Hyper-heuristic Learning for Swarm Robots

MONASH University

Shuang Yu

Faculty of Information Technology

Monash University

A thesis submitted for the degree of

Doctor of Philosophy

2019

For Daniel

Declaration

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Shuang Yu

2019

Acknowledgements

I am truly lucky to have my two supervisors, Aldeida Aleti and Andy Song. They have always been the most supportive people, and without their amazing supervision I could not have finished this research. Aldeida and Andy have not only been my advisers, mentors and friends, I am most grateful that they are both someone I can always look up to as role models. Aldeida is an amazingly competent female researcher in a male-dominated field, and her determination have always given me the strength to go on. Andy has always impressed me with his professionalism, wide range of knowledge and patience. He is a great teacher and has never run out of good advice to give me.

I am very grateful to Bernd Meyer, Alan Dorin, Minyi Li, Bala Srinivasan, Julian Garcia, Jan Carlo Barca, Ray Chen and Thomas Bochynek for their help and advice for my project.

I enjoyed so much working and spending my PhD life with my lab mates at the former Monash Swarm Robotics Lab. I would like to thank Apurva Joshi, Han Duy Phan, Mostafa Rizk, Phillip Smith, and Yathindu Rangana for all the laughs and positive energy they gave me throughout my PhD.

Finally I would like to thank my family for their unconditional love and support.

This research was supported by an Australian Government Research Training Program (RTP) Scholarship.

Abstract

As robots are increasingly used to assist in tasks undesirable for humans to do, the demand for robotic solutions to more large-scale and complex tasks similarly increases. These tasks are often well-suited to robot swarms, and therefore coordination of groups of robots is of growing importance. However, controlling a swarm of robots is challenging, as complexity and uncertainty of the environment often makes manually programming robot behaviours impractical. We address this problem by proposing a novel hyper-heuristic approach to enable decentralised coordination of robot swarms. It allows robots to create suitable sequences of actions from a collection of low-level heuristics, where each heuristic is a behavioural element. These heuristics are manually developed algorithms for compositing robot actions, and are stored in a heuristic repository in each robot. In the proposed framework, there is no centralised controller; each robot selects and applies heuristics, then performs actions found by the selected heuristics. The heuristics are evaluated by each robot individually, using an objective function that measures the contribution they have made towards the common goal of the swarm. This selection-execution-evaluation process is iterative, and robots can learn through experience acquired during the process to select better heuristics over time.

To evaluate the effectiveness of the proposed framework, we task robots with cleaning building surfaces where multiple separate surfaces exist, and complete information of the surfaces is difficult to obtain. This complex task not only requires robots to clean efficiently by distributing themselves on the surface, but also to self-assemble a bridging structure to safely move between surfaces. Several environments that are unknown to the robots are used

for experiments, including surfaces with dynamically positioned obstacles, surfaces with gaps, and combinations of the two.

The proposed framework firstly introduces a Multi-Armed Bandit (MAB) based online-learning implementation to validate the feasibility of our hyper-heuristic approach. We verify through experiments that the proposed approach effectively learns good cleaning behaviours for the robot swarm. Furthermore, the method is robust under different types of environmental setups. To further investigate and improve the effectiveness of the proposed framework, we adopted Q-learning as the learning mechanism, allowing each robot to develop its own policy of heuristic selection, according to its own experience and local observations. Superior performance and adaptability in unknown and dynamic environments can be demonstrated when compared with action-based swarm robot learning.

We improve the framework further by adding group learning mechanisms. Without these, swarm robots learn heuristics individually and make less informed collective decisions using probabilistic voting. With these mechanisms, robots improve learning efficiency by sharing knowledge with the group. Our experiments show such mechanisms are effective for both MAB-based and Q-learning based hyper-heuristics.

Finally, a state-of-the-art learning method, deep Q-learning, is incorporated into the proposed framework. It also demonstrates superior performance on heuristic learning versus action based learning, further supporting the effectiveness of the proposed framework. Additionally, deep Q-learning based hyper-heuristic outperforms other alternatives.

We conclude that the proposed decentralised hyper-heuristic learning method is effective and is suited to complex and dynamic swarm robotic tasks.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

SGD Stochastic Gradient Descent

CF Choice Function

CHeSC Cross-domain Heuristic Search Challenge

GP Genetic Programming

GR Greedy

HH Hyper-heuristic

MAB Multi-armed Bandit

RD Random Descent

SA Simulated Annealing

SR Simple Random

UCB Upper Confidence Bound

DDQN Double Deep Q-Network

DQN Deep Q-Network

DRL Deep Reinforcement Learning

MDP Markov Decision Process

RL Reinforcement Learning

FSM Finite State Machine

MRS Multi-robot System

ODE Open Dynamics Engine

SR Swarm Robotics

Chapter 1

Introduction

Robotics is a fast growing area due to the explosive development in Artificial Intelligence. For many years, robots have automated many tasks that are considered risky and undesirable for humans. Particularly, having multiple cooperative robots to tackle a problem has made robots capable of accomplishing tasks considered too difficult and large-scale not only for humans but also for single robots. A reliable and capable robot swarm can gain significant advantage in various domains such as mining, agriculture, smart cities and even military applications. This thesis focuses on 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 large scaled navigation and coverage. In a robot swarm, collective behaviours emerge from local control rules and local interactions. Due to this decentralised nature of swarm robotic systems, they have the benefits of being robust, scalable and flexible [12].

In order to enjoy the benefits of swarm robotic systems, all robots need to operate collaboratively to achieve a common goal. Manually developing control strategies for each robot can be feasible for simple tasks and small groups, but quickly becomes difficult and ineffective when the task becomes complex or the group becomes large. Control strategies can also be automatically generated, for instance, using evolutionary computing and reinforcement learn-

ing, and are able to tackle complex problems such as multi-robot foraging [79]. However, these methods are not able to deal with dynamic or unknown environments, which are usually what robots encounter in real-world missions [89]. Moreover, these algorithms need to be re-designed and re-tuned when the task or operating environment changes. Alternatively, swarm robots can utilise a set of behavioural elements where each element by itself may not solve the problem, but appropriate compositions of elements can. This idea aligns with the hyper-heuristic approach that automatically constructs a strategy from a set of heuristics to solve a problem, rather than manually developing strategies that search directly for solutions. In literature, the hyper-heuristic approach has been shown to be more adaptive and generalises better for a range of applications [82].

Therefore in this thesis we explore *how to use the hyper-heuristic approach in decentralised multi-robot systems, in order to allow robots to better cope with unknown and dynamic environments*. This question is primarily addressed in Chapter 4, where a novel swarm robot hyper-heuristic framework is proposed for automatically sequencing behaviours to fulfil task objectives in unknown and dynamic environments. 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 algorithm is verified in the application of building facade cleaning where multiple surfaces exist, and no map of the surfaces is available. Self-assembling robots have great advantage in this scenario, because they have the flexibility to assemble into a larger structure which can move between different surfaces, and then scatter on the destination surface to clean efficiently without human intervention. However, self-assembling behaviours are complex and hard to learn. Existing coordination and control algorithms for self-assembling robots are not able to deal with this complex scenario [28, 96]. Our proposed

hyper-heuristic algorithm fills this gap, and as verified from real-physics 3D simulations and experiments, the effectiveness and robustness in the application has been demonstrated.

The hyper-heuristic framework [36] is extended to facilitate group learning through local robot communications. Comparisons of various approaches including collective decision making and advising, etc. are conducted in Chapter 4, and the pros and cons of the methods are also described. An adaptable hyper-heuristic framework for swarm robots should accommodate a variety of learning algorithms in order to improve the construction of heuristics over time, for example, reinforcement learning algorithms, which are widely used in multi-robot systems and are the state-of-the-art for many applications. Novel hyper-heuristic algorithms that integrate the reinforcement learning and deep reinforcement learning of heuristics are proposed in Chapters 5 and 6 respectively, and the simulations show the significant improvement of performance over time, as well as the advantages over the conventional action-based reinforcement learning.

1.1 Research Questions

In decentralised multi-robot systems, a robot's control commands are generated by the robot's onboard controller. Control actions, such as *move with a given velocity*, *turn for a given period of time* and *slow down*, are determined by the robot controller at every time step, and the actions are given to the corresponding hardware to execute. Robot control algorithms exist to select actions according to certain heuristics [77]. Robots receive feedback from local sensor readings and local communication messages between robots. Heuristics generate actions according to that feedback and allow a group of robots to exhibit collective behaviours. For instance, the classic flocking rules for artificial agents [107] make up a heuristic, which allows a swarm of agents to collectively perform the flocking behaviour seen in a flock of migrating birds and a travelling school of fish.

When robots are required to perform complex tasks in unknown or dynamic environments, multiple behaviours may be needed to achieve the task objective. While one can manually design robot control algorithms that generate a sequence of actions to solve a problem, an alternative approach is to search for appropriate heuristics that in turn generate a sequence of actions to solve the problem. This is the concept of the hyper-heuristic approach.

This thesis focuses on the utilisation of the hyper-heuristic approach in swarm robot control and coordination, and aims to answer the following questions:

1. **How can hyper-heuristic approaches be formulated to coordinate swarm robots without centralised control, so swarm robots can better cope with unknown or dynamic environments?**

   To further explain, hyper-heuristic methods rely on a repository of basic heuristics, which in this thesis is denoted as H. Through iteratively selecting the heuristic to be applied, hyper-heuristic algorithms aim to construct a sequence of heuristics that solves given problems. On the other hand, given a task T, a group of decentralised robots U are required to cooperate and fulfil the task in environments that are unknown and can be dynamic while the robots undertake the task. Unknown environments are the ones without a map given before performing the task, and the environments that may change in condition or layout are considered dynamic. Robots receive feedback from onboard sensors, and an objective function $f(t)$ measures the quality of how well the task has been solved. Existing swarm robot control methods cannot automatically construct effective decentralised strategies for robots to achieve complex objectives in the dynamic environments defined above. This thesis aims to develop a hyper-heuristic approach that allows a swarm of robots operating in unknown and dynamic environments to autonomously construct a sequence of heuristics from H that maximise the objective $f(t)$.

2. **How can the learning of heuristics be improved by introducing different learning strategies?**

   The hyper-heuristic approach for swarm robots needs to be able to accommodate learning of heuristics, because it is beneficial for dealing with unknown and dynamic environments.

   Robots in a decentralised swarm can learn to construct a sequence of heuristics independently using local observations and feedback. Unlike single robot systems, there is a lot of redundancy in terms of observations and functionality in the group, as well as different knowledge learned by different robots. Communication of knowledge with reachable robots might further improve the group performance. This leads us to the sub-question: *How does the hyper-heuristic framework facilitate group learning and improve the swarm performance*? It should also be noted that the communication capacity between robots is not unlimited, the communication range is not infinitely wide, and communication is not instant. These constraints makes any means of emulating a centralised learning strategy through massively sharing knowledge and experience an infeasible solution.

   Additionally, different learning approaches can benefit the hyper-heuristic framework in different aspects, such as learning speed, task performance, scalability, etc. One of the most widely used methods in robotics is reinforcement learning, where robots iteratively learn to effectively improve their policy of selecting actions [3], and it has not been applied on selecting heuristics for robots. We therefore also seek to address the sub-question: *How can the hyper-heuristic framework integrate reinforcement learning of heuristics to further improve the swarm task performance in a decentralised manner?*

3. **How can state-of-the-art deep Q-learning be integrated to further improve the hyper-heuristic learning?**

In recent years, deep learning has become one of the most widely used methods in many applications [144, 70, 67], because they are effective function approximators. Deep learning would be beneficial for researchers if the hyper-heuristic framework can integrate the method and use it to improve the learning performance of swarm robots. Deep Q-learning has achieved state-of-the-art performance in various fields such as robotics, board games (chess and Go) and video gaming. How can deep Q-learning be used to improve the learning of heuristics?

1.2 Thesis Structure

The rest of the thesis is organised as follows.

Chapter 2 introduces hyper-heuristic methods and decentralised multi-robot systems, which are the focus of this thesis. We identify the gap in knowledge which hyper-heuristic methods can address in swarm robot coordination and control. Machine learning in hyper-heuristics is also discussed to provide background for proposed methods in addressing research questions.

Chapter 3 introduces the simulation platform and the decentralised robots used for verifying the proposed algorithms.

Chapter 4 answers the core research question of how the hyper-heuristic methodology can be used to coordinate swarm robots by proposing a novel decentralised multi-robot hyper-heuristic framework. A multi-armed bandit based online learning implementation is proposed and validated through simulations.

This chapter also discusses how robot group learning strategies that utilise local robot communication can be improvements to the proposed MAB-based hyper-heuristic framework.

Chapter 5 introduces a Q-learning hyper-heuristic approach to further improve upon the MAB-based hyper-heuristic framework.

Chapter 6 shows that the deep Q-learning can also be incorporated under the proposed hyper-heuristic framework and improve the learning of heuristics.

Chapter 7 gives further analysis of the hyper-heuristic algorithms, from the aspects of the explainable policies and scalability.

Chapter 8 gives a summary of contributions and future directions.

1.3 Research Output

This section lists the publications produced in this research, and the list of contributions to knowledge.

1.3.1 Publications

The publications are listed below:

1. Yu, Shuang, Aldeida Aleti, Jan Carlo Barca, and Andy Song. “Hyper-heuristic online learning for self-assembling swarm robots.” In International Conference on Computational Science, pp. 167-180. Springer, Cham, 2018.

2. Yu, Shuang, Andy Song, and Aldeida Aleti. “Collective Hyper-heuristics for Self-assembling Robot Behaviours.” In Pacific Rim International Conference on Artificial Intelligence, pp. 499-507. Springer, Cham, 2018.

3. Yu, Shuang, Andy Song, and Aldeida Aleti. "A Study on Online Hyper-heuristic Learning for Swarm Robots." In 2019 IEEE Congress on Evolutionary Computation (CEC), pp. 2721-2728. IEEE, 2019.

1.3.2 Contributions

These are the contributions to knowledge produced in this research:

1. We proposed a novel hyper-heuristic framework for decentralised multi-robot systems. We instantiated the framework incorporating online-learning of heuristics through a variety of mechanisms. Through experiments we showed that the framework is effective in coordinating robots in unknown and dynamic environments of different scenarios. Our study has shown learning heuristics to be more effective than learning actions in robot swarm coordination. So the hyper-heuristic framework is beneficial. In addition, the framework can provide solutions that are more explainable. The framework can also be scalable for tasks of different sizes.

2. We proposed a novel integration of MAB-based learning, Q-learning and deep Q-learning into the hyper-heuristic framework, which allows for reinforcement learning of heuristics for use with decentralised swarm robots. Through comparing the performance with traditional reinforcement learning in decentralised multi-robot control, the results demonstrated the effectiveness and advantages of the learning of heuristics using reinforcement learning.

3. We proposed multiple group learning strategies and integrated them into the swarm robot hyper-heuristic framework. This includes a *voting-based group decision making* strategy, a *MAX-SUM*-based strategy and a *MAX-MIN*-based strategy for the online hyper-heuristic; and an importance advising mechanism for the Q-learning hyper-heuristic framework. Through simulations and comparative study of the different strategies, we demonstrated the benefit of group learning and collaboration in robot swarms through local communication between robots.

Chapter 2

Related Work

The aim of this thesis is to establish a hyper-heuristic framework for decentralised robot swarm coordination. Hence we review the most relevant fields which are the foundation of this thesis: heuristics and hyper-heuristics; robots and multi-robot coordination; learning in hyper-heuristics, reinforcement learning and deep reinforcement learning. The latter three are the basis of our learning mechanism for the proposed framework. There are many relevant fields that are connected to this study, including swarm intelligence, optimisation, prediction etc., but these are not the focus of our research, however details of these fields that are relevant to our research can be found in the references [49, 6, 60].

2.1 Heuristics

The term "heuristic" as it is commonly used can be traced to a paper written in the French language in 1960, where the word itself is derived from the Greek language and means "to find or discover" [105]. In more modern usage, heuristics are "simple algorithmic process models" which have often been described as "rule[s] of thumb" [44], or educated guesses. They are usually estimates, and may not be the exact descriptions of the problem or solution, but they utilise information and knowledge in the problem domain. Using heuristics can help

to quickly find satisfactory solutions when the problem is too complex for exact optimal solutions to be found [125]. In solving real-world problems, heuristic methods are especially advantageous because what we are truly optimising is a model of the real-world, and many would prefer approximate solutions of superior models than exact solutions of inferior ones [105], and heuristics make it possible for satisfactory solutions of many complex models to be found.

Heuristics have been widely used in search, decision making, planning etc., and have been a crucial part of AI since a very early stage. A classic example of the use of heuristics in search is the $A*$ search algorithm. It finds the optimal path using an evaluation function $f = g + h$ where g is the current actual minimum cost of the path from the starting node and h is a heuristic to estimate the cost of the optimal path to the goal node. The heuristic estimation can be the Euclidean distance from the current node to the goal node, the Manhattan distance from the current node to the goal node, or other estimations according to the specific problem and scenario. These estimations are not guaranteed to exactly describe the actual cost, but are both informed approximations built upon prior knowledge of the problem, which can help the algorithm to yield satisfactory results. *Local search* is another heuristic search method, where the heuristic is that a solution can be iteratively improved by searching the neighbourhood of a current solution. The method was first created to solve the *Travelling Salesman Problem* (TSP) (the details of this problem are described in [2]), and has been generalised to solve a wide range of optimisation problems. Local search starts from an initial solution, and at each step evaluates solution(s) that can be reached by making a move from the current solution using an evaluation function. *Hill-climbing* in local search always selects the neighbouring solution that has been determined to be the best by the evaluation function, and this neighbouring solution will become the current solution [104]. This Greedy heuristic is the assumption that the global optima can be found by always choosing the local optima. However this is not always the case: greedy search can become trapped at local

optima where it will be unable to find the global optima. This can be solved by adding randomness in the heuristic, where if a candidate solution is better than the current one, it will only be selected with a probability [21].

Some researchers use the term "meta-heuristics" to refer to a collection of sophisticated heuristic methods, including Simulated Annealing (SA), Genetic Algorithms (GA) and Ant Colony Optimisation (ACO) [111]. Meta-heuristic methods are originally defined as "solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space" [38]. For example, the heuristic of genetic algorithms is that solutions can be improved by applying the GA operators such as crossover, mutation and selection.

Heuristics in heuristic search and meta-heuristic search are often manually developed from experience and require domain knowledge. Heuristics can be implemented as algorithms to be applied to solve problems.

2.2 Hyper-heuristics

Hyper-heuristics are "heuristics to choose heuristics" [111]. Hyper-heuristics can also refer to constructing heuristics (generative hyper-heuristics), where a typical example is genetic programming as the hyper-heuristic [14], but that is not the focus of our study. This study focuses on the selection of heuristics.

This study discusses the search for heuristics, not heuristic search. In heuristic search, a heuristic is used to assist in the search for solutions, and is usually manually constructed. In hyper-heuristics, we automatically construct and select heuristics.

Hyper-heuristics differ from meta-heuristics in the sense that meta-heuristics search in the solution space, while hyper-heuristics operate on a higher level and search in the space of

heuristics. Hyper-heuristics can employ meta-heuristic methods as options for generating solutions.

To date, hyper-heuristics have been successfully applied in a large number of software problem domains, for example educational timetabling [17, 16, 100], bin-packing [112, 73], production scheduling [109], vehicle routing [37] and general video game playing [11], etc. In solving these complex problems, hyper-heuristic methods have shown advantages in generality and efficiency in algorithm design.

Burke et al. [13] defined a hyper-heuristic as "an automated methodology for selecting or generating heuristics to solve computational search problems". The term "hyper-heuristics" was first introduced in [22] as an approach to manage the choice of "lower-level" heuristics at each decision point. They pointed out that heuristics and meta-heuristics are developed for particular problems and are not generalisable to other problems, or even other instances of the same or similar problems. However, hyper-heuristics operate at a higher abstraction level, and use little or no domain-knowledge - only a performance measure as feedback, therefore are able to handle a wider range of problems than heuristics and meta-heuristics. Another advantage is that while heuristics and meta-heuristics tend to be knowledge-rich, and require expertise in the problem domain to find good solutions, therefore are hard to implement, hyper-heuristics only require simple, easy-to-implement, and knowledge-poor heuristics [22]. Through a complex scheduling problem, Cowling et al. [22] demonstrated and compared a few hyper-heuristic methods for iteratively selecting low-level heuristics and results show that the hyper-heuristic solutions can be better than those generated from knowledge-rich heuristics.

A template for hyper-heuristic algorithms has been given in Cross-domain Heuristic Search Challenge (CHeSC) [95], a competition for hyper-heuristic algorithms. Figure 2.1 shows the elements and workflow of a hyper-heuristic algorithm. For a given problem domain, the set of low level heuristics $\{H_1, H_2 ... H_n\}$ is predefined. The set of heuristics

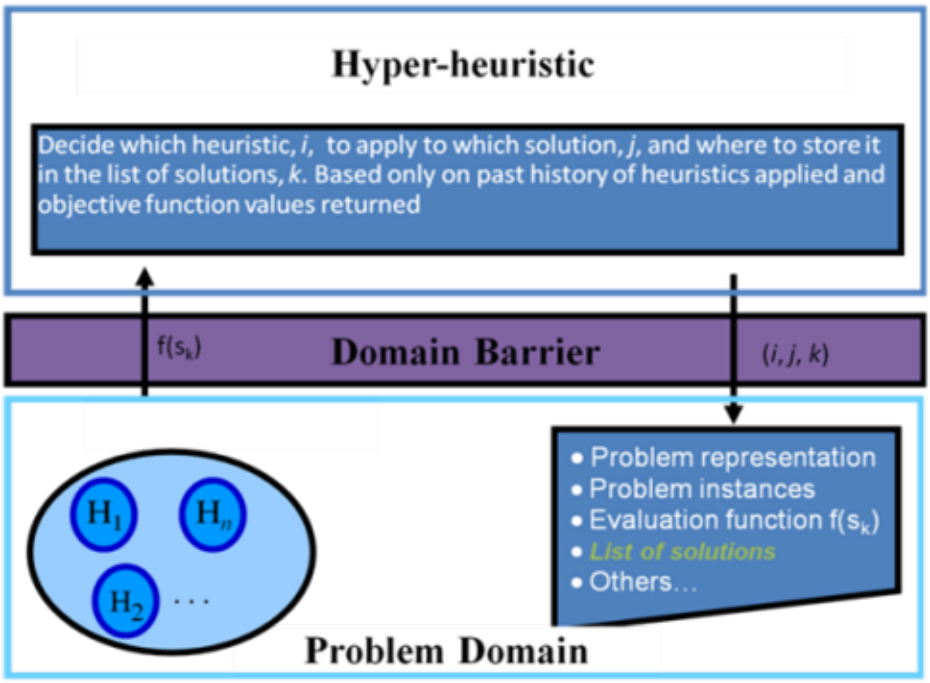


Fig. 2.1 The hyper-heuristic layer and the problem domain layer [36]

can be different for different problem domains. At each time interval, the applied heuristic generates a solution, and it is evaluated with the evaluation function $f(S_k)$ where k is where the solution is stored in the list of solutions. Hyper-heuristic algorithms are placed at the top layer of the graph in Figure 2.1, because it is a high level strategy for the book-keeping and decision making of heuristics, and remains the same for all problem domains. The hyper-heuristic algorithm only receives the objective function value $f(S_k)$ and the applied heuristics as feedback. Though the logic of hyper-heuristic algorithms varies, they all output three values: the selected heuristic to be applied in the next time interval i, the solution generated by the applied heuristic j and the place to store the solution k. Since the hyper-heuristic layer incorporates no domain knowledge, this property of hyper-heuristic algorithms allows them to break the domain barrier. When switching from one problem domain to another, only the bottom layer in Figure 2.1 needs to be replaced, and the core layer, which is the hyper-heuristic layer, is still functional without the need to change.

Hyper-heuristic algorithms can involve multiple search processes which are distributed in a complex setting, for example, a multi-agent environment. Biazzini et al. [7] presented distributed hyper-heuristics where all the heuristics in the repository are population-based algorithms and the hyper-heuristic layer algorithms are laws to choose the heuristics to be applied to the problem. Their proposed hyper-heuristics were successfully applied to a range of real parameter optimisation scenarios in a distributed environment. Another hyper-heuristic framework that involves multiple agents is mentioned in [98] where there are multiple "Low-level Heuristic Agents" and a "Hyper-Heuristic Agent". The "Low-level Heuristic Agents" searches cooperatively in the solution space and the "Hyper-Heuristic Agent" is in charge of selecting the low level heuristic to be applied at the next decision point. This has been verified on a flow shop scheduling problem. Ozcan et al. [99] have shown that using group decision making strategies such as voting as the move acceptance mechanism can improve the overall performance of heuristic selection. We will expand on the learning aspect of heuristics in the next section.

2.3 Hyper-heuristics Learning

Learning has been widely used in hyper-heuristics, where the previous performance of the heuristics is used as feedback to select and generate heuristics. Both online and offline learning have been used in hyper-heuristic methods, particularly selection perturbative hyper-heuristics, where low-level heuristics are applied iteratively to improve the initial solution to a problem [101]. Online learning hyper-heuristics learn as the problem is being solved, and offline learning hyper-heuristics learn from training data and generalise to unseen problem instances.

Many hyper-heuristic algorithms use online learning because it makes the algorithm suited to a range of problems and scenarios, especially problems with unseen and dynamic environments. A typical online learning hyper-heuristic method comprises two phases:

heuristic selection and move acceptance, as identified in [8]. At each step or decision point, the hyper-heuristic controller uses a heuristic selection method to choose and apply a heuristic from the given set of heuristics, then the second phase of hyper-heuristic uses a "move acceptance" criteria to determine if the move should be accepted. If the move is accepted, the problem is advanced to the state where chosen the heuristic has been applied, and if the move is rejected, the problem gets reversed to the state before the heuristic was applied. The heuristic selection method in hyper-heuristic algorithms can be Simple Random (SR) selection, where at each decision point a heuristic is randomly selected from the set until the termination criteria is met; in other words, until the solution is "good enough". Random Descent (RD) selection method randomly chooses a heuristic, and it is applied until no improvement in the objective function is observed. Greedy (GR) selection is an approach where the heuristic with the best improvement in objective value will always be selected [22]. Tabu search has also been used to select heuristics, as mentioned in [29, 48], where a tabu list is kept to store heuristics that are prohibited for a predetermined number of iterations. By forcing exploration, this method allows the hyper-heuristic to avoid cycling between small sets of solutions and getting trapped at local optima. A Dynamic Multi-armed Bandit selection mechanism is used to select heuristic at each iteration in [116], where the likelihood of selecting a heuristic is proportional to its average reward obtained so far and inversely proportional to the number of times the heuristic has been applied. The method has been verified on exam timetabling and dynamic vehicle routing problems.

Choice function selection method has been found to have superior performance in many problem domains, for example, nurse scheduling [120], scheduling a sales summit [22] and timetabling [103]. This method iteratively ranks the heuristics by giving each of them a score. In [103], a choice function is as calculated as:

$$f = \sum(\alpha f_1 + \beta f_2 + \gamma f_3) \tag{2.1}$$

where f_1 is the individual performance of each heuristic, f_2 is the joint performance of pairs of heuristics, and f_3 is the amount of time elapsed since each heuristic was last applied. At each decision point, the heuristic with the highest score is chosen. α, β and γ are parameters to be tuned, and [23] proposed a parameter-free choice function which utilised substantial domain knowledge. Both tabu search and choice function have been combined with reinforcement learning. [15] introduced a simple reward and punishment reinforcement learning component to determine the rank of each heuristic, as well as which heuristic will be include in the tabu list. The method was verified on a nurse timetabling and scheduling problem. [92] applied positive and negative reinforcement on heuristic utility values to find the expected benefit of choosing a heuristic, and at each decision point, a softmax choice function is used for selection, where the probability of selecting a heuristic is proportional to its utility value.

There are also offline learning hyper-heuristics, and a widely used technique is Genetic Programming (GP) [110, 14], which is used to generate new heuristics for a problem. This approach has achieved good results in a range of problems such as the Boolean satisfiability problem and bin-packing. However, the drawback is that the heuristic generated by each Genetic Programming run is different, and it requires a few runs to determine the quality of the heuristics the GP hyper-heuristic can produce. Moreover, a framework for evolving heuristics needs to be developed for the problem, and different frameworks yield different results. This lack of robustness means that it would take a long training process to make the generated heuristic reliable for robotic systems.

Learning of heuristics has shown advantage in dealing with complex problems and dynamic environments. Tavares et al. [129] has demonstrated through both theory and experiments that learning over heuristics is faster, and can give superior performance when the space and action spaces are large, when compared with using the same learning method over actions. Although in the long run, learning over actions can achieve optimal results, but learning over heuristics can give sub-optimal solutions within a reasonable amount of

time, which is typical for complex problems [75]. Uludag et al. [133] uses online learning to assist hybridising Estimation Distribution Algorithms, and has shown that the hyper-heuristic method has superior performance in dynamic and cyclic environments.

2.4 Swarm Robot Coordination

The aim of the thesis is to propose a hyper-heuristic robotics coordination mechanism, therefore the relevant literature is reviewed here. Single robot control, multi-robot systems etc. are wide areas but not the focus of this study. Therefore we review studies relevant to swarm robots which are decentralised multi-robot systems. These types of systems are suitable for complex environments where centralised control is not possible or difficult.

In [12], the characteristics of swarm robots are described as follows:

- Robots make local observations and communications,
- No robot has access to centralised control or global knowledge,
- Robots are fully cooperative, and
- Robots are fully autonomous, with no human intervention at any stage of performing the task.

We explain these traits by comparing decentralised controlled robots to centralised controlled ones. A centralised multi-robot control approach requires a control station or a single robot that holds the global knowledge, makes control decisions and gives commands to all robots. Examples of such systems include the centralised formation control framework in [52], centralised sensing and control for robots with no on-board processing capability [54], and the centralised multi-robot path planning algorithm in [76] which guarantees completeness of solutions. Although centralised control of multi-robot systems has the advantage of allowing individual robots to have very simple hardware, and can find complete solutions

for problems because the global knowledge is accessible, in practice decentralised control approaches have proven much more favourable for multi-robot systems. In a decentralised multi-robot system, no robot controls the whole system. Most multi-robot systems use decentralised control because it allows them to be robust and scalable. Unlike centralised systems where the failure of the central controller will cause the whole system to fail, the loss or failure of individuals is often not devastating for a decentralised robot system, therefore it is robust [117]. Centralised control is not a scalable method because the complexity of the control algorithm can grow massively with the size of the group, and control algorithms for large groups could become too hard for humans to design. On the other hand, decentralised robot systems are modular, and the group behaviour emerges from local control laws. This allows decentralised robot control algorithms to remain simple and easy to design for robot groups of different scales [12]. Another deciding factor for choosing decentralised control in multi-robot applications is that in many real-world tasks, global knowledge is too hard or impossible to obtain for any robot or control centre. For instance, in the application of multi-surface facade cleaning on a large and obstructed building surface, not all robots are accessible for control and communication, each robot only observes from its local sensor readings, thus centralised control and coordination are unfeasible and costly. In these scenarios, each robot is required to be able to act on its own, and more importantly learn on its own in order to deal with the dynamic and unknown environments. In summary, to solve some problems in the real-world scenarios, in particular multi-surface cleaning, a decentralised multi-robot system is required.

To coordinate and control decentralised multi-robot systems, there are holistic approaches, which involve manually designing and synthesising more sophisticated local control laws to solve more complex problems; and behaviour-based approaches, where complex problems are solved by compositing behavioural elements.

Holistic approaches rely on humans to design local control laws for every robot which give rise to overall system dynamics, and allow the robot group to exhibit desired swarm behaviours [89]. The design of swarm behaviours usually takes inspiration from biological systems in nature and laws of physics [12]. For instance, rule-based algorithms allow complex and collective behaviours to emerge from local behavioural rules. A well-known rule-based swarm behaviour is flocking [107], which roughly imitates the group flight behaviour of birds in nature. Since each bird cannot pay attention to every other flockmate, a simulated *boid* is similarly constrained, and follows a set of local rules in order of decreasing precedence:

1. Collision avoidance: avoid collisions with nearby flockmates.
2. Velocity matching: attempt to match velocity with nearby flockmates.
3. Flock centring: attempt to stay close to nearby flockmates.

With each boid practising these rules, they flock as a scalable flock, and split apart to go around an obstacle. Werkel et al. [138] designed sets of movement rules from observation of the motions of mound-building termites. When the rules are executed by each robot in a decentralised manner, the group collectively constructs buildings of desired structures like ants. Apart from rule-based algorithms, Finite State Machine (FSM) is also a way to design swarm robot controllers, for example, Soysal and Sahin [121] designed a FSM with probabilistic transitions between four states which allows swarm robots to display the aggregation behaviour observed in social insects and mammals. Physics-based control methods for swarm robots are inspired from natural physics laws that utilise virtual forces, potential fields etc. [53, 122]. These methods allow robots to avoid collision and collectively form patterns. Either inspired from biological or physical systems in nature, the design of such swarm robot control methods require long times of observing natural systems, followed by manual and iterative development and modification to the algorithm to achieve the desired behaviours.

These rule-based methods or manually developed methods are good for simple tasks, but can become too difficult for humans to design and tune for complex problems. There are holistic approaches that can automatically generate a control strategy for a problem, and they can be classified into two categories: evolutionary robotics and multi-robot reinforcement learning [12].

Evolutionary robotics [94] uses evolutionary computing techniques to iteratively evolve the parameters of the controllers, for example, to evolve parameters for virtual force law based controllers [41] and to evolve parameters for neural networks that act as controllers [1, 31, 123]. These evolutionary algorithms allow robots to generate desired behaviours, such as obstacle avoidance [41], pattern formation [31] and self-assembly [1]. The evolutionary methods have a few drawbacks: they are essentially black-boxes and the generated controllers are difficult to understand; they are computationally intense and require a large number of iterations in simulation to generate behaviours, and the complexity of the resulting behaviours can often be achieved by designing them by hand [12].

Multi-robot reinforcement learning has achieved a lot of success in developing good policies to solve problems, however most of the the work has been focused on centralised multi-robot systems [27], robots with shared memory or observations [140], or systems where there exists an "observer" that has access to the global states [45]. These methods are not applicable to decentralised multi-robot systems. For completely decentralised robotic systems, reinforcement learning states are representations of the local sensor observations, rewards for reinforcement learning are local rewards obtained by each robot, and the action space is made up of individual robot control actions instead of the joint action space for all robots. However, most existing multi-robot reinforcement learning algorithms are not fully decentralised, instead using the "centralised training, decentralised execution" paradigm [72]. However, in real-world robotics applications, the simulation model or platform is not always available for centralised training.

On top of not having much success with generating complex behaviours, the automatic methods are specifically evolved or trained for a particular task and environment. They cannot deal with change in the environment and fail to generalise to unseen scenarios.

In summary, holistic approaches for multiple robots do not deal well with increasing complexity of the task, especially with unknown or dynamic environments. Designing such algorithms can become too hard for humans. On the other hand, with behaviour-based methodologies, different robot behaviours can contribute to solving different problems or different parts of a problem. When the problem becomes complex, one can construct strategies from compositing behaviours instead of manually developing more and more complex control strategies.

In the next section, we will review methods that achieve objectives by selecting from a set of behaviours and sequencing them to solve problems.

2.4.1 Behaviour-based Methods

Behaviour-based methods are often built on a set of already established robot behaviours, especially the behaviours that contain rich domain knowledge.

For a single robot, there are approaches that use behaviour tree [78], behaviour network [93] etc. to solve complex problems by sequencing simple behaviours. With online learning, these algorithms can solve problems in dynamic environments, but they are tailor-designed for each problem, not directly transferable to other tasks or multiple robots.

Hoff et al. [43] compared a gradient-based foraging behaviour, an area-sweeping foraging behaviour, and a behaviour switching algorithm that adaptively switches between the two foraging behaviours. All three behaviours are on a colony level which means that all robots collectively exhibit a behaviour, and the behaviour switching is done by keeping track of a decreasing value synchronised through broadcasting. Although the individual behaviours have shown advantages in some scenarios, the behaviour switching algorithm has the highest

overall success rates across a range of environmental layouts. This shows that compositing two foraging behaviours makes the swarm more adaptive and the control method more generalisable. In [89] Nagavalli et al. proposed the problem of swarm behaviour sequencing, which aims to find the sequence of swarm behaviours for robots to collectively perform that would maximise the objective function value. A best-first search method has been shown to autonomously generate behaviour sequences for a human operator to execute in a known and static environment. When the environment changes, it relies on the human to manually update the algorithm with new environmental information and calculate new solutions, and lacks the ability to automatically adapt to changes in the environment.

Behaviour-based methods have been widely used in self-reconfigurable robot systems. For example, CONRO used a hormone based approach, which propagates hormone messages that contain its current behaviour and path. When a module receives a hormone message, it checks through a set of decision rules to select a behaviour. Since a module's behaviour is only selected according to its neighbour's behaviour, this method is robust to configuration changes. It is also scalable up to large systems because there are no global identifiers [118]. The rule-based algorithms are limited however by not being adaptive. CONRO robots are not able to deal with navigation in a dynamic or unknown environment, unless the rule set can be updated or evolved, and shared with the whole swarm. A more adaptive hormone-inspired behaviour switching method is described in [62], which allows a self-reconfigurable swarm to autonomously switch between "swarming" behaviour and "modular robot". This work uses the concentration of a virtual "hormone" inside a robot to switch the behaviour of a robot, for instance, the high level of a hormone called "impatience" in a robot would result in it trying to assemble into a "Snakebot" to climb across the wall to explore another area. Similarly, low level of "impatience" will trigger a behaviour to disassemble and individually explore the space. The hormone level in a robot is influenced by the environment and physical constraints. In this project, the authors used XML-based Genetic Programming to evolve the

selection of the impatience value. The algorithm successfully assembled the swarm into a larger organism that was capable of climbing over obstacles, however the research has found that the disassembly process could not be handled well by the hormone method. Moreover, the hormone method with "impatience" can only be applied to exploration tasks, and is not generalisable to other tasks.

While the above attempts have been made to construct a strategy on the basis of behavioural elements, they all lack the ability to adapt in unknown and dynamic environments. Moreover, these approaches are all manually developed and tailored to specific applications. Hyper-heuristics, on the other hand, makes it possible to automatically construct a sequence of heuristics to solve a problem instead of manually designing algorithms for the given task. In the context of swarm robotics, this design approach can be used to automatically construct a sequence of behaviours to solve a given problem.

Current literature suggests numerous advantages the hyper-heuristic approach can bring to decentralised robot systems. Tavares et al. [129] proved in theory and experiments that selecting heuristics may not find the optimal solution, while given enough time, action learning methods can. However, learning to choose heuristics has shown advantage in dynamic environments and large-scale problems because it finds sub-optimal but satisfactory solutions much faster than methods that learn on actions. Multi-robot systems have a large action space and/or state space, and the learning speed can be the bottleneck for the feasibility of a robot system. This is because robots act in real-time and cannot afford potentially thousands of iterations of training. Training time is also associated with consumed energy and cost. Any reduction in training time can be a great benefit to multi-robot groups, especially when the number of robots is large. Hyper-heuristics have also shown effectiveness in tracking changes in the environment and can adapt quickly, as suggested by Kiraz et al. [56] in addressing dynamic optimisation issues. Furthermore, hyper-heuristic algorithms are more transparent in terms of the internal mechanics of the strategy comparing to other

automatically generated strategies, because the logic of their heuristics are fully known. This can make the hyper-heuristic generated behaviours easier to explain.

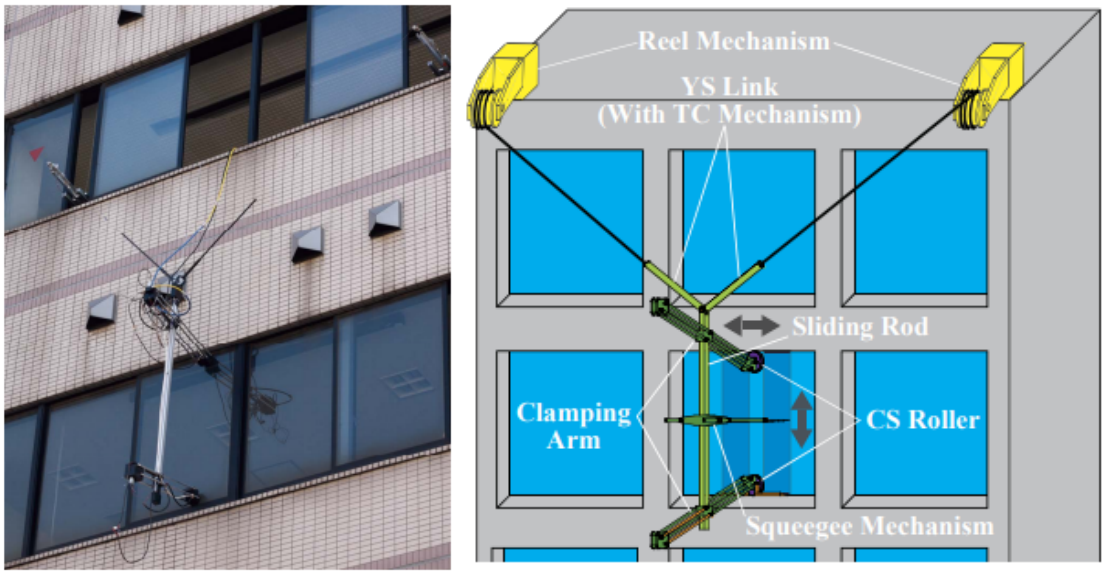
Therefore, this thesis will explore *how the hyper-heuristic approach can be used to solve swarm robot problems by sequencing swarm robot behaviours*, so that the swarm robotic coordination and control methods can enjoy the benefits of hyper-heuristics, in particular the automatic generation of heuristic sequences, and the generality to apply to unseen scenarios, because they enable robots to perform tasks in unknown and dynamic environments.

2.4.2 Surface Cleaning Robots

The swarm robotic application area for verifying the use of hyper-heuristics is cleaning the outer surface of buildings where multiple surfaces exist. This application is chosen because such environments are complex, and require self-assembling behaviours which have been particularly difficult for decentralised robots to perform.

Surface cleaning robots have already existed for more than a decade, however none of them have earned more popularity than human cleaners. This is because the existing robotic systems are able to clean individual surfaces, such as a window or a wall segment, but with serious limitations, which make them impractical in many real-world multi-surface cleaning applications.

The world's first facade cleaning robotic system was proposed in [10], and was specifically designed for the central building of Leipzig Trade Fair, Germany. It cleans the surface while hanging on the vertical surface through a trolley installed on the roof. There are elevators on both sides of the trolley to move the robot up and down in order for it to perform cleaning actions. A similar design for a semiautomatic SkyScraper facade cleaning robot is proposed in [9], where the robot is held by a wire rope and has vacuum suction legs to maintain contact with the facade surface. The robot is able to perform automatic cleaning but a human operator is needed to control and adjust the robot movement. Following these early explorations,



(a) Whole Windows Cleaning Robot (b) Robot movement

Fig. 2.2 “SkyScraper-I” [46]

autonomous facade cleaning robots were proposed. Figure 2.2(a) shows an autonomous whole-window cleaning robot that clamps onto the window frames. There are two rollers installed on the roof of the building to adjust the length of the supporting rope, and as the robot moves from one window to another, a squeegee moves up and down along the central rod to clean the window, as shown in Figure 2.2(b) [46].

Sky Cleaner 3 has a similar design, it operates on a hanging cable and a ground supporting unit, and by following a boustrophedon motion, it is able to clean glass walls with brushes and water [145]. All these robots are able to move on multiple surfaces with the help of tethering cables, so that they are able to clean the whole building. However there are serious limitations that make them impractical in cases such as tall buildings and irregularly shaped surfaces. This is because some buildings with unusual shapes require frequent repositioning of the cabling system, which can be cost-intensive and time-consuming. The cables also caused another problem, which is scalability. The existing facade cleaning robots are applicable for cleaning small building facades, but not efficient for skyscrapers, because a single robot would take too much time to clean them. The tethering system and power cables prevent the

existing facade cleaning robots from working in groups, as their support cables obstruct one another.

There are cable-free robots, usually small in size and for domestic use [69, 47, 85, 20]. They all use vacuum suction cups to adhere to window surfaces, because controlling the vacuum level only needs a simple PID controller with a pressure sensor. Rotating micro-fibre cleaning pads with no cleaning fluid are often used in these robots too, so they don't need water or drying process. A dirt sensor is also proposed in [47], an array of pairing LEDs and phototransistors are installed on the bottom of the robot. By measuring the voltage drop on the phototransistors, they can tell clean glass from dirty glass, which will reflect more IR. However it should be noted that this method does not always work because if posters or other non-transparent objects are attached to the inner surface of the window, the voltage would still drop, and cause a dirty region to be detected where there is none. These vacuum window cleaning robots are commercially successful and are popular for domestic use. HOBOT-188, for example, uses rotating micro-fibre cleaning pads to clean the glass surface. Its two cleaning pads also act as legs so that HOBOT-188 is able to move forward on a vertical surface through alternate motion of the two legs [42]. Windoro is another window cleaning robot pair that is able to clean both sides of a window since the inner and outer units are magnetic [19].

