Hyper-heuristic Online Learning for Self-assembling Swarm Robots

Shuang Yu[1](✉), Aldeida Aleti[1], Jan Carlo Barca[1], and Andy Song[2]

[1] Monash University, Clayton 3168, Australia
shuang.yu@monash.edu
[2] RMIT University, Melbourne 3000, Australia

Abstract. A robot swarm is a solution for difficult and large scale tasks. However, controlling and coordinating a swarm of robots is challenging, because of the complexity and uncertainty of the environment where manual programming of robot behaviours is often impractical. In this study we propose a hyper-heuristic methodology for swarm robots. It allows robots to create suitable actions based on a set of low-level heuristics, where each heuristic is a behavioural element. With online learning, the robot behaviours can be improved during execution by autonomous heuristic adjustment. The proposed hyper-heuristic framework is applied to surface cleaning tasks on buildings where multiple separate surfaces exist and complete surface information is difficult to obtain. Under this scenario, the robot swarm not only needs to clean the surfaces efficiently by distributing the robots, but also to move across surfaces by self-assembling into a bridge structure. Experimental results showed the effectiveness of the hyper-heuristic framework; the same group of robots was able to autonomously deal with multiple surfaces of different layouts. Their behaviours can improve over time because of the online learning mechanism.

Keywords: Hyper-heuristics · Online learning · Swarm robots
Robotic behaviors · Self-assembling robots · Robotic surface cleaner

1 Introduction

Robotics is a fast growing area due to the explosive development in Artificial Intelligence. Robots can automate many tasks that are considered risky and undesirable for humans. In addition they may accomplish tasks that are considered difficult for humans. One example is swarm robots where many robots, typically small droids, can collaborate and behave cohesively in one accord to achieve tasks that are difficult or expensive such as surveying, rescue and patrol.

A reliable and capable robot swarm can gain significant advantage in various domains such as mining, agriculture, smart cities and even military applications. However, coordinating a collection of robots is not a trivial task. All robots need to operate collaboratively to achieve a common goal. Manually developing behavioral strategies for each robot using a collaboration strategy to coordinate them can be difficult and ineffective, especially when the environment is dynamic.

Instead we propose a learning based hyper-heuristic approach for swarm robots. With the hyper-heuristic approach, we only supply low level operators instead of a full control strategy. These operators are elements for constructing instructions for various tasks and environments. More importantly, the instructions can be adjusted over time because of the learning mechanism. Therefore it is not necessary to manually define the robot behaviors beforehand as the performance of these robots can improve during execution.

As a case study, we apply the proposed hyper heuristic framework to a surface cleaning problem, where multiple building surfaces need to be cleaned. Basic moves and operators are supplied for the hyper-heuristic engine to build cleaning strategies. The robots are required to clean varied surface layouts using different swarm behaviours. Moreover the robots should be capable of self-assembly in order to cross gaps in the surfaces and travel from surface to surface. Through a range of test scenarios, we demonstrate the effectiveness of the proposed hyper heuristic approach. Analysis on the learning behaviors is also present to give insight into this method.

The rest of the paper is organised as follows: Sect. 2 reviews the related work, Sect. 3 describes the proposed hyper-heuristic methodology. A case study on self-assembly swarm robot behaviours demonstrates an implementation of the methodology, and is given in Sect. 4; this also includes the implementation of the heuristic repository. Section 5 shows simulations of the system used in the real-world application of cleaning multiple surfaces.

2 Related Work

Hyper-heuristics have been used in various complex problems, such as bin-packing, timetabling and vehicle routing. The aim of hyper-heuristic approaches is to design a generic method that can automatically generate algorithms from a repository of low-level heuristics or operators to solve a given problem [2]. It searches for solvers instead of solutions. To give a brief background of hyper-heuristic methods, a tabu search hyper-heuristic is applied on nurse timetabling and scheduling [4], [16] used a choice function to rank heuristics for selection, and [3] explored the use of Genetic Programming as a hyper-heuristic, and demonstrated it on Boolean Satisfiability problem and online bin-packing. Combined with online learning, which continuously learns the knowledge while performing actions, hyper-heuristics solve problems with dynamic environments. For example, [20] performs online learning to assist hybridising Estimation Distribution Algorithms with hyper-heuristics in dynamic environments. In [8], to determine shipper sizes for storage and transportation, reinforcement learning with tabu search is used to modify the performance score for each low-level heuristic at every decision point.

