>0.000e+0</b>	/
10 × 10			3	<b>4.203e-1</b>	5.616e-1	25.17%
			5	1.866e-1	<b>1.199e-1</b>	35.77%
			10	1.037e-1	<b>0.000e+0</b>	100.00%
			15	7.211e-1	<b>4.147e-3</b>	99.42%
15 × 15		3	8.510e-1	<b>7.240e-2</b>	91.49%	
		5	2.032e-1	<b>7.759e-2</b>	61.82%	
		10	<b>1.281e-1</b>	2.421e-1	47.07%	
IGD		5 × 5	3	<b>1.757e-1</b>	5.331e-1	67.04%
	10 × 10		3	<b>5.185e-1</b>	5.991e-1	13.45%
			5	<b>3.056e-1</b>	8.276e-1	63.07%
			10	<b>5.205e-1</b>	8.720e-1	40.31%
			15	<b>2.129e-1</b>	6.932e-1	69.29%
	15 × 15	3	<b>8.114e-1</b>	9.454e-1	14.18%	
		5	<b>3.191e-1</b>	7.634e-1	58.20%	
		10	<b>1.234e-1</b>	5.452e-1	77.37%	

front ( $\Delta$ ) is worse for GH except for the two smallest and the largest instance classes ( $(5 \times 5/3)$ ,  $(10 \times 10/3)$  and  $(15 \times 15/10)$ ). Finally the HV metric, shows the good performance of GH with results comparable to WO almost every class, while GH performs better on the two largest instances.

Experimental results thus demonstrate that QLHH is able to generate competitive swarming heuristics for any of the instance classes. Moreover, they permit to outline that QLHH is able to generate scalable heuristics, since this good global performance is obtained while the model was trained on the smallest instances only ( $5 \times 5/3$ ). This is a strong asset since the training process would be very long for large instances. With that model, the training can be much faster while keeping good performances.

## 6 Conclusion

This work has presented a model for optimising the coverage of a connected swarm, so called *Coverage of a Connected-UAV Swarm* (CCUS). In order to generate efficient UAV swarming behaviours, a Q-Learning based Hyper-Heuristic (QLHH) has been designed for generating distributed CCUS heuristics. The experiments have shown a good stability of the model. It means that it is possible to fast train the model on small instances and maintain the good properties resulting on bigger instances. Future work will consist in conducting experiments on additional larger classes of instances. Another extension will consider extending QLHH with Pareto-based approaches, instead of the scalarisation currently used to balance both objectives.

## References

1. Aznar, F., Pujol, M., Rizo, R., Rizo, C.: Modelling multi-rotor UAVs swarm deployment using virtual pheromones. *PLoS ONE* **13**(1), e0190692 (2018)
2. Babić, A., Mišković, N., Vukić, Z.: Heuristics pool for hyper-heuristic selection during task allocation in a heterogeneous swarm of marine robots. *IFAC-PapersOnLine* **51**(29), 412–417 (2018)
3. Birattari, M., et al.: Automatic off-line design of robot swarms: a manifesto. *Frontiers Robot. AI* **6**, 59 (2019)
4. Brust, M.R., Zurad, M., Hentges, L., Gomes, L., Danoy, G., Bouvry, P.: Target tracking optimization of UAV swarms based on dual-pheromone clustering. In: 2017 3rd IEEE International Conference on Cybernetics (CYBCONF), pp. 1–8. IEEE (2017)
5. Burke, E.K., et al.: Hyper-heuristics: a survey of the state of the art. *J. Oper. Res. Soc.* **64**(12), 1695–1724 (2013)
6. Liu, C., Xin, X., Dewen, H.: Multiobjective reinforcement learning: a comprehensive overview. *IEEE Trans. Syst. Man Cybern. Syst.* **45**(3), 385–398 (2015)
7. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 176–190. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-44629-X\\_11](https://doi.org/10.1007/3-540-44629-X_11)

8. Duflo, G., Danoy, G., Talbi, E.G., Bouvry, P.: Automated design of efficient swarming behaviours: a Q-learning hyper-heuristic approach. In: Genetic and Evolutionary Computation Conference Companion, pp. 227–228. ACM (2020)
9. Duflo, G., Danoy, G., Talbi, E.G., Bouvry, P.: Automating the design of efficient distributed behaviours for a swarm of UAVs. In: Symposium Series on Computational Intelligence - SSCI 2020. IEEE, Canberra, Australia (2020)
10. Duflo, G., Kieffer, E., Brust, M.R., Danoy, G., Bouvry, P.: A GP hyper-heuristic approach for generating TSP heuristics. In: 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 521–529. IEEE (2019)
11. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In: Guyon, I., (eds.) Advances in Neural Information Processing Systems. vol. 30, pp. 6348–6358. Curran Associates, Inc. (2017)
12. Kieffer, E., Danoy, G., Brust, M.R., Bouvry, P., Nagih, A.: Tackling large-scale and combinatorial bi-level problems with a genetic programming hyper-heuristic. *IEEE Trans. Evol. Comput.* **24**(1), 44–56 (2019)
13. Lin, J., Zhu, L., Gao, K.: A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem. *Exp. Syst. Appl.* **140**, 112915 (2020)
14. Tuyls, K., Stone, P.: Multiagent learning paradigms. In: Belardinelli, F., Argente, E. (eds.) EUMAS/AT -2017. LNCS (LNAI), vol. 10767, pp. 3–21. Springer, Cham (2018). [https://doi.org/10.1007/978-3-030-01713-2\\_1](https://doi.org/10.1007/978-3-030-01713-2_1)
15. Van Moffaert, K., Drugan, M.M., Nowe, A.: Scalarized multi-objective reinforcement learning: novel design techniques. In: IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), pp. 191–199. IEEE (2013)
16. Van Veldhuizen, D.A.: Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations. Ph.D. thesis, USA (1999), aAI9928483
17. Varrette, S., Bouvry, P., Cartiaux, H., Georgatos, F.: Management of an academic HPC cluster: the UL experience. In: Proceedings of the 2014 International Conference on High Performance Computing & Simulation (HPCS 2014), pp. 959–967. IEEE (July 2014)
18. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Trans. Evol. Comput.* **3**(4), 257–271 (1999)