In summary, large surface cleaning robots can clean multiple surfaces autonomously, with limited feasibility on irregular shaped buildings and surfaces, while small surface cleaning robots can fit in a wider range of surfaces, but are likewise limited because they cannot move from one surface to another by themselves. To date, no surface cleaning robotics system has utilised self-assembling robots for the task.

2.4.3 Self-assembling Robots

"Self-assembling is the autonomous organization of components into pattern or structures without human intervention" [139]. There are many instances of self-assembling behaviours that exist in nature, for example, molecules self-assemble into bilayers, bacteria self-assemble colonies with different patterns, and fire ants self-assemble into rafts to survive floods [86].

The idea of self-assembly has been adapted to robotics: the first self-organising robotic system was built in 1990, named CEBOT [35]. They are modular robots with a hooking mechanism and are built to work inside tanks because of their structural flexibility. PolyBot, as described in [142], was created using similar concepts but was the first system that demonstrated a heterogeneous self-reconfigurable swarm. It is composed of two types of modules, one called a segment which has 1 DOF and 2 connection ports, and one called a node which is rigid and has 6 connection ports. PolyBot is able to self-assemble into a "snake", a rolling track, a four-legged "spider", etc. and perform the corresponding locomotion. As a benefit of having this heterogeneity, PolyBot allows for more types of topological configurations than CEBOT. Homogenous modular robots also have the potential to form heterogeneous swarms if different topological configurations are allowed. CONRO, which was built in 2004, is a chain-type homogeneous robot system that has a distributed control system. A CONRO robot has two degrees of freedom, a yaw of 60° left and 60° right, and a pitch of 90° downwards and 30° upwards. It carries two batteries, one STAMP II-SX micro-controller, two servomotors and four docking connectors. Each connector has a pair of infrared transmitters/receivers to support communication as well as docking guidance. With this hardware setup, various configurations can be formed by CONRO robots, such as a quadruped, and each of them can perform the corresponding gait control through a centralised controller on a remote host using a master-slave approach. Kilobots are self-assembling thousand-robot swarm robots that are designed to auto-generate any given complex shapes in a distributed manner [114]. While the target shape is given to all Kilobots in the form of a

bitmap, this research proposed an edge-following algorithm that each robot follows to find and move to their own location. This collective behaviour allows the swarm to form any given shape, but the shape needs to be determined by a human operator or other controllers. [91] approaches the shape generation problem by seeking inspiration from chemical reactions, where stochastic robots with different connection point layouts move randomly and robots only connect with the matching ones.

M-TRAN III are distributed modular robots that run pre-determined and open-loop programs in parallel [61]. An M-TRAN module comprises two blocks and a link which allows them to rotate ±90° about their axes. The two blocks have the same shape but different gender. A female block can allow three of its surfaces to mechanically connect to counterparts of a male block. By following a set of predefined commands such as "rotate link motors" and "remote connection", the robots change their configuration in a distributed way and are able to climb steps. Until this point, research in self-assembling robots is focused on hardware concepts and implementation, while the control aspects are simple, deterministic, and only work in certain static and known environments.

The REPLICATOR project presented in [64] has shown robots with a more sophisticated sensory system and control algorithms compared to the previous self-assembling robot systems. REPLICATOR robots can perform laser-based exploration to detect other robots, and use camera based object recognition to align with each other. Control-wise, REPLICATOR robots cycle through a series control phases, which allow robots to switch between the "swarm mode" and the "organism mode". The aim of the robots is to reach a goal point, specifically a power source. When in "swarm mode", as shown in Figure 2.4(a), the robots perform a random walk individually, map the environment and aim to detect power sources. If there is a power source that is not within reach of individual robots, the robots will make the decision to assemble and perform 3D locomotion as shown in Figure 2.4(b) to reach the power source. [71] has introduced a behaviour-based control algorithm for REPLICATOR

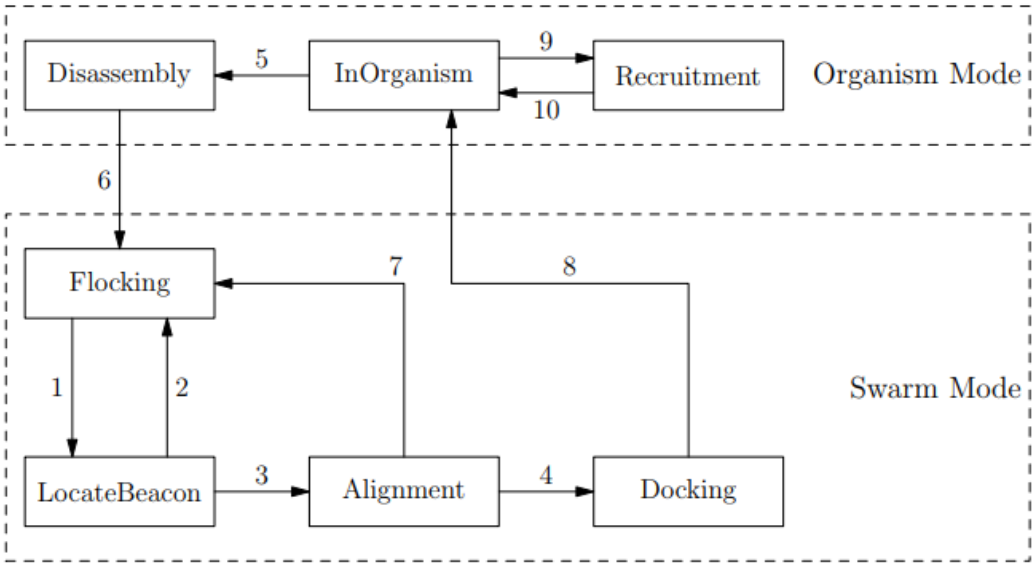
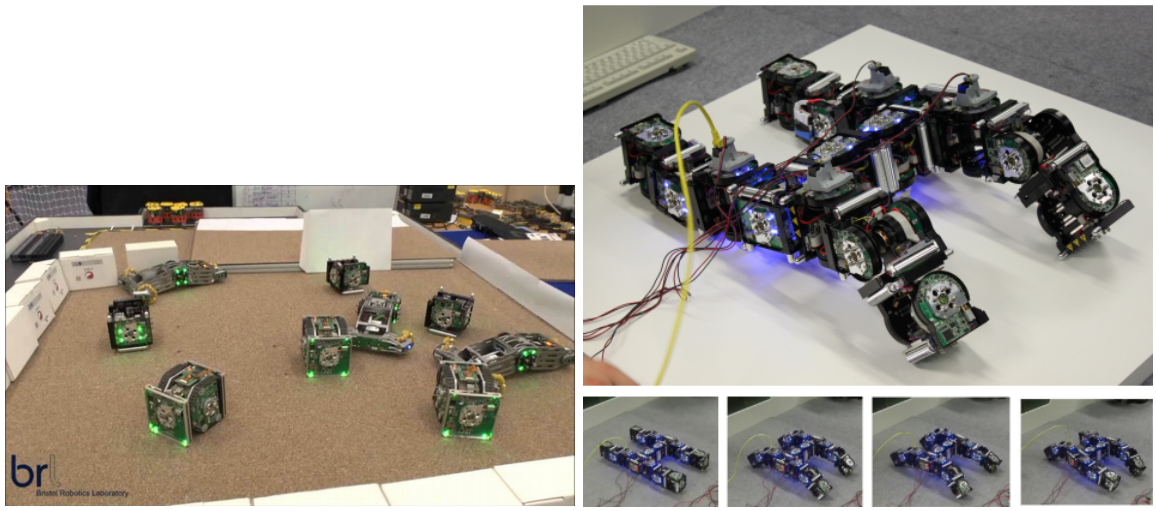


Fig. 2.3 REPLICATOR Robot finite state machine (FSM) for autonomous morphogenesis controller. Conditions causing state transitions: 1 – docking message received; 2 – collision, or no docking message received; 3 – docking beacon signals detected; 4 – aligned and ready to dock; 5 – disassembly required; 6 – undocking completed; 7 – expelling message received, or docking signals lost; 8 – docking completed; 9 – recruitment required; 10 – recruitment completed [71]

robots, and the behaviours are selected using a finite state machine (FSM). As shown in Figure 2.3, states are behaviours that facilitate assembling and disassembling of the robots, and the decision-making process for transitions 1 and 5 (to dock and to disassemble) is not autonomous. The aim of the algorithm is to assemble robots into a pre-defined configuration. The limitations of this FSM-based control are: 1. the behaviours are more like "phases" in the self-assembling process than what are commonly known as "swarm behaviours". Swarm behaviours are collective behaviours which are usually more complex and flexible; 2. robots can not autonomously choose which configuration to morph into, or when to morph, which makes the swarm limited to simple static tasks. [50, 51] summarised the REPLICATOR project, showing that in the development of this self-assembling swarm, hormone-based systems and gene-based evolutionary algorithms were used to generate pre-defined target robot configurations, sensor-motor coordination and fusion formed the basis of the modular



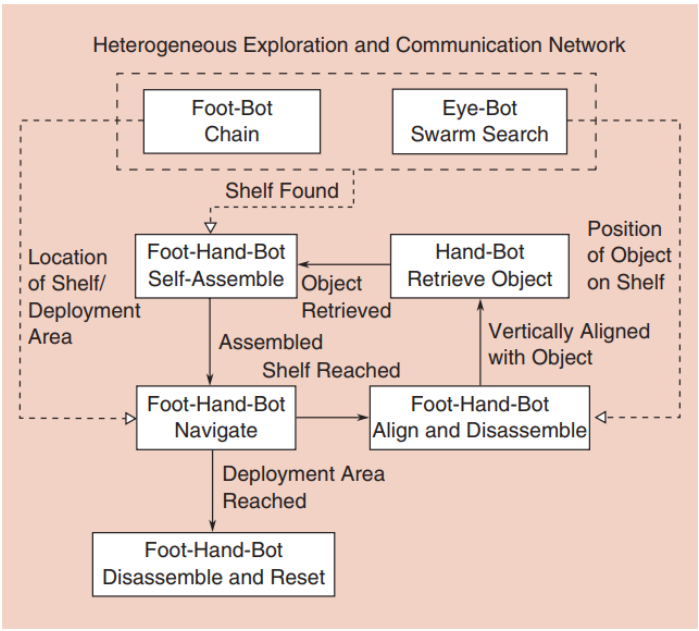
(a) Swarm mode (b) Organism mode

Fig. 2.4 Two REPLICATOR modes [64]

robotic cognitive system, and a distributed software framework was designed to achieve distributed control. The project ended with a focus on hardware assembly and locomotion control, leaving open questions, especially if modular robots can adaptively select which configuration to morph into according to the given environments and tasks.

The Swarmanoid project has partially answered that question by using behaviour networks, while the low level control and locomotion are evolved using evolutionary algorithms and neural networks. [32] first presented the hardware design and the self-assembling mechanism of a robot swarm, "swarm-bots". As shown in Figure 2.5(b), the robots are differential wheeled robots with grippers to connect to other robots. Figure 2.5(a) shows the concept design of swarm-bots connecting into a line and bridging across a gap. [31] proposed a neural-network approach to evolve robot controllers together so they cooperatively move forward as a connected organism, such as a "snake", while avoiding obstacles. [30] presented a heterogeneous swarm where foot-bots are ground-moving differential wheeled robots, hand-bots have grippers for fetching objectives and eye-bots have cameras for navigation and mapping. Figure 2.5(c) demonstrates a behavioural network that switches between different robot behaviours to achieve a book-fetching task. As can be seen, this method works even

(a) swarm-bot (b) bridge



(c) search-and-retrieval behavioural control

Fig. 2.5 The swarm-bots [32]

if the target objects and obstacles are moved, or are placed in unknown positions. This is because network conditions allow eye-bots to continuously locate the shelf and the book, so the other robots can adaptively adjust their behaviours.

In the scope of self-assembling robotic research, autonomous docking and moving as an assembled entity are well-researched problems. However, all the existing 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. This gap opens the opportunity for hyper-heuristic approach to be used to automate the process.

2.5 Reinforcement Learning

Our study utilise reinforcement learning for heuristics learning, hence relevant literature is reviewed in this section. Reinforcement learning learns through interacting with the environment. A policy is to be learned for finding the maximum long term reward [126]. A learning agent in reinforcement learning is guided by a *reward signal*, which defines the goal of the problem. The agent interacts with the *environment* to receive *rewards*, and the amount of *reward* received by the agent is influenced by the *actions* the agent takes. The goal of the reinforcement learning agent is to maximise the long term reward. Many reinforcement learning problems have been successfully modelled by Markov Decision Process (MDP), which has been the standard formalism for learning sequential decision making [135]. The elements of MDP include states, actions, state transitions and a reward function. The *environment* in reinforcement learning problems is the real-world or simulated world the agent lives in, while a *state* $s \in S$ represents a configuration of the environment that the agent can sense. A state can be an exact copy of the environmental information, for example, raw images of the Atari games used in [87], or an abstract representation of the perceived information about the environment [108].

As shown in Figure 2.6, as an agent takes an action $a \in A$, it causes state transitions to happen. At a given state, some actions are legal and some are illegal. By applying an action a to state s, the system makes a transition from s to s', the transition function is denoted as $T : S \times A \times S$, and the transition probability is $p(s'|s,a)$. The reward function R defines the reward received being in a state. A *policy* defines how the machine learning agent takes actions at a given state, and is denoted as $\pi : S \rightarrow A$. The optimal polity $\pi* : S \rightarrow A$ gives the maximum long term reward.

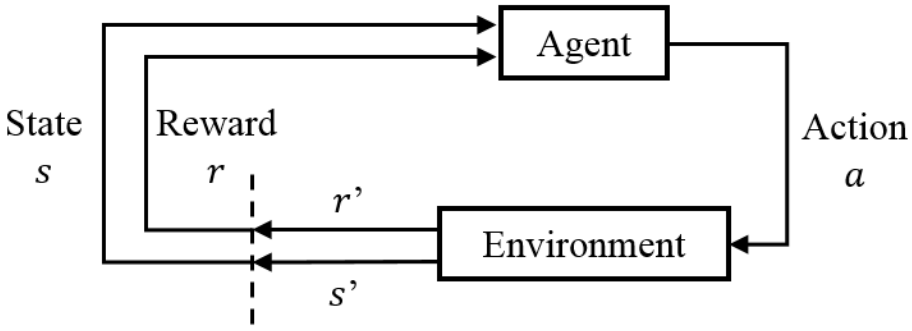


Fig. 2.6 The agent-environment interaction in reinforcement learning [126]

A value function is the expected return from a state by following a policy π, using the infinite-horizon:

$$V^{\pi}(s) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} | s_t = s\} \tag{2.2}$$

where γ is the discount factor. Q-learning [137] is an important model-free reinforcement learning algorithm, which uses the state-action value function:

$$Q^{\pi}(s,a) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} | s_t = s, a_t = a\} \tag{2.3}$$

This algorithm does not require a model of the environment; in other words, the state transition function can be unknown from the start. Q-learning relies on the agent exploring the environment, interacting with the environment, and iteratively gaining a better estimate of the state-action value function. The optimal value function is defined as:

$$Q^{*}(s,a) = E[R(s,a) + \gamma \max_{a'} Q^{*}(s',a')] \tag{2.4}$$

The optimal policy $\pi^{*}(s)$ is defined as the policy that would give the maximum long term reward, and can be found using the $Q^{*}(s,a)$ function:

$$\pi^{*}(s) = \arg\max_{a} Q(s,a) \tag{2.5}$$

Bellman equation [5] gives a recursive form of the Q-function which allows the Q-learning algorithm to iteratively find the optimal value function regardless of what policy is used in the Q-function update. Given a MDP tuple (s_t, a_t, r_{t+1}, s_{t+1}):

$$Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1},a)) \tag{2.6}$$

To explore the environment and optimise the Q-function, the machine learning agent needs to balance exploration and exploitation. A common method is using the ε-greedy algorithm for taking actions; and the Q-learning algorithm with ε-greedy is detailed in Algorithm 1 with infinite horizon. The state-action value function $Q(s,a)$ is first initialised arbitrarily. An action is taken at each step, and the action is selected according to current policy (on-policy) as shown in line 4 with probability (1-ε), otherwise the action is selected randomly. By taking an action, the agent is brought to the next state and receives reward r. In line 9, the Q-function is updated with the received reward and the state information. This process is iterative and $Q(s,a)$ becomes more and more accurate over more training iterations.

Algorithm 1 Q-learning

1: Initialise $Q(s,a)$ arbitrarily
2: **for** each step **do**
3: **if** randomNumber(0,1) $> \varepsilon$ **then**
4: $a \leftarrow \arg\max_a Q(s,a)$
5: **else**
6: Choose a random action a
7: **end if**
8: Take action a, observe r, s'
9: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a Q(s',a) - Q(s,a)]$
10: $s \leftarrow s'$
11: **end for**

Q-learning has been widely used in robotic systems, including single robots and multi-robot systems. Most explorations and successes of Q-learning have been in the domain of single robot control. For instance, a 7-DOF robot used reinforcement learning to flip

pancakes in the air and catch them with a frying pan [58], which showed that Q-learning is able to cope with highly-dynamic, real-world tasks. Q-learning has also been used widely in enabling robot mobility. [119, 57, 102] has shown that Q-learning allows legged robots to learn gait and locomotion from scratch. [59] proposed a neural controller that uses reinforcement learning to allow a wheeled robot to avoid collisions while navigating in an unknown environment with obstacles. More examples of the application of reinforcement learning on single robots include robot navigation [113, 127] and many others. However, having success in learning for single robots does not imply that the method works for multiple robots, therefore single robot Q-learning will not be extensively discussed.

Regarding multi-robot systems, Q-learning has been applied to a narrower range of applications than single robots. [27] presented a multi-robot Q-learning method to dynamically adjust the team size according to the layout of obstacles. The task objective is to achieve more coverage on a surface with obstacles, the states represent coverage efficiency, and the three actions correspond to setting a voting parameter to three different values, where the value of the voting parameter is the deciding factor of the team size. All robots perform a leader-referenced flocking behaviour and move in V-shapes of different sizes. Results show that with Q-learning for the team size, the coverage efficiency is improved by 5-10%. [72] proposed a deep reinforcement learning method to learn collision avoidance, where robots are trained on a shared policy with local observations. This approach follows the centralised training, decentralised execution paradigm. [136] compared single-agent Q-learning with Team Q-learning in a cooperative box-pushing task. The paper has shown that the total reward is higher using single robot action space instead of a joint action space for all robots, and each robot having its own policy gives a better performance.

In a decentralised setting, the environment is partially observable and non-stationary in the view of each agent. [34] has shown that with the help of local communication between robots, multilayered reinforcement learning is effective for a robot to avoid collision with

obstacles in the environment and other moving robots. The multilayered reinforcement learning approach follows a learning curriculum of four stages where the output of a previous stage is passed into the next stage as inputs. The limitation of a multilayered curriculum is that the curriculum and learning structure needs to be manually designed and it is only applicable to the intended applications, in this case collision avoidance.

[97] presented a deep reinforcement learning method to allow robots to learn from local experiences given shared rewards for all agents. Reusing knowledge in the learning process through explicit or implicit communication is a common approach to yield shorter and more reasonable training time in reinforcement learning. Agents learn from previous tasks, demonstration, imitation and so on [25]. A multi-agent deep reinforcement learning method for optimal link discovery and selection is mentioned in [140]. Each agent samples its training data and pushes them to a mutual memory pool for mutual learning. Advising methods work on asking other learning agents for advice, and avoid the need for shared memory, reward sharing, state sharing as well as large amounts of data transfer [24]. Additionally, [18] proposed a decentralised deep learning method for collision avoidance. Deep reinforcement learning in multi-robot systems will be further discussed in detail in Section 2.6.

While all the above multi-agent reinforcement learning methods operate in the space of actions, [141] proposed a multi-robot reinforcement approach that is based on learning rules to segment the continuous state-space. In this paper, each rule is for classifying a local region of the state-space into an action, and through interacting with the environment and updating the rules according to reward, the robots learn to autonomously segment the continuous state-space, and perform a cooperative carrying task. In this method, the robots learn to cooperate from continuously predicting the other robots' postures and use them as the input for reinforcement learning. This means that the number of teamed-robots needs to be known in advance, and the number can not change at any time of task execution. Moreover,

the state-space classification method in this research is not able to handle sophisticated cooperative behaviour in complex environments.

The assignment of rewards in a distributed robotic system is discussed in [79, 80], where the authors find that simple local broadcasting of sensory information and reward can greatly help distributed reinforcement learning in noisy and dynamic environments. The research was carried out using two robots, and did not show how it scales to larger swarms.

2.6 Deep Reinforcement Learning

A recent advancement in reinforcement learning is the integration of deep learning, as deep learning has boosted the field of machine learning, and achieved ground breaking results in many fields such as image processing, computer vision, games and robotics. Deep learning is a method of representation learning with multiple processing layers, typically with multilayer neural networks, to learn representations of data with multiple levels of abstraction [63]. To train the multiple layers of neural networks, backpropagation [115] is the common method, where the gradient of an objective function is computed with respect to weights in each layer, and updates weights iteratively. With sufficient data, deep neural networks are able to automatically learn features that can be better than hand-crafted feature representations.

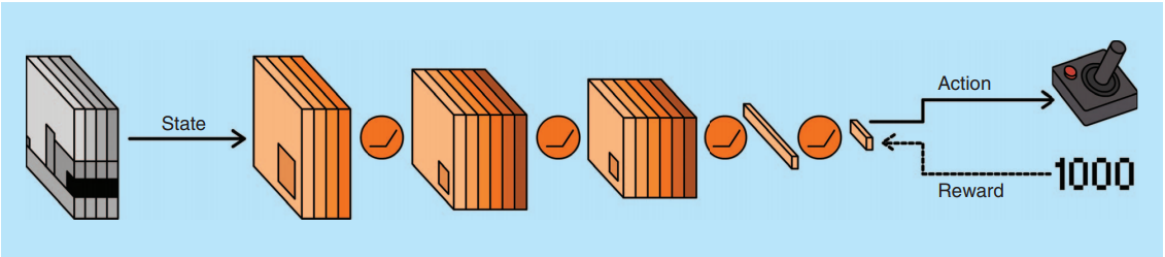


Fig. 2.7 Deep Reinforcement Learning [4]

Reinforcement learning greatly benefits from deep neural networks, because they are powerful at function approximation and representation learning [4]. In traditional reinforcement learning, linear function approximators are used to model the action-value function

$Q(s,a)$, while in deep reinforcement learning (DRL), deep neural networks are used as the non-linear function approximator. They act as efficient and accurate policy approximators in many reinforcement learning problems, making complex policies that were previously expensive to represent tractable. Moreover, less or no feature engineering is required in deep reinforcement learning with the use of deep neural networks because they are able to use raw data as inputs and automatically learn useful feature representations.

The process of Deep Q-learning is shown in Figure 2.7. Similar to other reinforcement learning, the machine learning agents learn to select actions according to the current state based on a reward signal from the environment. The defining feature of deep reinforcement learning is that it uses a neural network to output Q-values for actions while taking pre-processed state representations as neural network inputs. In Q-learning, the optimal value function is as defined in Equation 2.4, and deep reinforcement learning uses neural networks as action-value function approximator, which is defined as:

$$Q(s,a;\theta) \approx Q^*(s,a) \tag{2.7}$$

where the function approximator $Q(s,a;\theta)$ is a neural network (Q-network) with weights θ. The Q-network can be trained by minimising the loss function $L_i(\theta_i)$ at each training iteration i. The loss function is defined as:

$$L_i(\theta_i) = \mathbb{E}[(y_i - Q(s,a;\theta_i))^2] \tag{2.8}$$

$$y_i = \mathbb{E}[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a] \tag{2.9}$$

The loss function represents the mean square error between the neural network prediction and the target sequence y_i.

Mnih et al. [87] has described a deep reinforcement learning algorithm, detailed in Algorithm 2, where experience replay is used to stabilise and smooth training [65]. The

replay memory buffer D is initialised with a fixed capacity N, and the action-value function Q is initialised with random weights. Since the algorithm is primarily used to play Atari games, the states are preprocessed to be a stack of 4 consecutive frames of the down-sampled grey-scale images. The preprocessed sequence is represented as ϕ, and is the input to the neural network. For episodic tasks, the actions are selected with ε-greedy exploration. Deep Q-learning is off-policy because with probability ε, actions are randomly selected instead of following the policy. The agent then executes the action, observes the state, preprocesses the state and stores the tuple $(\phi_t, a_t, r_t, \phi_{t+1})$ into the experience replay memory D. The optimisation starts when there is enough experience in the memory, and when the experience exceeds the replay memory N, early experience is discarded. At each training step, a mini-batch is randomly sampled from the replay memory and used to optimise the neural network according to Equation 2.8. Since the neural network is supposed to predict the Q-values, the target of the network is calculated using the reward r_j received at the sampled iteration j and the discounted future reward using the prediction from the current neural network. Loss L_i is the mean square error of the actual Q-values predicted by the current neural network and the target y_i. After a number of learning steps, the neural network becomes better at predicting the Q-values. [88] improved the stability of learning by introducing a target network. The target network is only used for estimating the target y_j, and the weights are copied from the online network every τ steps, but in other steps remain fixed.

Research in [134] has found that the deep Q-learning algorithm in [87, 88] suffers from substantial over-estimations (proposed in [131]) of the value of the current policy. This overestimation causes the learning to be unstable and the performance to drop significantly. The solution to this is to decouple action selection and action evaluation, as [134] suggested, where the policy is evaluated by the online network, and the Q-values are evaluated by the

Algorithm 2 Deep Q-learning with Experience Replay [87]

Initialise replay memory D to capacity N
Initialise action-value function Q with random weights
for each episode **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ε select a random action a_t
    Otherwise select $a_t = max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in replay memory D
    Sample random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    if ϕ is terminal **then**
      $y_j = r_j$
    else
      $y_j = r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta)$
    end if
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

target network. This means the target y_t^{DDQN}:

$$y_i^{DDQN} = \mathbb{E}[r + \gamma Q(s', \arg\max_a Q(s', a; \theta_i); \theta_{i-1})|s, a] \tag{2.10}$$

It has been shown in various Atari games where DQN is unable to keep improving, double DQN is able to reduce over-estimations, stabilise the learning process, and find significantly better policies.

In solving a range of Atari games, Deep Q-networks (DQN) are able to stably represent complex policies, and DRL has achieved super-human performance in all Atari games [87, 88]. In the field of robotics, although the application of DRL has only started in recent years, it has quickly been found to be effective in many applications. [127] has shown that deep reinforcement learning can be a mapless motion planner for robot navigation. The state representation is a 14 dimensional vector made up of raw laser range sensor readings, the velocity and the relative position of the target point to the robot, and the network model is a

multi-layer dense neural network. This paper has verified that with the raw sensory feedback from the robot, deep reinforcement learning allows a robot to robustly navigate to goal points in unknown and complex environments.

Deep Q-learning has also been applied to decentralised robot control. Chen et al. [18] discusses multi-robot collision avoidance in a decentralised and non-communication setting. This work trains a multi-layer fully-connected network offline using perfect information of all agents, where the neural network inputs are vectors of all robots' observations concatenated into "joint states". In many real-world problems, the environment is not fully observable, therefore the assumption of perfect information is unrealistic. This is particularly the case in multi-robot systems, because each robot only has direct access to their own observations, and the information exchange between robots is greatly limited by bandwidth and distance. Under the partial observability assumption, the commonly used deep reinforcement learning paradigm for multiple agents is centralised training, decentralised execution. An example of this is [72], where robots are trained centrally on a shared policy from all agents' local observations and executed in a distributed way. During both training and performing the task of navigation, robots only have access to local observations, and the environment is unknown before the training. Results have shown that the trained deep neural network controller is able to develop a stable and robust collision avoidance policy for multi-robot navigation. *Centralised training* benefits from sharing experience or parameters across agents [128]. In [33], agents share deep Q-network parameters and pass gradients across agents. In learning communication and language, this has shown to have learned better policies or have learned faster than *independent Q-learning*, where each agent learns their own network parameters and treats the other agents as part of the environment. The disadvantage of independent Q-learning is due to poor cooperation between agents, when other agents are treated entirely as part of the environment, independent Q-learning agents only learn to cooperate passively [128].

In contrast to existing approaches, heuristics facilitate cooperation and communication between robots. When Q-learning is used to learn a heuristic-based policy, robots still cooperate and the quality of cooperation is a deciding factor of how good a heuristic is at a given state. Therefore, the benefit of cooperation can be preserved even though the deep Q-learning of each robot is independent.

2.7 Summary

This chapter reviewed the hyper-heuristic method in literature and introduced the design idea of hyper-heuristics: automatically select heuristics from a pool and apply heuristics to solve the problem iteratively. Since the multi-robot action and state space can be huge, no existing coordination and control method allows swarm robots, especially those that can self-assemble, to perform complex tasks in unknown and dynamic environments without human intervention. By reviewing the behaviour-based multi-robot coordination methods in literature, we proposed to address this gap using hyper-heuristics, which has been proved to be effective in many applications, but not yet in multi-robot coordination. The rest of the thesis will explore and demonstrate how the hyper-heuristic approach can be used to select behaviours in swarm robots and benefit the task performance in unknown and dynamic environments.

After identifying the main research questions, this chapter gave a background of the application area, building surface cleaning, where multiple surfaces exist. There is no existing robotic cleaning system that has used self-assembling robots to achieve sufficient flexibility to clean in complex environments, and a coordination method for them to work on a range of unknown surfaces with dynamic obstructions is so far missing.

A review of machine learning methods that have been used in hyper-heuristics was also presented as background for methods that can be used in swarm robotics.

Chapter 3

Swarm Robot Environment

This section introduces the robot model used in this research, and the simulation and experimental setup. The aim of the simulations is to verify if the hyper-heuristic algorithms proposed in this thesis are applicable in real-world swarm robotic systems. The Webots simulator developed by Cyberbotics Ltd. [84] is chosen because it supports real-physics 3D simulation of multi-robot systems. In this chapter, we introduce the simulator, the self-assembling robots and the environments used for experiments.

3.1 Simulator

Webots simulator was developed by Cyberbotics Ltd. in 1998, and it was built based on the OpenGL 3D library [83]. The accurate physics simulation is achieved using the Open Dynamics Engine (ODE) library. This allows Webots to simulate robots locomotion, sensors and compasses etc. The common elements of constructing a robot, such as sensors and wheels, are pre-built components that can be directly used.

The Webots simulator provides an interface for constructing worlds and robots, as well as for programming controllers for robots and the environment. As shown in Figure 3.1, the left window lists the components included in the simulation world. The worlds in this

research consist of a ground with boundaries, robots and lighting. Webots provides a module "DifferentialWheels" which has the physical simulation of two wheels and their motors. This module can be installed on a robot model, and the locomotion can be programmed by setting the motor speed. A robot is based on the "DifferentialWheels" module, and Webots simulator provides sensor modules such as "DistanceSensor", "Camera" and "GPS" that can be installed on the robot. In the simulator, the simulated sensor readings can be read by the robot control program, which imitates the way an equivalent real-world robot would operate. The sensor readings are all simulated with real-world physics. A bounding box of each object (for example, robots, ground and obstacles) can be defined, and collision is handled by Webots's ODE physics simulation which prevents simulated objects from intersecting with each other. All the physics objects in this project are simulated as solids that cannot intersect with each other.

The coordinate system in the simulator is (x, y, z) where the x and z axes forms the horizontal plane of the world, and the y axis points vertically upwards. In this thesis, for simplicity, if there are only two dimensions involved in the calculations, we will use x and z.

The middle window visualises the simulation environment, and the right window is the programming window, which shows the program for robot and ground controllers. The Webots simulator gives each robot a programmable controller which emulates the microcontroller that would be installed on a real-world instance of the robot. This enables each robot to operate individually and therefore allows distributed control of the swarm. The programming language used for the robot controllers is C++.

3.2 The Self-assembling Robots

The robot model is based on the non-holonomic robots in [143], as shown in Figure 3.2, and extended with cleaning and self-assembling capabilities. Each robot is equipped with differential wheels, a dirt sensor, obstacle sensors, wireless communication, suction cups,

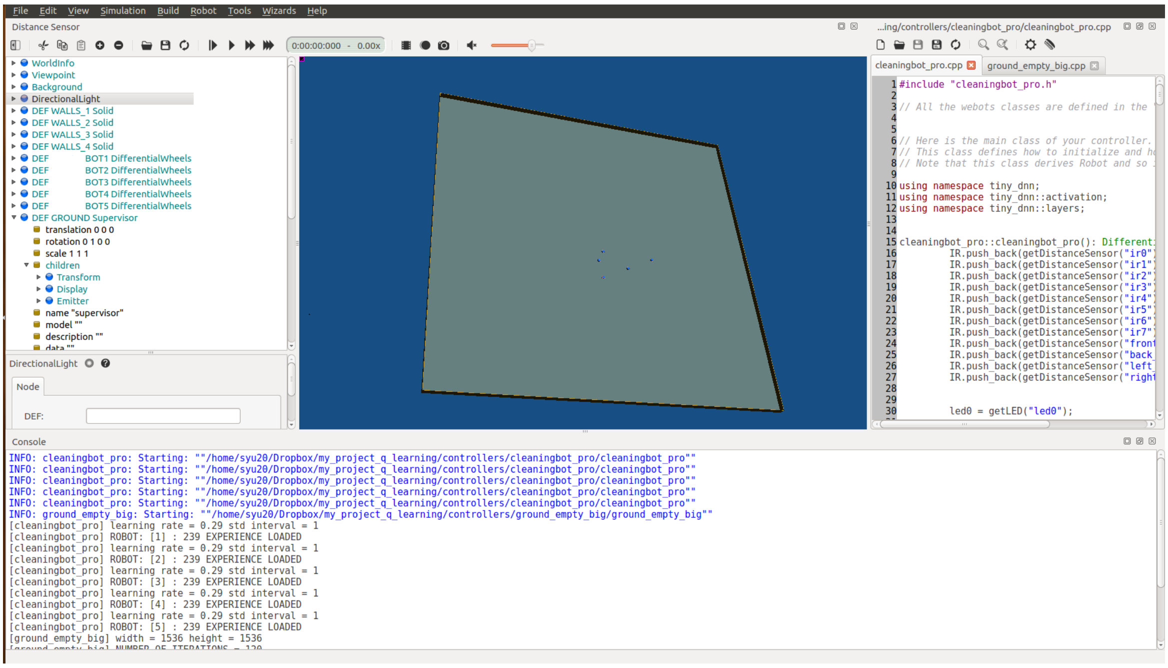
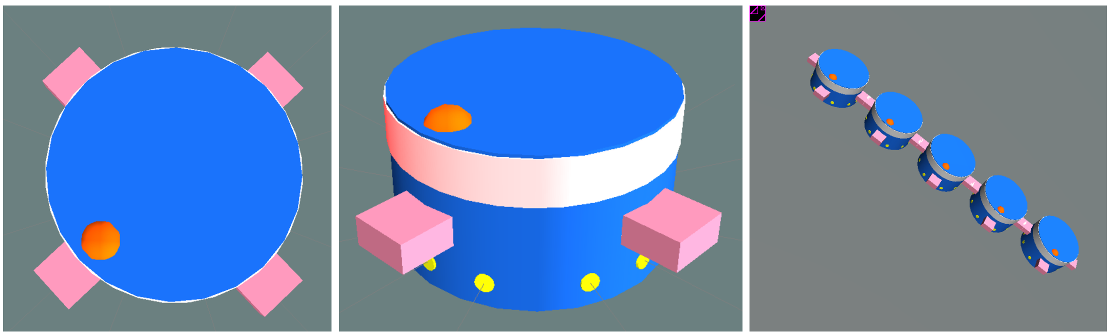


Fig. 3.1 The self-assembling robot state representation

cleaning wipes and a gripper arm to connect to a neighbouring robot. The robot model is as shown in Figure 3.2(a) and Figure 3.2(b). The radius of a robot is 0.3m.

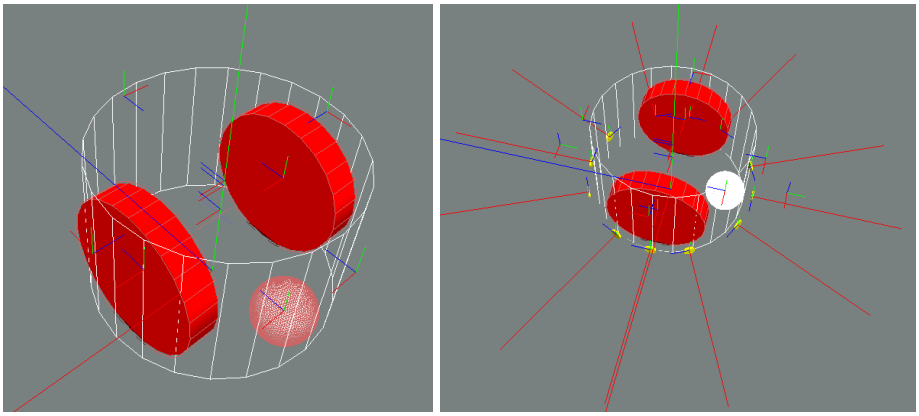


(a) Robot Model (top) (b) Robot Model (side) (c) Robots Connected

Fig. 3.2 The Self-assembling Robots

3.2.1 Robot Locomotion

The robots are differential wheel robots, which means the speed of each wheel can be set separately. As shown in Figure 3.3(a), each wheel can move forward and backwards, the



(a) (b)

Fig. 3.3 (a) Differential drive: two wheels and a caster (b) The distance sensors

caster rotates passively, and the when both wheels move forward or backwards at the same speed, the robot moves forward or backwards in a straight line. When the two wheels move at different speeds, the robot steers. The relationship between the robot linear velocity v, angular velocity ω and the robot left wheel speed v_{left}, robot right wheel speed v_{right} is:

$$v_{right} = \frac{2v + \omega L}{2R} \tag{3.1}$$

$$v_{left} = \frac{2v - \omega L}{2R} \tag{3.2}$$

where L is the distance between two wheels, and R is the radius of the wheels.

The simulated robot hardware configurations aim to emulate the robots in [143]. The robots are designed to clean at the maximum speed of $0.0745m^2$ per second. The table in Figure 3.1 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 $0.5m/s$.

Table 3.1 The robot configurations

Radius	$0.3m$
Maximum cleaning speed	$0.0745m^2/s$
Maximum linear velocity V_{max}	$0.5m/s$
Maximum angular velocity Ω_{max}	$0.66rad/s$
IR sensing range	$0.53m$
Wireless communication range	$200m$
Distance between two wheels L	$0.4m$
Radius of the wheels R	$0.167m$
Linear velocity multiplier V_{multi}	10.0
Angular velocity multiplier Ω_{multi}	6.0

The robot can be controlled by setting a target, and the control algorithm calculates the linear and angular velocity v, ω. For example, in Figure 3.4, a robot needs to reach the target position (x_{target}, z_{target}) on the right with a given target orientation, and the difference between the target and current orientation is θ_{target}. The target position coordinates (x_{target}, z_{target}) are both relative to the robot's current position. The robot calculates v and ω according to Equations 3.3.

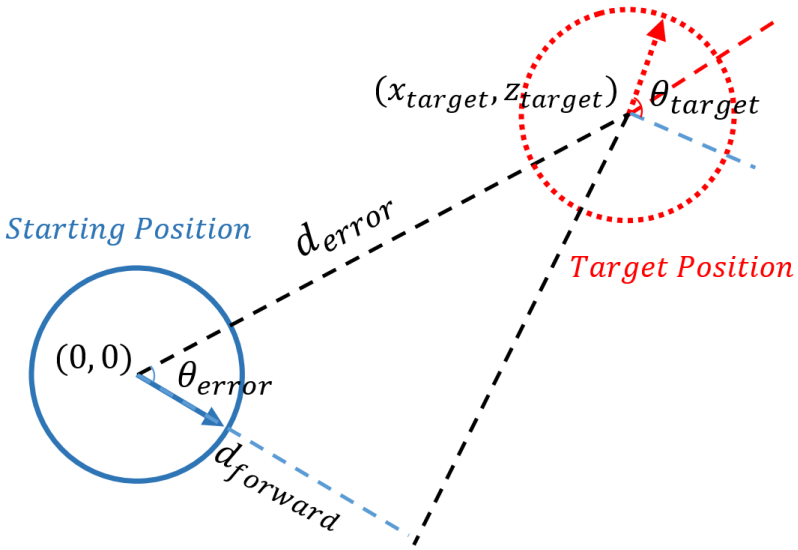


Fig. 3.4 Reaching a target position

$$
\begin{aligned}
d_{error} &= \sqrt{x_{target}^2 + z_{target}^2} \\
d_{forward} &= d_{error} \cos \theta_{error} \\
v &= \begin{cases} V_{max}, & \text{if } d_{forward} V_{multi} \geq V_{max} \text{ and } d_{error} \geq d_{threshold} \\ d_{forward} V_{multi}, & \text{if } d_{forward} V_{multi} < V_{max} \text{ and } d_{error} \geq d_{threshold} \\ 0, & \text{otherwise} \end{cases} \\
\omega &= \begin{cases} \Omega_{max}, & \text{if } \theta_{target} \Omega_{multi} \geq \Omega_{max} \text{ and } \theta_{target} \geq \theta_{threshold} \\ \theta_{target} \Omega_{multi}, & \text{if } \theta_{target} \Omega_{multi} < \Omega_{max} \text{ and } \theta_{target} \geq \theta_{threshold} \\ 0, & \text{otherwise} \end{cases}
\end{aligned} \tag{3.3}
$$

Where V_{multi} and Ω_{multi} are linear and angular velocity multipliers that reduce the robot's velocity when it moves closer to its target.

3.2.2 Robot Sensing

The robot sensing devices are listed in Figure 3.6. There are four dirt cameras on the front, back, left and right facing sides of the robot, measuring the dirt on the ground in a circle of radius 0.6m extending from the centre of the robot. The sensing area of the dirt cameras are shown in Figure 3.5, where four cone shaped areas in the front, left, back and right are covered by the cameras.

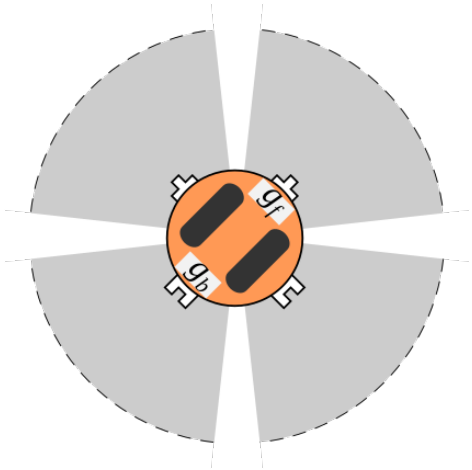


Fig. 3.5 Dirt camera sensing area and gap sensor locations

The gap sensors are located on the underside of the robots, as shown in Figure 3.5. The locations are indicated as g_f for the front gap sensor, and g_b as the back gap sensor.

A gripper camera is installed on each gripper for alignment. The camera is oriented to detect the colour of another robot's body LED if the other robot is properly aligned. The LED colours indicate different stages of the alignment and connecting process, which will be detailed in later chapters.

A GPS and compass are installed on each robot for localisation and orientation measurement. The GPS gives the robot coordinates in the simulator as (x, y, z), where the x and the z axis forms the horizontal plane and the y axis points vertically upward. Twelve IR sensors are installed for collision avoidance, the locations of which are shown in Figure 3.3(b). The distance sensing range is 0.53m.