Regarding heuristic learning, there are enormous amount of existing studies in robotics. However, most of them concern single robots, hence are not included in our review. In robot swarms, the local control laws executed by every robot give rise to overall system dynamics, which is defined as a swarm behaviour [14].

Rule-based algorithms allow complex and collective behaviours to emerge from local behavioural rules, such as collective building construction [21]. Heuristic learning techniques have been widely used in robot swarms to guide navigation (search and path planning), task scheduling, motion control, etc., but they are all tailored to their intended applications, and are not re-usable for other purposes, such as sequencing swarm behaviours. [14] defined the problem of swarm behaviour composition, and proposed an off-line learning method to generate behaviour sequences for a human operator to execute in a known and static environment. This lacks the ability to cope with dynamic environments. Moreover, the existing methods require manual algorithm re-design from task to task, while hyper-heuristics, on the other hand, are often used to automatically generate algorithms for a new problem.

The feasibility of this will be demonstrated through a case study on self-assembling robot behaviours. Self-assembling robots can physically join and form larger structures, as seen for example in Swarmanoid [6] and REPLICATOR projects [9]. They have demonstrated great mobility in complex environments by crossing gaps and moving across surfaces that could not have been traversed by a single robot. The Swarm-bot project [7] demonstrates the hardware implementation of completing tasks, such as retrieving heavy objects and passing over holes in a collective manner.

All these systems require manual input, such as a human operator or a control station, to determine and inform the robots when to start assembling/disassembling and what behaviour to perform for different problems and scenarios.

3 Methodology of the Hyper-heuristics Framework

We first introduce the hyper-heuristic methodology that is proposed to guide the self-assembling in a robot swarm.

Hyper-heuristics are built on heuristics. They treat heuristics as general purpose building blocks to be simultaneously and iteratively applied to a problem [18]. More importantly, hyper-heuristics are problem-independent, and the same repository of heuristics can be used to solve new problems.

3.1 Multi-robot Hyper-heuristic Structure

An overview of the proposed hyper-heuristics framework structure for swarm robots is given in Fig. 1. Each robot is **initialised** with a starting heuristic, then takes **action** corresponding to that heuristic. If the **termination criteria** is satisfied, meaning the objective is accomplished, then the process stops. Otherwise the robot behaviours are **evaluated** against the objective(s). Based on the evaluation result, heuristics then are selected from the pre-defined **heuristic repository** to construct new ones. All the robots will **update** their heuristics, take actions then enter the next iteration of the process unless a termination criterion is met.

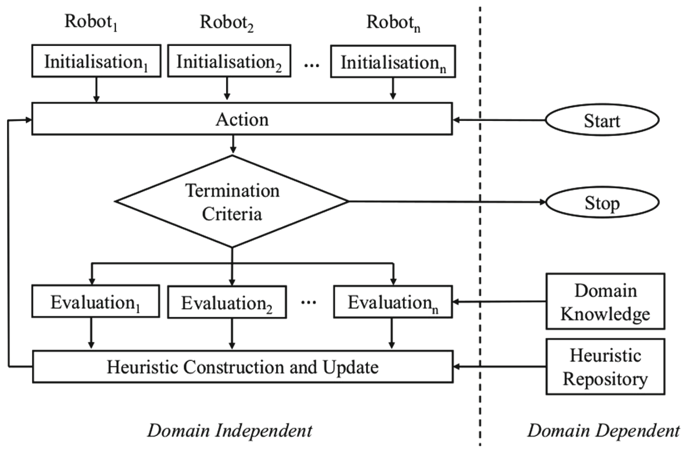


Fig. 1. Overview of the methodology.

In terms of evaluation of the objective, domain knowledge is required. In robotics, domain knowledge is usually dependent on the available sensors and robot communication mechanisms. Performance evaluation represents some "benefits" from the environment and can be translated and stored as performance scores within each robot. These scores are used to influence the heuristic selection. The heuristic repository is also domain relevant, and contains a variety of low-level heuristics.

3.2 Heuristic Construction

Our proposed hyper-heuristic structure allows the actions for a given problem to be automatically constructed and updated at each time interval. The performance of a heuristic measured in the environment is stored in each robot as heuristic scores. Through online learning, the robots can gradually learn to select better heuristics.

