Artificial Life and Robotics (2022) 27:652–662
https://doi.org/10.1007/s10015-022-00804-4

ORIGINAL ARTICLE

Largest coverage network in a robot swarm using reinforcement learning

Dalia S. Ibrahim[1,2] · **Andrew Vardy**[3]

Received: 23 May 2022 / Accepted: 15 September 2022 / Published online: 14 October 2022

Abstract
Establishing a large adaptive connected network for decentralized swarms is useful for their behavior to share information about the working environment. A hard-coded implementation is time-consuming to achieve. Therefore, we are motivated to explore the benefits of reinforcement learning (RL) to learn a suitable adaptive policy. We also explore the combined use of a scalar field, which was inspired by template pheromones in social insects. In this paper, we investigate using RL with low and high-resolution scalar fields to solve the largest covering network problem. Our results show that RL outperforms the hard-coded approach in the presence of the high-resolution scalar field.

Keywords Largest coverage network · Reinforcement learning · Scalar field · Swarm robotics

1 Introduction

Controlling the physical proximity between a swarm of robots can allow the swarm to gather more data and exchange it between them. Each robot could be responsible for a specific territory, if it faces a threat or needs help, it will send that information to its connected robots. Also, if there are some obstacles in front of some robots, they can propagate that information to others to avoid them. In the largest covering network (LCN) task, the robots should be connected to one cluster with the least possible overlap to cover the largest possible working area. Each robot can detect another robot if the other robot is in its range. These swarm robots work independently in a distributed manner with very simple computational power.

In mobile robots swarm, Panerati et al. [1] select a master robot, and the swarm performs a rendezvous around that master robot as an essential step to establish the connectivity of the swarm inside the coverage area. In [2], Francesca at al. presented an automatic control design for a robot swarm. They tested their controller on five different robotics tasks with LCN as one of these tasks. Mitaka et al. use static templates to help the robots to perform the required behavior. Researchers are inspired by social insects like ants, termites, wasps, and bees because these insects use pheromones and other cues to guide them to perform different activities in their societies [3]. For example, in building the royal chamber for the queen termite, the queen discharges a particular type of pheromone as a template to guide the worker termites to deposit the mud in a specific distance around it [4]. Similar to the pheromones in insects, Vardy [5] presents the scalar field as a global signal to guide a robot swarm in their work. The scalar field is introduced as a static template to guide the robots to create a two-dimensional enclosure using ambient objects by nudging them to form the desired shape. That scalar shape is projected on the working environment helps the robots construct an enclosure.

This work was presented in part at the joint symposium of the 27th International Symposium on Artificial Life and Robotics, the 7th International Symposium on BioComplexity, and the 5th International Symposium on Swarm Behavior and Bio-Inspired Robotics (Online, January 25-27, 2022).

✉ Dalia S. Ibrahim
dsibrahim@mun.ca

Andrew Vardy
av@mun.ca
http://bots.cs.mun.ca/

[1] Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada

[2] Department of Computer Systems, Faculty of Computer and Information Science, Ain-Shams University, Cairo, Egypt

[3] Department of Computer Science, Department of Electrical and Computer Engineering, Memorial University of Newfoundland, St. John's, NL, Canada

In [6], Strickland et al. use the power of reinforcement learning (RL) with scalar field to construct an enclosure and L-shape. The RL gives better results in terms of shape formation with a single type of robots, unlike [5], who used two types of robots.

Solving LCN is also useful in the initial deployment of fixed sensors, such as a wireless sensor network. Based on the specific target wanted to be covered, the users calculate the sensors' locations to be sure that target is fully covered. The sensors can be deployed in an unstructured or structured way, depending on the application. In the structured deployment, the place of the sensors are predetermined with the advantage of less cost of maintenance and good coverage. However, the sensors are randomly distributed in the environment in the unstructured deployment, so more sensors may cover the same area [7]. Also, in [8], they show how the overlapped sensors can form a disjoint cover set, and the only disjoint sensor should be active at a time to save their battery life.