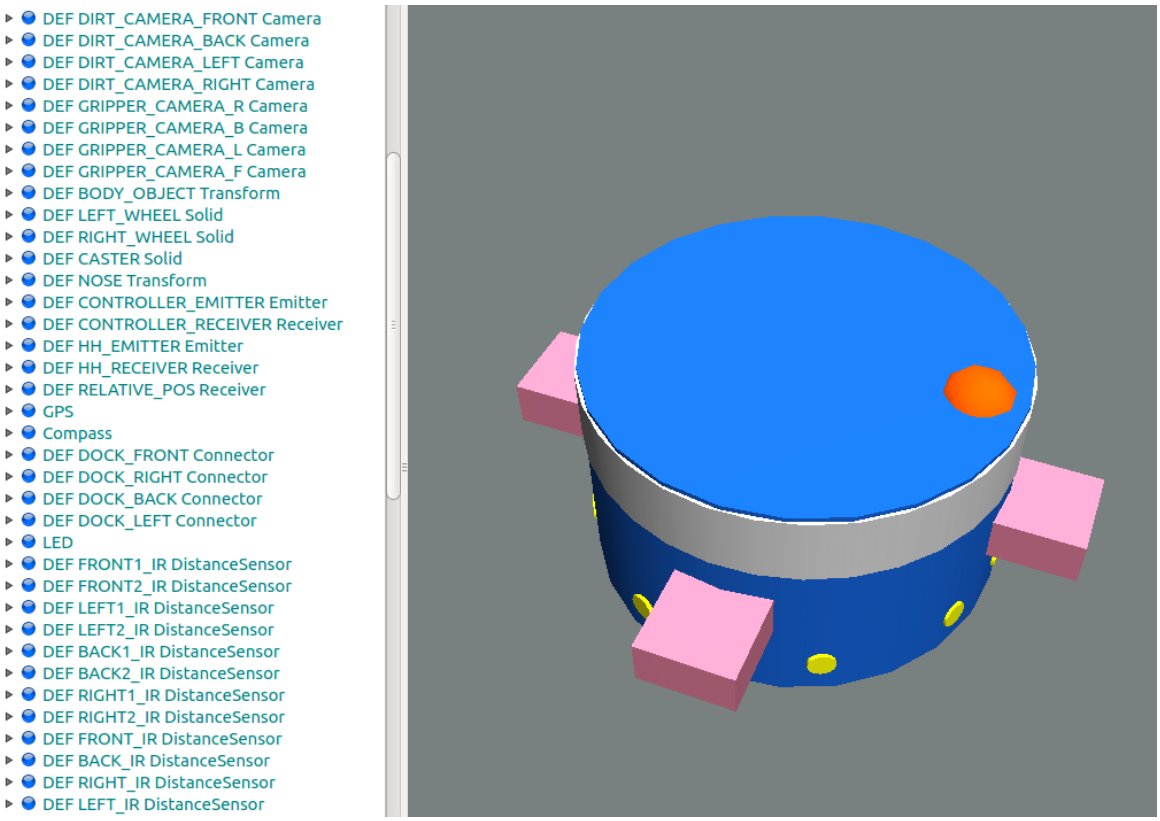


Fig. 3.6 Robot sensors

3.2.3 Robot Communication

In Webots simulator, an “Emitter” is the device that emulates a wireless transmitter, such as an RF transmitter and a “Receiver” emulates the RF device that receives the transmitted information. In Figure 3.6, it can be seen in the component list that two transmitter-receiver pairs are used: the “CONTROLLER_EMITTER - CONTROLLER_RECEIVER” pair and “HH_EMITTER - HH_RECEIVER” pair. The reason for having two pairs is to highlight the independence of the hyper-heuristic layer and the robot control layer. The hyper-heuristic layer messages are assigned to “HH_EMITTER - HH_RECEIVER”, and the robot control messages are assigned to “CONTROLLER_EMITTER - CONTROLLER_RECEIVER”. With this setup the low level robot control algorithms would still work even without the hyper-heuristic control layer. The transmitted messages have fixed formats, which are shown

in Figure 3.2. Voting messages, heuristic-score messages and Q-score messages are for hyper-heuristic control, therefore are transmitted on different channels of the "HH_EMITTER - HH_RECEIVER" device. Control messages are for robot communication in low level control algorithms. The use of each item in the message field will be explained in chapter 4.

Table 3.2 Robot Wireless Message Formats

Message Type	Voting Message	Heuristic-score Message	Q-score Message	Control Message
Information Field	sender ID is proposing candidate heuristic accept/reject proposal weight	sender ID is proposing byte count heuristic scores	sender ID state Q scores reward	sender ID start control state forward/backwards recipient (optional)

3.2.4 Interaction with Environment

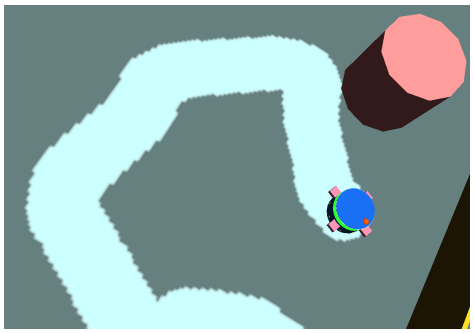


Fig. 3.7 The simulated visual feedback for cleaned area

When a robot moves around the surface, it constantly collects "dirt" from the surface. This project simulates visual feedback which highlights the cleaned area over the dirty area, as shown in Figure 3.7. Internally, the dirt is simulated to be distributed evenly as a layer on

the ground, and the robot collects the dirt underneath as it moves on the surface. This is to emulate the cleaning wipers and sweeping brushes on the bottom of cleaning robots.

The collected dirt is measured by counting "dirty" pixels in the image within the area under the robot captured by the dirt cameras. When the robot is in the middle of a gap, there is no dirt to collect, therefore the dirt measurement of that time step is 0. The robot cleaning efficiency $f(t)$ can be measured from

$$f(t) = \frac{\sum(L_t D_t)}{T} \tag{3.4}$$

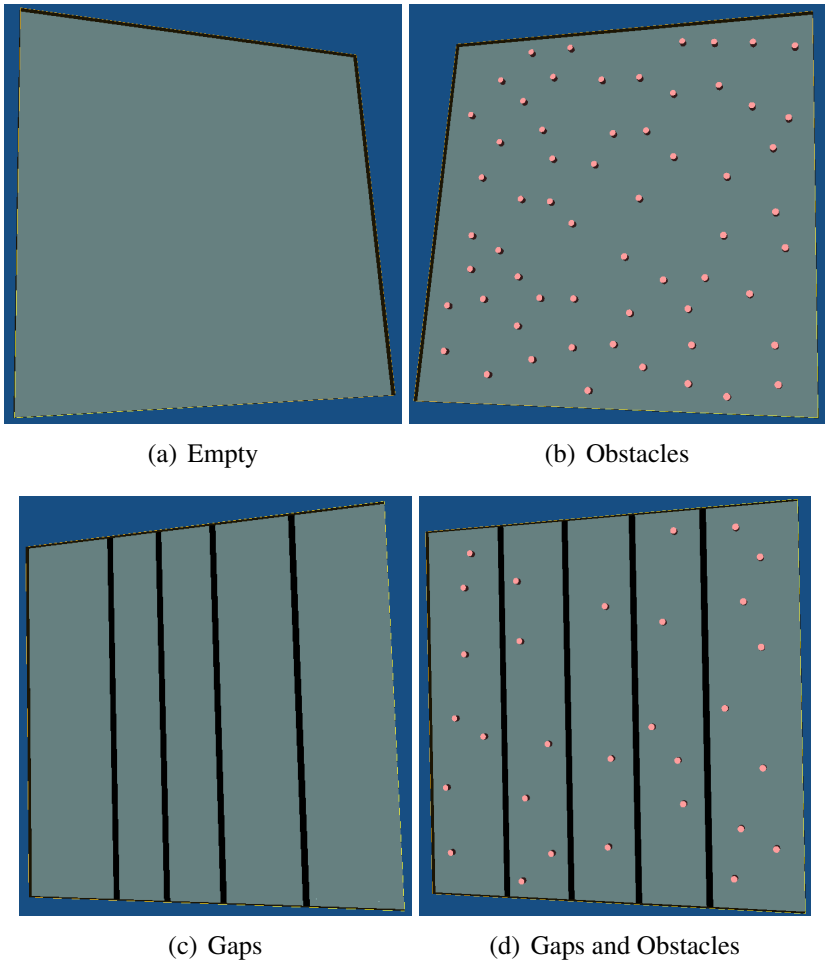
where L_t is the distance travelled since the last measurement, D_t is the percentage of "dirty" pixels at the time of measurement t, and T is the total time of measurement.

3.3 Environments

The simulator simulates obstacles and gaps in the environment. All environments are always large enough for the robots to clean within the time given. This is to prevent robots from running out of space to clean and affecting the fairness of measurement in comparisons. While the total area of environments is subject to individual experimental requirements, the configuration of elements in the environments have four possible types, as shown in Figure 3.8:

- *Empty environment*: the environment is bounded by walls, and the surface is smooth and connected. The walls are solid and can be detected by robots' distance sensors.

- *Obstacles environment*: the environment is also bounded by walls, and the surface smooth and connected. This type of environment also has obstacles distributed on the surface. Obstacles are all cylindrical pillars with a radius of $0.67m$, and can be detected



(a) Empty (b) Obstacles

(c) Gaps (d) Gaps and Obstacles

Fig. 3.8 The environments

by distance sensors. The number of obstacles in the environment is user-defined, and the obstacles are approximately evenly distributed on the surface.

- *Gaps environment*: the environment is also bounded by walls, and the surface smooth and connected. This type of environment also has gaps that separate the environment into multiple surfaces. Gaps all have a width of $0.67m$, which are wider than the $0.6m$ width of a robot. If one robot attempts to go across the gap it will fall into the gap, but with two or more robots connected, they will be able to move onto the surface on the other side. The gap sensors are able to detect if there is a gap present. The number of gaps in the environment and the distances between them are variable.

- *Gaps and Obstacles environment*: the environment is also bounded by walls, and the surface is smooth and connected. This type of environment has both obstacles and gaps which the shapes are identical to those in the previous environment types.

3.4 Summary

This chapter introduced the robot model used in this thesis, which is a 3D simulated robot model based on the existing non-holonomic robots in [143]. The self-assembling ability of the robots has been demonstrated through showing the robot gripper setups. Robot sensors including distance sensors, gap sensors and dirt cameras are set up to facilitate collision avoidance, gap avoidance and feedback regarding cleaning progress. This chapter also described the hardware mechanism that robots use to communicate with their neighbouring robots.

In order to verify the algorithm's robustness, we prepared a variety of scenarios for testing in simulation: empty environment, obstacle environments, gaps environments, and gaps & obstacles environment. These types of configurations cover simple environment, dynamic environment, multiple surfaces, and complex (dynamic and multiple surfaces) environments.

Chapter 4

The Hyper-heuristic Swarm Robot Framework

In the field of swarm robotics, manually developing control strategies can be too difficult for complex and unknown environments. Another approach is to automatically develop the strategy from a set of swarm robot behavioural elements, as reviewed in Chapter 2. This chapter addresses the question of how the hyper-heuristic approach can be used to automatically develop a strategy for swarm robots in unknown and dynamic environments.

We first introduce the methodology of our proposed hyper-heuristic framework that is designed to coordinate decentralised swarm robot behaviours, followed by a Multi-Armed Bandit based implementation of the framework which allows the robot group to adaptively change behaviours, and to assemble and disassemble according to the environment.

The method is verified on the application of surface cleaning with a range of scenarios. Through simulation and evaluations, this chapter shows that the hyper-heuristic framework is effective in building a sequence of swarm behaviours for a given task. Online learning in the hyper-heuristic method has also been shown to improve robot team performance over time.

This chapter also introduces group learning strategies that can improve the swarm robot task performance, which addresses the second research questions of how different learning strategies can improve the hyper-heuristic framework.

4.1 Methodology

We first introduce the hyper-heuristic methodology that is proposed to coordinate behaviours of swarm robots. Hyper-heuristics are built on heuristics. They treat heuristics as general purpose building blocks to be simultaneously and iteratively applied to a problem [116]. Our proposed hyper-heuristic framework combines the idea of "using heuristics to choose heuristics" with decentralised swarm robot control, and treats swarm behaviours as building blocks for the hyper-heuristic controller.

The proposed hyper-heuristic controller continuously applies swarm robot control heuristics intelligently to given tasks. The swarm robot hyper-heuristic framework aims to let a group of robots automatically sequence behaviours in a distributed manner.

A heuristic $h_u \in H$ is stored in the heuristic repository $H = \{H0, H1, H2...\}$. Each heuristic is a decentralised robot control algorithm that reads from sensors described in Section 3.2.2, prepares and sends communication messages with other robots, and at each time step outputs actions $a_u \in A$. At a time step t, the actions $a_u(t)$ are generated by a heuristic, $a_u(t) = \mathscr{H}_u(x_t, m_t)$, where x_t represents sensor readings and m_t represents inter-robot messages. Every robot $u \in U$ is a machine learning agent, and by executing actions a_u, robots perform collective behaviours.

An overview of the framework structure is given in Figure 4.1. For n robots $(u_1, u_2...u_n)$, each robot first performs *initialisation*, where a starting heuristic is chosen. Then the robot takes **actions** generated by 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) using objective functions. Based on the evaluation result,

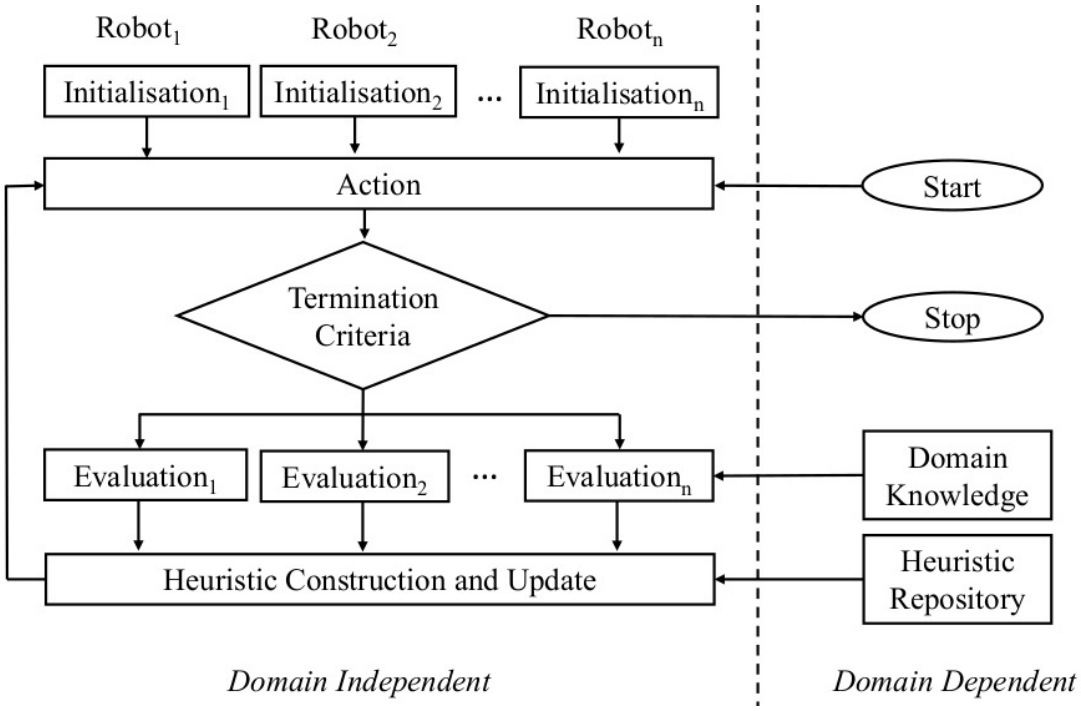


Fig. 4.1 Overview of the hyper-heuristic methodology for swarm robots.

heuristics then are selected from the pre-defined **heuristic repository** to construct new ones. It should be noted that this heuristic construction process uses the selection perturbative hyper-heuristic approach [101]. Learning can be involved at this stage, where robots remember and utilise previous experience to collectively improve future heuristic selections. Once a heuristic is determined for the interval, robots will make local inter-robot communications and **update** their heuristics. Robots take actions then enter the next iteration of the process unless a termination criterion is met.

The framework is designed for multiple robots to perform hyper-heuristic coordination and learning. There is no centralised control in this framework, and no global knowledge is required by the hyper-heuristic algorithm. The right part of the framework in Figure 4.1 is domain dependent, meaning that domain knowledge is used in these elements. The steps on the left are domain independent, which means that they are common for different problem domains and different heuristic repositories. Domain knowledge is needed to start and stop the robot swarm, for example, turning on and turning off the robot motors. Domain

knowledge is also used for evaluating the performance of a heuristic, since the objective function is domain-related. In robotics, domain knowledge is usually dependent on the available sensors and robot communication mechanisms. The heuristic repository contains a set of low-level heuristics that has been developed from previous experience or knowledge in the problem domain.

4.2 Multi-Armed Bandit Based Online-learning

To validate the proposed framework in Figure 4.1, we first give a Multi-Armed Bandit (MAB) based online learning implementation. The method allows swarm robots to be adaptive to changing environments.

The algorithm design idea follows the system architecture of cooperative multi-agent hyper-heuristics [98], where each robot is a "Low-Level Heuristic Agent" and the hyper-heuristic layer selects the next heuristic for robots to apply at the next decision point. All heuristics are decentralised multi-robot control algorithms that allow robots to cooperatively perform a task. Since all robots are fully cooperative and aim to achieve the best group performance, a group decision making strategy is needed to determine which heuristics are beneficial for the group and should be accepted to apply next [99].

The "Heuristic Construction and Update" part in Figure 4.1 of the proposed hyper-heuristic methodology is the process that defines different implementations. The one we propose in this section involves the commonly used iterative heuristic selection and acceptance combination. The selection is based on knowledge gained with an online learning method that utilises the MAB Upper Confidence Bound (UCB) method [26] and choice-function [103], and the acceptance method is a voting-based group decision making mechanism. Algorithm 3 shows the pseudo-code of the process, and the rest of this section provides a detailed explanation of the algorithm, including the method "SelectHeuristic" and the method "GroupDecisionMaking" in Section 4.2.4.

Algorithm 3 Multi-Armed Bandit Based Adaptive Robot Hyper-heuristic

Initialise heuristic scores for the heuristic repository H with k heuristics
Set current heuristic to the initial heuristic H_0
while termination criteria not met **do**
  for each robot **do**
    Observe sensor data x_t, read robot messages m_t
    Apply heuristic $a_u^t \leftarrow \mathscr{H}_u(x_t, m_t)$
    Apply action a_u^t, calculate reward f_t
    Update $M_t^i = \sqrt{(2\log\sum_{j=1}^{k} n_j)/n_i}$
    Update heuristic scores $\hat{h}^i \leftarrow \theta\hat{h}_{i-1} + (1-\theta)(\alpha f_t + \beta f_t^{ij} + \gamma M_t^i)$
    $H_c \leftarrow$ SelectHeuristic $(\hat{H})$
    $H_i \leftarrow$ GroupDecisionMaking (H_c, U)
  end for
end while

4.2.1 Initialisation

In our multi-armed bandit based algorithm, the initialisation involves initialising the heuristic scores and choosing a initial heuristic. Heuristic scores are denoted as $\hat{H}_t = \{\hat{h}_t^1, \hat{h}_t^2 ..., \hat{h}_t^k\}$ for k heuristics at time interval t. When there is no prior knowledge, heuristic scores are initialised with random values. In some scenarios, where there is prior knowledge a heuristic is better or worse, the initial heuristic score for that heuristic can be weighted up or down accordingly.

Each robot needs to initialise with a starting heuristic $H_0 \in H$. This initial heuristic can be randomly selected heuristic from the repository H, or selected according to the heuristic scores $\hat{H}$.

4.2.2 Generating Actions

Heuristics in our system are algorithms that read the robot sensor inputs and generate actions according to the sensed information. The detailed description of heuristics are given in Section 4.3. By applying a selected heuristic, actions $a_u \in A$ are generated and executed for one time interval T.

Next, the robots check if the termination criteria are satisfied, which is determined using problem-specific information. If the task objective is accomplished, the algorithm stops. If the termination criteria are not satisfied, the objective function is used to evaluate the performance of the applied heuristic.

4.2.3 Heuristic Evaluation

Performance evaluation reflects reward from the environment and is used to update the heuristic score measures to learn the "goodness" of a heuristic. The scores $\hat{H}_t$ are then used to influence the heuristic selection.

At the heuristic evaluation stage, robots read from sensors and incoming communication messages, and measure the performance of the applied heuristics according to the given task objectives. In this thesis, the evaluation approach is using objective functions. The objective value f_t reflects how good a heuristic is during the last time interval. For the task of surface cleaning, the objective function is a measure of collected dirt with respect to time. (The details of how the collected dirt is measured is described in Section 3.2.4.) Since the objective function is domain related, it changes from task to task.

4.2.4 Heuristic Construction and Update

This step constructs the sequence of heuristics in a decentralised manner. In this proposed online learning implementation, the heuristic construction has two steps, heuristic selection and group decision making.

As shown in Algorithm 3, heuristic selection is a method to choose a heuristic H_c from a robot's own experience, and in this algorithm, the experience is represented in heuristic scores $\hat{H}$. The algorithm is explained as follows.

Heuristic Scores

The heuristic scoring method is inspired by Choice Function (CF) and MAB, as described in [103] and [26, 116] respectively. Each robot calculates the score of the last action locally for each heuristic based on the objective function.

$$h_t^i = \sum(\alpha f_t + \beta f_t^{ij} + \gamma M_t^i) \tag{4.1}$$

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 Equation 4.2 to compensate for heuristics that are unexplored [26].

$$M_t^i = \sqrt{\frac{2\log\sum_{j=1}^{k} n_j}{n_i}} \tag{4.2}$$

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.

Online Learning

The scores $\hat{H}_t = \{\hat{h}_t^1, \hat{h}_t^2 ..., \hat{h}_t^k\}$ are updated at the end of each iteration according to previous experience and current performance, as shown in Eq. 4.3.

$$\hat{h}_t^i = \theta h_t^i + (1-\theta)\hat{h}_{t-1}^i \tag{4.3}$$

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.

Through the following method of online learning, the robots gradually learn more accurate heuristic performance measurements, which helps select better heuristics.

Heuristic Selection

As shown in Algorithm 3, at each time step t, a robot performs local heuristic selection based on the heuristic scores $\hat{H}$ learned in the previous iterations. The heuristic selection method in this implementation only uses local observations.

There are different strategies that could be used to select the local candidate heuristic, denoted as H_c. As reviewed in Section 2.3, researchers have found effective heuristic selection and acceptance methods for heuristic learning, which we will adapt and evaluate for our proposed framework. In this section we will introduce three heuristic selection methods:

- *Simple Random (SR)*: This method randomly selects a heuristic from the repository.
- *Greedy (GR)*: This method selects the heuristic with the highest score.

$$H_c = \arg\max(\hat{H}) \tag{4.4}$$

This method always uses the current best heuristic, and sub-optimal heuristics do not get explored. Relatively worse heuristics would not be re-used even if it may lead to better solutions when conditions changes.

- *Roulette Wheel (RW)*: Roulette Wheel selection (RW) is described in [66]. The probability of a heuristic i to be chosen is relative to its heuristic score $\hat{h}^i$:

$$p_i = \frac{\hat{h}^i}{\sum_{k=1}^{K} \hat{h}^k} \tag{4.5}$$

Since all heuristics have a chance to be selected, there is exploration outside the current policy. A good heuristic has greater chance to participate, while sub-optimal ones also have the chance to be selected. This gives the selection process a chance to escape local optima.

Since the input to the "SelectHeuristic" method is only the heuristic scores $\hat{H}$, the decision making is only based on the robot's own experience. In order to make sure the final heuristic benefits the whole group, a group decision making mechanism is designed to reject the heuristics that are only good for one or a few robots.

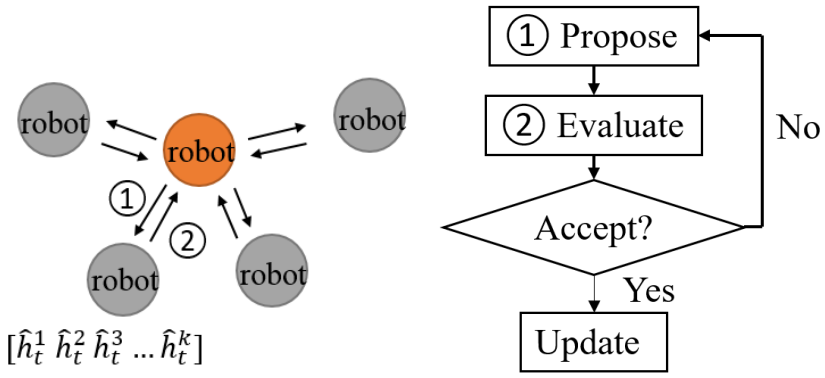


Fig. 4.2 Implementation of evaluation and group decision making.

Group Decision Making and Update

The robots 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 [13]. In a robotic system, once a move is performed, it can not be reverted, because reverting means going back to the robot's original state (position). In the real-world this is impractical and consumes energy.

In this multi-robot hyper-heuristic implementation, although a move is always accepted, a locally selected heuristic needs to be accepted by the whole robot group. In this section we introduce a voting-based decentralised group decision making strategy in which robots collectively make decisions through communication with nearby robots.

For swarm robots, 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 [99]. A heuristic is accepted if

$$\sum_{n=1}^{N} d_n \geq \delta \tag{4.6}$$

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.

The realisation of this strategy is shown in Figure 4.2. Using the heuristic selection method in the last section, any robot in the group can be the one to propose a heuristic, as long as one time interval has finished. When a robot's time counter for the heuristic has expired, the robot *proposes* its candidate heuristic H_c to the neighbouring robots to evaluate by sending a voting message as described in Table 3.2. When proposing, the robot sends the candidate heuristic index H_c in the "candidate heuristic" field, and its accumulated reward

in the "weight" field. If multiple robots propose at the same time, the receiving robots will compare the received accumulated rewards in the "weight" fields, and choose the one with highest weight to evaluate. At the *evaluate* step, all neighbouring robots will perform a heuristic selection step based on their own heuristic scores, and compare their local heuristic to the candidate heuristic. If they are the same, it will return a decision $d_n = 1$, meaning "accept" in the return voting message, otherwise will return $d_n = 0$, meaning "reject". Within a time frame, the proposing robot collects return voting messages from neighbouring robots, and determines if the group will accept or reject according to Equation 4.6. If the candidate heuristic is rejected, the robot will perform another heuristic selection iteration and propose again until the candidate heuristic is accepted.

Once a heuristic is accepted by the group, the proposing robot sends another voting message to inform neighbouring robots the final heuristic H_i. As shown in Algorithm 3, the "GroupDecisionMaking" method takes the candidate heuristic H_c as input, and needs to interact with other agents U. After the new heuristic is received by the swarm, all the robots will update their own heuristics to be the new heuristic, and take another action.

4.2.5 Problem Domain

As shown in Figure 4.1, the domain-dependent information is on the right hand side, which includes the starting and stopping criteria, the evaluation function for heuristics, and the heuristic repository. In order to validate the proposed method, we need to contextualise the domain information.

A suitable application area is proposed in Chapter 2. We identify a gap in the current state of robot control algorithms for building surface cleaning: no existing algorithm can coordinate self-assembling robot behaviours for building surface cleaning in a decentralised manner, where surfaces to be cleaned are not connected, and the environmental layout is not previously known. In this problem domain, the starting point is when the robots are

deployed on the surface(s), and the task termination criteria can be custom defined case by case. For example, we give all scenarios a fixed time frame for performing a task, therefore the termination criteria is to terminate after n iterations.

The evaluation function in the application of surface cleaning is a measurement of how much dirt is collected, and the feedback is given by dirt cameras on robots, as explained in Section 3.2.4. In the heuristic repository, there are algorithms that can be helpful in the multi-surface cleaning task. Each heuristic allows the group of robots to cooperate in a way where each of them cleans a part of the surface or moves in the environment in a coordinated manner. The details of the heuristic repository used for this problem domain are presented in the next section.

4.3 The Heuristic Repository

This section introduces the heuristic control algorithms that are programmed into each robot. Heuristics are pre-defined programs that generate actions for robots to execute. In this study, robots with identical hardware setups are used in heuristic-based learning and action-based learning, which means that their state space and action space are the same. In the learning process, heuristics make use of domain knowledge and reduce the search space by searching in a known good subset of all legal actions.

Heuristics in this repository assume a fixed group size throughout the task, and it is assumed that the group size is known by all robots in a group.

Table 4.1 shows the robot parameters of relevance to the heuristics.

Each heuristic has mechanisms for collision avoidance and gap avoidance that are programmed into the robot. The collision avoidance algorithm applies a multiplier v_{multi} to linear velocity, and another multiplier ω_{multi} to the robot's angular velocity. The multipliers

Table 4.1 Robot Parameters

Maximum Linear Velocity	$0.5m/s$
Maximum Angular Velocity	$0.66rad/s$
IR Sensing Range	$0.53m$
Wireless Communication Range	$200m$

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_{min} is the distance between obstacle and robot, and θ_{obs} is the angle between robot orientation and the obstacle. Algorithm 4 details how the multipliers are applied to robot linear and angular speed at each control step. The robots read from each distance sensor and find the one with shortest distance to obstacles, as shown in line 1-6. The distance between the closest obstacle and the robot d_{obs} is used to calculate multiplier v_{multi}. The direction the sensor is facing θ_{obs}, is the direction of the closest obstacle, and is used to calculate ω_{multi} in line 8. Next, in line 9-10, the robot updates its linear and angular velocity to be applied at the next step.

The gap avoidance mechanism is described in Algorithm 5. When a gap sensor detects a gap, the robot chooses a random angle θ_g between $\pi/2$ and π. The robot rotates θ_g and checks if the gap is still in the way of the robot (if the front gap sensor still senses the gap). If not, the robot moves forward, otherwise the robot rotates another random number of degrees until the robot is no longer facing the gap.

Algorithm 4 Collision Avoidance

1: **for all** distance sensors **do**
2: Observe sensor reading d_i
3: Store sensor reading d_i in D
4: Store obstacle angle θ_i in Θ
5: **end for**
6: Find sensor $s_{obs} \leftarrow \arg\min(D)$
7: Get sensor reading d_{obs} and sensor angle/obstacle direction θ_{obs}
8: Calculate v_{multi} and ω_{multi}
9: Robot linear speed $v \leftarrow v \times v_{multi}$
10: Robot angular speed $\omega \leftarrow \omega \times \omega_{multi}$

Algorithm 5 Gap Avoidance

1: **for all** gap sensors **do**
2: Observe sensor reading g_i
3: Store sensor reading g_i in G
4: **end for**
5: **while** any $g \in G > 0$ **do**
6: Gap avoidance angle $\theta_g =$ randomNumberGenerator($\pi/2$, π)
7: Rotates θ_g
8: **end while**
9: Move forward

4.3.1 Sweeping

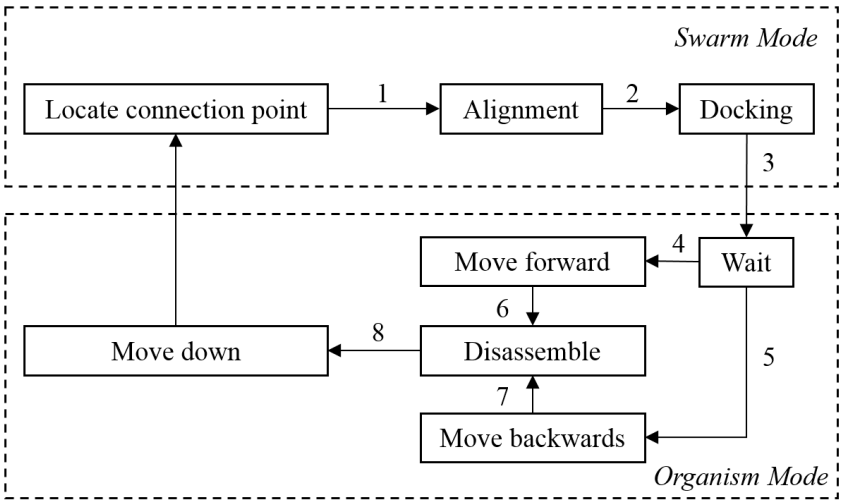
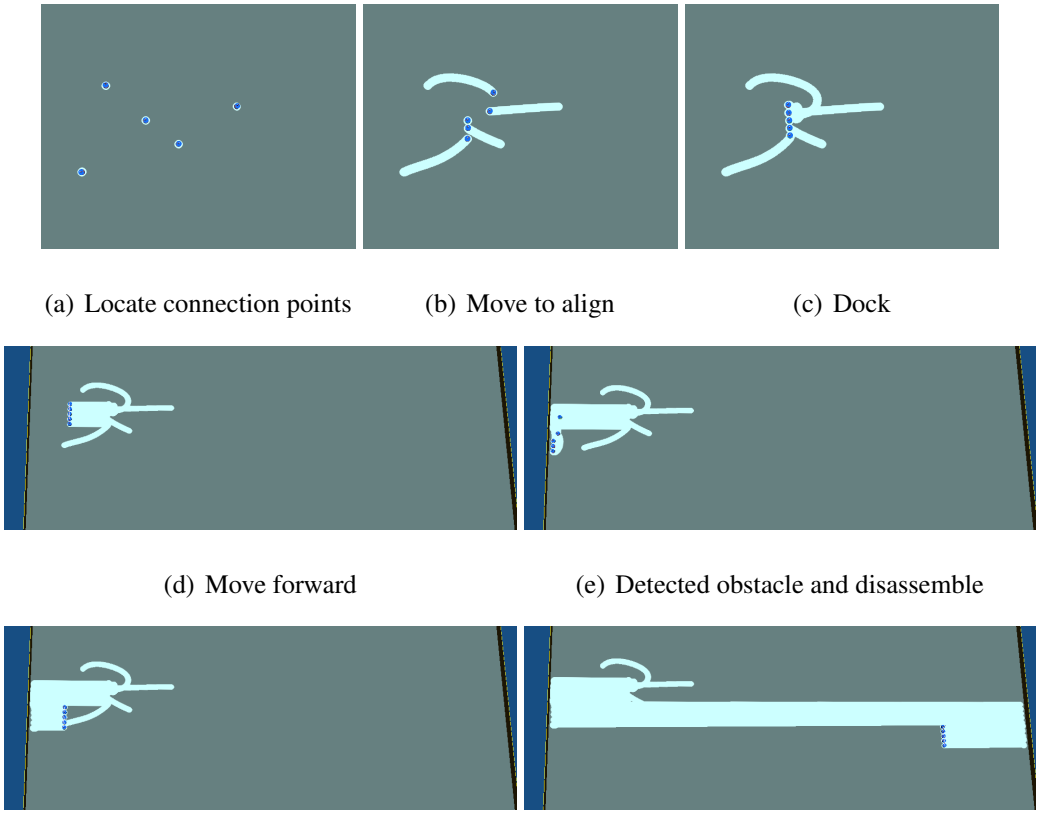


Fig. 4.3 The sweeping heuristic: 1 - connection points located; 2 – aligned and ready to dock; 3 – docking completed; 4 - no obstacle/gap/clean area in front of the robot; 5 – no obstacle/gap/clean area behind the robot; 6 – front obstacle/gap detected; 7 – back obstacle/gap detected; 8 – disconnected

The sweeping algorithm is adapted from the finite state machine-based self-assembling control algorithm in [71], shown in Figure 2.3, where robots have two modes, Swarm Mode and Organism Mode. Swarm Mode includes states when the robots are not connected with other robots, and Organism Mode includes states when robots are physically connected with other robots and move as a combined organism on the surface. Robots in this thesis employ different locomotion from [71], and the combined organism also moves differently.



(a) Locate connection points (b) Move to align (c) Dock

(d) Move forward (e) Detected obstacle and disassemble

(f) Move backward (g) Move forward again

Fig. 4.4 The sweeping heuristic

The sweeping behaviour allows robots in a group to physically connect into a line and move as a larger entity. In the context of cleaning, this is similar to turning a small cleaning pad into a large cleaning pad. The FSM robot control algorithm is shown in Figure 4.3. The aim of this FSM control algorithm is to ensure that all robots are assembled into a line and move together.

If robots start as scattered, in "swarm mode", they will first locate their own connection points, and will all try to align with the centre robot. Figure 4.4(a) gives an example of the starting positions of the robots. To move to the centre, each robot will follow the closest robot that is vertically ahead. The centre robot stays at the same location, and rotates to the target horizontal orientation. Once a robot reaches its target docking location and is ready

to be connected with other robots, it changes its LED colour to green. A robot will only connect to a robot whose LED colour is green. The robots connect one by one, and since all the robots know how many robots there are in the group, the last robot knows there are no more robots to connect. When the last robot is connected, it means that the assembling is complete and it will send "Control Messages" (described in Section 3.2.3) to other robots in the organism to inform them to start moving. The last robot determines which direction to move by checking the distance sensors, gap sensors and dirt sensors. As shown in Figure 4.3, transition 4 and 5, if there is no obstacle or gap in front of the robot, or if the back area is already clean, the organism will choose to move forward, otherwise it will move backward. When in organism mode, if any robot in the organism has detected an obstacle or gap, it will send another "Control Message" to call for disassembling (shown in Figure 4.4(e)), and to move down to the dirty area. The robots will re-align, dock and re-connect. This time there is an obstacle in the front, therefore the robot goes to transition 5 and moves backwards. With this FSM controller, the robots collectively perform a back-and-forth motion as shown in Figure 4.4(g).

4.3.2 Bridging (horizontal and vertical)

Bridging heuristic enables the swarm to connect and form a bridge structure when gaps are encountered, as demonstrated in Figure 4.6. This algorithm is adapted from the finite state machine-based self-assembling control algorithm in [71]. In this section, robots have two bridging heuristics: horizontal bridging and vertical bridging, as shown in Figure 4.6(g) and Figure 4.6(h) respectively. When there are multiple surfaces separated by gaps, single robots cannot cross to the other side. Similar to the sweeping heuristic, the bridging control algorithm is a Finite State Machine, and is detaled in Figure 4.5.

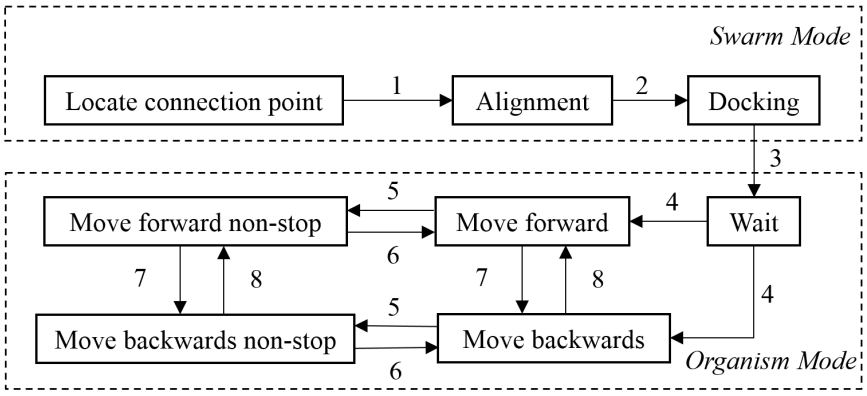
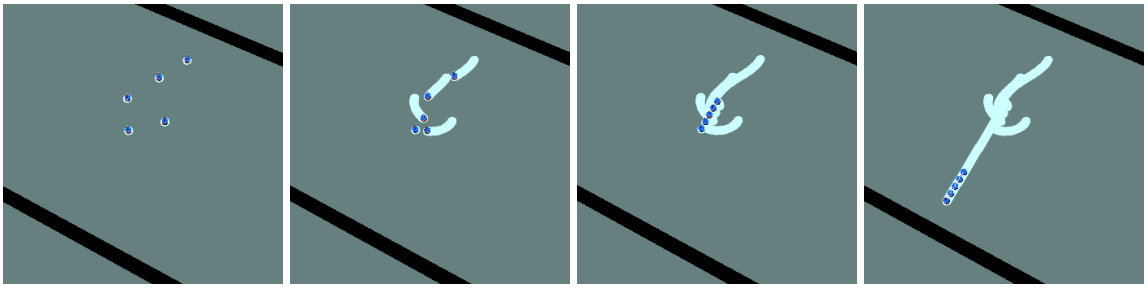
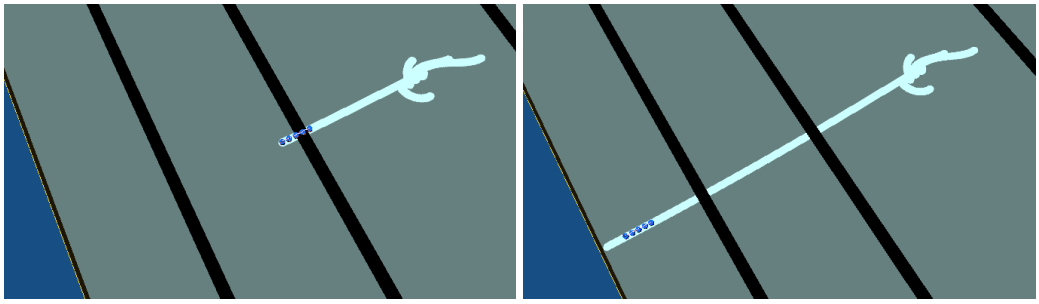


Fig. 4.5 The bridging heuristic: 1 – connection points located 2 - aligned and ready to dock 3 - docking completed 4 - move signal received/sent 5 - crossing a gap 6 - crossing completed 7 - front obstacle detected 8 - back obstacle detected

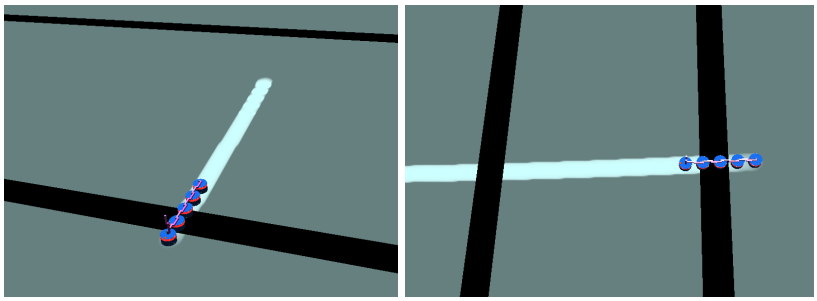
A robot applying this heuristic will first check if it is connected to another robot, if not, it will observe nearby robot positions and seek to align with the robot that is horizontally or vertically ahead, depending on if it is usiing horizontal bridging or vertical bridging heuristic. The docking process is the same as the sweeping heuristic, where the synchronisation is achieved by LED colours. Each robot will check the connected number of robots when it connects, and the last robot to connect will find the number of connected robots equals to the group size. At this point, it will send “Control Messages” to connected robots and signal all robots to move forward, as shown in Figure 4.6(d). If a gap is detected by the gap sensor, transition 5 (Figure 4.5) brings robot a state where it will move non-stop until the gap is no longer detected by any robot in the group. When crossing the gap is completed, the robot returns to original move states through transition 6. When there is an obstacle encountered, such as a wall in Figure 4.6(f), the organism changes moving direction. Once the group is fully assembled, it does not disassemble unless the heuristic changes.



(a) Locate connection points (b) Move to align (c) Dock (d) Move forward



(e) Move forward non-stop (f) Move backward after hitting obstacle



(g) Bridging (horizontal) (h) Bridging (vertical)

Fig. 4.6 The bridging heuristic

4.3.3 Exploring

Exploring is a heuristic which implements random-walk with noise. It allows robots to take actions that scatter the group in random directions. The resulting movement is as indicated in Figure 4.7. Each robot moves in a straight line while avoiding obstacles using Algorithm 4. Since each robot is equipped with dirt sensors on the bottom, a small amount of noise is

introduced for the robot to change direction if the total dirt value is below a control threshold T_{dirt}.

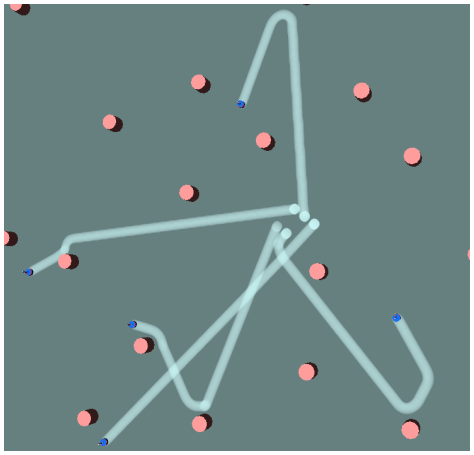


Fig. 4.7 The exploring heuristic

This heuristic allows robots to spread out on the surface, while avoiding cleaning the cleaned area.

Algorithm 6 Heuristic: Exploring

1: **while** termination criteria is met **do**
2: Set robot linear speed $v = v_{exp}$
3: Observe dirt sensor readings
4: **if** Total detected dirt $< T_{dirt}$ **then**
5: $\omega \leftarrow \omega +$ randomNoise$(-0.1, 1)$
6: **else**
7: $\omega = 0$
8: **end if**
9: **end while**

4.3.4 Circling

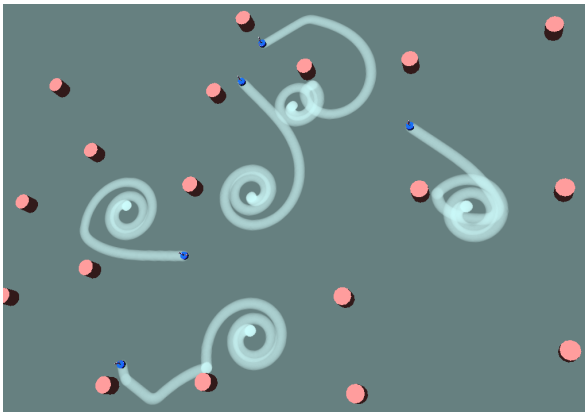


Fig. 4.8 The circling heuristic

Circling is a cleaning pattern that is widely used in cleaning robots, as mentioned in [40]. Robots perform a spiral motion, following with straight lines in random directions, as shown in Figure 4.8.