Heuristic Scores. If there is no prior knowledge, heuristic scores are initialised with the same value. The initial score can be obtained from prior learning. The heuristic scoring method is inspired by Choice Function (CF) and Multi-armed Bandit (MAB), as described in [5,16] respectively. Each robot calculates the score of the last action locally for each heuristic based on the objective function.

$$h_t^i = \sum(\alpha f_t + \beta f_t^{ij} + \gamma M_t^i), \tag{1}$$

where

- f_t is the objective function value evaluated at time step t,
- f_t^{ij} is the objective function value of heuristic i given the previous heuristic j at time step t, and

– M_t^i is the MAB term for heuristic i at time step t.

α, β and γ are parameters that control the weight of each term on the overall score of a heuristic. The performance of the heuristic and the joint performance of pairs of heuristics are measured using the objective function. For instance, in the case of cleaning tasks, f_t would be the cleaning efficiency measured from robot dirt sensors. The MAB term M_t^i is introduced as in Eq. 2 to compensate for heuristics that are unexplored [5].

$$M_t^i = \sqrt{\frac{2 \log \sum_{j=1}^{k} n_j}{n_i}} \tag{2}$$

where k is the total number of heuristics and i represents the index of the current heuristic. The MAB term is inversely proportional to how many times the current heuristic has been used n_i, and directly proportional to the total number of times the other heuristics have been used $\sum_{j=1}^{k} n_j$. This ensures that unexplored heuristics are given more weight than the ones that have been used often, which prevents the algorithm from becoming trapped in a local optima.

Learning the Heuristic Scores. The scores $\hat{H}_t = \{\hat{h}_t^1, \hat{h}_t^2 \ldots, \hat{h}_t^k\}$ are updated at the end of each iteration according to previous experience and current performance, as shown in Eq. 3.

$$\hat{h}_t^i = \theta h_t^i + (1 - \theta)\hat{h}_{t-1}^i \tag{3}$$

where $\hat{h}_{t-1}^i$ is the previous score of heuristic i, and θ is the learning rate. Higher θ values favour recent knowledge over the accumulated knowledge.

Heuristic Selection. The heuristic scores learned in the previous t iterations affect the heuristic to be selected in the next iteration $t + 1$. There are different strategies that could be used to select the appropriate heuristic from a heuristic repository. One of them is Greedy selection, always selecting the heuristic with the highest score. Relatively worse heuristics would not be re-used even if it may lead to better solutions when the environment changes. The second method is Roulette Wheel selection (RW) [10]. A good heuristic has greater chance to participate, while "bad" ones also have the chance to be selected. This method is quick to explore heuristics. This is further discussed through simulations in Sect. 5.

Group Acceptance and Update. The robots can communicate to reach an agreement on which heuristic to execute next. In software optimisation problems, after a selected heuristic is applied to the problem, hyper-heuristics use a move acceptance method to determine if the heuristic, or the "move" should be accepted or rejected [1]. In a robotic system, the move acceptance strategy is "always accept", because rejecting a heuristic means going back to the robot's original state (position), which in the real-world is impractical and consumes energy. In addition, we use a **decentralised group acceptance strategy** which only accepts heuristics that benefit the whole group.

For swarm robotics, decentralised systems are more robust, scalable and flexible. No single robot possesses the entire swarm's knowledge of the environment. Every robot can evaluate a heuristic based on its local knowledge, and communicate the result to its neighbours.

The decentralised strategies are similar to that in [15]. A heuristic is accepted if $\sum_{n=1}^{N} d_n \geq \delta$, where d_n is the robot decision on acceptance, N is the number of robot decisions received, with "accept the heuristic" as $d_n = 1$, and reject as $d_n = 0$. δ is the acceptance threshold.

After the new heuristic is selected and accepted by the swarm, all the robots will update their own heuristics to be the new heuristic, and take another action.

4 Case Study on Self-assembling Multi-robot Systems

To show the feasibility and effectiveness of the proposed hyper-heuristics methodology, we present a case study using self-assembling robots.

4.1 Problem Description

In self-assembling robots, swarm behaviours emerge from the physical connections and interactions between robots. A robot can be seen as a module and an emergent behaviour can result from the interaction between these modules.

For example, robots can connect to cross gaps, scatter to cover surfaces, or flock to stay close but separated. A repository of such basic behaviours $B = \{b^1, b^2 \dots b^k\}$, defines the set of robot control laws that read sensor data, and execute actions accordingly. The task is then, *given a set of self-assembling robot behaviours, and an objective function, construct a sequence of such behaviours autonomously to maximize the objective value $f(t)$ in unknown environments.*

4.2 Implementation

To solve this problem, we consider a type of swarm robot behaviour as a *heuristic*, which is defined by the control rules that take in environmental input and control actions periodically.