In this paper, we address the spatial coverage of mobile robots to achieve global connectivity with minimum overlapping sensing range. The robots are placed randomly without any central control. They collectively explore the environment and use RL with a scalar field to build one connected network of robots. The resolution of the scalar field to be provided to the robots is an interesting quantity to explore. If a high-resolution scalar field (HRSF) is required, then this places certain constraints on the robots ability to sense this field. On the other hand, if a low-resolution scalar field (LRSF) suffices, then it may be possible to implement our technique on simpler robots with fewer and less accurate sensors.

Using RL, the robots can automatically learn how to establish a large coverage area with minimum overlap between communication ranges of the other robots. The working environment is examined in both LRSF and HRSF situations. The rewards in RL are determined based on the detected number of robots in range and the total coverage area. The results are obtained from a simulation environment and compared to the hard-coded implementations.

2 Proposed methods

The LCN's objective is to maximize the coverage area by a robot swarm while maintaining the connection among these robots. Tabular Q-Learning shows promising results when applied in a dynamic environment [6, 9]. Therefore, we propose to use Q-Learning as a RL algorithm and the hard-coded approaches for solving LCN in two different environments LRSF and HRSF. Robots try to form one connected cluster with minimum intersection areas among them.

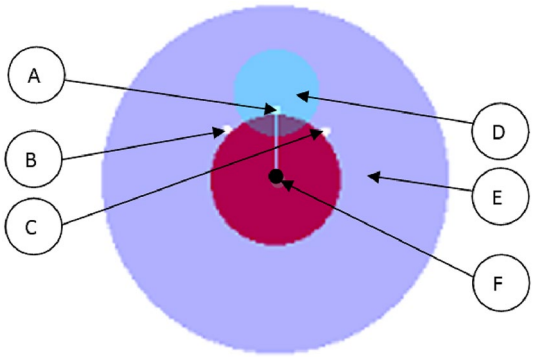

Fig. 1 Robot's sensors. **A**, **B**, and **C** are floor sensors. **D** is the obstacle sensor. **E** is the communication range sensor. **F** is the position sensor

In HRSF, the scalar field values are in the range [0,1], where the center of the environment equals one and decreases towards the environment's border. On the other hand, in LRSF, the working environment is virtually divided into four quadrants.

The operated robots are equipped with five sensors, as shown in Fig. 1. The A, B and C sensors measure the intensity of light projected on the floor, which we named center, left, and right sensors, respectively. These sensors return the scalar value in a range from zero to one. The D sensor is the obstacle sensor to detect the presence of the walls and other robots. The E sensor is the communication sensor that covers a circular region of the space around the robot. The F sensor is the position sensor that returns the robot's x and y coordinates and is used only in the low resolution scalar field environment.

2.1 Low-resolution scalar field (LRSF)

The robots depend on the readings of their communication range sensor and obstacle sensors. So using the scalar field guides the robots and gives them an idea of where they are.

In our proposed solution LRSF, we divided the working environment into four quadrants (Q1, Q2, Q3 and Q4), so the expected shape with a large number of robots will be spread over the environment. Also, with this division, the robots could explore the whole environment, switch among different quadrants, and take random positions in these quadrants, giving them more options to find their best positions. At the beginning of the simulation, each robot is placed randomly in a random quadrant. The difference between hard-coded and RL implementations is how the robot selects the next quadrant in each time step.

2.1.1 Hard-coded implementation

The robot has systematic moves. It always moves to a random position in the lower integer number. For example, if a robot is placed in the third quadrant, it will move to a random position in the second quadrant. Any robot located in the first quadrant keeps selecting positions randomly and waiting for any other robot to join its cluster. Once two robots or more form a small cluster in the first quadrant, the robots in this cluster reduce the intersection area among them using Eq. 1. We added a random floating number [0,1] to the backward speed to ensure the two intersected robots take different values to reduce their intersection areas.

$$\text{Backward Speed}\,(t) = -1\left(\mathrm{e}^{\frac{\text{Max Intersected Area}\,(t)}{\text{Robot's Range}}} + \text{Rand\_Float}\,(0,1)\right) \tag{1}$$

If they are connected with 2% of their connected range (maximum allowed intersection), they stop moving and wait for other robots to reach them, whether the coming robots reach the first quadrant or are on their way.