Algorithm 7 Heuristic: Circling

1: **while** termination criteria is met **do**
2: Initialise $v_{vt} = v_0$, $\omega_{vt} = \omega_0$
3: $v_{vt} \leftarrow v_{vt} + \alpha_{vt}$
4: $v \leftarrow \min(v_{vt}, V_{max})$
5: $\omega_{vt} \leftarrow \omega_{vt} - \beta_{vt}$
6: $\omega \leftarrow \max(\omega_{vt}, 0)$
7: **end while**

This heuristic as described in Algorithm 7 calculates the linear velocity v and ω. v is first initialised to a small value v_0 and ω is initialised to a relatively large value ω. A large initial angular velocity allows the robot to move in a small circle first. At each control step, a small value α_{vt} is added to the linear velocity and a small ω_{vt} is reduced from the angular velocity. This makes the robot slowly move in a growing circle and outward, as shown in Figure 4.8. The increasing linear velocity makes the robot slowly reach the maximum speed. When the angular speed is reduced to 0, the robot moves in a straight line. Obstacle avoidance and gap

avoidance in Algorithm 4 and Algorithm 5 are used in this heuristic, therefore in the second stage of the heuristic, robots perform a movement in a straight line.

4.3.5 Flocking

This local controller enables robots to follow a flocking behaviour based on the behaviour described in [107]. This heuristic commands robots to stay relatively close to each other and avoid obstacles, while preserving some degree of random exploration.

The rules introduced for flocking are :

1. Robots avoid collisions,
2. Robots attempt to stay close to teammates, and
3. Robots attempt to match orientation with teammates.

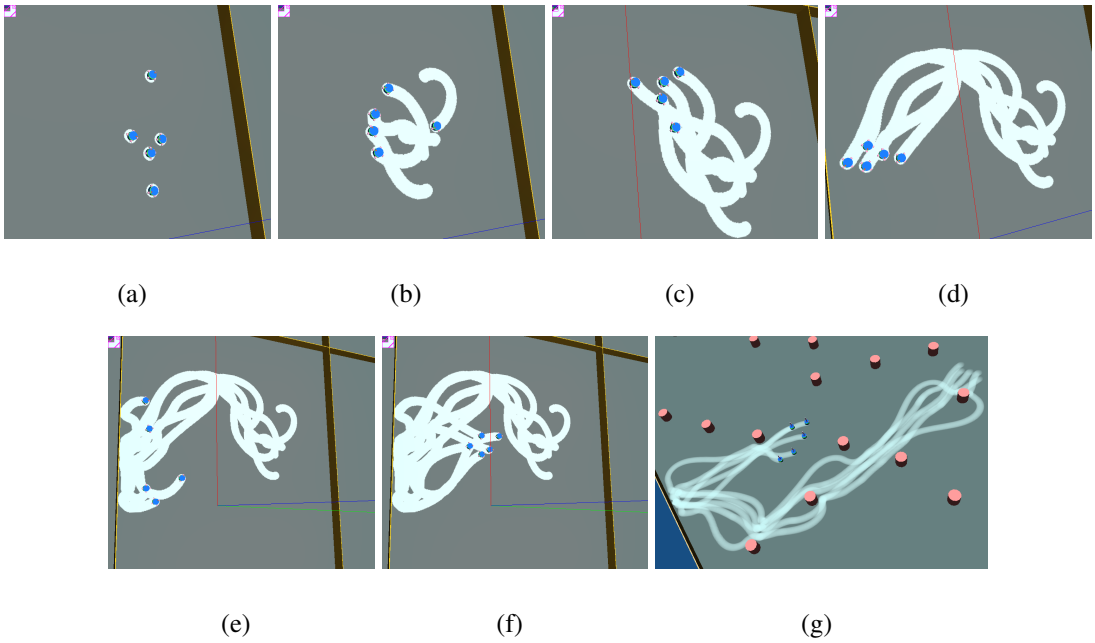


Fig. 4.9 The flocking heuristic

Unlike other heuristics where rules are somewhat directly translated into code, these rules are more like design guides that help shape the velocity calculations. To realise the flocking

rules, each robot iteratively calculates its target relative position according to neighbour positions and orientation as follows. Assume the position of the robot $P_0 = (x_0, z_0)$. To avoid collisions, each robot calculates a repulsion vector $R_{rep} = (x_R, z_R)$ from $D_{minobs} = (x_{minobs}, z_{minobs})$, which is the direction vector of the closest obstacle detected from proximity sensors, to let $R_{rep} = -D_{min}$:

$$x_R = -x_{minobs}$$

$$z_R = -z_{minobs}$$

To make robots stay together, $\overline{P}(\overline{x}, \overline{z})$ calculates the average position of nearly flocking robots, assuming there are n flocking teammates with coordinates $\{(x_1, z_1), (x_2, z_2)...(x_n, z_n)\}$:

$$\overline{x} = (\sum_{i=1}^{n} x_i)/n$$
$$\overline{z} = (\sum_{i=1}^{n} z_i)/n$$

Robots also attempt to match orientation with teammates, and to achieve this, robots calculate the average orientation of other flocking robots $\overline{O}$.

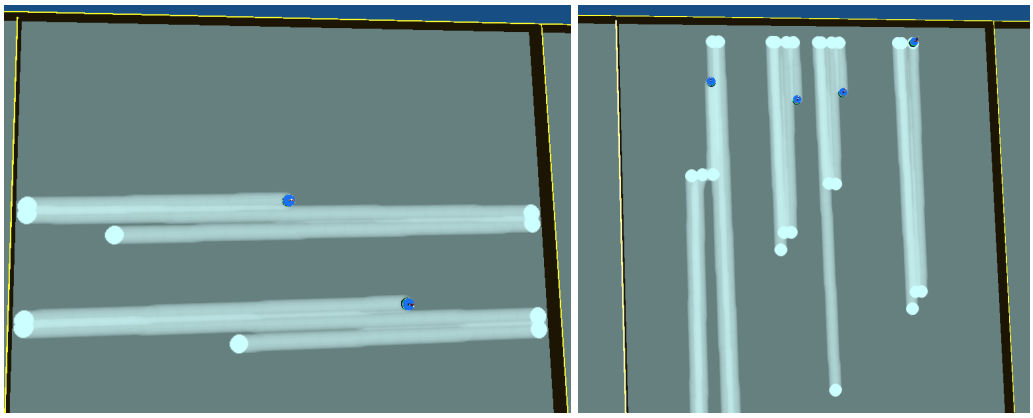
$$\overline{o} = (\sum_{i=1}^{n} o_i)/n$$

Each robot satisfies the three rules by summing the three vectors R_{rep}, $\overline{P}$ and $\overline{o}$. At each time step, the next position is calculated as:

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

where $S_x = S_z = 0.5$.

4.3.6 Raster Scan (Horizontal and Vertical)



(a) Raster Scan (horizontal) (b) Raster Scan (vertical)

Fig. 4.10 The raster scan heuristic

This heuristic allows robots to perform a raster scan-like motion in the north-south direction, similar to the behaviour described in [106]. The control algorithm works as follows: each robot keeps a record of its visited locations, and at every time step, the robot checks its surrounding in the order of north, south, east and west. If there is no obstacle or gap in that direction or the location has not been visited, the robot will add the direction to move set M. If $M = \emptyset$, the robot will move towards a random direction, otherwise the robot will choose the first move $M(0)$ in the move set.

The heuristic allows robots to perform movements as shown in Figure 4.10.

4.3.7 Neighbourhood Search

This heuristic performs a neighbourhood search algorithm [39] according to the amount of dirt detected around the robot. It periodically checks the robot dirt sensors and steers the robot in the direction where there is the highest amount of detected reward. For every step t, the robot determines its moving direction $Dr = \{front, right, left, back\}$ from dirt sensor

Algorithm 8 Heuristic: Raster Scan

read the robot position sensor, initialise available move set D_0, map of visited locations M_0
$i = 0$
while robot position changed **do**
  $i \leftarrow i + 1$
  update map M_i
  check in the order of north, south, east, west direction:
  if no obstacle or gap in that direction OR location has not been visited **then**
    add direction to available move set D_i
  end if
  if D_i is empty **then**
    move towards a random direction
  else
    select the first element in D_i and perform that move
  end if
end while

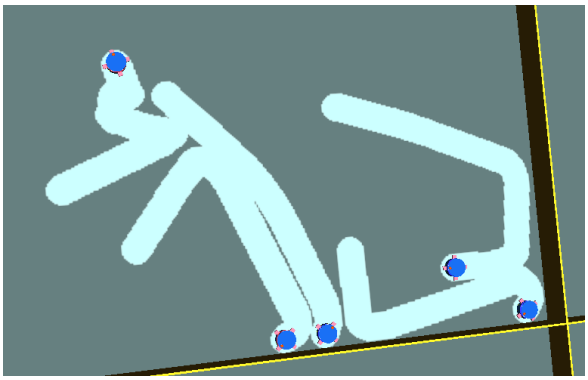


Fig. 4.11 The neighbourhood search heuristic

readings D_s

$$Dr(t) = \begin{cases} \arg\max(D_s(t)), & \text{if } r_u(t) < threshold \\ Dr(t-1), & \text{otherwise.} \end{cases}$$

The heuristic behaviour is shown in Figure 4.11.

Algorithm 9 Heuristic: Neighbourhood Search

```
1: for every time step do
2:    if obtained reward R_i < T_neigh then
3:       read from dirt sensors, update expected rewards R̂_d
4:       move towards the direction d_i ← arg max R̂_d
5:    else
6:       d_i ← d_{i−1}
7:    end if
8: end for
```

4.4 Group Learning of Heuristics

In the previous section of this chapter we have presented an MAB-based hyper-heuristic method which has validated the effectiveness of the proposed hyper-heuristic framework over a range of environments. The robots learn individually using their local observations and rewards, and whether the group should accept a heuristic that is beneficial for the whole group is determined through a voting-based decision making mechanism as described in Section 4.2.4. This section discusses how other means of group decision making can improve the learning of heuristics. While the voting mechanism in Section 4.2.4 only shares robots' proposal and votes with other robots, which is good for systems with limited communication capacity, this section introduces methods that makes decisions through sharing knowledge using local robot communication.

Ozcan et al. [99] has found that group decision making strategies can act as good move acceptance methods in hyper-heuristic algorithms where multiple moves are available. In our proposed multi-robot hyper-heuristic framework, group decision making is used to determine the acceptance of heuristics.

Recall in Section 4.2.4, we introduced a group voting mechanism that chooses the heuristic that is preferred by the majority of the group. Since robots explore the environment and continuously learn which heuristics are best from local observations and rewards, their preference is related to their own benefit.

In the voting mechanism, 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.

4.4.1 Methodology of Group Learning with Knowledge Sharing

In this section, we discuss methods that allow robots to share a part of their heuristics scores to help other robots select heuristics. Robots iteratively learn their own heuristic score table according to the applied heuristics' past performances. After updating the heuristic score table for the robot swarm, each individual robot may have a different score. Hence a collective decision making strategy is proposed to determine the best heuristic for the whole swarm for the next time interval. Similar to the voting mechanism, which requires each robot to vote on a candidate heuristic, this section proposes two other strategies which are also inspired by social decision making [124], to identify the heuristic that reflects the group interest.

For a group of N robots and K heuristics, the heuristic score matrix of the swarm is:

$$\hat{H} = \begin{bmatrix} \hat{h}^{11} & \hat{h}^{12} & \hat{h}^{13} & \dots & \hat{h}^{1K} \\ \hat{h}^{21} & \hat{h}^{22} & \hat{h}^{23} & \dots & \hat{h}^{2K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{h}^{N1} & \hat{h}^{N2} & \hat{h}^{N3} & \dots & \hat{h}^{NK} \end{bmatrix}$$

The first strategy calculates the sum of each heuristic score across all robots in the group, and the heuristic with maximum total score is selected:

$$k_{selected} = \arg\max_{1 \leq k \leq K} \sum_{n=1}^{N} \hat{h}^{nk} \tag{4.7}$$

This strategy aims to maximise the total group performance score for the next iteration, so is named MAX-SUM.

The second strategy is named MAX-MIN as it finds the heuristic with the best minimum score. If the minimum score of each heuristic, and selects the heuristic that has the maximum minimum score.

$$k_{selected} = \arg\max_{1 \leq k \leq K} \{\arg\min_{1 \leq n \leq N} \hat{h}^{nk}\} \tag{4.8}$$

In this way, the selection of heuristics utilises not only the preference of each robot, but also their own evaluations of heuristics. The MAX-SUM strategy aims to maximise the overall performance of all robots, while the MAX-MIN aims to make sure no single robots are behaving very badly.

4.4.2 Implementation with Swarm Robots

With the proposed MAB-based hyper-heuristic algorithm in Section 4.2, 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 Section 4.2.4.

With the voting mechanism in Section 4.2.4, any robot can propose a candidate heuristic for the next time interval to the group, and the robots send communication messages to each other as described the diagrams in Figure 4.2. A robot first 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 majority of the group accepts the proposed heuristic, the group will collectively decide 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.

To integrate the knowledge sharing decision-making with the decentralised robot group, after the heuristic scores are updated, any robot can request heuristic scores from other robots to determine the next heuristic. In order to physically achieve the collective decision process described in Section 4.4.1, the robots send communication messages to each other following the diagrams in Figure 4.12. When a robot has received a request for scores, they will send the line in their heuristic score table $[\hat{h}^1, \hat{h}^2, \hat{h}^3 ... \hat{h}^k]$ which indicates the score for each heuristic after the heuristic in the last time interval has been applied. After a robot has collected heuristic scores of neighbouring robots in the group, it uses the MAX-SUM or MAX-MIN method to select a heuristic and update the other robots.

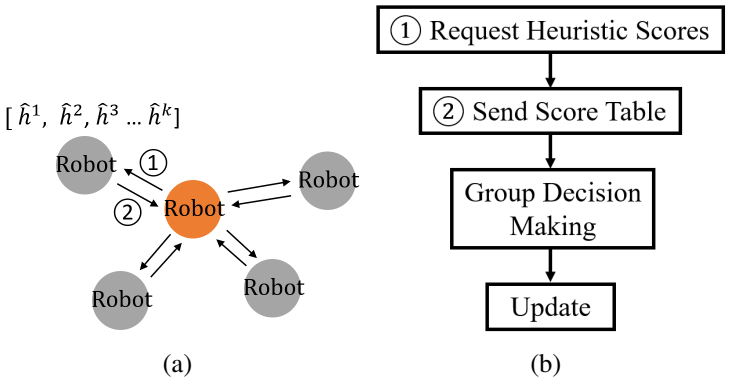


Fig. 4.12 Implementation of Group Decision Making.

4.5 Experiments and Results

This section introduces the swarm robot experiments in 3D simulation in order to verify the effectiveness of the proposed hyper-heuristic algorithm. The application area is surface 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 may sweep, crowd or scatter on the surface according to the environmental layout. No single behaviour is able to clean multiple surfaces, so the problem requires combinations of heuristics. In the real-world, cleaning of multiple surfaces such as the outer surfaces of a building is often complex and large. It is usually costly to build a map of each surface for robot cleaners. Even if a map is built, during the cleaning process, the changes caused by windows opening/closing or decorations will make the map become outdated and inaccurate. In such a dynamic environment, swarm behaviours need to be adaptive and decentralised.

In the experiments conducted in this section, the robots are each programmed with the multi-armed bandit based online-learning robot hyper-heuristic algorithm described in Section 4.2. The robot hardware, simulator and environmental setups are described in Chapter 3.

The hyper-parameter tuning for the robotic 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, the Iterated Racing (*irace*) tuning method is used because it is effective and prunes the space of parameter value combinations that have to be checked, and the details to the method can be found in [74]. The tuning results are shown in Table 4.2, and are used as the parameters in all experiments.

Table 4.2 Tuned values of hyper-parameters and controller parameters.

(a) Heuristic Parameters

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

(b) 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	$40s$

4.5.1 Robustness In Different Environments

Four different layouts of facade surfaces are used in the experiments as shown in Fig. 4.13. Environment (a) is a flat surface with the dimensions 8m $\times$ 8m, 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 4.13 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. Therefore the termination criteria of the algorithm is terminating after 50 iterations. 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.

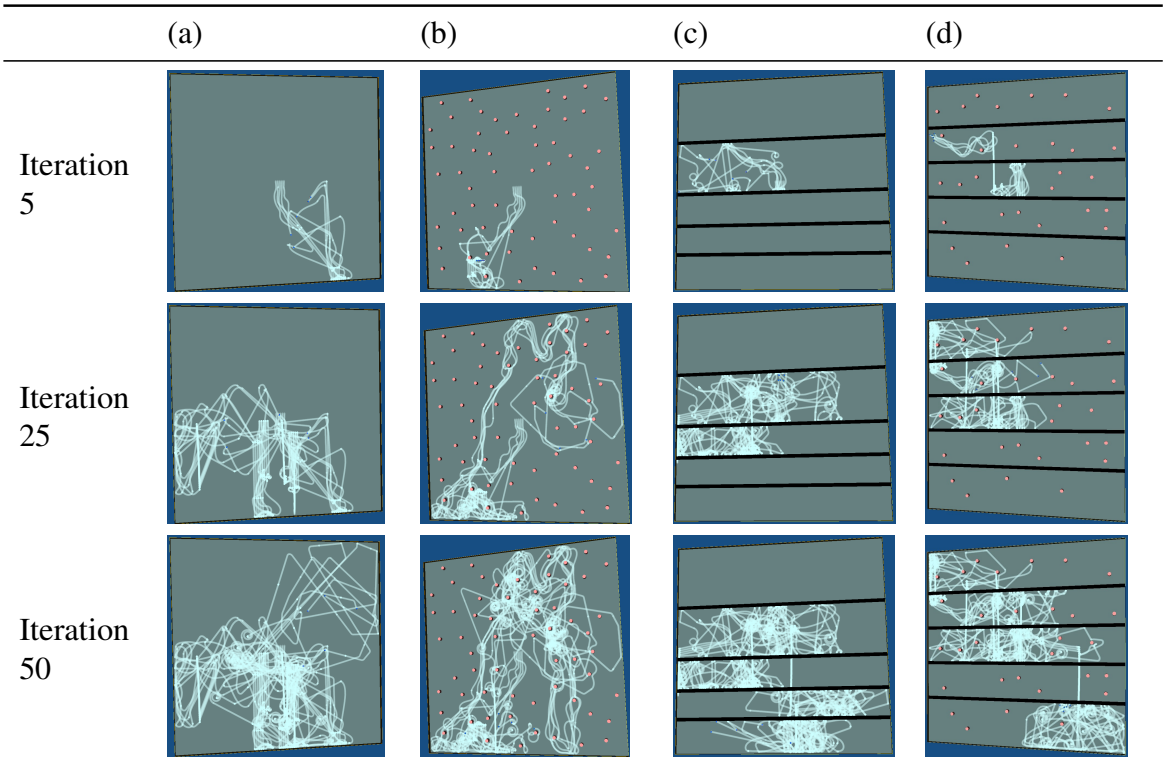
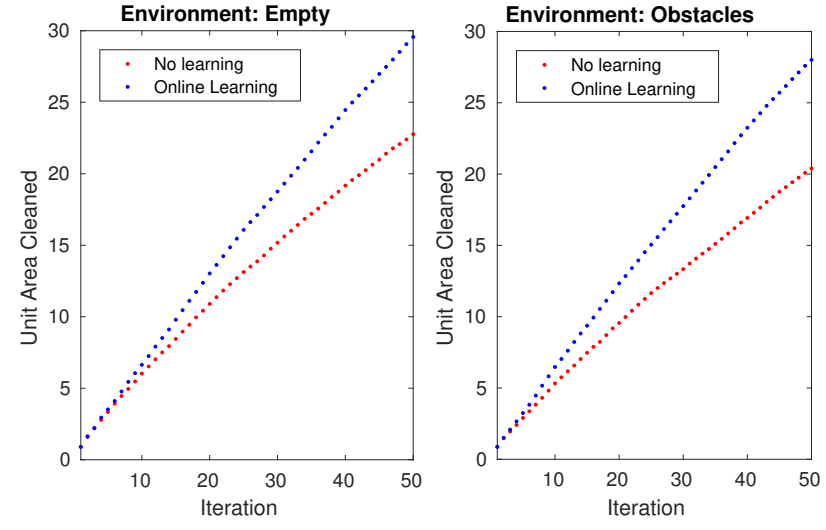
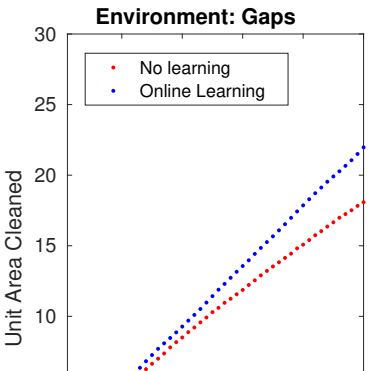


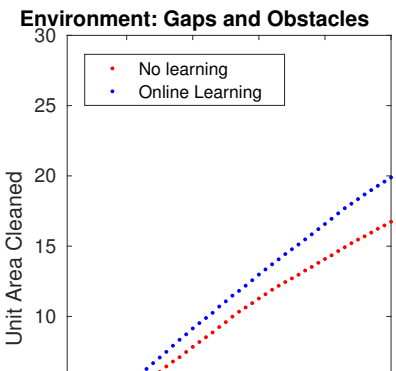
Fig. 4.13 Cleaning progress of the swarm at 5, 25 and 50 iterations in four types of environmental layouts: (a) flat empty surface, (b) surface with obstacles (indicated by red blobs), (c) five surfaces separated by gaps (black stripes), and (d) five separated surfaces with obstacles.



(a)

(b)





4.5.2 Comparison of Heuristic Selection Methods

In this part we compare three different heuristic selection methods as discussed in Section 4.2.4: Roulette Wheel selection, Greedy selection and Simple Random selection, which randomly samples heuristics from the repository. The robots are simulated cleaning at the maximum speed of $2.98m^2$ per time interval of $40s$, which for simplicity, we define as one unit area. Each group performs cleaning tasks for 200 runs across the four types of layouts (Fig. 4.13). Fig. 4.15 plots the performance distribution grouped by environment types.

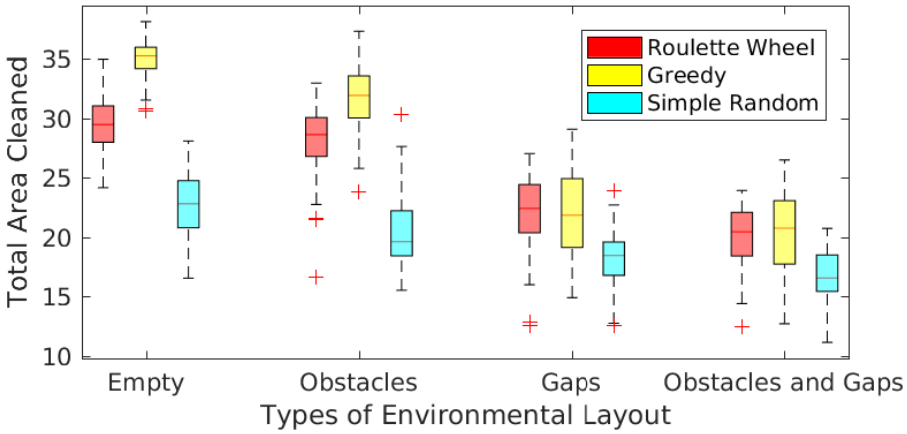


Fig. 4.15 Comparing three hyper-heuristic selection methods: Roulette Wheel, Greedy and Simple Random over four types of environments.

Through Mann-Whitney U tests [81], 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.

4.5.3 Learning Comparing to No Learning

We further investigate the performance in each environment in Figure 4.14. 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%.

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. 4.16 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 behaviour.

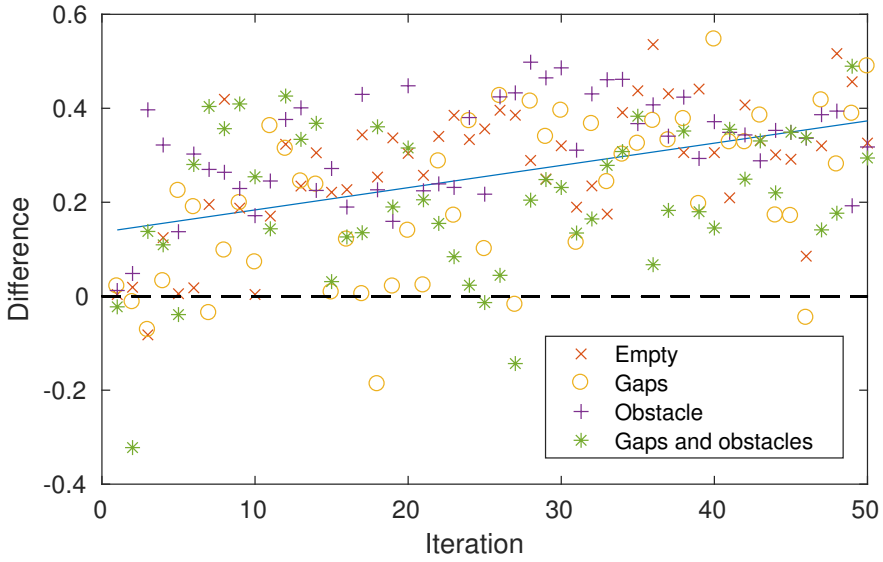


Fig. 4.16 Performance difference for every iteration between online learning hyper-heuristics and no learning.

This demonstrates 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 in a real-world scenario, 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.

4.5.4 Comparison of Group Learning Strategies

The aim of experiments in this section is to compare the group learning strategies proposed in Section 4.4. We first discuss the robustness of the strategies in different scenarios.

The same environments and environmental setups as used in previous sections are used again for consistency, and five robots are used in experiments. The hyper-heuristic implementation is tested in four types of environments shown in Fig. 4.17. To refresh,

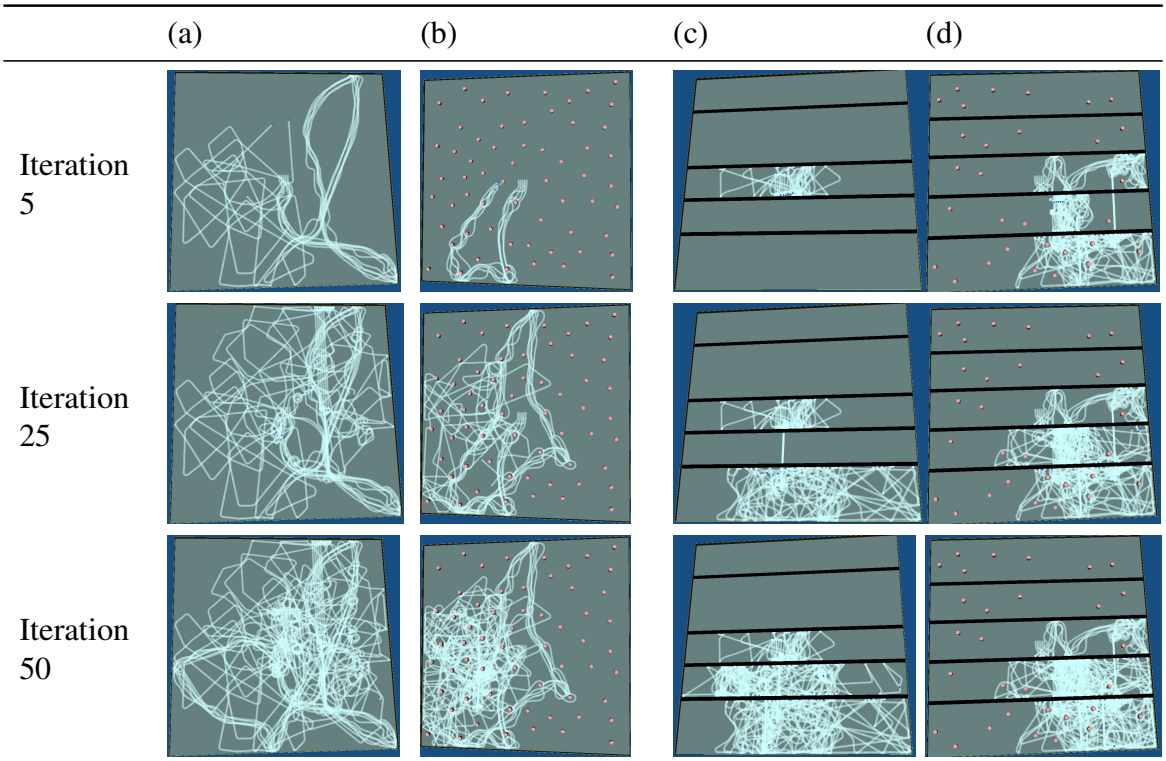


Fig. 4.17 Cleaning progress of the swarm at 5, 25 and 50 iterations in four types of environmental layouts: (a) flat empty surface, (b) surface with obstacles (indicated by red blobs), (c) five surfaces separated by gaps (black stripes), and (d) five separated surfaces with obstacles.

Environment (a) is a flat surface that is 8m $\times$ 8m, 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 obstacles and gaps are randomly placed in the environment, and are different in each experimental run. Robots have no prior knowledge of which heuristics are suitable for a specific environment.

Results from Scenarios (a) and (b) show that the robots are able to perform the cleaning task continuously and robustly with the presence of walls and obstacles, and those from (c) and (d) show that robots are able to move across surfaces to clean.

To show the effectiveness of the collective decision making, we compared the results with the benchmark method in Section 4.2.4 where iterative voting is used to determine the best heuristic over 30 runs under each experimental condition. The benchmark method uses

Table 4.3 Comparing Mean and Std MAX-SUM, MAX-MIN with Voting in surface cleaning over 30 runs

Environment	Strategy	Iteration 5	Iteration 10	Iteration 25	Iteration 50	Std
Empty	Voting	3.5190	6.6389	16.0767	29.5652	2.6209
	MAX-SUM	3.9862 ‡	7.8382 ‡	18.7146 ‡	34.8393 ‡	2.0073
	MAX-MIN	3.7504 ⊖	7.4416 ‡	19.0949 ‡	35.1158 ‡	1.8367
Obstacles	Voting	3.2409	6.4697	15.0475	28.0052	3.1566
	MAX-SUM	3.9173 ‡	7.5398 ‡	17.8297 ‡	32.8875 ‡	2.3756
	MAX-MIN	3.7029 ‡	6.7881 ⊖	17.2879 ‡	32.2359 ‡	2.0985
Gaps	Voting	2.5055	4.9182	11.4324	21.9716	3.2797
	MAX-SUM	3.0322 ‡	5.9019 ‡	14.3134 ‡	25.3342 ‡	2.4588
	MAX-MIN	2.7628 ⊖	5.8101 ‡	12.7129 ‡	23.8899 ‡	2.1364
Gaps and Obstacles	Voting	2.4350	4.8998	11.0722	19.8920	2.8743
	MAX-SUM	2.5812 ⊖	5.6010 ‡	13.2607 ‡	24.4293 ‡	3.6064
	MAX-MIN	2.5870 ⊖	5.5249 ‡	12.6025 ‡	23.8831 ‡	1.9721
Overall	Voting	2.9251	5.7317	13.4072	24.8585	5.0173
	MAX-SUM	3.3792 ‡	6.7203 ‡	16.0296 ‡	29.3726 ‡	5.2839
	MAX-MIN	3.2008 ‡	6.3912 ‡	15.4245 ‡	28.7812 ‡	5.4000

the Roulette Wheel selection method because experiments in Section 4.5.2 indicate that RW selection with the voting algorithm has the best performance so far overall. Table 4.3 detailed the performance increase with MAX-SUM hyper-heuristics and MAX-MIN hyper-heuristics. The performance are measured from the total number of unit area cleaned, as mentioned in Section 4.5.2. Mann-Whitney U tests have been performed on the results. A ‡ symbol indicates a p value < 0.05 comparing to voting, while $\ominus$ indicates a p value > 0.05.

In this part we used the voting method in Section 4.2.4 as a benchmark, and compare the two proposed group learning strategies MAX-SUM and MAX-MIN with it. The benchmark method requires a candidate heuristic to be selected and proposed to the group, and according

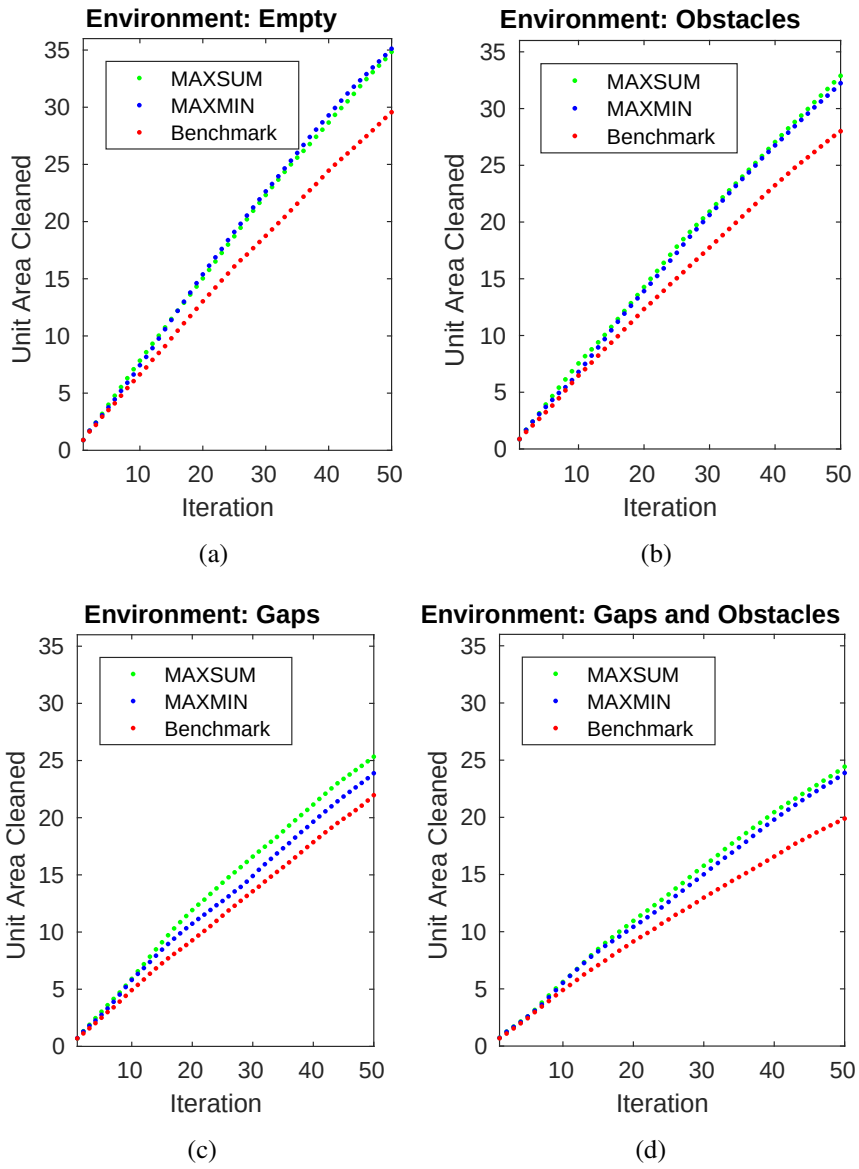


Fig. 4.18 Comparing mean performance of MAX-SUM, MAX-MIN and the benchmark method over 30 experimental runs

to each robot's acceptance criteria, the heuristic accepted by the majority of the swarm will be the group heuristic.

Each group performs cleaning tasks for 30 runs under each of the four types of layouts. Fig. 4.18 plots the mean performance of the proposed group decision making strategies compared to benchmark for every iteration, where the objective is to maximise the area

cleaned by the robot swarm. Both MAX-SUM and MAX-MIN outperform the benchmark in all four environments, overall by 18% and 15.8% respectively.

After applying Mann-Whitney U tests on MAX-SUM performances and MAX-MIN overall performances over 30 runs, we have found that under the environment with gaps, the p value is $0.0096 < 0.05$, indicating that MAX-SUM is better than MAX-MIN. For the three other environments there are no statistical differences between these two approaches, although the average performances of MAX-SUM is slightly better. Therefore we conclude that knowledge sharing group learning strategies outperform the voting-based strategy, and MAX-SUM performs the best among the three strategies.

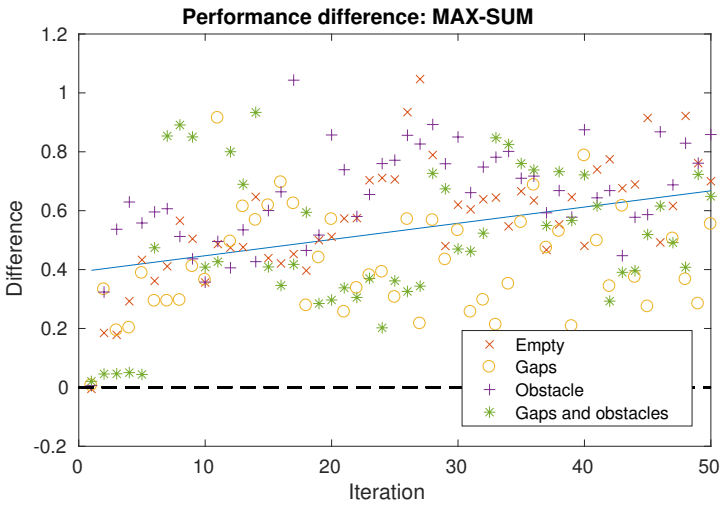
In Figure 4.19, we plot the performance improvements from our methods compared with no learning during the task execution process. No learning means randomly selecting heuristics at all decision points. The Y-axis of Figure 4.19 is the performance difference at each iteration calculated from

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

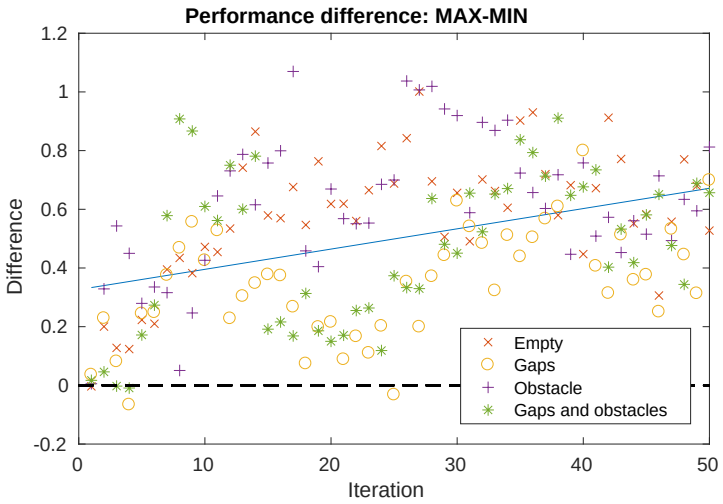
where f_i^L is the total area been cleaned at iteration t by the swarms with learning, and f_i^{NL} is without learning. It can be seen that for the majority (97%) of iterations, the differences are greater than 0. Furthermore, the differences increase over time, as indicated by the blue line which represents the trend. By the end of 50 iterations, both MAX-SUM and MAX-MIN have achieved approximately 67% improvement over the no-learning counterparts.

4.6 Summary

This chapter addressed the first research question and 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



(a)



(b)

Fig. 4.19 No-learning vs. MAX-SUM and MAX-MIN on four scenarios.

environment types were used as test scenarios. The experiments verify that the robot swarm can adapt in different unseen and dynamic scenarios and automatically find appropriate actions to fulfil the given task without manual programming and centralised control. The experiments also show 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.

This chapter also integrated novel group learning of heuristics for the proposed multi-robot hyper-heuristic framework. The group learning is achieved through local communication between robots, and the selection of heuristics is based on relevant knowledge sharing by other robots. Through simulations and experiments, significant improvements have been achieved across all scenarios comparing to the benchmark MAB-based hyper-heuristics in Section 4.2.

Chapter 5

Learning Heuristics with Q-Learning

The previous chapter described the novel MAB-based hyper-heuristic approach for sequencing distributed swarm behaviours, and the enhancements achieved by adapting various group learning strategies. The online learning method allows decentralised robot swarms to be adaptive in dynamic and unknown environments, and demonstrated the effectiveness of the use of the hyper-heuristic framework in coordinating swarm robots. This chapter describes how we further improve the hyper-heuristic method with Q-learning of heuristics, which by utilising local observations in the selection of heuristics, improves the heuristic construction and the learning capability.

In the MAB-based hyper-heuristic algorithm in Section 4.2, the learning of heuristics is based purely on feedback from the objective function. Although the heuristics utilise all the sensor data and environmental information to select actions, in the selection heuristics, no other environment-related information is involved in the decision-making. Most hyper-heuristic algorithms have this design idea of minimising "domain knowledge" in the hyper-heuristic layer, in order for the hyper-heuristic algorithms to be generalisable to a wider range of problems and scenarios. In the domain of robotics, however, the sensor and communication setup for a robot does not usually change from task to task. It can be controlled by different heuristics or perform different tasks, but the set of sensory readings available to the robot is

universal for all tasks and scenarios, the only difference being some sensory information is useful and some is not. Therefore, utilising observations of the environment should not harm the generality of hyper-heuristics for robots. Therefore, reinforcement learning, where the decisions are made according to "states", which consist of observations of the environment, can be an applicable method to further improve the hyper-heuristic framework.

Notice in the MAB-based hyper-heuristic implementation, each heuristic operates on the swarm level, which requires the participation of all robots. This method does not accommodate for robots separating into sub-groups or rejoining a larger group. Reinforcement learning, however, can fully take advantage of the fact that each robot can act as an intelligent agent and form its own policies to act upon while maintaining cooperation with other robots on a hyper-heuristic level.

In order to plan for each robot in a decentralised manner, multi-agent Q-learning is a good option because the information of neighbouring robots can be embedded in the state representations and the policy. For each robot, the policy it develops allows it to act according to its own local observation, and the policy can be different from other robots.

Q-learning hyper-heuristics proposed in this chapter, however, allow robots that use different heuristics to learn together and develop policies that cooperate across different groups of robots. The cooperation comes from the self-organised teaming mechanism that will be introduced in this chapter. The Q-learning-based hyper-heuristic makes the robot group achieve better performance.

We consider an environment that can not be monitored or observed globally. A set of robots U is equipped with on-board processing, sensing and communication devices for local information. The robots aim to fulfil a collaborative task. There is no central "brain" for all robots, thus each robot is considered an independent machine learning agent. A robot only relies on its on-board sensors to observe its local environment, making the environment non-stationary in the view of each robot. There is noise in robot sensor readings and the

inter-robot communication capacity is limited, which prohibits sharing memory or large amounts of experience data.

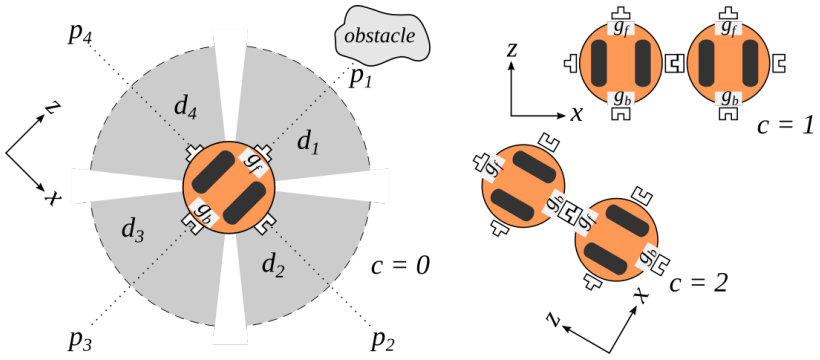


Fig. 5.1 The self-assembling robot state representation

Reinforcement Learning (RL) considers tasks that can be modelled with Markov Decision Process (MDP). Machine learning agents interact with the environment and receive rewards from the environment. Reinforcement learning aims to maximise the long-term reward.

In action-based reinforcement learning, the choice of state representations is usually problem-specific. For learning heuristics, abstract states are often used to make the learning algorithm more generalisable to a range of problems. In the domain of robotics, the feedback from the environment comes only from sensors and communication devices installed on the robot. The hardware sensing range and communication capability constrain the state space for each learning agent, where states are representations of features extracted from sensor readings. Here we introduce our state, action, heuristic and reward definitions in the Q-learning hyper-heuristic framework:

- **States**. In this study, the state representation is formed from raw data sampled periodically from the robot hardware. At each time step, a robot observes the local environment state $s_u \in S$. The set of robot states

$$S = \{s_u = (D_s, P_s, G_s, c, nr)\} \tag{5.1}$$

where D_s denotes the dirt sensing camera observations, P_s is the array of proximity sensor readings, G_s is the gap sensor reading, c denotes the number of neighbouring robots that are physically connected with the current robot. $nr \in \{0,1\}$ where 0 for no reachable robot and 1 for having at least one reachable robot around.

In this study, given self-assembling robots with sensory arrangements shown in Figure 5.1, the robot has 4 dirt sensing cameras installed, therefore $D_s = \{d_1, d_2, d_3, d_4\}$. An array of 4 proximity sensors is equipped to face robot front, left, back and right to avoid collision. $P_s = \{p_1, p_2, p_3, p_4\}$. Gap sensors $G_s = \{g_f, g_b\}$ are at the bottom front and bottom back of the robot. The robot has two axial gripper arms and two side arms. $c \in \{0,1\}$, which represents if the robot is physically connected to other robots.