Robots start with an initial heuristic, and the same score $\hat{h}_0^i$ for each heuristic in the repository. This is based on the assumption that the environment is unknown, which means there is no knowledge of which heuristic is better. The robots perform actions for a period of time guided by the initial heuristic, and learn heuristic scores online according to the method described in Sect. 3.2.

Any robot can propose a candidate heuristic for the next time interval to the group, and in order to physically achieve the group decision process described in Sect. 3.2, the robots send communication messages to each other following the diagrams in Fig. 2. A robot sends a candidate heuristic to neighbouring robots.

The neighbouring robots then run RW selections according to their locally kept heuristic scores and compare their selections with others. If they match, the neighbour will inform the proposing robot of its acceptance.

If the number of robots accepting passes the acceptance threshold d_n, then the group has collectively decided to accept the candidate heuristic and will apply that heuristic for the next iteration. If the candidate heuristic is rejected, another proposal will be made, and the process repeats until a candidate is accepted by the group.

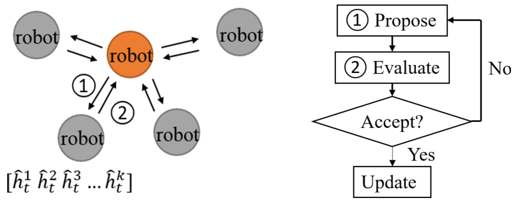
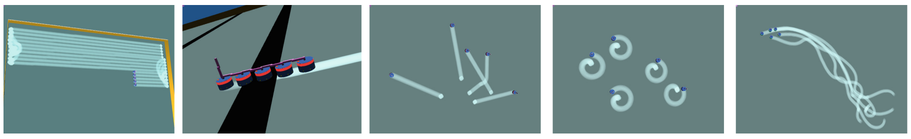


Fig. 2. Implementation of evaluation and group acceptance.

There are three benefits in this approach. Firstly, the robots do not require a human operator or centralised control, making the group scalable and robust against single-point failure. Secondly, it allows robots to collectively adapt to unknown or dynamic environments. Lastly, the behaviour construction is not problem-specific. New tasks only require the change of an objective function, not any re-design of the hyper-heuristic. For instance, for surface cleaning tasks, the area cleaned is used to evaluate the performance; for multi-robot rendezvous, the distance to the meeting point would be the objective function.

4.3 Heuristic Repository

The heuristic repository for this cleaning task comprises: sweeping, bridging, exploring, circling and flocking, as described below:



(a) Sweeping (b) Bridging (c) Exploring (d) Circling (e) Flocking

Fig. 3. Heuristic repository

Sweeping. As shown in Fig. 3(a), the sweeping behaviour allows robots to physically connect to form a vertical line, and perform horizontal back-and-forth movements.

Bridging. Bridging behaviour enables the swarm to connect and form a bridge structure, as demonstrated in Fig. 3(b). It allows a group of robots to cross gaps and small obstacles, as a connection is formed between robots to keep them from falling when they lose contact with the surface.

Exploring. Exploring is a behaviour that allows the robots to scatter in random directions (which change at random time intervals) on the surface, as indicated in Fig. 3(c). To prevent any robot from moving outside other robots' communication range, each robot keeps estimating its distance to the furthest robot in range, and if the distance is outside a threshold, the robot moves towards its furthest neighbour, until it is within the threshold again.

Circling. Circling performs a spiral motion, following with straight lines in random directions, as shown in Fig. 3(d). This is a cleaning pattern widely used by floor cleaning robots.

Flocking. This local controller enables robots to follow a flocking behaviour [17], as illustrated in Fig. 3(e). This heuristic allows robots to stay relatively close to each other, making it easier for the swarm to assemble when needed, while preserving some degree of random exploration. Assume the position of the robot $P_0 = (x_0, z_0)$. $\overline{P}(\overline{x}, \overline{z})$ is the average position of neighbours, and $\overline{O}$ is the average orientation of neighbours within the sensing range. A repulsion vector $R(x_R, z_R)$ prevents robots from being too close to each other, and $R = -P_{min}$, where P_{min} is the relative position of the nearest neighbour. The alignment vector for each robot $P'(x', z')$ is calculated as:

$$\begin{aligned} x' &= \overline{x} + S_x \cdot \cos \overline{O} + x_R \\ z' &= \overline{z} + S_z \cdot \sin \overline{O} + z_R \end{aligned}$$

where S_x and S_z are scaling factors, which can be determined empirically.