- **Action**. The action space is defined by single robot control actions. For any robot $u \in U$, $s_u \in S$, the set of actions

$$A(s_u) = \{a_u = (v, \omega, m, g) | v \in V, \omega \in \Omega, m \in M, g \in G\} \tag{5.2}$$

where $v \in V$ denotes the robot linear velocity and the set of allowed velocity $V \in [v_{min}, v_{max}]$. The set of robot angular velocity $\Omega \in [\omega_{min}, \omega_{max}]$, and the set of inter-robot messages $M \in \{m_1, m_2, ...\}$ and physical robot gripper connect/disconnect action set $G \in \{0,1,2\}$.

- **Heuristic**. The set of heuristics H is a collection of robot control algorithms. For any $h_u \in H$, h_u selects an action $a_u \in A$ given a robot local environment state $s_u \in S$. Let robot $u \in U$, $a_u(t) = \mathscr{H}_u(s_u^t)$ where $\mathscr{H}_u$ represents the algorithm of heuristic h_u that outputs the action $a_u \in A$ for every t.

- **Rewards**. As robots interact with the environment, apart from observing the state, they also have sensors that receive feedback about the task progress, which makes up the reward r_u in reinforcement learning. In the application of surface cleaning, robots

sample from the dirt sensor continuously on the bottom of the robot and calculate collected dirt d_u during every time interval with length T. The reward received in each time interval is:

$$r_u = \sum_{t=0}^{T} d_u/t \tag{5.3}$$

5.1 Action Based Learning

We first give the background of commonly used multi-agent reinforcement learning, which learns action-based policies [68]. At each time step t, in the set of agents U, agent $u \in U$ observes its local environment state $s_u \in S$ and takes an action $a_u \in A$ to interact with the environment and other agents, and obtains reward r_u from the environment, as . The environment advances to the next state $s'_u \in S$ at time $t+1$, following a state transition probability $p(s'_u|s_u, a_u)$. The policy for agent u is denoted $\pi_u : S \to A$. Reinforcement learning aims to find the optimal policy $\pi^*_u : S \to A$. This study considers infinite horizon, and the state-value function is defined as

$$Q_\pi(s_u, a_u) = E_\pi[\sum_{t=0}^{\infty} \gamma^t r^t_u | s_u = s^t_u, a_u = s^t_u] \tag{5.4}$$

Estimates of the optimal values can be learned using Q-learning. Agents explore in the environment and Q-values are iteratively updated by every agent using the Bellman Equation:

$$Q(s_u, a_u) \leftarrow (1-\alpha) * Q(s_u, a_u) + \alpha * (r_u + \gamma * \max_{a_u} Q(s'_u, a_u)) \tag{5.5}$$

where γ denotes the discount factor for future rewards and α is the learning rate, and r_u is defined in Equation 5.3.

To balance exploration and exploitation when agents interact with the environment, ε-greedy algorithm controls the agent to choose a random action with probability ε, otherwise $\pi = \arg\max_{a_u} Q(s_u, a_u)$.

In the domain of robotics, the state space and action space are both continuous. In this section, we use the reinforcement learning with the bins method to transform the state and action space respectively into discrete and finite space. All dimensions are divided into "bins" according to the lower and upper limits, then a tabular method can be applied. The Q-table $Q(s_u, a_u)$.

Algorithm 10 Decentralised Reinforcement Learning over Actions

for $\forall u \in U$ **do**
  Initialise $Q(s_u, a_u)$
  while task is not terminated **do**
    if probability $> \varepsilon$ **then**
      $a_u^t \leftarrow \arg\max_{a_u} Q(s_u^t, a_u)$
    else
      Select a random action a_u^t
    end if
    Apply action a_u^t
    Observe state s_u^{t+1}, obtain reward r_u
    $Q(s_u^t, a_u^t) \leftarrow (1-\alpha) * Q(s_u^t, a_u^t) + \alpha * (r_u + \gamma * \max_{a_u} Q(s_u^{t+1}, a_u))$
    $t \leftarrow t+1$
  end while
end for

It is known in general that learning independent Q models proves to be inefficient in multi-robot coordination because from the perspective of a given agent, the remaining environment is non-stationary. In the next section, we address this problem by introducing the learning of heuristics.

5.2 Heuristics Based Learning

Standard reinforcement learning agents explore the environment and obtain *State, Action and Reward* tuples, while this section proposes learning on *State, Heuristic and Reward*. Fig. 5.2 shows the structure of the decentralised multi-robot learning framework, and the rest of this section describes the robot-environment interaction model, policy optimisation, self-organised teaming and rewards sharing in detail.

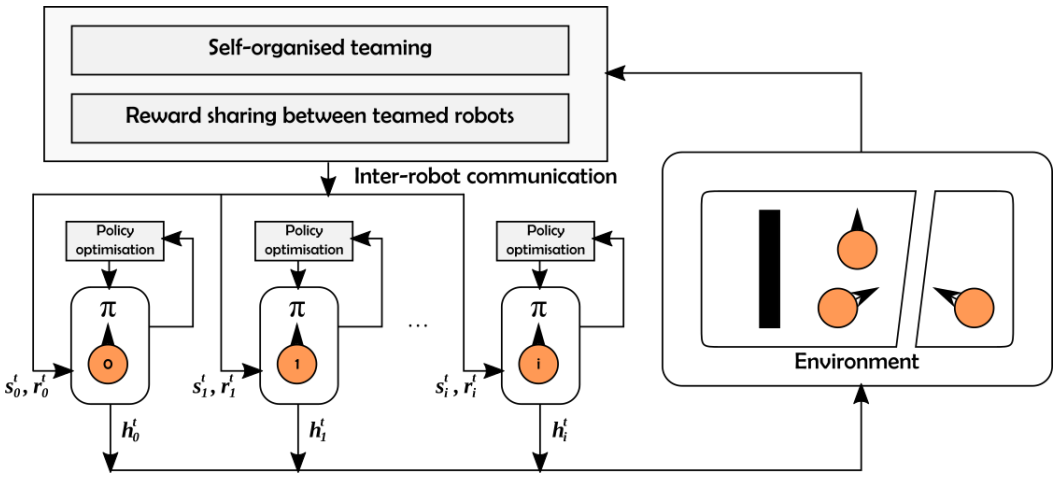


Fig. 5.2 The framework of decentralised multi-robot learning on heuristics

At each time step t, agent $u \in U$ observes the local environment state $s_u \in S$ and obtains reward r_u from the environment. Robots choose a heuristic h_u^t at each time step from the heuristic set H, apply the heuristic to the current state and generate an action $a_u(t) = \mathscr{H}_u(s_u^t)$. The robot performs the action to interact with the environment and obtains reward r_u. Each robot performs policy optimisation independently and locally. Implementation-wise, the robot's on-board processor is loaded with control programs that take s_u as input and outputs a_u according to the selected heuristic. The algorithm descriptions used in this study are given at the end of this section.

Algorithm 11 is a Q-learning implementation of the above framework. It is executed in parallel in each robot. While the task is not finished, at each time step t, a robot selects a heuristic. With the selected h_u^t, the robot applies the corresponding heuristic and generates

Algorithm 11 Decentralised Reinforcement Learning over Heuristics

for $\forall u \in U$ **do**
  Initialise $Q(s_u, h_u)$
  while task is not terminated **do**
    if probability $> \varepsilon$ **then**
      $h_u^t \leftarrow \arg\max_{h_u} Q(s_u^t, h_u)$
    else
      Select a random heuristic h_u^t
    end if
    Apply heuristic $a_u^t \leftarrow \mathscr{H}_u(s_u^t)$
    Observe state s_u^{t+1}, obtain reward r_u
    if there is collaborating robot with heuristic h_u^t **then**
      Send r_u to collaborating neighbouring agents, receive r_i from n connected neighbours
      $r_a \leftarrow (r_u + \sum_{i=1}^{n} r_i)/(n+1)$
      $Q(s_u^t, h_u^t) \leftarrow (1-\alpha) * Q(s_u^t, h_u^t) + \alpha * (r_a + \gamma * \max_{h_u} Q(s_u^{t+1}, h_u))$
    else
      $Q(s_u^t, h_u^t) \leftarrow (1-\alpha) * Q(s_u^t, h_u^t) + \alpha * (r_u + \gamma * \max_{h_u} Q(s_u^{t+1}, h_u))$
    end if
    $t \leftarrow t+1$
  end while
end for

a_u^t. The environment advances to the next state after the robots have executed their actions. Then the robots perform the following:

- **Self-organised teaming** In the heuristic repository, there are heuristics that requires collaboration and communication between multiple robots, for example, the bridging heuristic that control robots to physically connect and help each other move across gaps. If a robot has selected a heuristic h_u^t, it will seek to form a team with nearby robots that have selected the same cooperative heuristic. A simple handshake is implemented in which the robot broadcasts its heuristic to neighbouring robots, then it listens for agreement messages returned from robots that have selected the same heuristic. Robots that are self-organised into the same team share the reward. At each time step, teamed robots broadcast their local rewards r_u within the team and each robot replaces

its reward for learning with the average team reward $r_a \leftarrow (r_u + \sum_{i=1}^{n} r_i)/(n+1)$. The collaboration and communication mechanisms are embedded in all cooperative heuristics. A robot automatically leaves the team when a different heuristic is selected according to the current policy. In the scenarios where the robots are connected and moving together, the leaving robot automatically disconnects from the team.

If there are reachable robots that are applying the same heuristic, the averaged team reward r_a is used for policy optimisation, otherwise the robot uses its raw obtained reward r_u.

- **Policy optimisation** To balance the exploration and exploitation, the heuristics are selected using ε-greedy selection, as shown in Algorithm 11 lines 4-8. At every step t, Q-values are updated from each robot's $\{s_u, h_u, r, s'_u\}$, where r can take the value of r_u or r_a as discussed above. Robot Q-function is updated iteratively as follows:

$$Q(s_u, h_u) \leftarrow (1-\alpha) * Q(s_u, h_u) + \alpha * (r_u + \gamma * \max_{h_u} Q(s'_u, h_u)) \tag{5.6}$$

This process is distributed without inter-robot communication. With a probability ε, the robot randomly selects a heuristic from H, otherwise selects the heuristic with maximum $Q(s,h)$.

The inter-robot communication in this process is quick and brief since it is only at the hardware control command level. In other words, this decentralised framework does not require sending past experience, sharing states or sharing rewards. This makes the multi-robot system feasible in environments that can only allow for low communication capacity.

The reason for the improvement Q-learning of heuristics brings over the MAB-based online learning is that Q-learning utilises the environmental observations in selecting heuristics, while the MAB-based hyper-heuristic only uses the heuristic evaluation value. Unlike the MAB-based adaptive hyper-heuristic algorithm which treats robots performing different

heuristics as part of the environment, the Q-learning hyper-heuristic method in this chapter allows robots to cooperate and learn together with robots using different heuristics. This is achieved through the state-heuristic learning model, which allows robots to take the status of other robots as well as the environmental state into account in the decision making of heuristics selection. On the other hand, the MAB-based adaptive hyper-heuristic algorithm treats a robot group as a whole at all times and optimising a robot's own reward does not equate to optimising the group reward. The experiments comparing the two types of hyper-heuristics are given in Section 5.5.

5.3 Group Learning with Advising

In Q-learning hyper-heuristics, each robot learns independently and maintains its own Q-table. This section introduces a method that allows robots to learn faster through communication.

In solving both small-scale and complex problems, reinforcement learning agents are known to take a large number of learning iterations to reach convergence in policy. When multiple agents exist, learning independently in parallel is less efficient than sharing some knowledge and learning collaboratively. While many approaches exist to address this, they either need an expert in the system such as a teacher [130], or require a large amount of communication such as episode sharing. The multi-agent advising framework [24], however, allows multiple agents to advise each other while learning together in the same environment. Any robot can seek advice when it is not sure about what to do, and any other robot that is more confident can give advice.

In [24], authors have identified scenarios where the advising mechanism is useful:

1. A learning agent is in a new state, but the state has been explored by another reachable agent.

2. A learning agent joins a system in which other agents have been learning for some time.

3. A learning agent's learning approach is less efficient than another reachable agent.

Since this thesis only concerns robot swarms that use the same learning algorithm, the proposed advising method is useful in the first two scenarios.

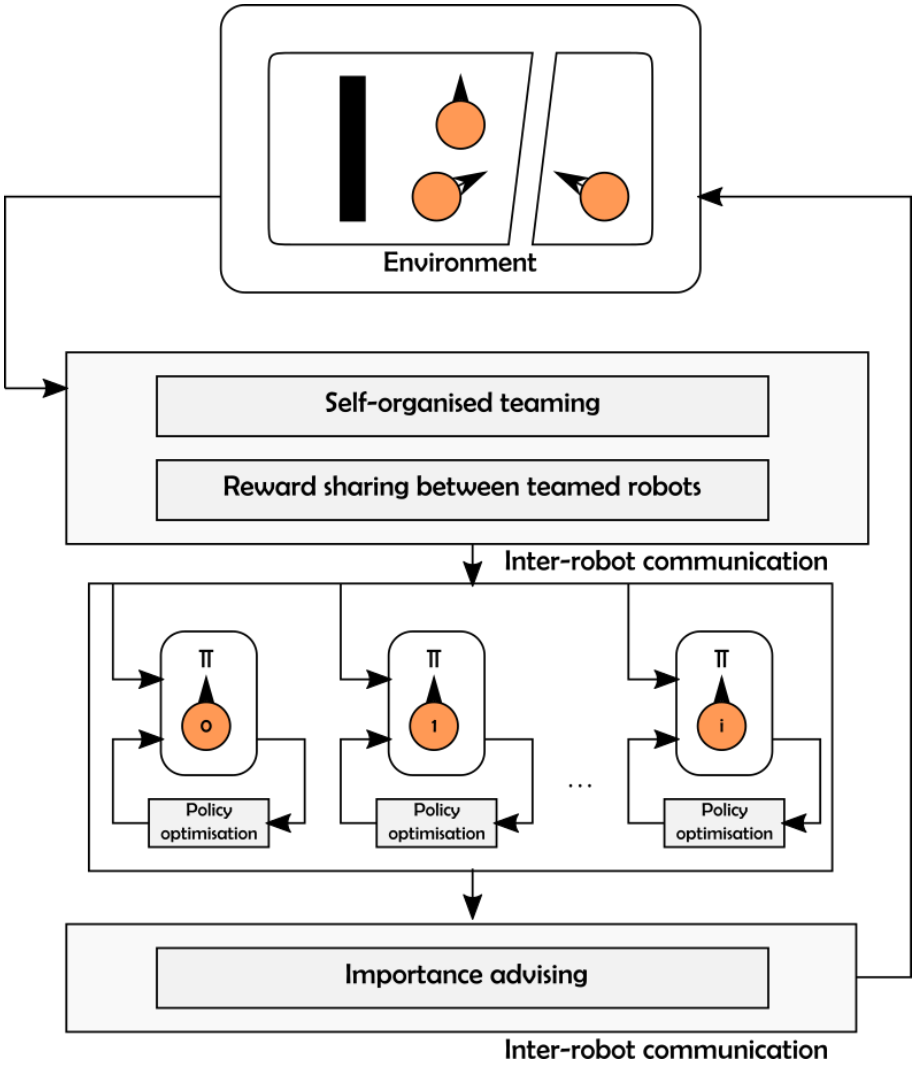


Fig. 5.3 The framework of multi-robot learning with importance advising

Figure 5.3 shows where importance advising lies in the Q-learning hyper-heuristic framework. After the policy optimisation step, described in lines 12-17 in Algorithm 11, the

Algorithm 12 Heuristic Selection with Importance Advising

```
1: for ∀u ∈ U do
2:     ...
3:     Policy optimisation
4:     if state s_u^t has never been observed then
5:         Send request for advice to reachable robots
6:     else
7:         Calculate ask probability P_a(s_u^t) = (1+v)^{-√N(s_u^t)}
8:         if probability < P_a(s_u^t) then
9:             Send request for advice to reachable robots
10:        end if
11:    end if
12:    if there is request sent then
13:        Π_a ← ∅
14:        for all receive answers An(s_u^t) do
15:            Π_a ← Π_a ∪ An(s_u^t)
16:        end for
17:        if Π_a ≠ ∅ then
18:            Perform ε-greedy selection on Π_a
19:        else
20:            Perform the usual ε-greedy selection
21:        end if
22:    else
23:        Perform the usual ε-greedy selection
24:    end if
25: end for
```

importance advising strategy allows a robot to communicate with neighbouring robots, ask and give advise about which heuristic to select next. The pseudo-code for the importance advising algorithm is shown in Algorithm 12. This process happens after lines 12-17 in Algorithm 11, for all robot learning iterations. For a robot $u \in U$, if the current state observed by the robot s_u^t is a new state, the robot will send robot messages to neighbouring robots asking for advice. Otherwise, as shown in lines 6-11 of Algorithm 12, if the current state has been observed before, the probability of asking other robots for advice is defined by

$$P_a(s_u^t) = (1+v)^{-\sqrt{N(s_u^t)}} \tag{5.7}$$

where $N(s_u^t)$ is the number of visits of state s_u^t, and v is the scaling factor that controls the probability of asking for advice. Robots send query requests to reachable robots with probability $P_a(s_u^t)$.

After the asking stage, if the robot has asked for advice, it will listen for answers from replied robots. For all neighbouring robots that have received an advising request, they calculate the importance $I(s_u^t)$ as:

$$I(s_u^t) = \max Q(s_u^t, h) - \min Q(s_u^t, h_u) \tag{5.8}$$

The advising robot prepares the answer $An(s_u^t)$ according to the calculated $I(s_u^t)$, as shown in Equation 5.9, and transmits the answer back to the advisee.

$$An(s_u^t) = \begin{cases} \pi(s_u^t), & \text{if } I(s_u^t) > threshold \\ \emptyset, & \text{otherwise} \end{cases} \tag{5.9}$$

The advisee robot u proceeds with the algorithm at line 13, where it initialises an advised policy $\Pi_a = \emptyset$ for storing received answers. For each answer $An(s_u^t)$ the robot has collected, which can be an empty set or the policy $\pi(s_u^t)$ given by the advising robot, it is stored in Π_a. If the advising policy Π_a is not empty, the robot selects the heuristic that has found to have the maximum expected reward for the state with a probability of $1-\varepsilon$, while selecting a random heuristic with a probability ε. If no advice has been received, or the robot has decided not to ask for advice after line 11, the robot performs the usual ε-greedy heuristic selection strategy on $Q(s_u^t, h_u)$.

5.4 The Updated Heuristic Repository

The heuristic repository in Section 4.3 is re-used here with some modifications. The sweeping and bridging heuristics in the last chapter assume the behaviours are all on the swarm level,

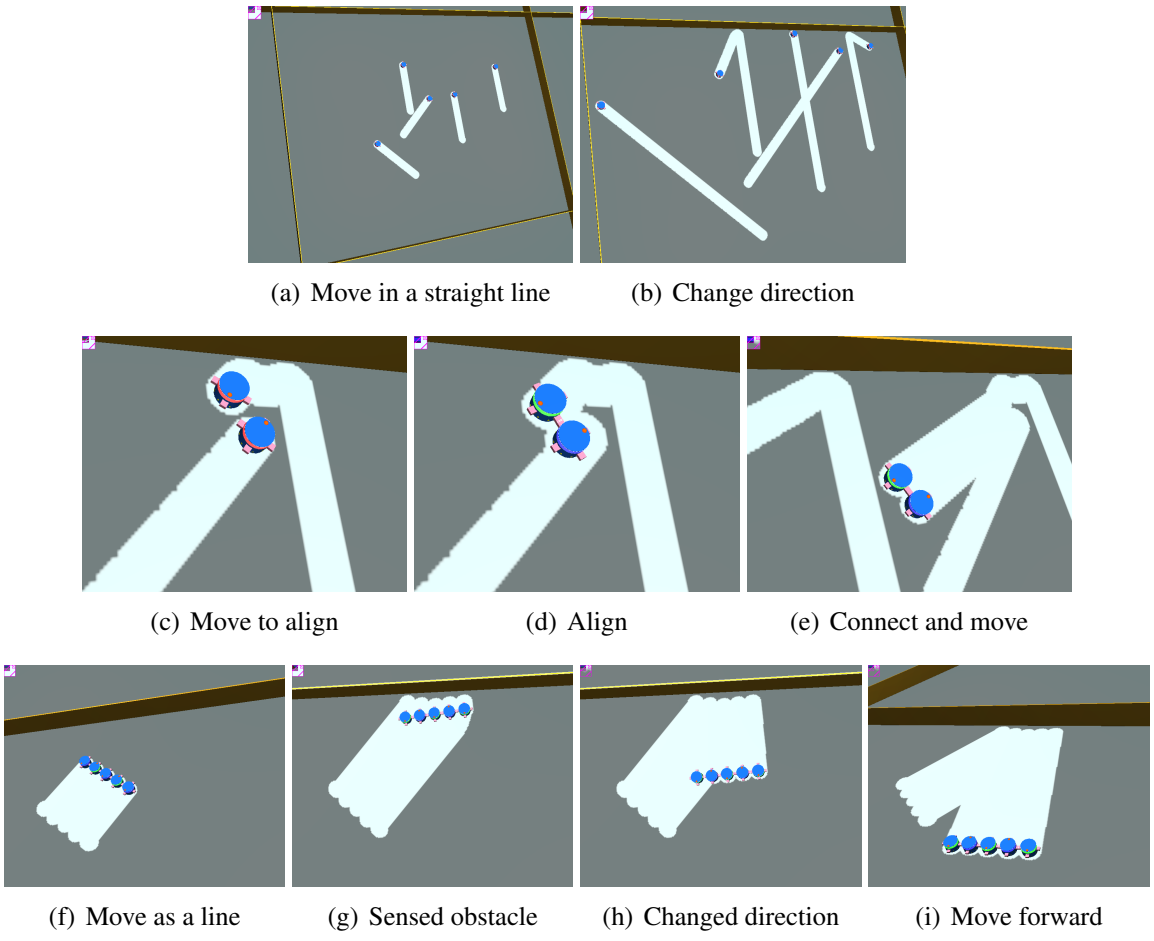
which means all robots participate to perform the heuristic. Those heuristics determine if assembling is completed simply by counting the number of connected robots. In this chapter, the group size can be varied, and the previous way of determining when the self-assembling is completed is no longer applicable.

In order to evaluate the dynamic grouping and regrouping procedure, this section updates the original heuristic repository with new *sweeping* and *bridging* heuristics. These heuristics are able to deal with self-assembling robots leaving or joining the group. In this repository, the "swarm mode" heuristics are the same as the heuristics in Section 4.3: *Exploring*, *Circling*, *Flocking*, *Raster Scan* and *Neighbourhood Search*.

The heuristics all include the same collision avoidance (Algorithm 4) and gap avoidance (Algorithm 5) mechanisms as described in Section 4.3.

5.4.1 Sweeping

As shown in Figure 5.4, the sweeping heuristic controls the robot to physically connect with other robots that are open for connection to form a vertical line. This heuristic has two parts: the robot control algorithm for when the robot is connected to other robots, and the algorithm when the robot is moving by itself. This heuristic utilises distributed control and robots are coordinated only through LED signals.



(a) Move in a straight line (b) Change direction

(c) Move to align (d) Align (e) Connect and move

(f) Move as a line (g) Sensed obstacle (h) Changed direction (i) Move forward

Fig. 5.4 The sweeping heuristic

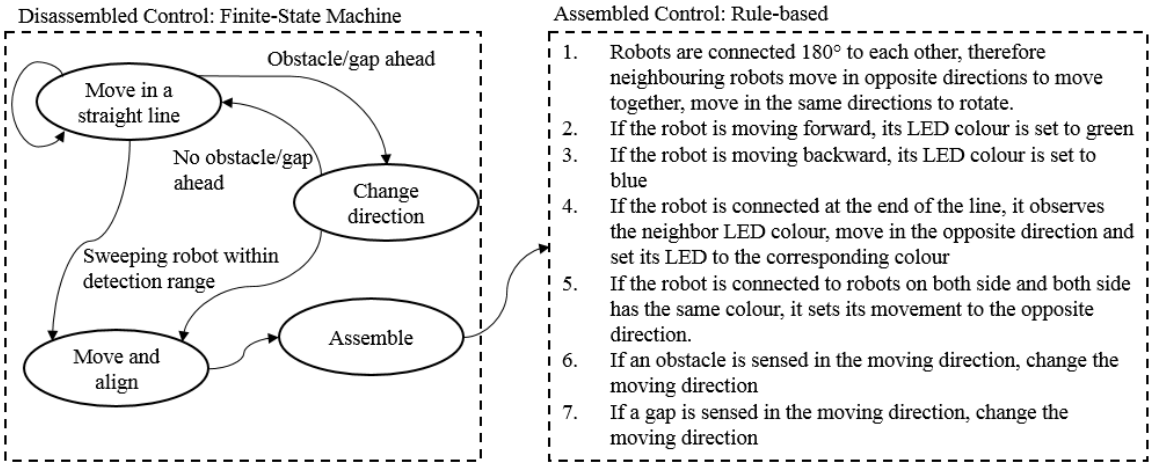


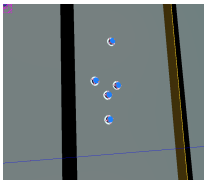
Fig. 5.5 The sweeping heuristic

Figure 5.5 summarises the sweeping heuristic. For each robot that applies the sweeping heuristic, it observes the gripper sensor at every control loop. If the robot is not connected to any robot, it uses a finite-state machine controller as shown in Figure 5.5. The robot moves in a straight line if no obstacle or gap is sensed. If the proximity sensors or gap sensors have sensed obstacles or gaps ahead, the robot will change direction to avoid the obstacle/gap. Once the obstacle/gap is no longer observed by sensors, the robot goes back to the "Move in a straight line" state. From the "Move in a straight line" or "Change direction" state, when there is a robot performing the sweeping heuristic within the detection range, the robot will seek to assemble with it. To do so, the robot calculates the connecting position and moves towards it, aligns the grippers and assembles.

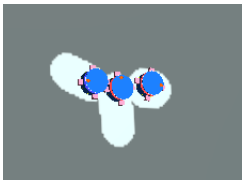
Once the robot is connected to other robots, it uses a rule-based controller. When robots are connected, they need to move together and coordinate their movements. In this heuristic, the robots are connected 180^o to each other, therefore if neighbouring robots' speeds are in opposing directions, the connected line of robots is able to move in a straight line. The robots match neighbouring robots' speed through robot LED colours. Since each robot has cameras that can observe neighbouring robots' LED colours, and sweeping heuristic rules regulate robots to use their LED colours to indicate their movement direction, robots do not need to use inter-robot messages to coordinate motions.

5.4.2 Bridging (Horizontal and Vertical)

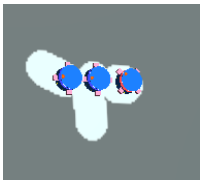
The bridging heuristic algorithm is shown in Figure 5.7. When this heuristic is selected, if the robot is not connected to any others, it moves around in a straight line, avoids obstacles and looks for other bridging robots. The obstacle avoidance mechanism is described as in Algorithm 4. Each bridging robot continuously broadcasts its choice of heuristic and its location to nearby robots, while listening to other robot messages. If there is a match



(a) 5 robots

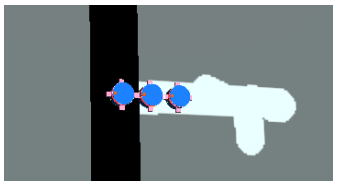


(b) Nearby robots assemble

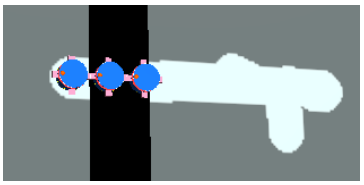


(c) Head and middle robots connected

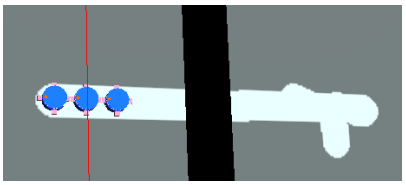
(d) All robots connected



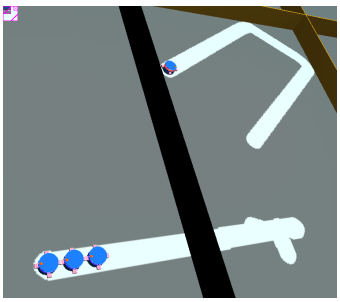
(e) Move forward non-stop



(f) Move forward non-stop



(g) Move forward



(h) An unconnected bridging robot

Fig. 5.6 The bridging heuristic

in heuristic, the robots calculate their own docking locations according to the other robot positions. For each robot with location (x_i, z_i), the alignment rules are:

1. Depending on if it is horizontal bridging or vertical bridging heuristic, a robot compares its own x_i or z_i against its neighbours. If the robot has minimum x or z, it is the head robot
2. If a robot has the maximum x or z, it is a tail robot
3. If a robot is neither head or tail robot, it is a middle robot
4. A head robot rotates at the same position (x_h, z_h) and orients itself in the horizontal or vertical direction

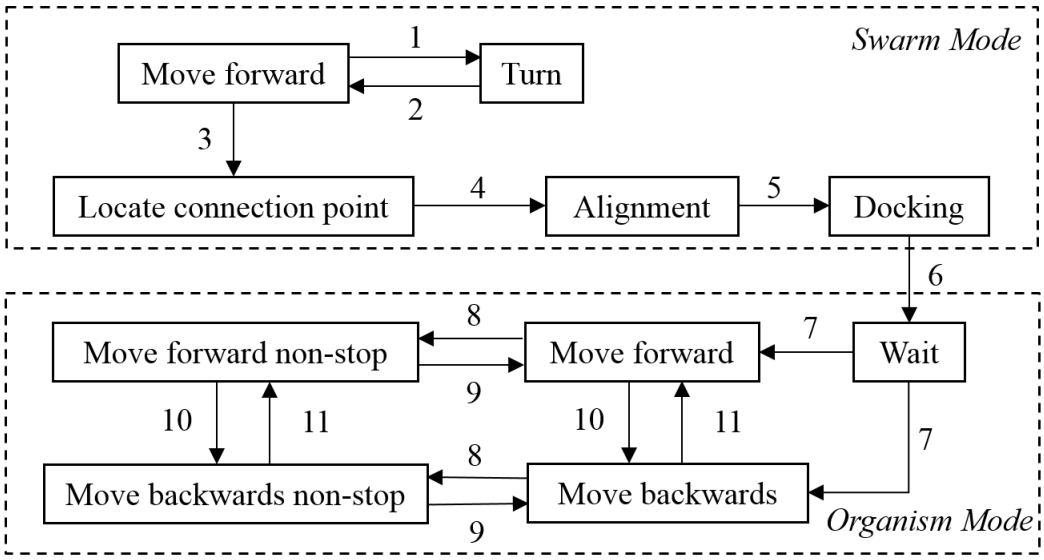


Fig. 5.7 The bridging heuristic: 1 - Obstacle detected; 2 - Oriented away from the obstacle; 3 - bridging robot detected - connection location reached; 4 - connection points located; 5 - aligned and ready to dock; 6 - docking completed; 7 - all nearby bridging robots connected; 8 - crossing a gap; 9 - crossing completed; 10 - front obstacle detected; 11 - back obstacle detected

5. A middle robot or a tail robot finds its rank r_i in the horizontal or vertical direction, and calculates its target location $(x_h + r_i * d_{dock}, z_h)$ for horizontal alignment, and $(x_h, z_h + r_i * d_{dock})$ for vertical alignment

6. After its own docking has been completed, the robot sets its LED colour to green, otherwise sets it to red

7. A robot only connects to robots with LED colour green

8. When a tail robot is docked, it sends move signals to connected robots

9. The tail robot decides whether to move forward or backwards: if there is dirt behind, move backwards, otherwise randomly choose a moving direction

10. A robot starts to move when a move signal is received

When each robot executes these rules in parallel, the robots collectively connect into a line, as shown in Figure 5.6. This connection process describes the behaviour up to the transition 7 in Figure 5.7. When each robot moves forward or backwards with a pre-defined fixed speed, the the new connected-robot organism moves forward or backwards together. If a gap is encountered, condition 8 is satisfied, which means that the organism is on top of a gap, and should not be allowed to change heuristic during this time. This condition brings the organism to the Move forward/backwards non-stop states. In these states, the hyper-heuristic controller is aware that the bridging heuristic should keep being re-selected. When the gap crossing is finished (condition 9), the organism returns to move forward/backwards states. When there is an obstacle detected in the way of the robots, as condition 10 and 11 show, the organism changes direction.

5.5 Experiments and Results

To verify that using Q-learning to learn heuristics is more effective than learning actions, the proposed hyper-heuristic algorithm is evaluated on multiple types of environments. The robots and simulator are described in Chapter 3. In this section, we first compare the tabular Q-learning on heuristics, as proposed in Section 5.2, and the tabular Q-learning on actions, which is described in Algorithm 10 [136], Section 5.1. Then we compare performance with and without group learning, as well as comparisons with the MAB-based hyper-heuristic proposed in Chapter 4. For easy reference, in the rest of this section, we use the "Q-heuristic method' to refer to the decentralised Q-learning over heuristics in Section 5.2, and the "Q-action method" refers to the decentralised Q-learning over actions (Algorithm 10).

The features used as inputs to the reinforcement learning algorithms are binary representations of sensor values, as shown in Table 5.1. Four bits are used to represent dirt sensor values, as shown in Figure 5.1, the four bits correspond to whether there is dirt in region d_1, d_2, d_3 and d_4. A "1" represents that there is dirt in the region and a "0" represents that

there is no dirt in the region. The obstacle and gap representations follow the same principle, "1"s for where there are obstacles and gaps in that direction and "0"s for where there are no obstacles or gaps. A "1" for the connection status bit indicates that the robot is in organism mode and is connected to other robots, while a "0" means the robot is in swarm mode, moving as an individual robot.

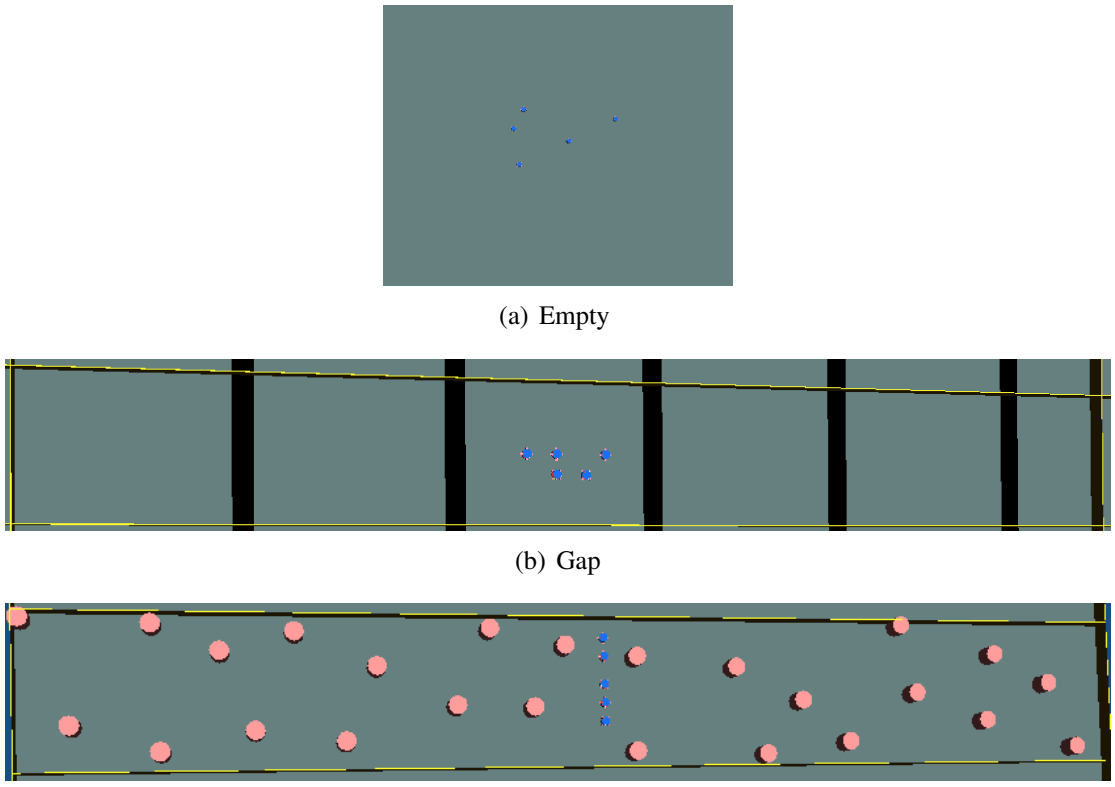
Table 5.1 Feature representation for Q-learning hyper-heuristics

Features	Number of bits
Dirt: front, left, right, back	4
Obstacle: front, left, right, back	4
Gap: front, back	2
Connection status: organism mode, swarm mode	1
Number of neighbours : 0, 1, 2, >2	2

The experiments are conducted in three types of environments shown in Figure 5.8:

1. Empty: Single surface, static environment
2. Obstacle: Environment with obstacles that change positions every 300 iterations
3. Gap: Environment with gaps

The first environment is a simple stationary environment for the purpose of demonstrating the trend. The second environment is dynamic, since the positions of the obstacles change randomly. The gap environment is for demonstrating the performance on multiple surfaces which can be difficult for the swarm. These scenarios are chosen to demonstrate the characteristics of Q-learning hyper-heuristics in this section, and more complex and dynamic scenarios and comparisons are given in later Section 6.2.3.



(a) Empty

(b) Gap

(c) Obstacle

Fig. 5.8 Environments

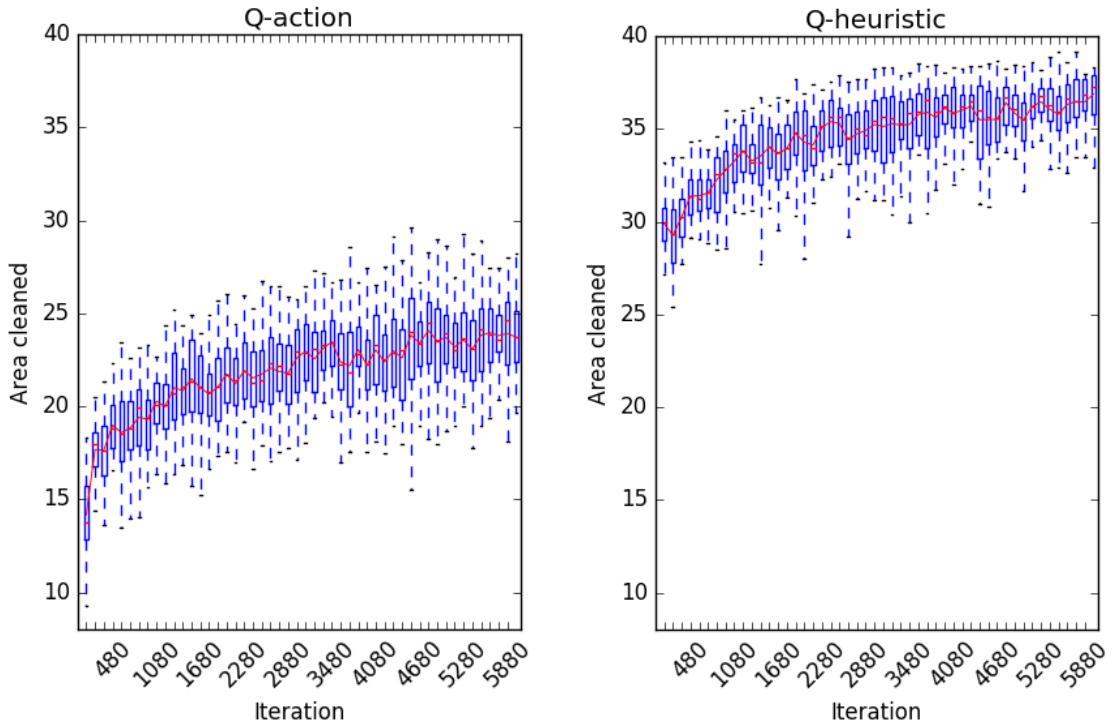
5.5.1 Comparing Learning Heuristics to Learning Actions

This section compares the decentralised Q-learning on heuristics (Algorithm 11) with the Q-learning on actions (Algorithm 10). The termination criteria for each run is terminating after 5880 iterations for empty environment, and 14,700 iterations for obstacle and gaps environment. Each experiment is repeated 30 times. The figures 5.9, 5.10 and 5.11 show the distributions of team reward collected by 5 robots. The y-axes show the area cleaned in m^2, which is the measurement taken from the robot dirt camera, and each time interval is 1 second.

The parameters for Q-learning Hyper-heuristics are shown in Table 5.2. The parameters are preliminarily values that were tuned with pilot runs.

Table 5.2 Parameters for Q-learning hyper-heuristics

Learning rate α	0.01
Discount factor γ	0.7
Initial exploration factor ε	0.5
ε decay rate	0.2 per 200 iteration

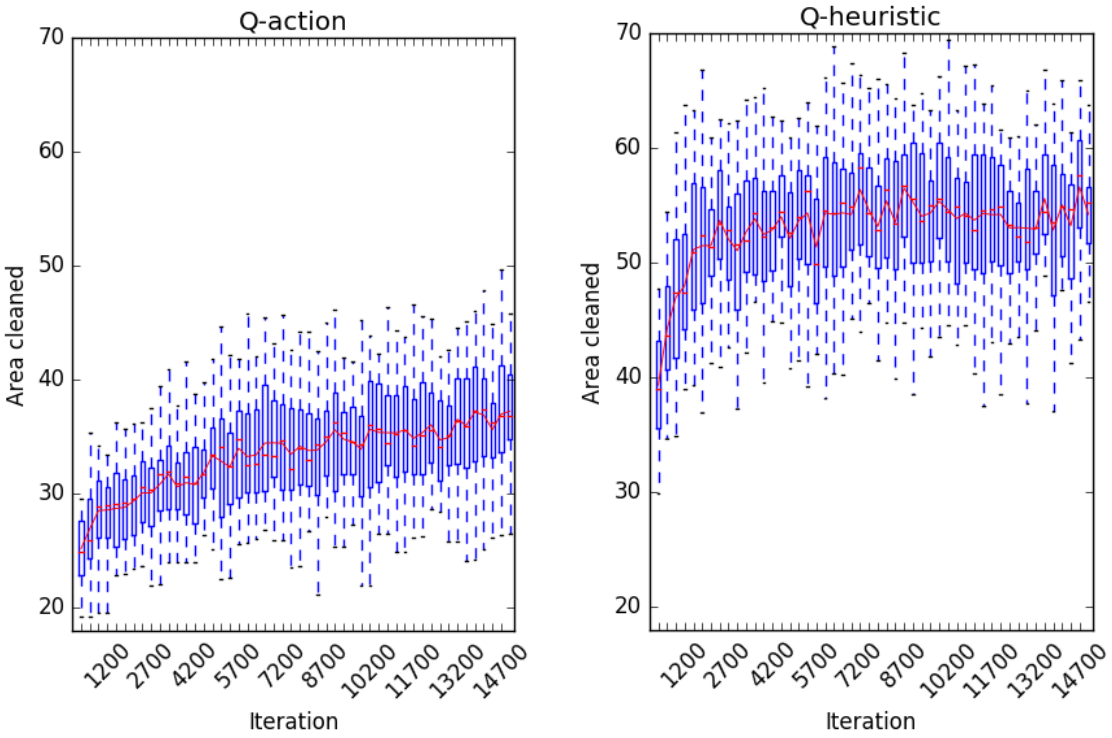


(a) Tabular Q-learning on actions

(b) Tabular Q-learning on heuristics

Fig. 5.9 Empty environment: team reward with Q-learning

It can be seen that in all tested scenarios, learning heuristics produces higher average rewards than the approach of learning actions. Figure 5.9 compares the performance of the action based Q-learning and Q-learning hyper-heuristic with the tabular implementation in the empty environment as Figure 5.8(a). Figure 5.10 shows the comparison of Q-heuristic method and Q-action method in the obstacle environment as Figure 5.8(c). The obstacle locations change every 300 iterations, therefore is a dynamic environment. Figure 5.11



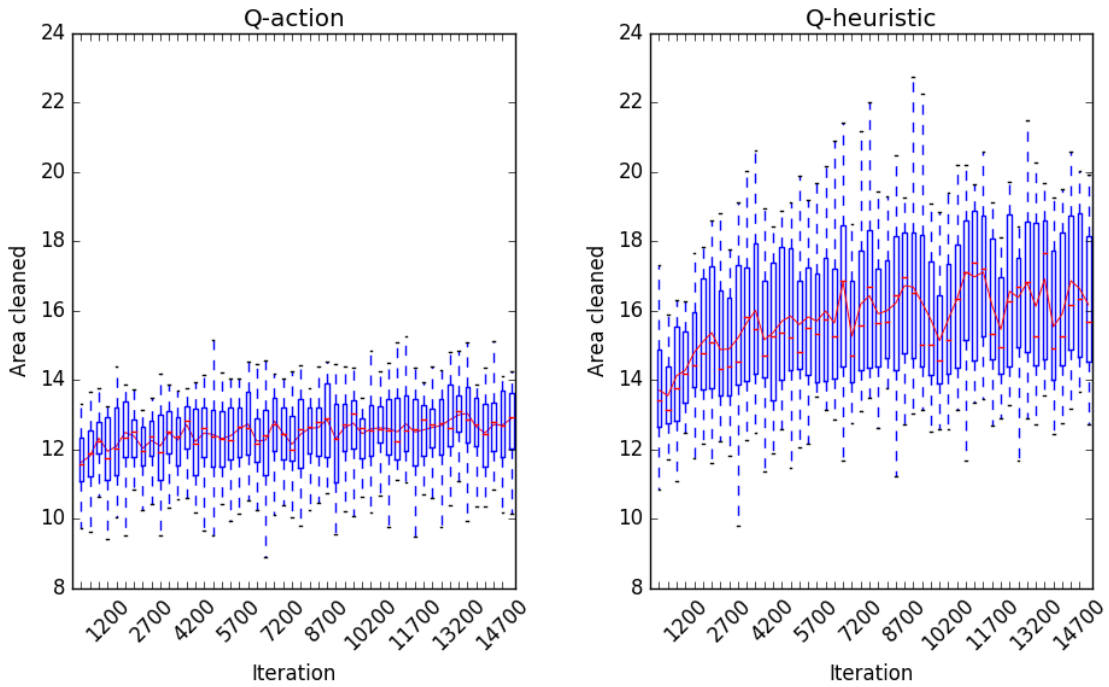
(a) Tabular Q-learning on actions

(b) Tabular Q-learning on heuristics

Fig. 5.10 Obstacle environment: team reward with Q-learning

compares the Q-heuristic method and Q-action method in the gaps environment as Figure 5.8(b).