The Obstacle and Gap Avoidance Mechanism. This is implemented for all heuristics. In a complex environment, there could be obstacles which the robots need to avoid and go around, and gaps that the robots cannot cross on their own. In this case study, we use potential field based obstacle avoidance [19], which applies a multiplier v_{multi} to linear velocity, and another multiplier ω_{multi} to the robot's angular velocity. The multipliers are calculated as follows:

$$v_{multi} = 1 - e^{-\frac{K_{rise} d_{obs}}{R_{avoid}}}$$

$$\omega_{multi} = \begin{cases} K_\omega(\pi/2 - \theta_{obs}), & \text{if } \pi > \theta_{obs} > 0 \\ K_\omega(-\pi/2 - \theta_{obs}), & \text{otherwise} \end{cases}$$

where K_{rise} and K_ω are control parameters to be tuned empirically, d_{obs} is the distance between obstacle and robot, and θ_{obs} is the angle between robot orientation and the obstacle. If a gap is detected, the robot will rotate in a random direction until it orients away from the gap, and continue to move.

5 Experiments and Results

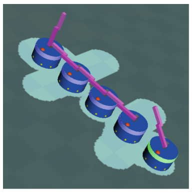
The specific task in our experiment is building facade cleaning where multiple surfaces exist. To move from surface to surface, robots need to physically connect to move each other across surfaces. To clean the surface efficiently, robots need to scatter on the surface according to the environmental layout. This means that no single behaviour is able to clean multiple surfaces, and the problem requires combinations of heuristics.

In the real-world, the outer surfaces of a building are often complex and large. It is costly to build a map of each building for robot cleaners. Even if a map is built, during the cleaning process, the changes caused by window opening/closing or decorations will make the map inaccurate. In such a dynamic environment, swarm behaviours should be adaptive and decentralised.

5.1 Experimental Setups

The experiments are simulated in Webots Simulator with emulated real-world physics [13]. The robot model is based on the non-holonomic robots in [22], as shown in Fig. 4. Each robot is equipped with differential wheels, a dirt sensor, obstacle sensors, wireless communication, suction cups, cleaning wipes and a gripper arm to connect to a neighbouring robot. The robots are simulated to clean at the maximum speed of 2.98 m^2 per time interval of 40 s, which for simplicity, we define as one unit area. The table in Fig. 4 contains the configuration of the simulated robots. To model cleaning in the real world, robots collect 14.9% of the available dirt each time when travelling at the maximum speed of 0.5 m/s. Every robot is controlled by the hyper-heuristic controller as described in Sect. 3.

The objective function $f(t)$ indicates robot cleaning efficiency, which is given by: $f(t) = \frac{\sum(L_t D_t)}{T}$ where L_t is the distance travelled since the last measure, D_t is the dirtiness reading from the dirt sensor at iteration t, and T is the amount of time the current heuristic is applied. The termination criteria is set to be terminating after 50 iterations.



Radius	$0.3m$
Maximum Cleaning Speed	$0.0745m^2/s$
Maximum Linear Velocity	$0.5m/s$
Maximum Angular Velocity	$0.66rad/s$
IR Sensing Range	$0.53m$
Wireless Communication Range	$200m$

Fig. 4. LEFT: robots forming a larger structure. The lighter areas have been cleaned by robots. RIGHT: robot configurations of the simulation

Parameter tuning for the system involves two independent stages. The heuristic parameters are tuned individually as separate control algorithm units, in order to achieve better performance by themselves. The hyper-parameters are tuned with the whole system integrated. In this application, Iterated Racing (*irace*) tuning method [11] is used because it is effective and prunes the space of parameter value combinations that have to be checked. The tuning results are shown in Table 1, and are the parameters used in all experiments.

Table 1. Tuned values of hyper-parameters and controller parameters.