From the experimental results, it can be seen that the Q-learning method is effective for both learning actions and heuristics in the sense that the team performances are improved over the task iterations. However, it is also clear that learning heuristics have both better starting performance and learned policies than learning over actions. In the empty environment, learning on heuristics gives the multi-robot system a head start, with an average reward of 29.83 over 30 repetitions, which is 2.14 times the action-based average reward of 13.91. In the obstacles environment, the improvement in the first 300 iterations from learning heuristics instead of actions is 57.25%, with an average reward of 24.77 from learning actions and average reward of 38.95 from learning heuristics. Learning over heuristics in the gaps



(a) Tabular Q-learning on actions

(b) Tabular Q-learning on heuristics

Fig. 5.11 Gaps environment: team reward with Q-learning

environment has a starting average team reward of 13.71, and learning over actions has an average reward of 11.63. The difference at the start is 17.93%.

In the empty environment, the Q-action learning converges to a team reward of about 23.83 over 4560 iterations, and the Q-heuristic reward converges to approximately 35.83 over 3960 iterations. In this regard, the learned heuristic policy is 50.36% better than the learned action policy. In the obstacle environment, learning over heuristics gives a significant improvement in the team performance. In 2700 iterations, the average robot team reward converges to around 53.34 using Q-learning over heuristics. When learning over actions, the average reward converges in 14700 iterations to a value of 36.95. The heuristic policy found in the obstacle environment is improved by 44.36% from the action-based policy. It can be seen that learning actions in this dynamic environment is not every effective, which agrees the established knowledge that traditional Q-learning is not suitable for distributed

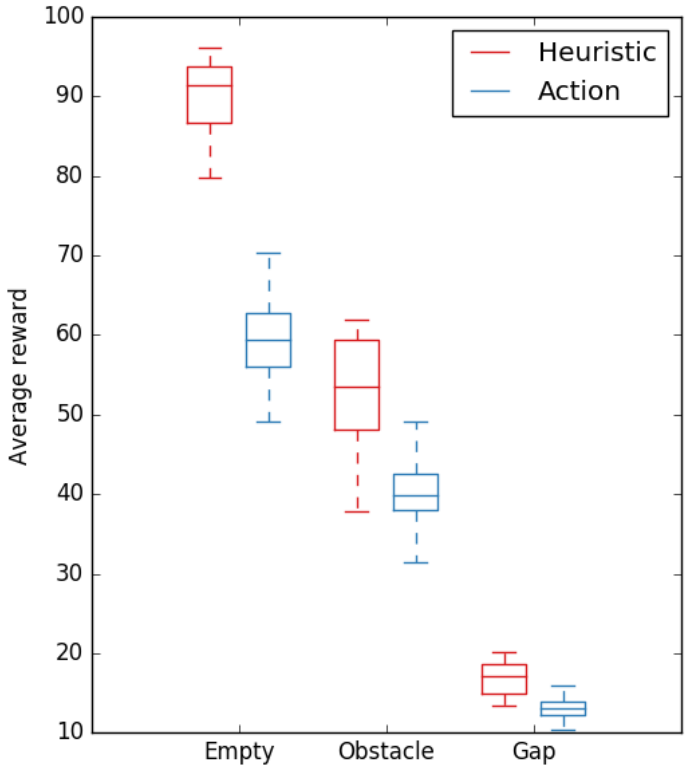


Fig. 5.12 Comparing the learned heuristic-based policy and the action-based policy

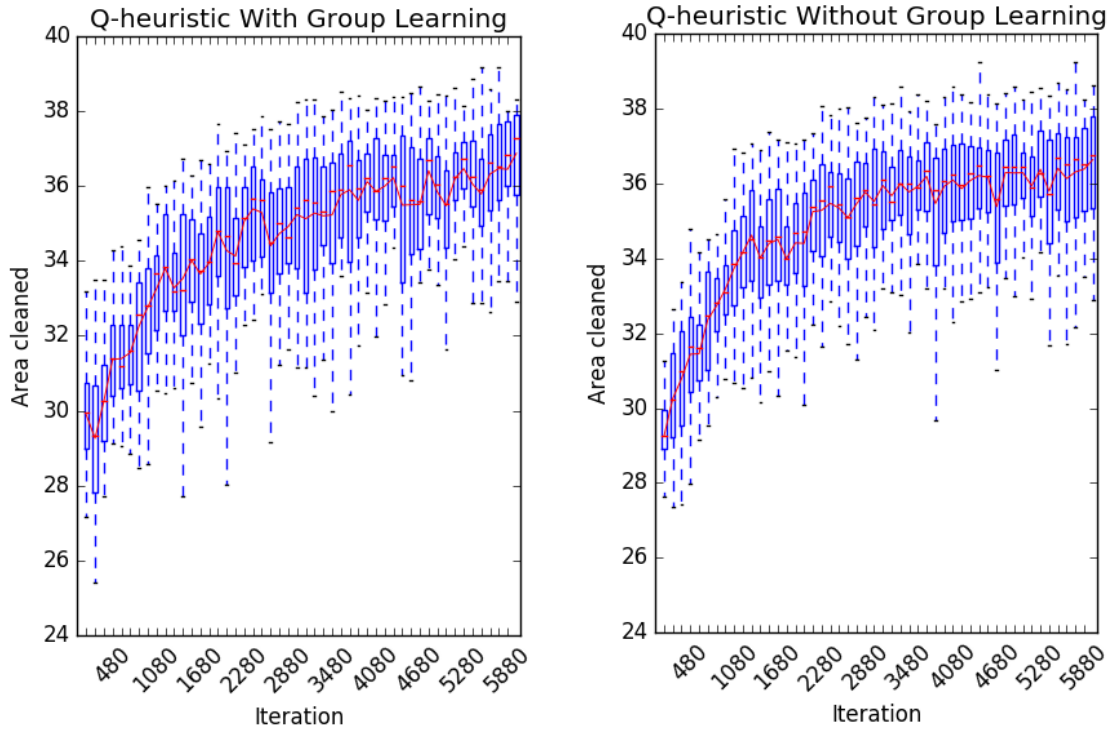
and dynamic environments. The problem of the environment being dynamic all the time for each robot is addressed by incorporating multi-robot collaboration and coordination in heuristics. The collective behaviours brought by the heuristics make the environment less "dynamic" in the view of each heuristic. Therefore the learning of policy is more stable and more effective. This can also be seen more clearly in the gaps environment, where Q-action fails to learn to cross onto other surfaces. The initial performance of the Q-action method is 11.63, and does not exceed an average of 12.75 over 14700 iterations. With heuristics, the "bridging" behaviour is pre-programmed into robots, so that the test robots already know how to cross gaps, and the learning process is for learning when and where to cross gaps. It can be seen that within 8700 iterations the Q-heuristic method converges to an average team reward of 16.67, and that this improvement is from cleaning multiple surfaces. We have also performed Mann-Whitney U tests on the performances of the learned policies, comparing

Q-heuristic and Q-action methods, and the p-values are all approximately 0, indicating that the advantage of learning over heuristics is statistically significant in all scenarios.

In conclusion, learning heuristics is significantly more efficient and more effective than learning actions with Q-learning.

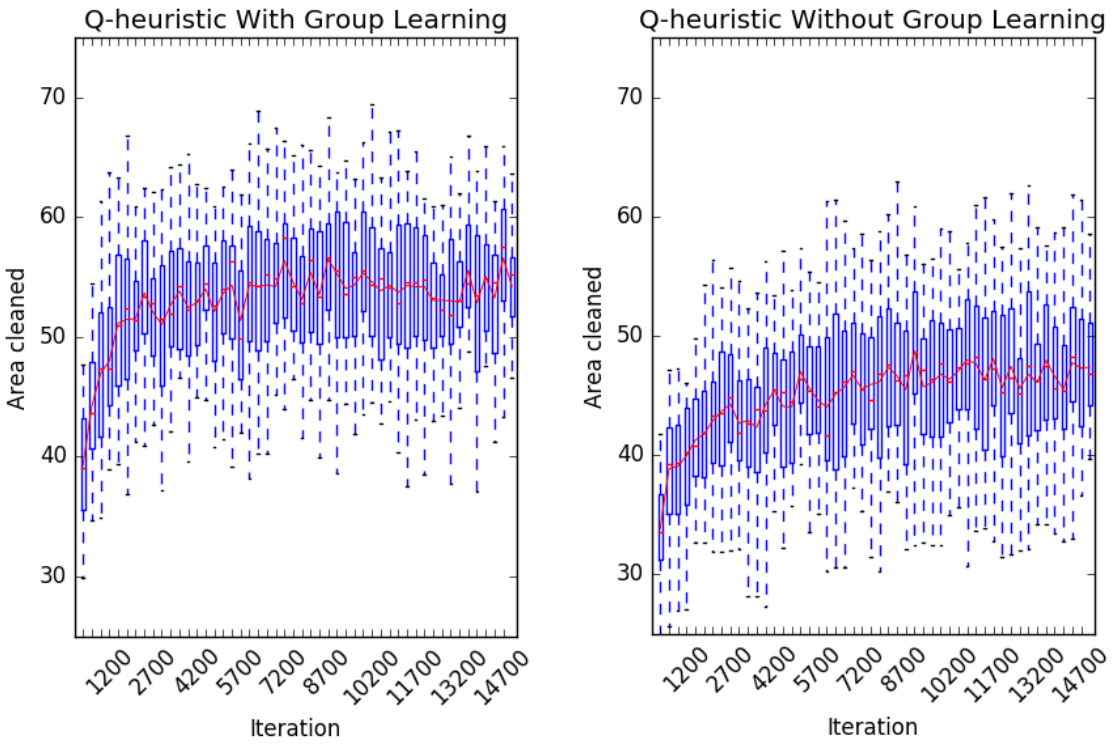
5.5.2 With and Without Group Learning

This section aims to explore the effectiveness of the importance advising mechanism, which is the group learning method for Q-learning hyper-heuristics, by comparing the performance of Q-learning with group learning described in Section 5.3 and the performance of robots without learning from each other. The algorithm is tested on five robots in the environments detailed in Chapter 3, and each experiment is repeated 30 times.



(a) (b)

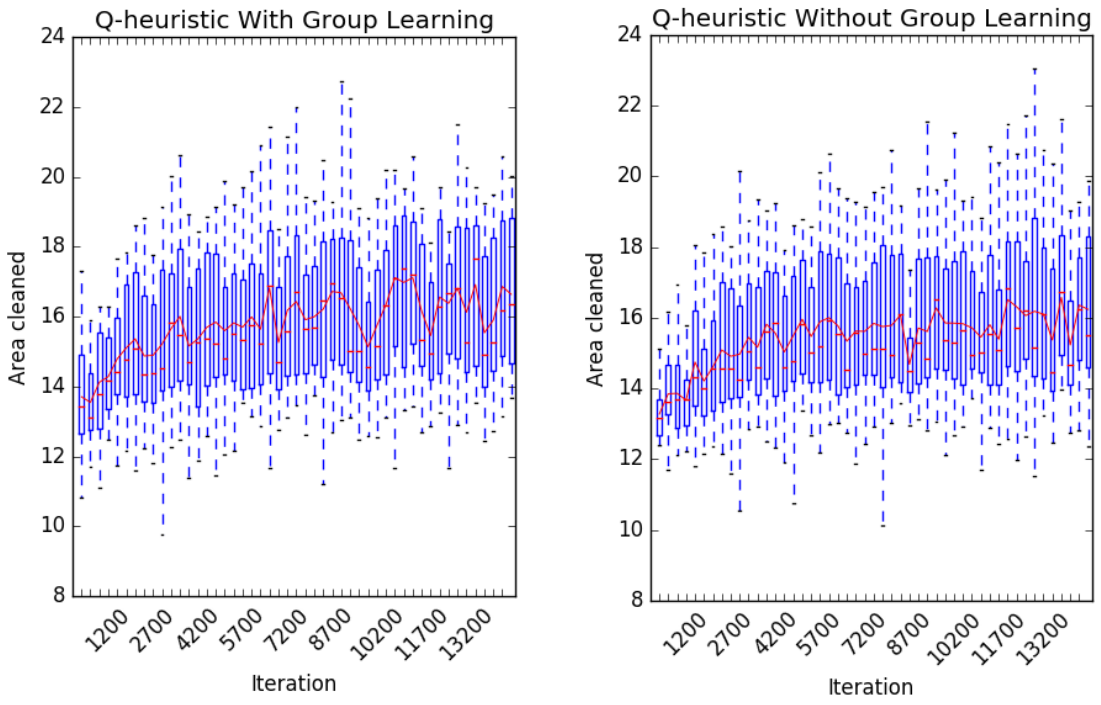
Fig. 5.13 Comparing with and without the importance advising mechanism: Empty environment



(a) (b)

Fig. 5.14 Comparing with and without the importance advising mechanism: Obstacle environment

Figure 5.13 shows the comparison of Q-learning hyper-heuristic with importance advising and Q-learning hyper-heuristic without importance advising in the empty environment. It can be seen that in the trained policy after 5880 iterations, the average reward over 30 repetitions with importance advising is 37.20, which is higher than an average reward of 36.67 given by the Q-learning hyper-heuristic without importance learning. To further confirm that the differences are statistically significant, Mann-Whitney U test is used, and the p-value is $0.035 < 0.05$. The reward obtained with importance advising in the first 300 iterations is also higher than the reward without importance advising, with values of $29.83 > 29.12$. When performing the Mann-Whitney U test, the p-value is $0.033 < 0.05$. This means that there is 95% confidence that the importance advising mechanism improves the Q-learning hyper-heuristic in empty environment.



(a) (b)

Fig. 5.15 Comparing with and without the importance advising mechanism: Gaps environment

Figure 5.14 shows the performance comparison in the environment with changing obstacles, and the change cycle is every 300 iterations. The plots show that the policy with the importance advising mechanism has greatly improved the Q-learning hyper-heuristic. After 5880 training iterations, the average reward obtained without importance advising is 46.74, while the average reward for Q-learning hyper-heuristic with importance advising is 54.81. After having applied the Mann-Whitney U test on the data of 30 repetitions, the p-value is $0.01 < 0.05$. At the beginning of the training, the average rewards of the first 300 iterations are 33.46 and 38.93 for Q-learning hyper-heuristic without importance advising and with advising respectively. The p-value of the Mann-Whitney U test on the two datasets is close to 0. The statistical tests show that the importance advising is very effective in the environment with dynamic obstacles.

The results in the gaps environment are shown in Figure 5.15. After 14500 training iterations, the average reward of the learned policy with importance advising Q-learning hyper-heuristic is 16.46, and the reward without importance advising is 15.79. At the beginning of the training, the average reward for the first 300 iterations is 13.62 and 13.06 for with and without importance advising respectively. However, when the Mann-Whitney U test is performed, the p-values for the end and beginning of the training are $0.209 > 0.1$ and $0.236 > 0.1$. This means that although the mean reward for the importance advising method is higher, the differences across 30 repetitions are not statistically significant.

It is worth noting that although the difference between Q-learning hyper-heuristics with importance advising and without in the empty environment is statistically significant, it is not very large. And in the gaps environment, although the average reward of having the importance advising method is better, the difference is not statistically significant. On the other hand, in the environment with dynamic obstacles, the importance advising mechanism has dramatically improved Q-learning hyper-heuristic performance, both in terms of learning speed and the learned policy. Both the empty and gaps environments are static, therefore during learning, each robot's observations are similar at any time and the learning processes are almost in synchronisation. Advising is not very useful in these scenarios because all robots are roughly at the same level of learning and even if advice is given to peers, they are not substantially superior. On the other hand, when the obstacles are dynamic, the observations from each robot can be very different because of the random distribution of obstacles. In this case, some robots might have explored many more states than others, and are good advisers. The results verified that the presence of these good advisers in the group makes the advising mechanism more effective.

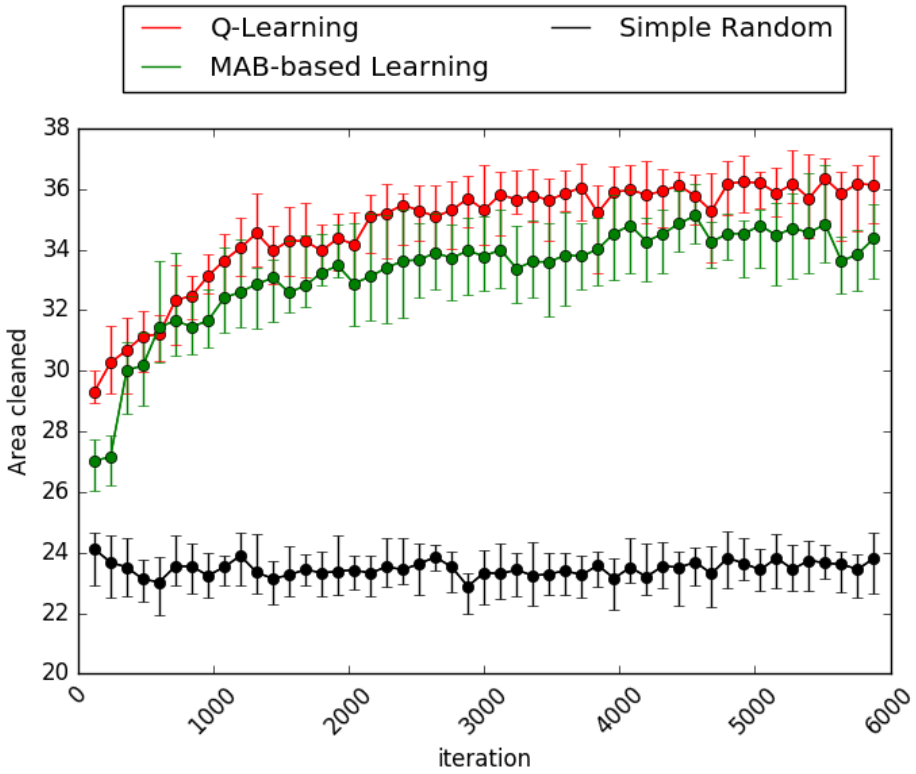


Fig. 5.16 Comparing hyper-heuristic methods in the empty environment

5.5.3 Comparing MAB-based Learning and Q-Learning

This section aims to verify that the Q-learning hyper-heuristic is an improvement from the MAB-based adaptive hyper-heuristic method described in Chapter 4.

The Q-learning method enables robots to automatically form teams that apply the appropriate heuristics, collaborate across teams, and achieve better rewards. Figures 5.16, 5.17 and 5.18 show the comparisons between the Q-learning method and the MAB-based hyper-heuristic on a swarm of five robots using the same heuristic repository and environments. For fair comparison of the two machine learning methods, the MAB-based online learning has been adjusted to have the same time interval length as the Q-learning hyper-heuristic and the same reward averaging mechanism among cooperating robots. The baseline method, Simple Random heuristic selection is also shown in the graphs.

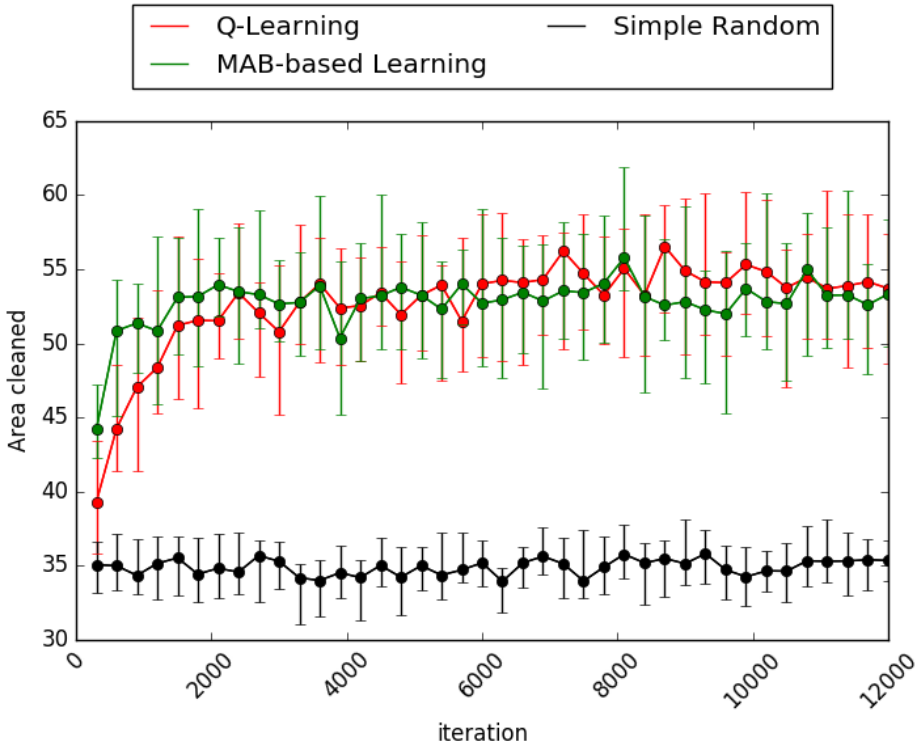


Fig. 5.17 Comparing hyper-heuristic methods in the dynamic obstacles environment

In the empty environment, the baseline method gives an average total reward of 23.47, and it can be seen in Figure 5.16, the two hyper-heuristic methods have both dramatically increased the cleaning ability of robot cleaners. It can be seen that the Q-learning hyper-heuristic has better performance than the MAB-based hyper-heuristic. The Q-learning method has an average reward of approximately 35.83 by the end of the learning, and the MAB-based hyper-heuristic gives 34.72. After performing the Mann-Whitney U test, the p-value is $0.0001 < 0.05$. Therefore the Q-learning hyper-heuristic method outperforms the MAB-based method in empty environment.

In the environment with dynamic obstacles, the baseline reward given by Simple Random heuristic selection is 34.94, and the hyper-heuristic methods all have much better performance. The Q-learning hyper-heuristic and MAB-based hyper-heuristic has similar performance: both gives total reward of approximately 53.34. After performing the Mann-Whitney U test

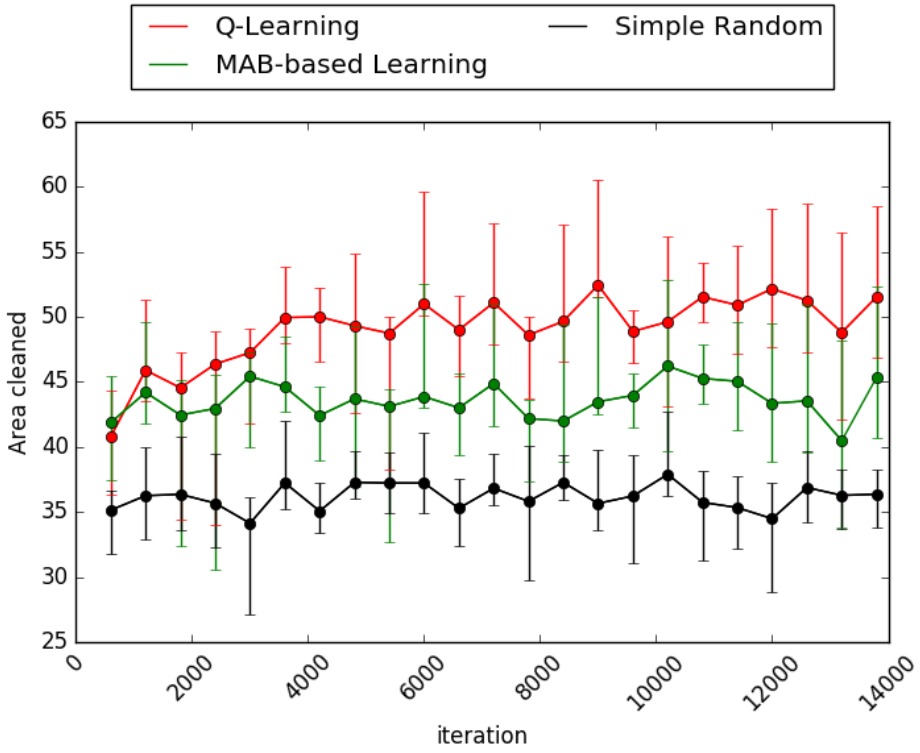


Fig. 5.18 Comparing hyper-heuristic methods in the gaps environment

on the performance of the learned policies, the p-value is $0.0901 > 0.05$, indicating that statistically, there is no significant difference at the end of the learning. This shows that when the environment is dynamic, the MAB-based hyper-heuristic is effective because of its adaptiveness. The MAB-based hyper-heuristic also learns faster than the tabular Q-learning hyper-heuristic. As can be seen in Figure 5.17, it takes the MAB-based hyper-heuristic 1200 iterations to reach a reward of 53.12 while the tabular Q-learning method used 2130 iterations to reach the same performance.

The comparison of algorithms in the gaps environment is shown in Figure 5.18. The Simple Random heuristic selection baseline is 36.28. In this environment, the MAB-based adaptive heuristic has better performance than the baseline, giving a reward of 43.84. The Q-learning based methods are both significantly more effective than the MAB-based method, achieving rewards of approximately 50.93, which is 16.17% higher than the MAB-based

adaptive method. The Mann-Whitney U test statistically confirms the advantage with a p-value of 0.0001. Although results confirmed that both the MAB-based hyper-heuristics and Q-learning hyper-heuristics outperforms the baseline methods, Q-learning hyper-heuristic has better performance. This is because the MAB-based hyper-heuristic controller does not take environmental conditions into account in selecting heuristics, and uses the objective value as the single feedback. The MAB-based policy is purely adaptive. On the other hand, the Q-learning hyper-heuristic policy is more sophisticated because the knowledge is stored in the Q-table.

In conclusion, results across a range of environments show that the Q-learning of heuristics has significantly improved the overall task performance over the MAB-based hyper-heuristic.

5.6 Summary

This chapter answered the second research question, how can different learning strategies improve the swarm robot hyper-heuristic method, and introduced a Q-learning based hyper-heuristic. The Q-learning of heuristics allows robots to learn their own policies while maintaining cooperation with other robots in the swarm. We verified that Q-learning of heuristics can integrate well into the hyper-heuristic framework, and it is significantly more effective than action-based Q-learning.

It has also been shown that with the group learning strategy (the importance advising mechanism) robots can learn faster and find better policies.

Comparisons with the MAB-based hyper-heuristics have been made in simulation, and results show that the Q-learning of heuristics has achieved a significant improvement over MAB-based hyper-heuristics.

Chapter 6

Learning Heuristics with Deep Reinforcement Learning

Previous chapters have shown that the hyper-heuristic framework can facilitate MAB-based online learning and Q-learning of heuristics. In recent years, researchers have found that neural networks can boost a lot of the existing machine learning algorithms. A typical example is Deep Q-learning, where Deep Q-Networks (DQN) are used as function approximators for the state-action value function, and have achieved state-of-the-art performance in many robotics applications. In this chapter, we verify if our proposed Q-learning hyper-heuristic framework (shown in Fig. 5.3) can incorporate deep learning and further improve. Since we have demonstrated the success in learning heuristics using Q-learning, deep Q-learning is a good candidate to provide improvement to the framework.

6.1 Methodology

In reinforcement learning, the state space can be so large that the Q-table may not fit into memory. Approximation methods compress the amount of space needed to represent Q, and neural networks are universal function approximators. In this section, we demonstrate

the use of deep reinforcement learning in hyper-heuristics, compare the heuristic-based deep reinforcement learning with action-based deep reinforcement learning and show the advantage in learning heuristics.

We consider tasks in which reinforcement learning agents, in this case robots, interact with the environment in a sequence of actions, observations and rewards. At each step, the robot selects an action a_t from a set of legal actions A, and by applying that action, the state of the environment changes. No robot observes the complete environment state, instead it observes a vector of sensor readings s_t.

Neural networks are able to approximate non-linear functions $Q(s,a) \approx Q(s,a;\theta)$, with the Q-network parameterised with θ.

Algorithm 13 Double Deep-Q Learning over Actions

for $\forall u \in U$ **do**
  Initialise replay memory E to capacity N
  Initialise action-value function Q with random weights
  while task is not terminated **do**
    Observe state s_u^t
    Policy $\pi_u \leftarrow$ predict using the neural network model
    if probability $> \varepsilon$ **then**
      $a_u^t \leftarrow \arg\max_{a_u} Q(s_u^t, a_u; \theta_u)$
    else
      Select a random action a_u^t
    end if
    Apply action a_u^t
    Observe state s_u^{t+1}, obtain reward r_u
    Store transition $\{s_u^t, a_u^t, r_u, s_u^{t+1}\}$ in experience buffer E
    Sample a mini-batch from experience buffer E
    Calculate target for each transition $y_u^t = r_u + \gamma * \max_a Q(S_t, \arg\max_a Q(S^{t+1}, a; \theta_t'); \theta_u)$
    Perform one gradient descent training step on the loss function $L_u(\theta_u) = (y_u^t - Q(s_u^t, a_u^t; \theta_u))^2$
    $t \leftarrow t+1$
  end while
end for

Algorithm 14 Double Deep-Q Learning over Heuristics

for $\forall u \in U$ **do**
  Initialise replay memory E to capacity N
  Initialise action-value function Q with random weights
  while task is not terminated **do**
    Observe state s_u^t
    Policy $\pi_u \leftarrow$ predict using the neural network model
    if probability $> \varepsilon$ **then**
      $h_u^t \leftarrow \arg\max_{h_u} Q(s_u^t, h_u; \theta_u)$
    else
      Select a random heuristic h_u^t
    end if
    Apply heuristic $a_u^t \leftarrow \mathscr{H}_u(s_u^t)$
    Observe state s_u^{t+1}, obtain reward r_u
    if there is collaborating robot with heuristic h_u^t **then**
      Send r_u to collaborating neighbouring agents, receive r_i from n connected neighbours
      $r_u \leftarrow (r_u + \sum_{i=1}^{n} r_i)/(n+1)$
    end if
    Store transition $\{s_u^t, h_u^t, r_u, s_u^{t+1}\}$ in experience buffer E
    Sample a mini-batch from experience buffer E
    Calculate target for each transition $y_u^t = r_u + \gamma * \max_h Q(S_t, \arg\max_h Q(S_{t+1}, h; \theta_u); \theta_u')$
    Perform one gradient descent training step on the loss function $L_u(\theta_u) = (y_u^t - Q(s_u^t, h_u^t; \theta_u))^2$
    $t \leftarrow t+1$
  end while
end for

The deep reinforcement learning hyper-heuristic algorithm is detailed in Algorithm 14. In order to stabilise the learning process, double Q-learning is used to learn heuristics [134]. The standard deep-Q learning updates the parameters after the agent has taken an action a' and observed the next state S_{t+1}

$$\theta_{t+1} = \theta_t + \alpha(Y_t - Q(S_t, A_t; \theta_t))\Delta_{\theta_t} Q(S_t, A_t; \theta_t)) \tag{6.1}$$

where α is the step size and the target Y is defined as

$$Y_t \equiv R_{t+1} + \gamma \max_a Q(S_t, A_t; \theta_t)) \tag{6.2}$$

With double Q-learning, the action selection and evaluation are untangled by using two neural networks. Using the standard Q-learning, the learning process is noisy and unstable, especially in complex environments where there are gaps in the surfaces. With double Q-learning, the training is stabilised and robots are able to learn and improve steadily. The target Y_t^{ddqn} is defined as

$$Y_t^{ddqn} \equiv R_{t+1} + \gamma \max_a Q(S_t, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t')) \tag{6.3}$$

When applied to distributed robots, the algorithm is described in Algorithm 13. Each robot trains its own neural network model locally.

To select heuristics instead of actions, the target Y_t^{ddqn} is defined as

$$Y_t^{ddqn} \equiv R_{t+1} + \gamma \max_h Q(S_t, \arg\max_h Q(S_{t+1}, h; \theta_t); \theta_t')) \tag{6.4}$$

The loss function for each robot is given as

$$L(\theta) = \mathbb{E}[(r + \gamma \max Q(s', a'; \theta) - Q(s, a; \theta))^2] \tag{6.5}$$

When learning heuristics, the loss function that is to be minimised is:

$$L(\theta) = \mathbb{E}[(r + \gamma \max Q(s', h'; \theta) - Q(s, h; \theta))^2] \tag{6.6}$$

In optimising the neural network, calculating the cost over all experience E is computationally inefficient. Instead, the algorithm uses stochastic gradient descent (SGD), which samples

mini-batches from the stored experience E and performs one gradient descent training step per training iteration.

6.2 Experiments

With the same environmental setups in Chapter 5, this section aims to verify that deep Q-learning can also be used with the hyper-heuristic framework, and that robots are able to effectively use deep learning with heuristics in a decentralised setting.

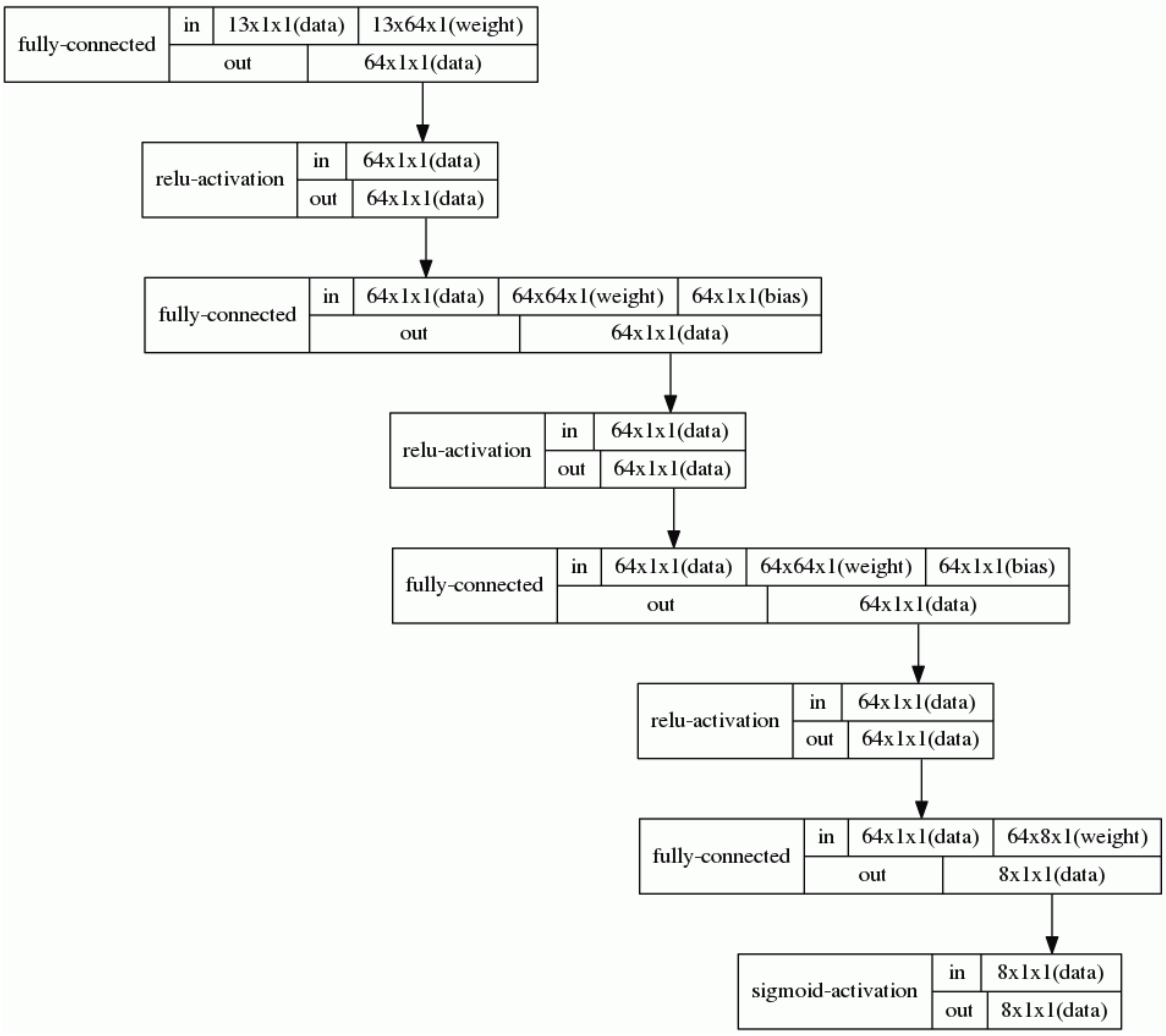


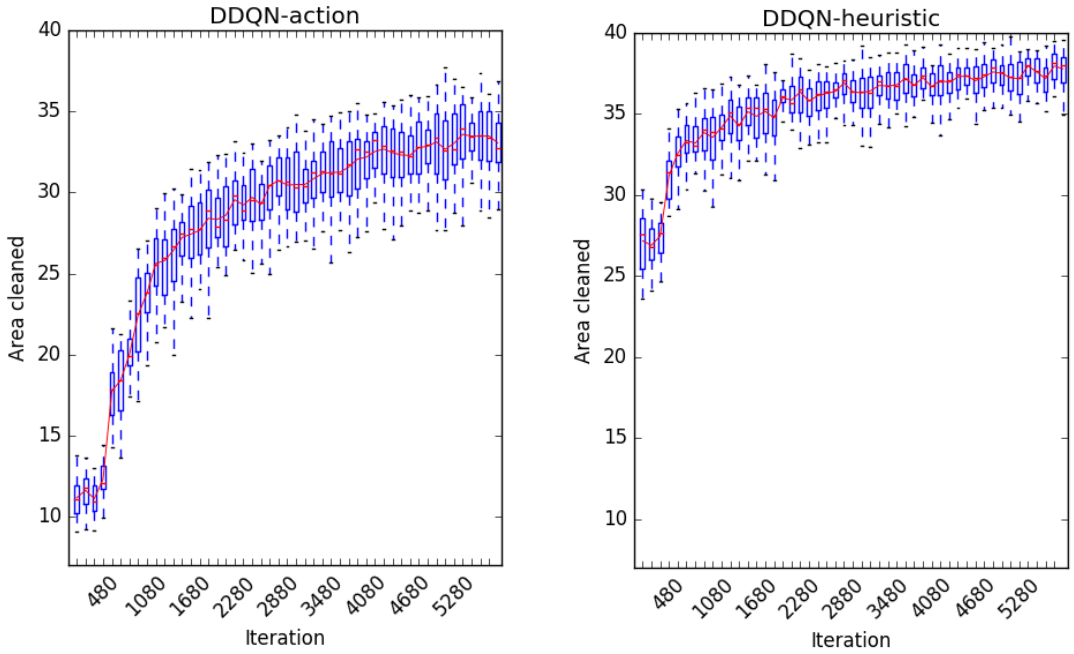
Fig. 6.1 The neural network architecture for DDQN learning

Table 6.1 Parameters for Q-learning hyper-heuristics

Optimizer	Adam
Learning rate α	0.001
Adam parameter β_1	0.9
Adam parameter β_2	0.999
Adam parameter β_1^t	0.9
Adam parameter β_2^t	0.999
Discount factor γ	0.99
Initial exploration factor ε	0.5
ε decay rate	0.2 per 200 iteration
Batch size	32
Maximum stored experience number	15,000
Minimum experience to start training	400

The state representation used as the input to the neural networks is the same as the tabular method for fair comparison, and is described in Table 5.1. The output of the neural networks is predicted Q-values of the heuristics. Through some primary tuning, the neural network architecture is shown in Figure 6.1. Since the state representation has 13 bits, the input to the neural network has a dimension of "$13 \times 1 \times 1$". The input layer is connected to a hidden layer that has 64 neurons, with ReLU activation [90]. There are four such hidden layers with the same dimensions and activation function. The output neurons have a dimension of "$8 \times 1 \times 1$", which agrees with the number of heuristics in the repository.

The parameters for DDQN Hyper-heuristics are empirically tuned for best performance, and are shown in Table 6.1. The optimizer used in the neural network training is Adam [55].

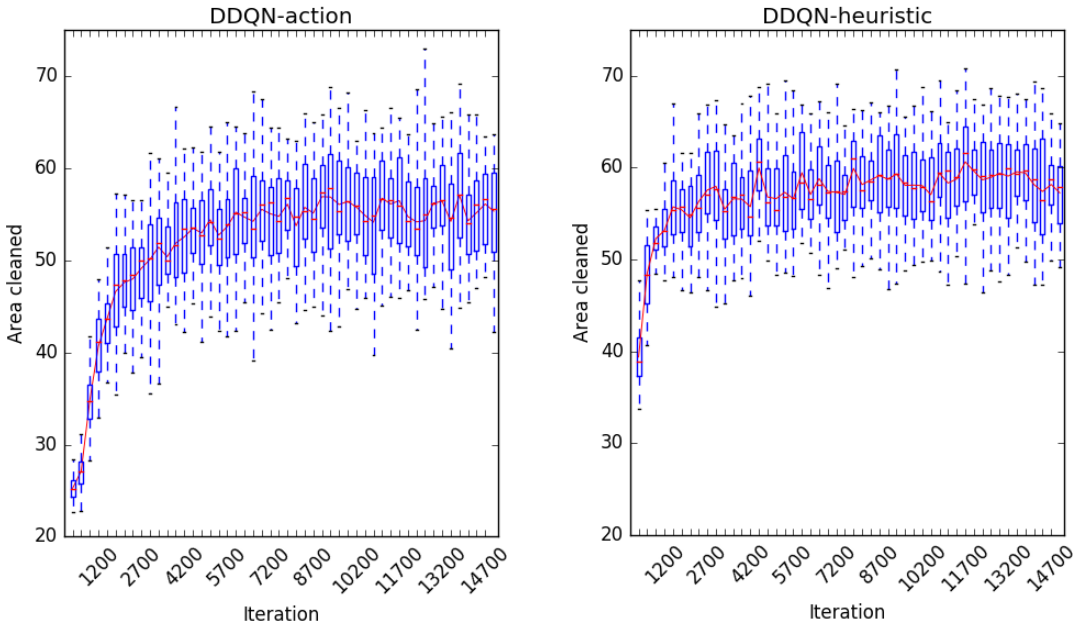


(a) DDQN on actions (b) DDQN on heuristics

Fig. 6.2 Empty environment: team reward

6.2.1 Comparing Learning Heuristics to Learning Actions

Figures 6.2, 6.3 and 6.4 show the distribution of team rewards over 30 repetitions of experiments. It can be seen that with DDQN, robots learn actions more effectively than learning actions with the tabular method, however the learning heuristics still has a significant advantage in all environments, especially in the environment with gaps. In the empty environment, the action-based policy is able to achieve a reward of 33.2 and the heuristic-based policy has a reward of 37.93 with the same number of training iterations, which has a 14.25% improvement. The improvement in the obstacle environment is less than the empty environment, 5.9%, where the total reward gained by action-based DDQN policy is 55.95 and by the reward of heuristic-based policy is 59.24. This is because the locations of the obstacles changes every 300 iterations, therefore the policy is harder to learn. In the gaps environment, the difference between heuristic-based policy and the action based policy is the largest. It

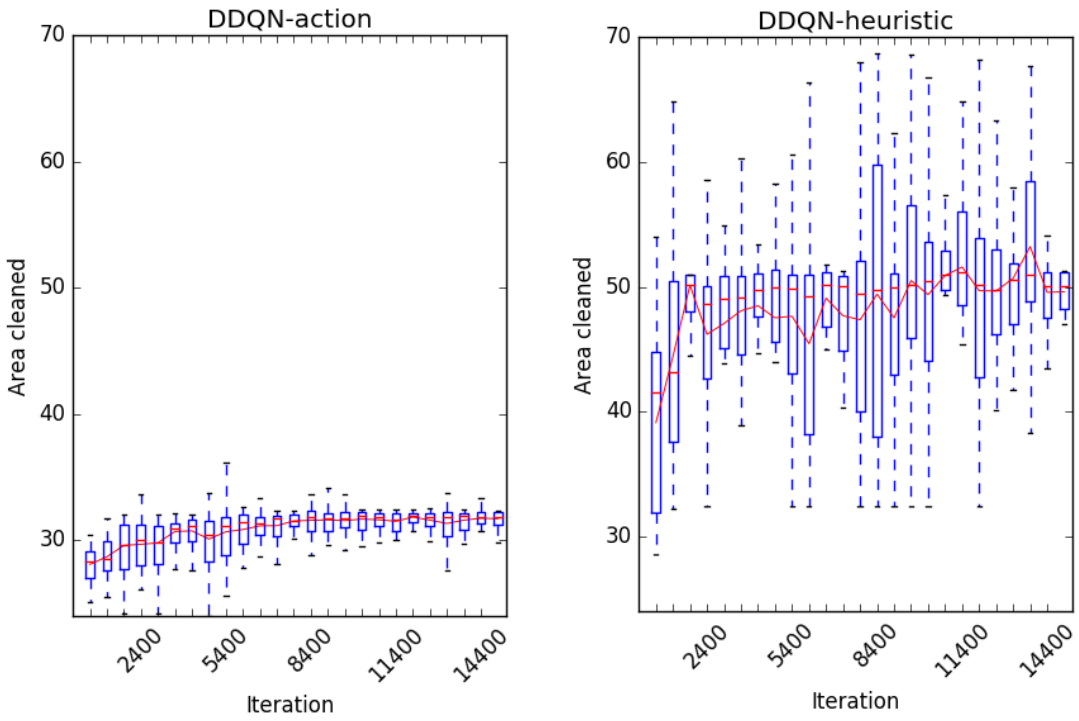


(a) DDQN on actions (b) DDQN on heuristics

Fig. 6.3 Obstacle environment: team reward

can be seen in Figure 6.4 that the action-based DDQN has learned very little, with almost no improvement in performance. When looking into the robots' behaviour, the action-based policy has not learned to assemble and cross gaps in the 14400 iterations of training. On the other hand, learning on heuristics is much more effective, which gives an average reward of 50.59, which is 60.35% better than the reward 31.55 given by the action-based policy. With the heuristic-based policy, it can be seen from experiments that the robots use the bridging heuristic to move across multiple surfaces, which drastically increased the group performance.