Heuristic Parameters

flocking: S_x	0.5
flocking: S_z	0.5
K_{rise}	1.0
K_ω	10.0
R_{avoid}	$0.46m$

Hyper-parameters

Learning rate θ	0.29
Choice Function α	0.99
Choice Function β	0.01
Choice Function γ	0.9
Group acceptance δ	0.5
Time interval length T	$40s$

5.2 Robustness in Different Environments

Four different layouts of facade surfaces are used in the experiments as shown in Fig. 5. Environment (a) is a flat surface that is 8 m × 8 m, bounded by four barriers; environment (b) is the same size, with 50 obstacles; (c) has four gaps on the surface, which single robots cannot cross; (d) has four gaps and 30 obstacles.

The positions of obstacles and gaps are randomly generated, thus different in each experimental run. Robots have no prior knowledge about the surfaces.

Figure 5 shows the cleaning progress of the robots using the hyper-heuristics at 5, 25 and 50 iterations. Since the purpose of the experiments is not to show the completeness of cleaning, but the effectiveness of behaviour sequence construction, 50 iterations are adequate to show the characteristics of the performance curve, as detailed in the resulting plots. It can be seen that the robots are able to perform the cleaning task continuously and robustly for 50 iterations, as the area cleaned over the four environments is continuously increasing. In environments (c) and (d), robots are able to automatically assemble to move across surfaces, and disassemble on new surfaces. This shows the feasibility of the proposed hyper-heuristic methodology on self-assembling robots, and that it can be applied in real-world applications.

We further investigate the performance in each environment in Fig. 6. For comparison, a hyper-heuristic with no learning is implemented, where heuristics are randomly sampled from the repository at each decision point. To reach all surfaces, and move effectively in environments with many obstacles, the swarm needs to learn the heuristics that perform better in these scenarios. Comparison results with the baseline method shows that the online learning hyper-heuristic is successful in finding sequences that perform better, based on the improvement of 28.86%, 37.34%, 21.51% and 18.86% in each environment, and overall improvement of 27.52%.

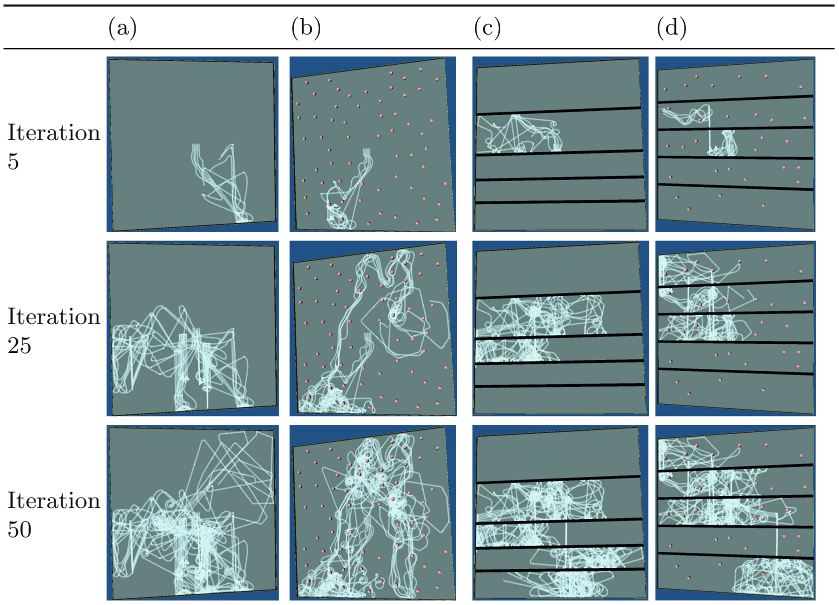


Fig. 5. Cleaning progress of the swarm at 5, 25 and 50 iterations in four types of environmental layouts: (a) flat empty surface, (b) surface with obstacles (indicated by red blobs), (c) five surfaces separated by gaps (black stripes), and (d) five separated surfaces with obstacles. (Color figure online)

We also plot the efficiency improved by learning in each iteration:

$$e(t) = \frac{\sum_{i=1}^{t}(f_i^L - f_i^{NL})}{\sum_{i=1}^{t} f_i^{NL}},$$

where f_i^L and f_i^{NL} are the areas cleaned (objective values) at iteration t by the swarms with and without learning respectively. It can be observed from the results shown in Fig. 7 that in the majority of the cases, the behaviour construction with learning is superior (95% of the points are above 0). Also the method gets better with time, as shown by the blue solid line representing the trend, indicating that our method continuously improves at learning the best behavior.

This proves that without knowing each particular layout, the robots are able to learn the suitable heuristics and autonomously clean multiple surfaces. This offers the advantage of performing tasks without prior knowledge of the environment, and without human supervision. It means that human workers will only need to transport the robot cleaners from building to building and install them on the starting surface, without providing prior knowledge of the building facade layout. Very little or no re-programming of the robots is required between tasks, and no human intervention is needed during the cleaning process.

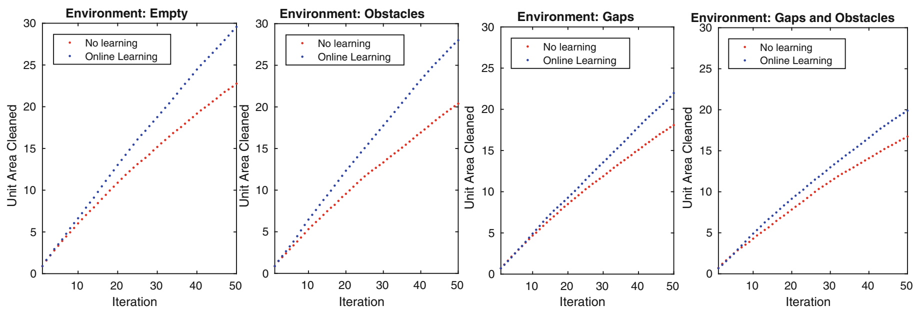


Fig. 6. Comparing mean performance of online learning and no learning over 50 experimental runs

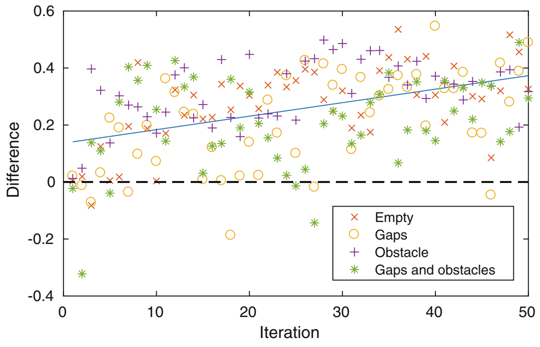


Fig. 7. Performance difference for every iteration between online learning hyper-heuristics and no learning.

5.3 Comparison of Heuristic Selection Methods

In this part we compare three different heuristic selection methods as discussed in Sect. 3.2: Roulette Wheel selection, Greedy selection and Simple Random selection, which randomly samples heuristics from the repository. Each group performs cleaning tasks for 200 runs across the four types of layouts (Fig. 5). Figure 8 plots the performance distribution grouped by environment types.

Through Mann-Whitney U tests [12], it can be confirmed that both RW and Greedy methods have statistically better performance than Simple Random. Greedy has the best performance in empty and obstacles environments, while RW outperforms Greedy in environments that have gaps. RW also gives 48.19% less variance in performance over the four environments. This is because RW selection hyper-heuristic is quick to explore the heuristics that have not performed the best, but could lead to better performance later, therefore is more adaptive in complex environments. Greedy hyper-heuristic is very effective in simple environments, such as the empty surface, but if the user requires robustness in different environment types, RW is a better option.

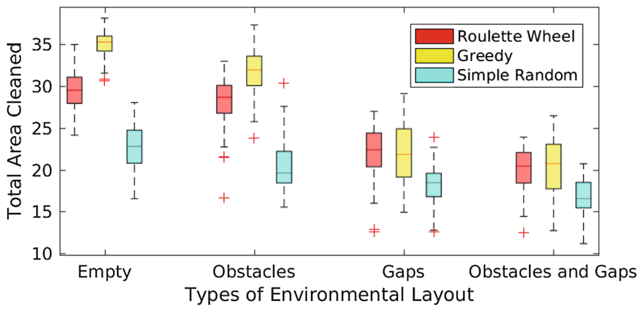


Fig. 8. Comparing three hyper-heuristic selection methods: Roulette Wheel, Greedy and Simple Random over four types of environments.

6 Conclusions

This study proposed a novel hyper-heuristics methodology combined with online learning to coordinate swarm robots, in particular, self-assembling robots. Building surface cleaning is used as a case study to evaluate the framework. The task was carried out on a real-physics robot simulator, and a range of surface types were used as test scenarios. The experiments verify that the robot swarm can adapt in different environmental layouts and automatically find appropriate actions to fulfill the given task without manual programming and centralised control. The study also shows that the performance of the robot swarm can be improved through online learning of heuristic scores.