DDQN-based hyper-heuristics also have great advantage at the start of the training. The average reward of the first 300 iterations of learning actions is 10.9, while learning heuristics gives 27.45 at the start of the training. This means that without any learning, selecting heuristics gives a reward that is 2.52 times higher than selecting actions. At the start of the training in the obstacle environment, the average reward of the first 300 iterations is 38.47



(a) DDQN on actions (b) DDQN on heuristics

Fig. 6.4 Gaps environment: team reward

for learning heuristics and 24.73 for learning actions, which means that learning heuristics gives a head start of 55.56% more reward. In the gaps environment, the difference is the largest, where the heuristic-based policy is already better than action-based policy in the first 300 iterations. The average reward at the start of the training is 41.25 for learning with heuristics, which is already higher than the best reward achieved by action-based learning (31.55). Learning on actions gives a starting reward of 28.07. This agrees with the fact that learning on heuristics is much more efficient in environments that have multiple surfaces.

To further compare the learning of heuristics and actions, Figure 6.5 shows the trained neural network performance on heuristics and actions with the same number of iterations. It can be seen that not only does the DDQN-based heuristic learning method outperform the action-based DDQN in all three scenarios, but that the hyper-heuristic method is also more stable than action-based DDQN. The standard deviations of heuristic-based policies are

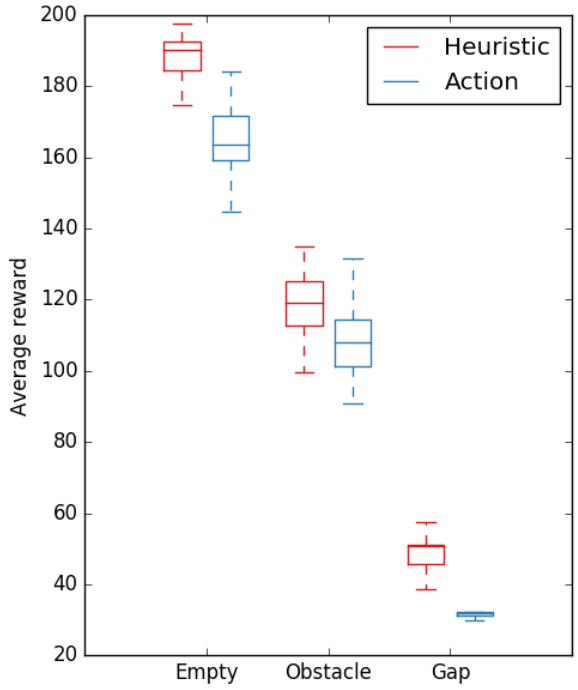


Fig. 6.5 Comparing the learned heuristic-based policy and the action-based policy

lower than action-based policies. After performing Mann-Whitney U test on the data set, we have found that the p value of the heuristic-based and action-based method in all scenarios are close to zero, which confirms that the advantage of learning heuristics is statistically significant.

6.2.2 Comparing Deep Q-learning with Q-learning

This section compares the deep Q-learning hyper-heuristic with the tabular Q-learning hyper-heuristic method described in Chapter 5. For better comparison, we also plot the MAB-based hyper-heuristic results in the plots. Figure 6.6, 6.7 and 6.8 show the comparison between the deep Q-learning hyper-heuristic and the Q-learning hyper-heuristic.

In the empty environment, it can be seen that the deep Q-learning method is the best performing method among the three, and the MAB-based hyper-heuristic is the worst. The policy deep Q-learning has learned achieves an average performance of 37.45 after 6000

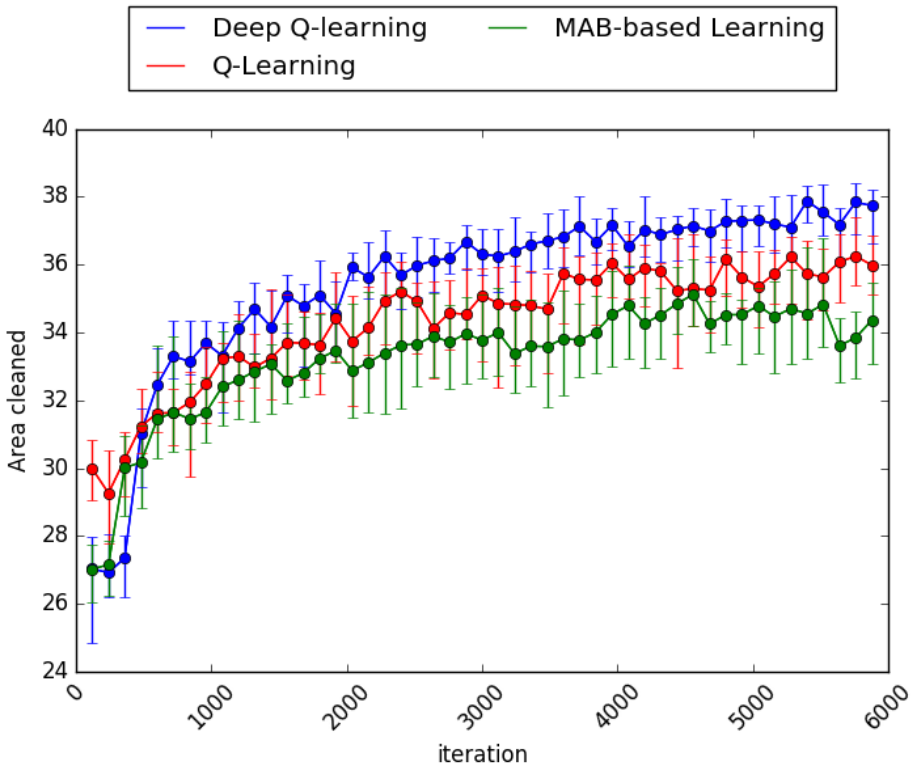


Fig. 6.6 Comparing hyper-heuristic methods in the empty environment

iterations, which has improved 4.4% from the tabular Q-learning, whose reward is 35.88. In the environment with dynamic obstacles, as shown in Figure 6.7, although tabular Q-learning did not achieve a significant improvement on the MAB-based hyper-heuristic method, the deep Q-learning hyper-heuristic outperforms the two. The deep Q-learning method gives an average reward of 59.41, while the other two methods' performance are both 53.34. The deep Q-learning method has an improvement of 11.37% comparing to the Q-learning hyper-heuristic and the MAB-based hyper-heuristic. The comparison of algorithms in the gaps environment is shown in Figure 6.8. In the previous comparison in Section 5.5.3, the Q-learning hyper-heuristic is significantly more effective than the MAB-based method. Comparing deep Q-learning to Q-learning of heuristics, there is no significant difference.

Across all three environments, the deep Q-learning hyper-heuristic performs the best among the three learning methods, as can be seen in Fig. 6.9, showing that the deep Q-

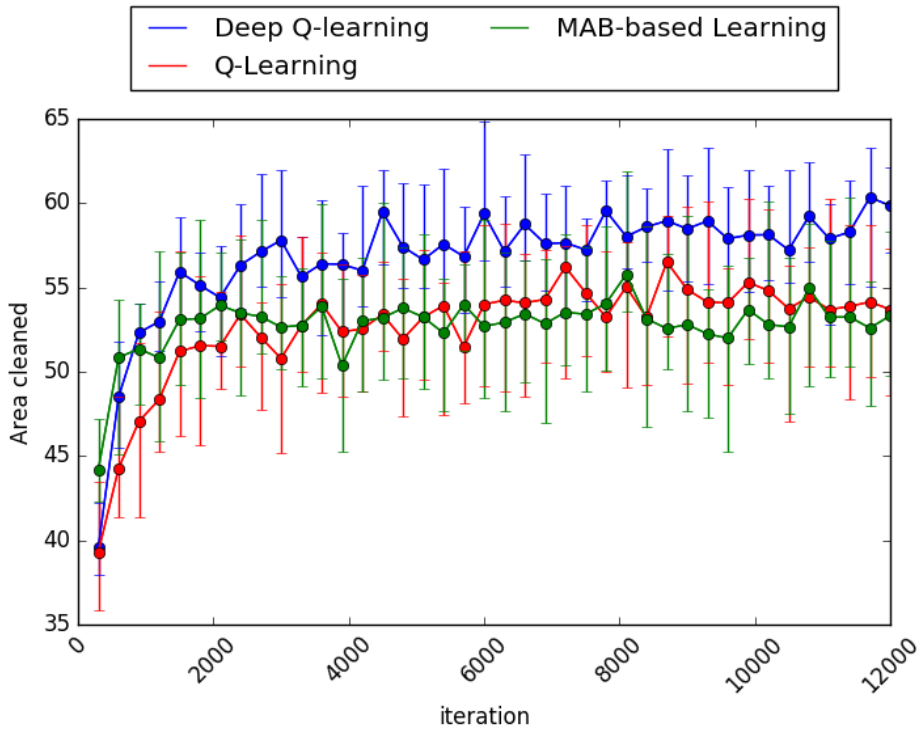


Fig. 6.7 Comparing hyper-heuristic methods in the dynamic obstacles environment

network is an effective function approximator of the state-heuristic value function. The deep Q-learning method matches or improves the learning performance over Q-learning in all scenarios, showing that deep learning of heuristics is better at finding good policies.

However, does this mean that the deep Q-learning method should always be preferred? In real-world robotics applications, a robot may have limited computational power, and does not support the training and evaluation of neural networks, or does them slowly. The tabular Q-learning hyper-heuristic approach requires less computational power, and can be performed on most commercial robot micro-controllers. Therefore, although deep Q-learning of heuristics can yield better results, it does have higher requirements on the robot hardware, which can increase the cost of building the robots.

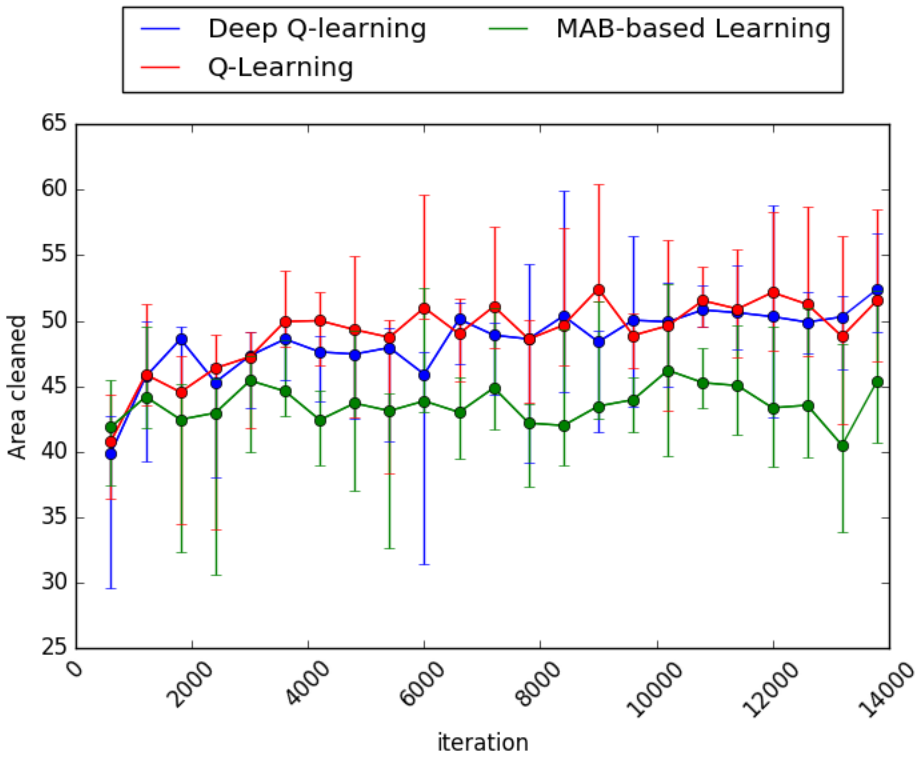


Fig. 6.8 Comparing hyper-heuristic methods in the gaps environment

6.2.3 Dynamically Changing Environments

This section compares the performance of the Deep Q-learning hyper-heuristic in two types of dynamically changing environments, and for comparison we include the previously proposed MAB-based hyper-heuristics and Q-learning heuristic. Additionally, the action-based Q-learning is used as a baseline to show improvements of the heuristic-based methods. The aim is to verify if Q-learning hyper-heuristics are able to perform consistently well in changing environments.

We test the DDQN hyper-heuristic, Q-learning hyper-heuristic, and the MAB hyper-heuristic under these environments:

- *Environment I*, which consists of multiple surfaces separated by gaps. Its obstacles change positions every 600 iterations

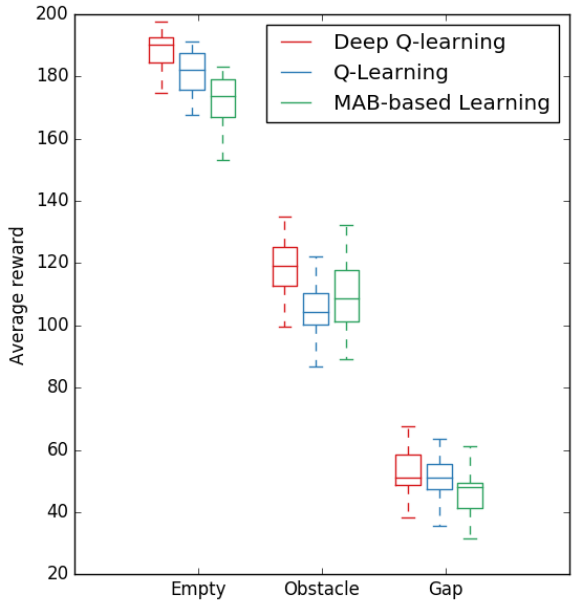


Fig. 6.9 Comparing the learned policies using Deep Q-learning, Q-learning and MAB-based learning

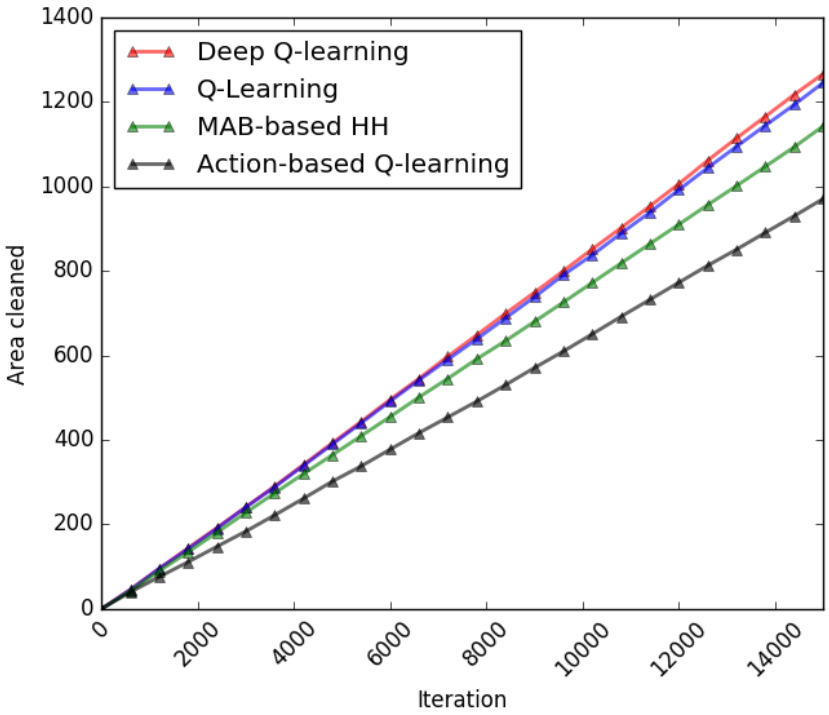


Fig. 6.10 Total rewards in *Environment I*: changing environments

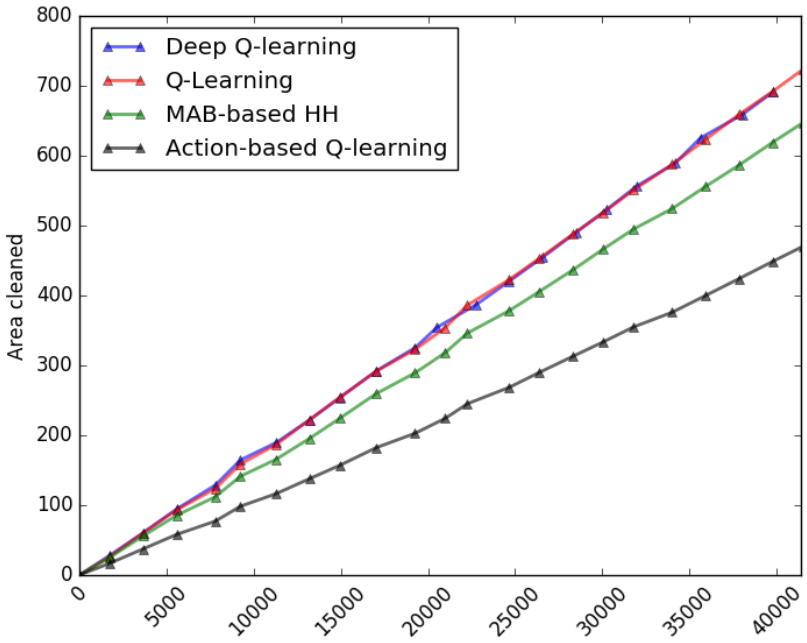


Fig. 6.11 Total rewards in *Environment II*: random environments

- *Environment II*, random environments selected from empty, obstacles and gaps

The robots have to continuously learn in these ever-changing environments, and aim to maximise the total reward. The results obtained from 30 runs are plotted in Figures 6.10 and 6.11. Since decentralised reinforcement learning agents observe the local state, they have the ability to transfer knowledge into new types of environments as long as some features in the new environment have been seen before. This can be seen from the fact that both methods are able to adapt and improve in both dynamic environments. In terms of the total reward, heuristic-based learning robots outperform action-based learning robots in both scenarios. This indicates that learning on heuristics is more effective in re-using and transferring knowledge into unknown and dynamic environments.

In *Environment I*, the obstacle positions change every 600 iterations, and Figure 6.10 plots 14000 iterations. It can be seen that the deep Q-learning hyper-heuristic has better performance than Q-learning and MAB-based hyper-heuristics, which agrees with the results

from previous experiments. The performance of deep Q-learning and Q-learning hyper-heuristic learning methods are 31.0% and 28.9% better than the action-based Q-learning. MAB-based hyper-heuristic is 18.3% better respectively than the action-based Q-learning.

Mann-Whitney U test [81] is performed on the dataset. The difference between the deep Q-learning and tabular Q-learning hyper-heuristic is not significant, with a p value of 0.18684. When compared with the MAB-based hyper-heuristic, the p value of deep Q-learning is 0.01684, which is significantly better with 95% confidence. Performing the Mann-Whitney U test on the Q-learning hyper-heuristic and MAB-based hyper-heuristic, the p value is 0.00016, indicating that Q-learning hyper-heuristic is significantly better with more than 99% confidence.

In *Environment II*, the methods all use the same random seed in the experiments. It can be seen in Figure 6.11 that the trend is very similar to *Environment I*. The performance difference between deep Q-learning and tabular Q-learning hyper-heuristics is negligible, and they are both significantly more effective than MAB-based hyper-heuristic in random environments.

Mann-Whitney U test [81] has been performed on the data. The p value of deep Q-learning and Q-learning hyper-heuristics is 0.90448, which shows that the performances are almost identical. Comparing MAB-based hyper-heuristic to deep Q-learning hyper-heuristic and Q-learning hyper-heuristic, the p values are 0.01596 and 0.01684 respectively, showing that there is a significant improvement with more than 95% confidence.

Results show that across all scenarios, the deep Q-learning of heuristics has either equal or higher reward than the tabular Q-learning. In addition, both deep-Q learning and Q-learning hyper-heuristic perform significantly better than the MAB-based hyper-heuristic in dynamically changing environments.

6.3 Summary

This chapter proposed and verified the use of deep Q-learning in the learning of heuristics for decentralised swarm robots. The deep Q-learning of heuristics has been shown in simulation to learn more quickly than action-based deep Q-learning, and has achieved significant improvements on task performance in various environments comparing to the methods in previous chapters. The results from experiments in this chapter lead us to conclude that, for a swarm robotic system, if the robots have enough on-board computational power for deep learning, the deep Q-learning hyper-heuristic approach would yield the best performance. If instead the robots do not have enough computational power but have a relatively large memory, the Q-learning hyper-heuristic approach becomes the preferred option in terms of the overall performance in complex environments. At last, if neither the computational power nor the memory on the robot are large and the task objective is complex, the MAB-based hyper-heuristic is recommended because it requires much less computational resources than the Q-learning methods, and still allows robots to adapt to changes in the environment.

Therefore we answered the research question of how deep learning can further improve the hyper-heuristic framework, and confirmed that deep Q-Networks can be integrated in our proposed hyper-heuristic framework and can achieve the aforementioned significant improvements.

Chapter 7

Analysis and Discussion

In the previous chapters we have shown the benefits of the hyper-heuristic framework, especially learning heuristics versus learning actions. However, the proposed hyper-heuristic approach has more serendipitous benefits such as explainability and scalability, which we analyse in this chapter.

7.1 Explainable Policies

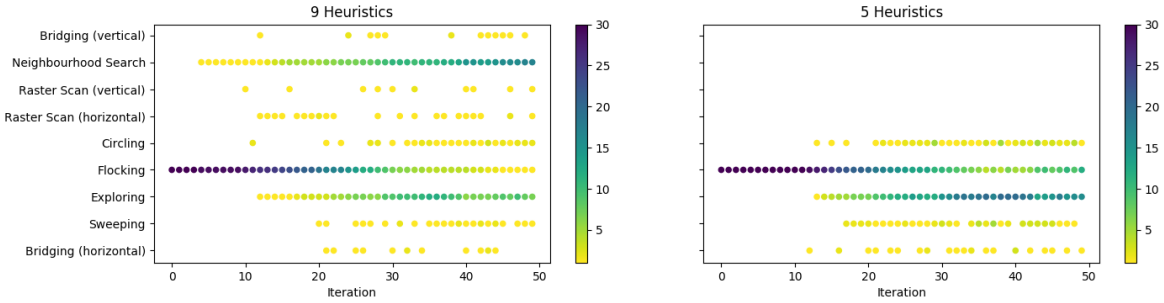
One advantage of learning heuristics is that both the learning process and the policy are easier for humans to comprehend than learning on actions. This is because the basic heuristics are hand-coded algorithms that have explainable logic, and the effects of parameters are very clear. For example, the collision avoidance mechanism used in the swarm robot heuristics has a parameter R_{avoid} that controls the radius of activating the collision avoidance mechanism, and changing this parameter will directly affect how sensitive the robot responds to obstacles. With the hyper-heuristic framework, it is easy to fine tune particular robot behaviours without re-designing the hyper-heuristic algorithm or the other heuristics in the repository. On the other hand, learning actions from scratch without heuristics introduces many "black boxes" in the learned policy, especially when the learned behaviour is complex. Taking the previous

collision avoidance as an example, if the learned policy, for instance a deep Q-network based policy, works just as intended but a human operator prefers the robots to maintain a further safety distance, the Q-learning algorithm needs to be re-designed to suit the extra requirement. This characteristic of the hyper-heuristic approach makes it easy to explain and potentially analyse the learning process. The following sections demonstrate how the hyper-heuristic learning in this thesis, namely the MAB-based adaptive hyper-heuristics and the Q-learning based hyper-heuristics can be explained and analysed using different statistics.

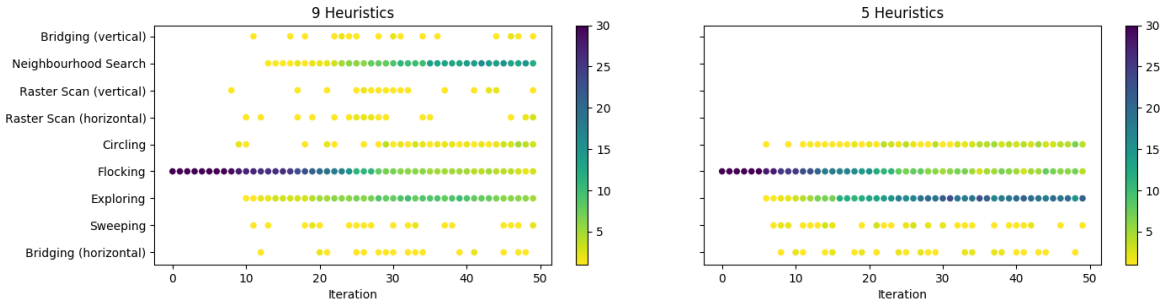
7.1.1 The MAB-based Hyper-heuristics

This section analyses the MAB-based adaptive hyper-heuristic described in Chapter 4. The common method of understanding a learned strategy is through observing the robot behaviours, however with hyper-heuristics, we are able to examine the policy at a lower level, that is which heuristic or combinations of heuristics are used, and make sense of the learned strategy. This can potentially help developers in verifying and modifying the algorithms.

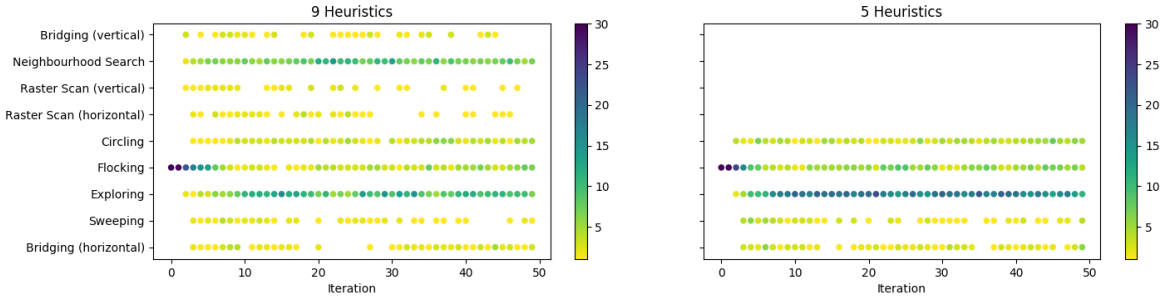
To demonstrate that the MAB-based hyper-heuristic is explainable, we compare two sets of heuristics on the same environments and experimental settings. Through examining the times a heuristic has been used in 30 repetitions, the learning process and the learned strategy can be easily analysed. Fig. 7.1 plots heuristics used during cleaning tasks in the four different environments. The colour bar indicates the number of times the heuristic has been used in that iteration. It can be seen in Fig. 7.1(a) and Fig. 7.1(b) that in empty environments and environments with obstacles, *Flocking* is found to be a good heuristic to start with, but in later explorations, *Exploring* and *Neighbourhood Search* become the better heuristics to use. In the two environments with gaps, *Flocking* is quickly found to be ineffective from the start, and quickly transitions to other heuristics. The reason behind this is that in environments without gaps, *Flocking* behaviour is able to rapidly cover a large area when there is a lot of dirt to collect, and as there is less reward in the environment, *Exploring* allows robots to



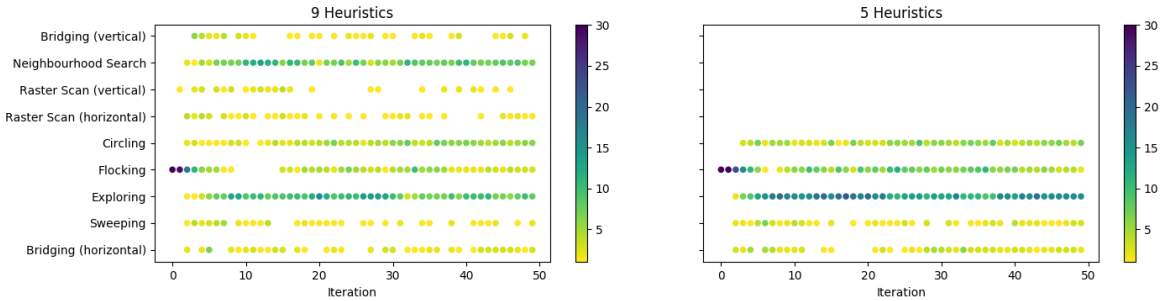
(a) Environment: Empty



(b) Environment: Obstacle



(c) Environment: Gaps



(d) Environment: Gaps and Obstacles

Fig. 7.1 Comparing heuristic distributions using a set of 9 and 5 heuristics

scatter into unexplored areas and obtain more reward. In the environments with gaps, each

surface is narrower, and since *Flocking* made the robots stay together, it is less efficient than *Exploring* which allows robots in the group to explore different directions.

In environments (c) and (d), *Bridging (horizontal)* is used more frequently than in environments without gaps, which agrees with the intuition that forming bridges is effective in navigating unconnected surfaces. This confirms that without re-programming or re-tuning, the same hyper-heuristic algorithm is able to pick out the better performing subsets of heuristics in different environments.

Like all the hyper-heuristic approaches, the performance of the system is largely influenced by the quality of the heuristic repository. This is because hyper-heuristic methods search in the space of heuristics, and rely on the resulting heuristics to generate a solution. If none of the heuristics in the given repository is effective or relevant for the given problem or scenario, the overall algorithm will have close to no capability of generating a solution. To show the robustness of our proposed hyper-heuristic method over different selections of heuristic repository, we investigate the heuristics used given a repository of 9 heuristics and a repository of 5 heuristics in Fig. 7.1. The 9 heuristics are described in Section 4.3, and the 5 heuristics repository comprises *Exploring*, *Simple Sweeping*, *Flocking*, *Bridging (horizontal)* and *Circling*. In the empty and the obstacles environment, the robots learned to use *Neighbourhood Search* together with *Exploring* when *Neighbourhood Search* is available.

It is also interesting to notice that *Raster Scan* is not selected as much as we expected when the behaviour is added to the repository. The behaviour as we observed requires the robots to stop and turn 90 degrees whenever an obstacle or another robot is encountered, and turning 90 degrees on the same spot is slow for two wheeled robots like those used in our experiments. A similar example is in most commercial floor cleaning robots: they usually perform *Exploring* like motions when cleaning a room rather than raster-scan motion because there are too many obstacles in rooms, which could cause robots to perform slow turning motions all the time. Without previous knowledge of how useful this heuristic is, the proposed

hyper-heuristic algorithm learned to automatically avoid this poorly-performing behaviour. Another example is Bridging (vertical) and Simple Sweeping, neither of them are selected frequently by robot cleaners, which agrees with the intuition that the gaps presented in the experiments are horizontal so the vertical movement is not effective, and Simple Sweeping gives too much overhead in re-positioning when obstacles and gaps are encountered.

The heuristics that are found to be inefficient in the cleaning task are still given the chance to be explored occasionally, balancing the exploitation of learned knowledge of heuristics.

Table 7.1 gives the detailed percentage of the heuristics used after their heuristic scores are stabilised. When the heuristic repository size is 9, heuristic H7 *Neighbourhood Search* is found to be the best in environments with no gaps and slightly less used in environments with gaps. In the environments with no gaps, H0 and H8 are used the least, both with a percentage of less than 2, because they are designed for bridging across surfaces. In the environments with gaps, the usage of H0 is increased to 5.6% and 5.33% because it is the relevant bridging behaviour for the given gaps. H8 was still explored occasionally, but was used much less than H0. This shows that the proposed method adapts to different environments by favouring different heuristics.

7.1.2 The Q-learning Method

The policy π_u as described in Chapter 5 in reinforcement learning is a probability distribution over actions given states, while the hyper-heuristic reinforcement learning is a probability distribution over heuristics given states. In the swarm robotics domain, state-heuristic policies tend to be more explainable than state-action policies because heuristics represent specific behaviours with logic and reasoning. This section analyses the learning outcome through examining the percentage of heuristics used over a number of learning iterations. Eight heuristics are used: *Bridging, Raster Scan (horizontal and vertical), Circling, Local Search, Flocking, Sweeping* and *Exploring*.

Table 7.1 Heuristic Percentage

Environment	Heuristic[a]								
	H0	***H1***	***H2***	***H3***	***H4***	***H5***	***H6***	***H7***	***H8***
Empty: 9 heuristics	1.47	3.20	28.80	16.67	4.27	1.60	1.20	41.33	1.47
Empty: 5 heuristics	2.40	6.27	54.53	29.33	7.47	–	–	–	–
Obstacle: 9 heuristics	1.87	1.87	26.00	18.93	6.53	2.13	2.13	38.93	1.60
Obstacle: 5 heuristics	2.40	2.67	64.40	19.60	10.93	–	–	–	–
Gaps: 9 heuristics	5.60	2.67	34.00	14.53	11.07	1.33	1.47	26.40	2.93
Gaps: 5 heuristics	6.67	3.60	57.47	18.13	14.13	–	–	–	–
Gaps and Obstacles: 9 heuristics	5.33	2.27	29.73	12.67	17.47	2.40	1.60	25.73	2.80
Gaps and Obstacles: 5 heuristics	9.07	5.47	47.20	20.27	18.00	–	–	–	–

[a]H0: Bridging (horizontal), H1: Simple Sweeping, H2: Exploring, H3: Flocking, H4: Circling, H5: Raster Scan (horizontal), H6: Raster Scan (vertical), H7: Neighbourhood Search, H8: Bridging (vertical)

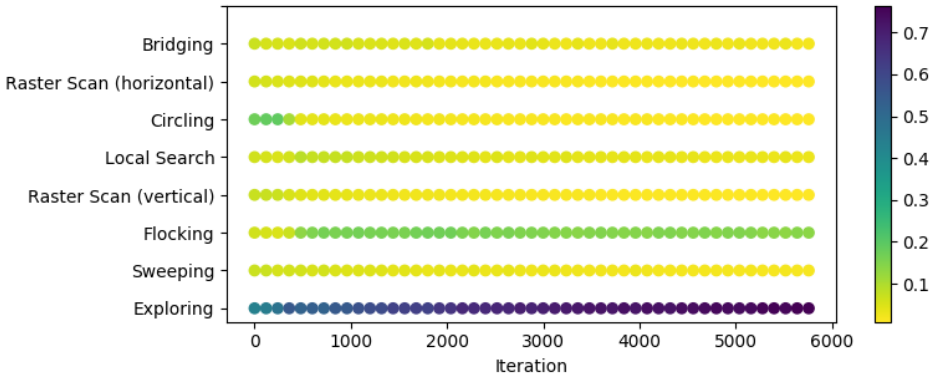


Fig. 7.2 Used heuristics by deep Q-learning hyper-heuristic in empty environment

Figure 7.2 plots the percentage of each heuristic used in the empty environment with 30 repetitions using the deep Q-learning hyper-heuristics, Figure 7.3 plots the same using Q-learning hyper-heuristics, and Figure 7.4 plots the MAB-based hyper-heuristics method. The percentage is calculated as the number of times a heuristic is used every 120 learning

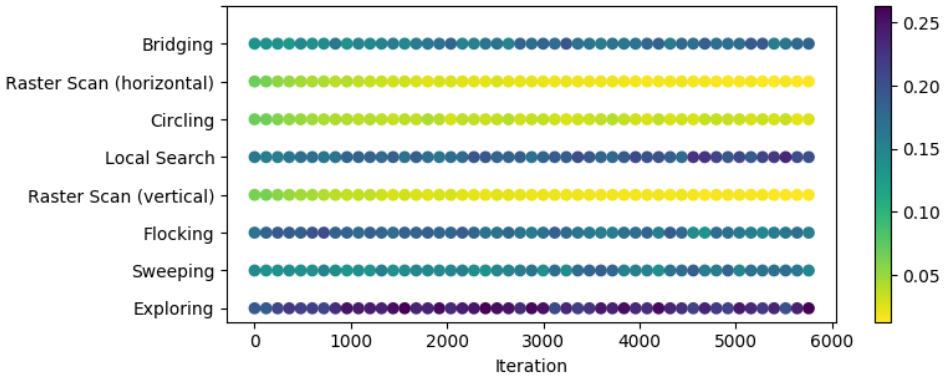


Fig. 7.3 Used heuristics by Q-learning hyper-heuristic in empty environment

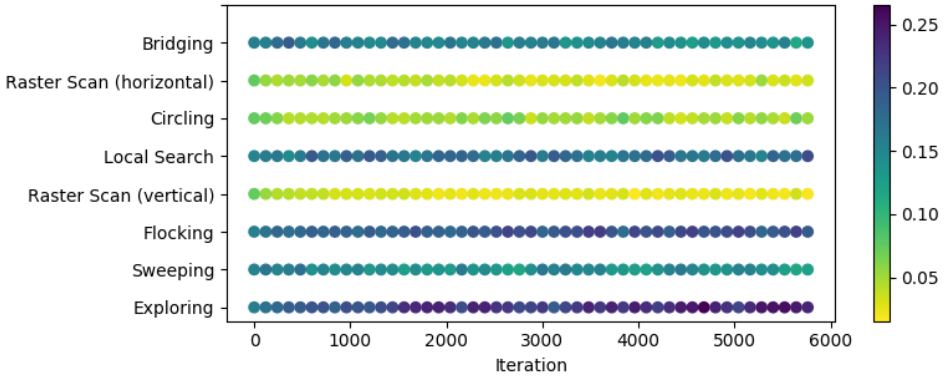


Fig. 7.4 Used heuristics by MAB-based hyper-heuristic in empty environment

iterations by 5 robots with 30 repetitions divided by 120 iterations $\times$ 5 robots $\times$ 30 repetitions $\times$ 8 heuristics. It should be noted that the plots only show the trends and proportion, not the exact policy. With all three methods, it can be seen that the heuristic *Exploring* is the most frequently used heuristic, but all the heuristics have been used in the learned policy. With the deep Q-learning hyper-heuristics, the algorithm found *Circling* and *Exploring* to be a good combination to use at the start, then transited to a combination of *Flocking* and *Exploring* with a heavy weight of 76.4% on *Exploring*. The Q-learning hyper-heuristic method uses the heuristics with similar proportions at the start, and then only found *Raster Scan* and *Circling* to be ineffective. The MAB hyper-heuristic method finds a similar combination of effective heuristics, but with slightly different weights. The MAB-based hyper-heuristic used more *Flocking*, while the Q-learning hyper-heuristic used more *Sweeping* than the MAB-based

hyper-heuristic. From the previous performance evaluation as plotted in Figure 6.6, it is known that the deep Q-learning hyper-heuristic has significantly better performance than the other two methods, and it is the result of the distinctly different policy learned by the deep Q-learning method.

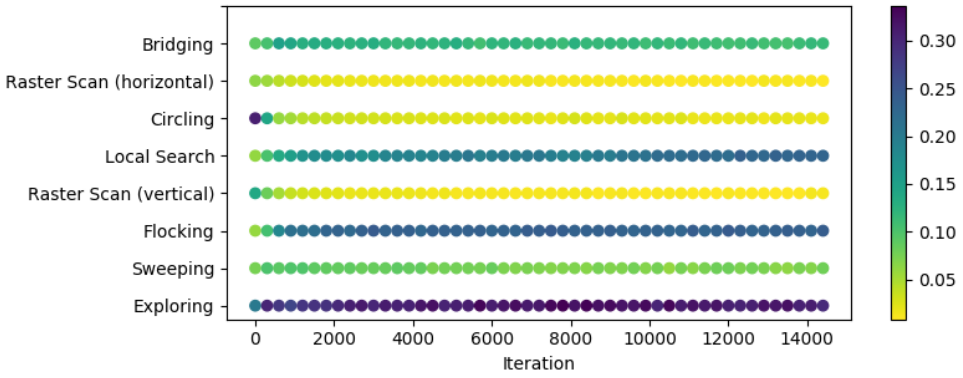


Fig. 7.5 Used heuristics by the deep Q-learning hyper-heuristic in obstacle environment

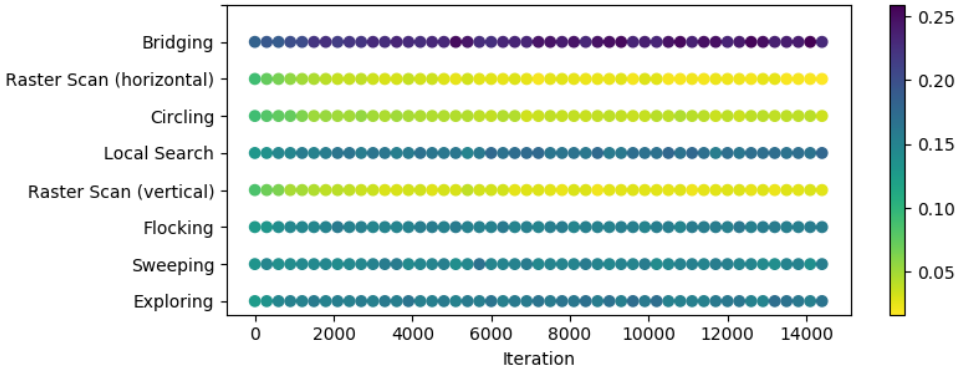


Fig. 7.6 Used heuristics by the Q-learning hyper-heuristic in obstacle environment

Figure 7.5, 7.6, 7.7 plots the heuristic percentage of the deep Q-learning hyper-heuristic, the Q-learning hyper-heuristic and the MAB-based hyper-heuristic every 300 iterations. In this scenario, the three algorithms all learned different strategies. With the deep Q-learning hyper-heuristic method, the starting heuristics are *Circling*, *Raster Scan* and *Exploring*, and the learned policy is a combination of *Local Search* at 23.02%, *Flocking* at 24.04% and *Exploring* at 29.79%. The Q-learning hyper-heuristic method finds the same heuristics, but

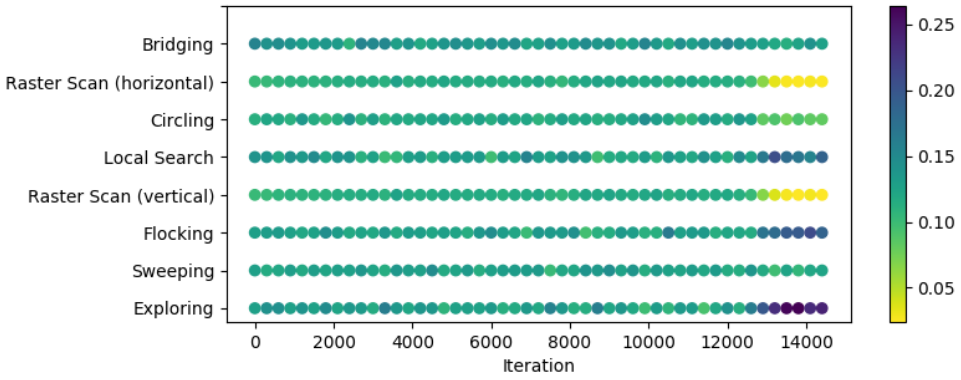


Fig. 7.7 Used heuristics by MAB-based hyper-heuristic in obstacle environment

with different weights on each heuristic. Starting from similar weights on each heuristic, the algorithm quickly optimises to use mainly *Exploring* at 24.15% together with *Bridging*, *Local Search*, *Flocking* and *Sweeping*. As can be seen in Table 7.2, the deep Q-learning Hyper-heuristic has used 13.2% more *Exploring* and 10.31% less *Bridging*. From the performance plot in Figure 6.7, it is known that the deep Q-learning hyper-heuristic has a better policy, and the fact that less *Bridging* leads to a better policy in obstacles environments agrees with the intuition. The MAB-based hyper-heuristic has a similar performance with the Q-learning hyper-heuristic in obstacles environments, and the policy composition is quite different. This shows that different policies can have similar performance.