Hence we conclude that hyper-heuristics are effective and advantageous in coordinating swarm robots in complex tasks. With the proposed approach, robots can construct and adjust behaviours based on a repository of heuristics. Combined with online learning, robots can adapt to different environments and different types of tasks. This feature is particularly beneficial for dynamic environments and complex tasks where prior programming is often difficult.

References

1. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyper-heuristics: an emerging direction in modern search technology. In: Glover, F., Kochenberger, G.A. (eds.) Handbook of Metaheuristics, pp. 457–474. Springer, Boston (2003). https://doi.org/10.1007/0-306-48056-5_16
2. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. **64**(12), 1695–1724 (2013)
3. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R.: Exploring hyper-heuristic methodologies with genetic programming. In: Mumford, C.L., Jain, L.C. (eds.) Computational Intelligence, vol. 1, pp. 177–201. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01799-5_6
4. Burke, E.K., Kendall, G., Soubeiga, E.: A Tabu-search hyperheuristic for timetabling and rostering. J. Heurist. **9**(6), 451–470 (2003)

5. DaCosta, L., Fialho, A., Schoenauer, M., Sebag, M.: Adaptive operator selection with dynamic multi-armed bandits. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, pp. 913–920. ACM (2008)
6. Dorigo, M., Floreano, D., Gambardella, L.M., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., et al.: Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. IEEE Robot. Autom. Mag. **20**(4), 60–71 (2013)
7. Dorigo, M., et al.: The SWARM-BOTS project. In: Şahin, E., Spears, W.M. (eds.) SR 2004. LNCS, vol. 3342, pp. 31–44. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30552-1_4
8. Dowsland, K.A., Soubeiga, E., Burke, E.: A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. Eur. J. Oper. Res. **179**(3), 759–774 (2007)
9. Levi, P., Meister, E., Van R, A., Krajnik, T., Vonasek, V., Stepan, P., Liu, W., Caparrelli, F.: A cognitive architecture for modular and self-reconfigurable robots. In: Systems Conference, pp. 465–472. IEEE (2014)
10. Lipowski, A., Lipowska, D.: Roulette-wheel selection via stochastic acceptance. Phys. A **391**(6), 2193–2196 (2012)
11. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)
12. McKnight, P.E., Najab, J.: Mann-Whitney U test. In: Corsini Encyclopedia of Psychology (2010)
13. Michel, O.: Webots: symbiosis between virtual and real mobile robots. In: Heudin, J.-C. (ed.) VW 1998. LNCS (LNAI), vol. 1434, pp. 254–263. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-68686-X_24
14. Nagavalli, S., Chakraborty, N., Sycara, K.: Automated sequencing of swarm behaviors for supervisory control of robotic swarms. In: IEEE International Conference on Robotics and Automation, pp. 2674–2681. IEEE (2017)
15. Özcan, E., Mısır, M., Kheiri, A.: Group decision making hyper-heuristics for function optimisation. In: UK Workshop on Computational Intelligence, pp. 327–333. IEEE (2013)
16. Rattadilok, P., Gaw, A., Kwan, R.S.K.: Distributed choice function hyper-heuristics for timetabling and scheduling. In: Burke, E., Trick, M. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 51–67. Springer, Heidelberg (2005). https://doi.org/10.1007/11593577_4
17. Reynolds, C.W.: Flocks, herds and schools: a distributed behavioral model. ACM SIGGRAPH Comput. Graph. **21**(4), 25–34 (1987)
18. Sabar, N.R., Ayob, M., Kendall, G., Qu, R.: A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. IEEE Trans. Cybern. **45**(2), 217–228 (2015)
19. Seng, W.L., Barca, J.C., Sekercioglu, Y.A.: Distributed formation control in cluttered environments. In: IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 1387–1392. IEEE (2013)
20. Uludag, G., Kiraz, B., Uyar, A.E., Özcan, E.: Heuristic selection in a multi-phase hybrid approach for dynamic environments. In: 2012 12th UK Workshop on Computational Intelligence (UKCI), pp. 1–8. IEEE (2012)
21. Werfel, J., Petersen, K., Nagpal, R.: Designing collective behavior in a termite-inspired robot construction team. Science **343**(6172), 754–758 (2014)
22. Yu, S., Barca, J.C.: Autonomous formation selection for ground moving multi-robot systems. In: IEEE International Conference on Advanced Intelligent Mechatronics, pp. 54–59. IEEE (2015)