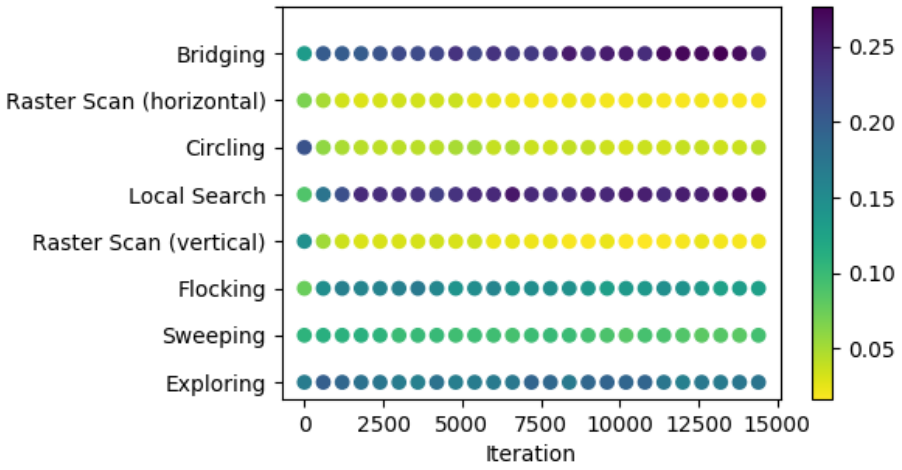


Fig. 7.8 Used heuristics by deep Q-learning based hyper-heuristic in gaps environment

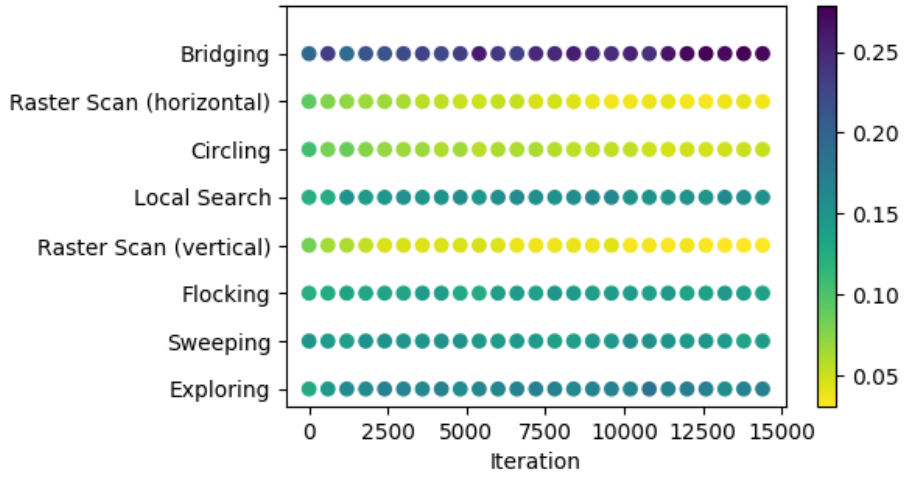


Fig. 7.9 Used heuristics by Q-learning based hyper-heuristic in gaps environment

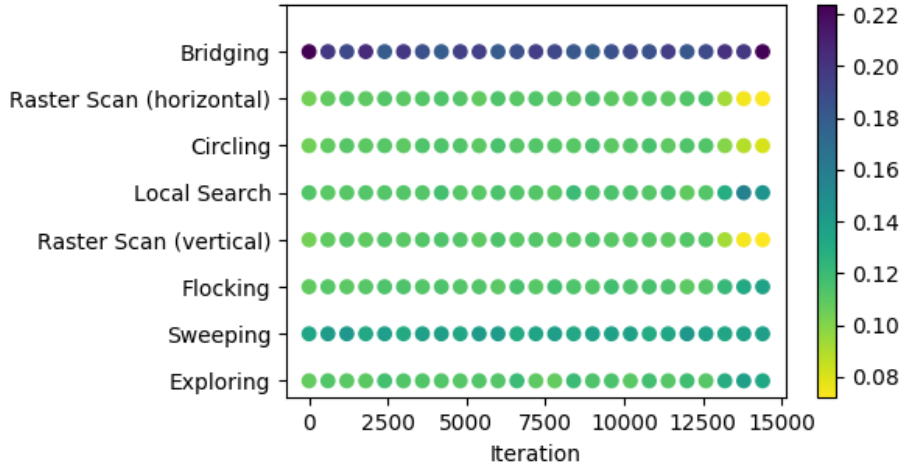


Fig. 7.10 Used heuristics by MAB-based hyper-heuristic in gaps environment

Figure 7.8 shows the heuristic percentage used every 600 iterations with the deep Q-learning hyper-heuristic method. The most used heuristics are *Bridging* and *Local Search*, which is different from the other environments, especially *Bridging* which is used much more, at 24.79%. From the performance evaluation, the Q-learning hyper-heuristic has a similar performance as the deep Q-learning hyper-heuristic. As shown in Figure 7.9, *Bridging* is the most used heuristic, and the rest of the heuristics distribution is similar to the deep Q-learning hyper-heuristic. Figure 7.10 shows the heuristics percentage using the MAB-based hyper-heuristic. It can be seen that the learned policy is similar to the other two methods, but

Bridging is used 22.25% of the time, and it is less than the other two methods, which are 24.79% and 27.46%. This lack of *Bridging* has been reflected in the performance evaluation, making it the worse method in gaps environment.

Table 7.2 Q-learning Heuristic Percentage

Environment	**Heuristic**[a]							
	H0	***H1***	***H2***	***H3***	***H4***	***H5***	***H6***	***H7***
Empty: deep-Q HH	2.25	0.95	0.98	3.19	1.09	14.04	1.94	76.37
Empty: Q HH	18.5	1.36	2.65	20.3	1.28	15.66	14.69	26.33
Empty: MAB HH	13.72	3.46	5.18	20.84	1.79	19.54	12.47	23.83
Obstacles: deep-Q HH	11.52	0.86	1.87	23.02	1.28	24.04	7.93	29.79
Obstacles: Q HH	23.10	1.68	3.50	17.94	2.76	15.82	15.60	16.59
Obstacles: MAB HH	12.79	2.55	8.30	18.86	2.41	18.85	12.42	24.15
Gaps: deep-Q HH	24.79	1.79	4.30	26.99	2.25	12.99	9.39	17.66
Gaps: Q HH	27.46	3.59	5.27	15.44	3.30	13.90	14.32	16.90
Gaps: MAB HH	22.25	7.27	8.11	14.51	7.19	13.65	13.87	13.32

[a]H0: Bridging, H1: Raster Scan (horizontal), H2: Circling, H3: Neighbourhood Search, H4: Raster Scan (vertical), H5: Flocking, H6: Sweeping, H7: Exploring

7.2 Scalability

The last section showed that the heuristic policy and learning can be explainable, and another benefit of the hyper-heuristic framework is that it is easily scalable. More robots can be added to the system, which makes the hyper-heuristic approach suitable for applications of different sizes. This section discusses the issues that can arise in the MAB-based hyper-heuristic system and the Q-learning hyper-heuristic system when there are more robots in the swarm.

7.2.1 The MAB hyper-heuristic

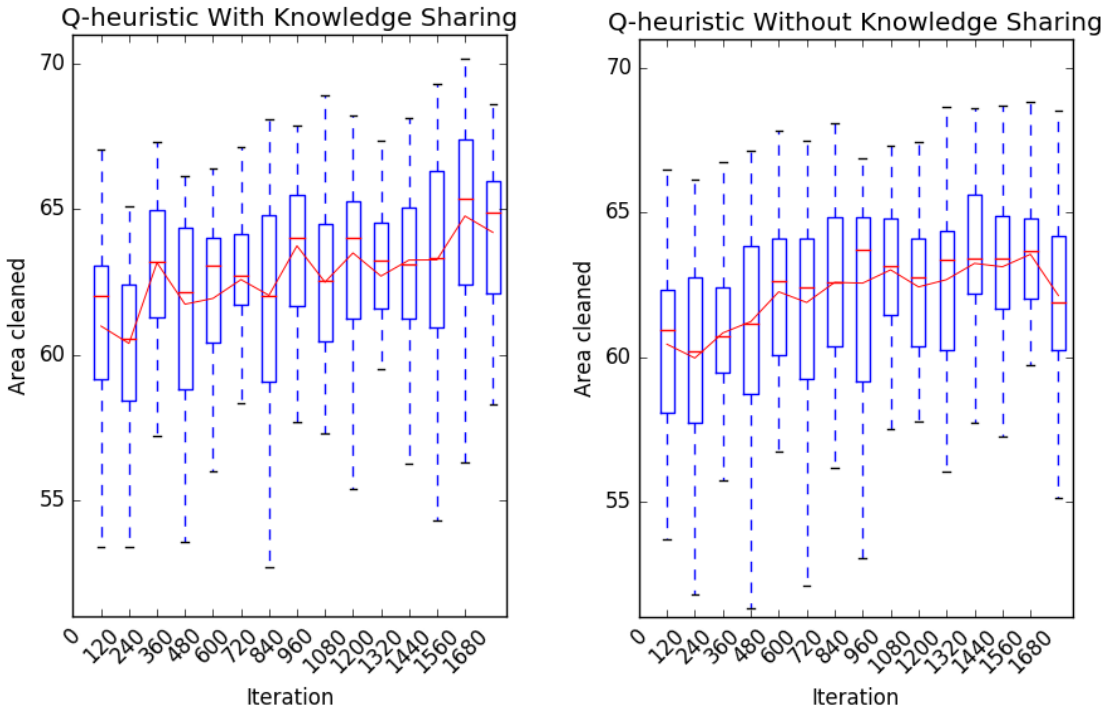
Table 7.3 Comparing the time to reach consensus in the collective decision making

Num of Robots	MAX-SUM & MAX-MIN	Voting
5	0.256s	4s
10	0.256s	32s
20	0.256s	65s

As the scale of the swarm increases, the time spent performing online learning depends heavily on the collective decision making strategy, as the initial individual learning is independent of other robots. We compare the time to reach consensus during the collective decision making process. Table 7.3 shows the comparison of robot swarms with size 5, 10 and 20 robots respectively. As can be seen, using MAX-SUM and MAX-MIN, the decision time is much faster and does not increase with the number of robots. In contrast, the time used for voting is longer even in a small swarm and increases quite rapidly with the number of robots. That means the complexity of voting is higher than that of MAX-SUM and MAX-MIN. For large scale swarm robots, MAX-SUM and MAX-MIN would be advantageous.

7.2.2 Q-learning hyper-heuristic

It has been shown in the previous section that the MAB-based hyper-heuristic method can scale to large numbers of robots, and this section demonstrates how the Q-learning hyper-heuristics behave with more robots. Since robots with Q-learning hyper-heuristics mostly learn by themselves without much communication, the communication latency is not a major factor that would affect the scalability of the system. Instead, since Q-learning robots cooperatively learn their policies, this section aims to find out if the learning is still effective when new robots are introduced to the swarm.



(a) (b)

Fig. 7.11 Five experienced robots + five new robots in empty environment: learning from other robots (knowledge sharing) and learning by themselves

In these experiments, we introduce 5 new robots to a group of 5 robots that had been exploring in the environment for a while, then examine if the hyper-heuristic coordination is still effective, and if robots are still able to learn to improve the performance over time. The first 5 robots are taken from the Q-learning hyper-heuristic experiments in Section 5.5.1, and after the good policies are found, 5 robots with no experience are placed into the environment. Each experiment is repeated 30 times for statistical significance. In this set of experiments, we demonstrate the performance of the Q-learning hyper-heuristic with the importance advising mechanism in Section 5.3 as well as the scenario when new robots learn independently (without asking and receiving knowledge from other robots).

Results of the 10 robots group performance are plotted in Figure 7.11. It can be seen that with 10 robots in the group, they are able to continuously learn to improve their performance

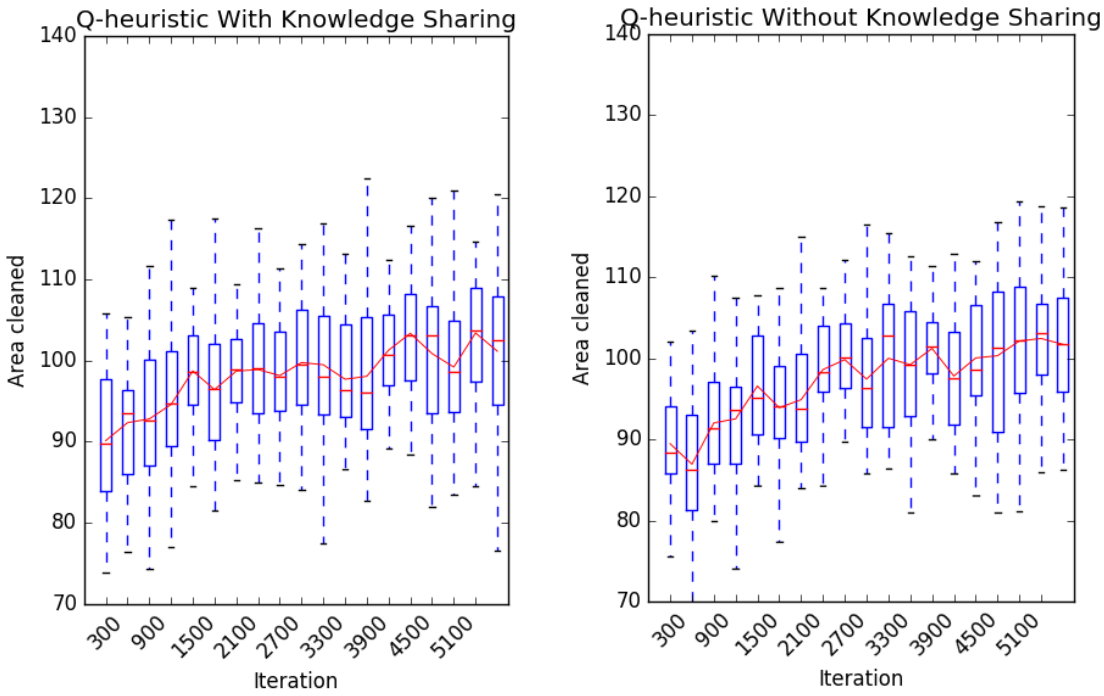
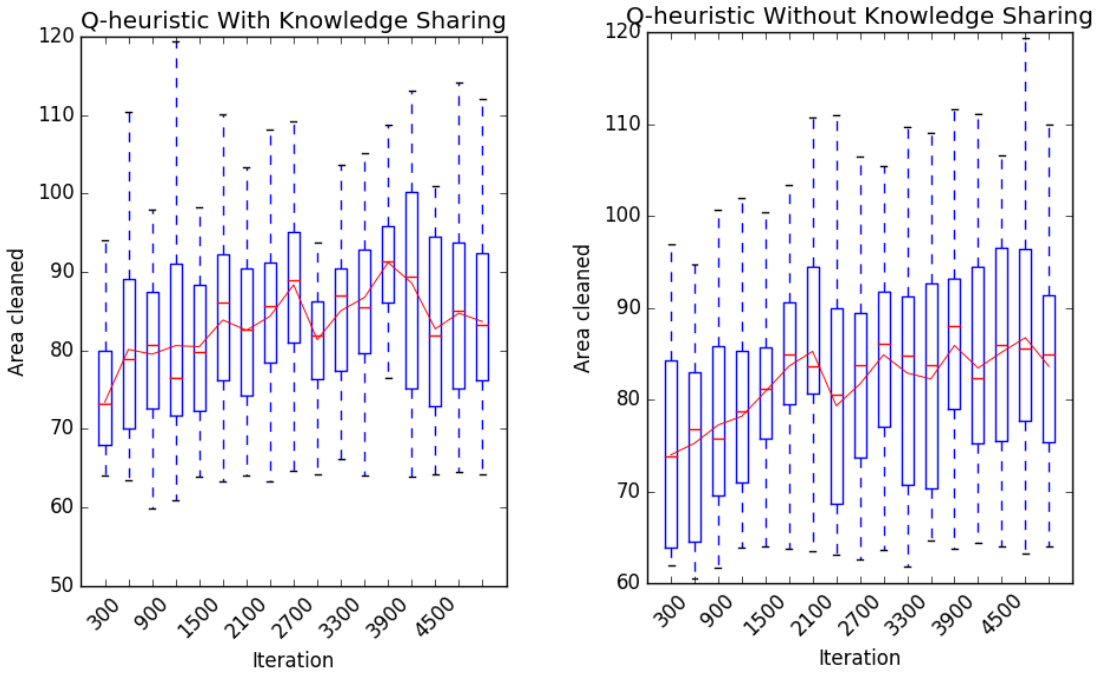


Fig. 7.12 Five experienced robots + five new robots in obstacles environment: learning from other robots (knowledge sharing) and learning by themselves

both with the assistance from experienced robots and from scratch. Within 1800 iterations, the group reward improves from 60.95 in the first 120 iterations to 64.80 and from 60.27 to 63.50 in the scenarios with and without knowledge sharing respectively. In the previous experiments in Section 6.2.2 with five robots in the empty environment, the group performance is approximately 35.83 at the end of 6000 iterations. With double the number of robots, the total reward increased by 80%. Although the performance did not double with double the robots, the heuristic learning is still effective because all robots started at the same place, and when there are more robots, they are more likely to encounter obstructions which make them waste more time dealing with collisions rather than cleaning. Comparing performance with and without group learning (knowledge sharing between old and new robots), robots with the importance advising mechanism learn faster than those learning independently, as



(a) (b)

Fig. 7.13 Five experienced robots + five new robots in gaps environment: learning from other robots (knowledge sharing) and learning by themselves

the reward with knowledge sharing is consistently higher in the first 600 iterations. After 600 iterations of learning, the independent learning robots catch up in performance, and the two groups' performance trends merge to be similar. This experiment shows that the hyper-heuristic framework can operate normally when more robots are added to the group in the empty environment.

Figure 7.12 shows the 10 robot hyper-heuristic learning in environment with obstacles. With knowledge sharing, the total reward obtained by 10 robots in the first 300 iterations is 89.14, and after 5700 iterations of training, the performance has improved to 102.28. In the previous experiments in Section 6.2.2 with five robots in the obstacles environment, the learning enabled the performance to rise from 39.10 to 53.34, which means that adding five more robots, the group performance was able to improve over time and improve 91% when the robot number is doubled. Without learning from experienced robots, the initial reward

is 88.41, and improved to be 101.17 at the end of 5700 iterations. The improvements show that the Q-learning hyper-heuristics are effective with 5 new robots added to the group in the obstacles environment. Furthermore, the importance advising mechanism does speed up the learning since the performance of group learning robots is higher than independent learning robots in the first 2400 iterations.

Similar trends can be found in the gaps environment, as shown in Figure 7.13. The group performance in the first 300 iterations with new robots in the system is 78.30 for both scenarios, and the learned policy has given a reward of approximately 90.8. In the previous experiments in Section 6.2.2 with five robots in the empty environment, the learning enabled the performance to rise from 40.49 to 50.91. This shows that with ten robots in the group, the overall performance can be improved 78.3%. The new robots can also improve their own policy through reinforcement learning of heuristics. When robots learn from each other using the knowledge sharing method, it can be seen that they can learn significantly faster in this environment. After 600 learning iterations, the robots with advising in learning reach a group reward of 85.15, while independent learning robots have a reward of 80.05, and the advantage stays until iteration 1500.

In summary, when new robots are added to the Q-learning hyper-heuristic group, new robots are able to integrate with the old system and the group improves as a whole. This makes the system scalable to larger groups of robots. The newly added robots also can accelerate their learning by the advising mechanism described in Section 5.3.

7.3 Summary

This chapter has shown more benefits of the framework. The policies generated by learning heuristics are explainable, and the framework is still effective with scalable group sizes.

We analysed how the policies learned by the proposed hyper-heuristic swarm robot controllers can be explainable, and that they are more explainable than the traditional action-

based policies. By analysing which heuristics are used at each stage of learning, it is easy to see how the policies evolve through hyper-heuristic online learning, and the policy can be understood intuitively by researchers and users. This chapter also compared the performances of different hyper-heuristic methods in ever changing environments, and has shown that the Q-learning hyper-heuristic methods are best at finding good policies. This chapter has also compared the benefits and drawbacks of the proposed hyper-heuristic methods in terms of scalability.

Chapter 8

Conclusions

The central theme of this thesis has been to address the integration of the hyper-heuristic methodology with swarm robot behaviour coordination in performing tasks in complex and dynamic environments. Based on the research within, we hereby list the contributions and conclude the investigation in this final chapter.

Recall the research questions posed in Section 1.1. We will summarise how our research has addressed these questions below:

- **How can hyper-heuristic approaches be formulated to coordinate swarm robots without centralised control, so swarm robots can better cope with unknown or dynamic environments?**

  Based on our study, we conclude that a hyper-heuristic framework can be established for swarm robot coordination, and the proposed swarm robot hyper-heuristic framework is effective across a range of unknown and dynamic environments. Our study shows that the hyper-heuristic learning is beneficial for the task performance, in particular, heuristic-based learning is more effective than action-based learning.

- **How can the learning of heuristics be improved by introducing different learning strategies?**

Q-learning can be introduced as a heuristic learning strategy under the hyper-heuristic framework and can improve its performance further.

Group learning, which allows robots to communicate knowledge and learn as a group, can also be beneficial and support learning. Group learning strategies can be applied to both MAB-based learning and Q-learning of heuristics.

- **How can state-of-the-art deep Q-learning be integrated to further improve the hyper-heuristic learning?**

  Our study confirmed that deep Q-learning can further improve the hyper-heuristic learning. More importantly, it also demonstrates that learning heuristics is better than learning actions, even when utilising deep learning.

Our study has also shown additional benefits of the proposed hyper-heuristic framework. The solutions generated are more explainable. The framework is also scalable and continues to perform well when more robots are integrated into the system.

Therefore, based on this study, we conclude that the hyper-heuristic learning in our framework is beneficial for swarm robot coordination under complex and dynamic environments, and the contribution of this study is significant.

This study has established a new framework and many further future work can be extended based on this work. We believe that it can open a new research area into many directions.

8.1 Future Works

The following research directions can be further explored in the future:

- Genetic Programming is a widely used hyper-heuristic for heuristic generation [14], and is not explored in this work. As mentioned in Chapter 2, existing methods are not ideal for multi-robot systems because the framework for evolving heuristics needs to

be redeveloped for each task. How can Genetic Programming method be integrated into the hyper-heuristic framework in a way that is beneficial for performing swarm robotics tasks?

- A wide range of environmental setups and scenarios in the application of cleaning multiple surfaces with self-assembling robots have been studied, what other gaps in multi-robot applications can be addressed with the hyper-heuristic framework?

- Chapter 6 presented a deep Q-learning based hyper-heuristic method for decentralised swarm robots, can the performance be further improved by using different deep reinforcement learning strategies or neural network architectures?

- Transfer learning allows machine learning agents to transfer the knowledge learned from one problem domain to a new one [132]. Can transfer learning be used with the hyper-heuristic framework to further improve the adaptability of the swarm robots?

References

[1] Ampatzis, C., Tuci, E., Trianni, V., Christensen, A. L., and Dorigo, M. (2009). Evolving self-assembly in autonomous homogeneous robots: Experiments with two physical robots. *Artificial Life*, 15(4):465–484.

[2] Applegate, D. L., Bixby, R. E., Chvatal, V., and Cook, W. J. (2006). *The traveling salesman problem: a computational study*. Princeton university press.

[3] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017a). A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*.

[4] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017b). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

[5] Bellman, R. (1966). Dynamic programming. *Science*, 153(3731):34–37.

[6] Bertsekas, D. P. (1997). Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334.

[7] Biazzini, M., Bánhelyi, B., Montresor, A., and Jelasity, M. (2009). Distributed hyper-heuristics for real parameter optimization. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1339–1346. ACM.

[8] Bilgin, B., Özcan, E., and Korkmaz, E. E. (2006). An experimental study on hyper-heuristics and exam timetabling. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 394–412. Springer.

[9] Bock, T., Bulgakow, A., Ashida, S., and Hergl, C. (2002). Facade cleaning robot for the skyscraper. *VDI BERICHTE*, 1679:627–632.

[10] Bohme, T., Schmucker, U., Elkmann, N., and Sack, M. (1998). Service robots for facade cleaning. In *IECON'98. Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society (Cat. No. 98CH36200)*, volume 2, pages 1204–1207. IEEE.

[11] Bontrager, P., Khalifa, A., Mendes, A., and Togelius, J. (2016). Matching games and algorithms for general video game playing. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[12] Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41.

[13] Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., and Schulenburg, S. (2003a). Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of metaheuristics*, pages 457–474. Springer.

[14] Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Ozcan, E., and Woodward, J. R. (2009). Exploring hyper-heuristic methodologies with genetic programming. In *Computational intelligence*, pages 177–201. Springer.

[15] Burke, E. K., Kendall, G., and Soubeiga, E. (2003b). A tabu-search hyperheuristic for timetabling and rostering. *Journal of heuristics*, 9(6):451–470.

[16] Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192.

[17] Burke, E. K., Petrovic, S., and Qu, R. (2006). Case-based heuristic selection for timetabling problems. *Journal of Scheduling*, 9(2):115–132.

[18] Chen, Y. F., Liu, M., Everett, M., and How, J. P. (2017). Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 285–292. IEEE.

[19] Choi, Y.-H. and Jung, K.-M. (2011). Windoro: The world's first commercialized window cleaning robot for domestic use. In *2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 131–136. IEEE.

[20] Choi, Y.-H., Lee, J.-Y., Lee, J.-D., and Lee, K.-E. (2012). Smart windoro v1. 0: Smart window cleaning robot. In *2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 116–119. IEEE.

[21] Consoli, S., Darby-Dowman, K., Mladenović, N., and Pérez, J. M. (2009). Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem. *European Journal of Operational Research*, 196(2):440–449.

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

[23] Cowling, P., Kendall, G., and Soubeiga, E. (2001). A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the 4th Metaheuristic International Conference, MIC*, volume 2001, pages 127–131. Citeseer.

[24] Da Silva, F. L., Glatt, R., and Costa, A. H. R. (2017). Simultaneously learning and advising in multiagent reinforcement learning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 1100–1108. International Foundation for Autonomous Agents and Multiagent Systems.

[25] Da Silva, F. L., Taylor, M. E., and Costa, A. H. R. (2018). Autonomously reusing knowledge in multiagent reinforcement learning. In *IJCAI*, pages 5487–5493.

[26] DaCosta, L., Fialho, A., Schoenauer, M., and Sebag, M. (2008). Adaptive operator selection with dynamic multi-armed bandits. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 913–920. ACM.

[27] Dasgupta, P., Cheng, K., and Banerjee, B. (2011). Adaptive multi-robot team reconfiguration using a policy-reuse reinforcement learning approach. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 330–345. Springer.

[28] Del Dottore, E., Sadeghi, A., Mondini, A., Mattoli, V., and Mazzolai, B. (2018). Toward growing robots: a historical evolution from cellular to plant-inspired robotics. *Frontiers in Robotics and AI*, 5:16.

[29] Di Gaspero, L. and Schaerf, A. (2000). Tabu search techniques for examination timetabling. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 104–117. Springer.

[30] Dorigo, M., Floreano, D., Gambardella, L. M., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., et al. (2013). Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*, 20(4):60–71.

[31] Dorigo, M., Trianni, V., Şahin, E., Groß, R., Labella, T. H., Baldassarre, G., Nolfi, S., Deneubourg, J., Mondada, F., Floreano, D., et al. (2004a). Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, 17(2-3):223–245.

[32] Dorigo, M., Tuci, E., Groß, R., Trianni, V., Labella, T. H., Nouyan, S., Ampatzis, C., Deneubourg, J., Baldassarre, G., Nolfi, S., et al. (2004b). The swarm-bots project. In *International Workshop on Swarm Robotics*, pages 31–44. Springer.

[33] Foerster, J., Assael, I. A., de Freitas, N., and Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145.

[34] Fujii, T., Arai, Y., Asama, H., and Endo, I. (1998). Multilayered reinforcement learning for complicated collision avoidance problems. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, volume 3, pages 2186–2191. IEEE.

[35] Fukuda, T. (1988). Self organizing robots based on cell structures-cebot. In *Proc. IEEE Int. Workshop on Intelligent Robots and Systems (IROS'88)*, pages 145–150.

[36] G. Ochoa and M. Hyde (2011). The hyper-heuristic framework featuring the domain barrier, the hyper-heuristic layer and the problem domain layer. http://www.asap.cs.nott.ac.uk/external/chesc2011/hyflex_description.html#Figure_2. [Online; accessed 5-July-2019].

[37] Garrido, P. and Castro, C. (2009). Stable solving of cvrps using hyperheuristics. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 255–262. ACM.

[38] Glover, F. W. and Kochenberger, G. A. (2006). *Handbook of metaheuristics*, volume 57. Springer Science & Business Media.

[39] Hansen, P., Mladenović, N., and Pérez, J. A. M. (2008). Variable neighbourhood search: methods and applications. *4OR*, 6(4):319–360.

[40] Hasan, K. M., Reza, K. J., et al. (2014). Path planning algorithm development for autonomous vacuum cleaner robots. In *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*, pages 1–6. IEEE.

[41] Hettiarachchi, S. D. (2007). *Distributed evolution for swarm robotics*. Citeseer.

[42] HOBOT (2015). Hobot-188. http://www.hobot.com.tw/Products_list_Window_Robot_188.php. [Online; accessed 12-July-2019].

[43] Hoff, N., Wood, R., and Nagpal, R. (2013). Distributed colony-level algorithm switching for robot swarm foraging. In *Distributed Autonomous Robotic Systems*, pages 417–430. Springer.

[44] Hutchinson, J. M. and Gigerenzer, G. (2005). Simple heuristics and rules of thumb: Where psychologists and behavioural biologists might meet. *Behavioural processes*, 69(2):97–124.

[45] Hüttenrauch, M., Adrian, S., Neumann, G., et al. (2019). Deep reinforcement learning for swarm systems. *Journal of Machine Learning Research*, 20(54):1–31.

[46] Imaoka, N., Roh, S.-g., Yusuke, N., and Hirose, S. (2010). Skyscraper-i: Tethered whole windows cleaning robot. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5460–5465. IEEE.

[47] Katsuki, Y., Ikeda, T., and Yamamoto, M. (2011). Development of a high efficiency and high reliable glass cleaning robot with a dirt detect sensor. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5133–5138. IEEE.

[48] Kendall, G. and Hussin, N. M. (2005). An investigation of a tabu-search-based hyper-heuristic for examination timetabling. In *Multidisciplinary Scheduling: Theory and Applications*, pages 309–328. Springer.

[49] Kennedy, J. (2006). Swarm intelligence. In *Handbook of nature-inspired and innovative computing*, pages 187–219. Springer.

[50] Kernbach, S., Hamann, H., Stradner, J., Thenius, R., Schmickl, T., Crailsheim, K., van Rossum, A. C., Sebag, M., Bredeche, N., Yao, Y., et al. (2009). On adaptive self-organization in artificial robot organisms. In *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, pages 33–43. IEEE.

[51] Kernbach, S., Meister, E., Schlachter, F., Jebens, K., Szymanski, M., Liedke, J., Laneri, D., Winkler, L., Schmickl, T., Thenius, R., et al. (2008). Symbiotic robot organisms: Replicator and symbrion projects. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pages 62–69. ACM.

[52] Keshmiri, S. and Payandeh, S. (2009). A centralized framework to multi-robots formation control: Theory and application. In *Collaborative Agents-Research and Development*, pages 85–98. Springer.

[53] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer.

[54] Khoshnevis, B. and Bekey, G. (1998). Centralized sensing and control of multiple mobile robots. *Computers & industrial engineering*, 35(3-4):503–506.

[55] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[56] Kiraz, B., Etaner-Uyar, A. Ş., and Özcan, E. (2013). Selection hyper-heuristics in dynamic environments. *Journal of the Operational Research Society*, 64(12):1753–1769.

[57] Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2619–2624. IEEE.

[58] Kormushev, P., Calinon, S., and Caldwell, D. G. (2010). Robot motor skill coordination with em-based reinforcement learning. In *2010 IEEE/RSJ international conference on intelligent robots and systems*, pages 3232–3237. IEEE.

[59] Krose, B. J. and Van Dam, J. W. (1992). Adaptive state space quantisation for reinforcement learning of collision-free navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 1327–1332. IEEE.

[60] Kuhn, M. and Johnson, K. (2013). *Applied predictive modeling*, volume 26. Springer.

[61] Kurokawa, H., Tomita, K., Kamimura, A., Kokaji, S., Hasuo, T., and Murata, S. (2008). Distributed self-reconfiguration of m-tran iii modular robotic system. *The International Journal of Robotics Research*, 27(3-4):373–386.

[62] Kuyucu, T., Tanev, I., and Shimohara, K. (2013). Hormone-inspired behaviour switching for the control of collective robotic organisms. *Robotics*, 2(3):165–184.

[63] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

[64] Levi, P., Meister, E., Van R, A., Krajnik, T., Vonasek, V., Stepan, P., Liu, W., and Caparrelli, F. (2014). A cognitive architecture for modular and self-reconfigurable robots. In *Systems Conference*, pages 465–472. IEEE.

[65] Lin, L.-J. (1993). Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

[66] Lipowski, A. and Lipowska, D. (2012). Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196.

[67] Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., Van Der Laak, J. A., Van Ginneken, B., and Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88.

[68] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier.

[69] Liu, J., Jiang, H., Li, Z., and Hu, H. (2009). A small window-cleaning robot for domestic use. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, volume 2, pages 262–266. IEEE.

[70] Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., and Pietikäinen, M. (2020). Deep learning for generic object detection: A survey. *International journal of computer vision*, 128(2):261–318.

[71] Liu, W. and Winfield, A. F. (2010). Autonomous morphogenesis in self-assembling robots using ir-based sensing and local communications. In *International Conference on Swarm Intelligence*, pages 107–118. Springer.

[72] Long, P., Fanl, T., Liao, X., Liu, W., Zhang, H., and Pan, J. (2018). Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6252–6259. IEEE.

[73] López-Camacho, E., Terashima-Marin, H., Ross, P., and Ochoa, G. (2014). A unified hyper-heuristic framework for solving bin packing problems. *Expert Systems with Applications*, 41(15):6876–6889.

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

[75] Lu, J. J. and Zhang, M. (2013). *Heuristic Search*, pages 885–886. Springer New York, New York, NY.

[76] Luna, R. and Bekris, K. E. (2011). Efficient and complete centralized multi-robot path planning. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3268–3275. IEEE.

[77] Mac, T. T., Copot, C., Tran, D. T., and De Keyser, R. (2016). Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems*, 86:13–28.

[78] Marzinotto, A., Colledanchise, M., Smith, C., and Ögren, P. (2014). Towards a unified behavior trees framework for robot control. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 5420–5427. IEEE.

[79] Matarić, M. J. (1997). Reinforcement learning in the multi-robot domain. In *Robot colonies*, pages 73–83. Springer.

[80] Mataric, M. J. (1998). Using communication to reduce locality in distributed multiagent learning. *Journal of experimental & theoretical artificial intelligence*, 10(3):357–369.

[81] McKnight, P. E. and Najab, J. (2010). Mann-whitney u test. *Corsini Encyclopedia of Psychology*.

[82] Mendes, A., Togelius, J., and Nealen, A. (2016). Hyper-heuristic general video game playing. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE.

[83] Michel, O. (1998). Webots: Symbiosis between virtual and real mobile robots. In *International Conference on Virtual Worlds*, pages 254–263. Springer.

[84] Michel, O. (2004). Cyberbotics ltd. webots™: professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):5.

[85] Miyake, T., Ishihara, H., Shoji, R., and Yoshida, S. (2006). Development of small-size window cleaning robot a traveling direction control on vertical surface using accelerometer. In *2006 International Conference on Mechatronics and Automation*, pages 1302–1307. IEEE.

[86] Mlot, N. J., Tovey, C. A., and Hu, D. L. (2011). Fire ants self-assemble into waterproof rafts to survive floods. *Proceedings of the National Academy of Sciences*, 108(19):7669–7673.

[87] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[88] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

[89] Nagavalli, S., Chakraborty, N., and Sycara, K. (2017). Automated sequencing of swarm behaviors for supervisory control of robotic swarms. In *Robotics and Automation, IEEE International Conference on*, pages 2674–2681. IEEE.

[90] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[91] Napp, N., Burden, S., and Klavins, E. (2011). Setpoint regulation for stochastically interacting robots. *Autonomous Robots*, 30(1):57–71.

[92] Nareyek, A. (2003). Choosing search heuristics by non-stationary reinforcement learning. In *Metaheuristics: Computer decision-making*, pages 523–544. Springer.

[93] Nicolescu, M. N. and Matarić, M. J. (2002). A hierarchical architecture for behavior-based robots. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 227–233. ACM.

[94] Nolfi, S., Floreano, D., and Floreano, D. D. (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press.

[95] Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J. A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A. J., Petrovic, S., et al. (2012). Hyflex: A benchmark framework for cross-domain heuristic search. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 136–147. Springer.

[96] Oh, H., Shirazi, A. R., Sun, C., and Jin, Y. (2017). Bio-inspired self-organising multi-robot pattern formation: A review. *Robotics and Autonomous Systems*, 91:83–100.

[97] Omidshafiei, S., Pazis, J., Amato, C., How, J. P., and Vian, J. (2017). Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2681–2690. JMLR. org.

[98] Ouelhadj, D. and Petrovic, S. (2008). A cooperative distributed hyper-heuristic framework for scheduling. In *Systems, Man and Cybernetics, IEEE International Conference on*, pages 2560–2565. IEEE.

[99] Özcan, E., Mısır, M., and Kheiri, A. (2013). Group decision making hyper-heuristics for function optimisation. In *UK Workshop on Computational Intelligence*, pages 327–333. IEEE.

[100] Pillay, N. (2016). A review of hyper-heuristics for educational timetabling. *Annals of Operations Research*, 239(1):3–38.

[101] Pillay, N. and Qu, R. (2018). *Selection Perturbative Hyper-Heuristics*, pages 17–23. Springer International Publishing, Cham.

[102] Porta, J. M. and Celaya, E. (2001). Efficient gait generation using reinforcement learning. In *Proceedings of the Fourth International Conference on Climbing and Walking Robots*, pages 411–418.

[103] Rattadilok, P., Gaw, A., and Kwan, R. S. (2004). Distributed choice function hyper-heuristics for timetabling and scheduling. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 51–67. Springer.

[104] Reeves, C. R. (1994). Genetic algorithms and neighbourhood search. In *AISB Workshop on Evolutionary Computing*, pages 115–130. Springer.

[105] Reeves, C. R. (1996). Heuristic search methods: A review. *Operational Research-Keynote Papers*, pages 122–149.

[106] Rekleitis, I., New, A. P., Rankin, E. S., and Choset, H. (2008). Efficient boustrophedon multi-robot coverage: an algorithmic approach. *Annals of Mathematics and Artificial Intelligence*, 52(2-4):109–142.

[107] Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH computer graphics*, 21(4):25–34.

[108] Riedmiller, M., Gabel, T., Hafner, R., and Lange, S. (2009). Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73.

[109] Rodríguez, J. V., Petrovic, S., and Salhi, A. (2007). A combined meta-heuristic with hyper-heuristic approach to the scheduling of the hybrid flow shop with sequence dependent setup times and uniform machines. In *Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications. MISTA: Paris, France*, pages 506–513.

[110] Ross, H.-L. F. P. and Corne, D. (1994). A promising hybrid ga/heuristic approach for open-shop scheduling problems. In *Proc. 11¡ sup¿ th¡/sup¿ European Conference on Artificial Intelligence*, pages 590–594.

[111] Ross, P. (2005). Hyper-heuristics. In *Search methodologies*, pages 529–556. Springer.

[112] Ross, P., Schulenburg, S., Marín-Bläzquez, J. G., and Hart, E. (2002). Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 942–948. Morgan Kaufmann Publishers Inc.

[113] Ross, S., Chaib-draa, B., and Pineau, J. (2008). Bayesian reinforcement learning in continuous pomdps with application to robot navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 2845–2851. IEEE.

[114] Rubenstein, M., Cornejo, A., and Nagpal, R. (2014). Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799.

[115] Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

[116] Sabar, N. R., Ayob, M., Kendall, G., and Qu, R. (2015). A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Cybernetics*, 45(2):217–228.

[117] Şahin, E. (2004). Swarm robotics: From sources of inspiration to domains of application. In *International workshop on swarm robotics*, pages 10–20. Springer.

[118] Salemi, B. and Shen, W.-M. (2004). Distributed behavior collaboration for self-reconfigurable robots. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 4178–4183. IEEE.

[119] Shen, H., Yosinski, J., Kormushev, P., Caldwell, D. G., and Lipson, H. (2012). Learning fast quadruped robot gaits with the rl power spline parameterization. *Cybernetics and Information Technologies*, 12(3):66–75.

[120] Soubeiga, E. (2003). *Development and application of hyperheuristics to personnel scheduling*. PhD thesis, University of Nottingham.

[121] Soysal, O. and Sahin, E. (2005). Probabilistic aggregation strategies in swarm robotic systems. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 325–332. IEEE.

[122] Spears, W. M., Spears, D. F., Hamann, J. C., and Heil, R. (2004). Distributed, physics-based control of swarms of vehicles. *Autonomous Robots*, 17(2-3):137–162.

[123] Sperati, V., Trianni, V., and Nolfi, S. (2008). Evolving coordinated group behaviours through maximisation of mean mutual information. *Swarm Intelligence*, 2(2-4):73–95.

[124] Spradling, M., Goldsmith, J., Liu, X., Dadi, C., and Li, Z. (2013). Roles and teams hedonic game. In *International Conference on Algorithmic DecisionTheory*, pages 351–362. Springer.

[125] Stern, R. and Lelis, L. H. (2016). What's hot in heuristic search. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[126] Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 2. MIT press Cambridge.

[127] Tai, L., Paolo, G., and Liu, M. (2017). Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE.

[128] Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337.

[129] Tavares, A. R., Anbalagan, S., Marcolino, L. S., and Chaimowicz, L. (2018). Algorithms or actions? a study in large-scale reinforcement learning. In *IJCAI*, pages 2717–2723.

[130] Taylor, M. E., Carboni, N., Fachantidis, A., Vlahavas, I., and Torrey, L. (2014). Reinforcement learning agents providing advice in complex video games. *Connection Science*, 26(1):45–63.

[131] Thrun, S. and Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*.

[132] Torrey, L. and Shavlik, J. (2010). Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI Global.

[133] Uludag, G., Kiraz, B., Uyar, A. E., and Özcan, E. (2012). Heuristic selection in a multi-phase hybrid approach for dynamic environments. In *Computational Intelligence (UKCI), 2012 12th UK Workshop on*, pages 1–8. IEEE.

[134] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[135] van Otterlo, M. and Wiering, M. (2012). Reinforcement learning and markov decision processes. In *Reinforcement Learning*, pages 3–42. Springer.

[136] Wang, Y. and De Silva, C. W. (2006). Multi-robot box-pushing: Single-agent q-learning vs. team q-learning. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3694–3699. IEEE.

[137] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

[138] Werfel, J., Petersen, K., and Nagpal, R. (2014). Designing collective behavior in a termite-inspired robot construction team. *Science*, 343(6172):754–758.

[139] Whitesides, G. M. and Grzybowski, B. (2002). Self-assembly at all scales. *Science*, 295(5564):2418–2421.

[140] Yang, B. and Liu, M. (2018). Keeping in touch with collaborative uavs: A deep reinforcement learning approach. In *IJCAI*, pages 562–568.

[141] Yasuda, T. and Ohkura, K. (2008). A reinforcement learning technique with an adaptive action generator for a multi-robot system. In *International Conference on Simulation of Adaptive Behavior*, pages 250–259. Springer.

[142] Yim, M., Duff, D. G., and Roufas, K. D. (2000). Polybot: a modular reconfigurable robot. In *ICRA*, pages 514–520.

[143] Yu, S. and Barca, J. C. (2015). Autonomous formation selection for ground moving multi-robot systems. In *IEEE International Conference on Advanced Intelligent Mechatronics*, pages 54–59. IEEE.

[144] Zhang, C., Patras, P., and Haddadi, H. (2019). Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys & Tutorials*, 21(3):2224–2287.

[145] Zhang, H., Zhang, J., Zong, G., Wang, W., and Liu, R. (2006). Sky cleaner 3: A real pneumatic climbing robot for glass-wall cleaning. *IEEE Robotics & Automation Magazine*, 13(1):32–41.

Appendix A

Source Code

The source code of this project can be found on https://drive.google.com/open?id=1Nk-i-mRdevmEt_6rydCnwk_0WiaNNSNL.