The control Algorithm is presented as Algorithm 1 with the function getNextMove provided separately. This function calculates the forward speed and angular velocity required for the robot to move from its current position to the selected random position in the low integer quadrant.

Algorithm 1 Hard-Coded for LRSF

Input: Robot's Position, $\overrightarrow{Pos}$
Obstacle sensor's reading , obs
Number of neighbors, $Neighbors$
Max intersection with neighbors, A_{Max}
Robot's steer angle, $currAngle$
Output: Forward velocity, v
Angular velocity, ω
State variables: Robot's quadrant, Qid

$\rho \leftarrow 0.02 \times Robot'sRange$ ▷ Minimum Intersection
if obs **then** ▷ Avoid Obstacles
  return ω_{max}
if t is 1 **then** ▷ Beginning of the simulation
  $Qid \leftarrow getRandomQuadrant()$
else if $A_{Max} > \rho$ **then**
  if $((Qid == 1 \wedge Neighbors \geq 2) \vee Neighbors \geq \frac{Robots}{2})$ **then**
    ▷ robot constructs sub cluster
    $v \leftarrow BackwardSpeed$ ▷ move backwards
    return v, ω
else if $A_{Max} \leq \rho \wedge Neighbors \geq \frac{Robots}{2}$ **then**
  ▷ robot connected with min intersection
  $v, \omega \leftarrow 0$ ▷ Stop movement
  return v, ω
else if $Qid \neq 1$ **then**
  $Qid \leftarrow Qid - 1$ ▷ Assign Qid to the lower integer
$\overrightarrow{NewPos} \leftarrow getRandPos(Qid, \overrightarrow{Pos})$
$v, \omega \leftarrow getNextMove(\overrightarrow{Pos}, \overrightarrow{NewPos}, currAngle)$
return v, ω

Function getNextMove

Input: Robot's Position, $\overrightarrow{Pos}$
Destination's Position, $\overrightarrow{NewPos}$
The robot's steer angle, $currAngle$
Output: Forward velocity, v
Angular velocity, ω

$\Delta X \leftarrow \overrightarrow{NewPos}.x - \overrightarrow{Pos}.x$
$\Delta Y \leftarrow \overrightarrow{NewPos}.y - \overrightarrow{Pos}.y$
$Kv \leftarrow 0.01$
$v \leftarrow Kv \times \sqrt{\Delta X^2 + \Delta Y^2}$
$Angle \leftarrow Atan2(\Delta Y, \Delta X)$
$Kh \leftarrow 0.5$
$\Delta Angle \leftarrow Angle - CurrAngle$
$\omega \leftarrow -Kh \times \Delta Angle$
return v, ω

For example, in Fig. 2 demonstrates the robot's movements to establish the one connected network. The colors of robots show which quadrants it belongs to, the yellow color means this robot stop moving, and the black arrows indicate the robot's actions on the next move. In Fig. 2(A), robots in Q2 to Q3 choose to decrease their quadrant number, hoping it might find any other robots. Robot #1 in Q1 waits for other robots to come to its quadrant, so it selects a random position; perhaps it finds other robots. In (B), robot #2 reaches the Q1 and finds robot #1, but their intersection area is greater than 2% of the robot's range size. Therefore, they choose to move back to reduce their intersection area with backward speed, as shown in the Eq.1. Robot #3 and #4 are still reducing the number of their quadrants searching for other robots. In (C), robots #2, #3, and #4 are connected with big intersection areas, so they choose to move backwards. However, robot #1 is connected with a good intersection area, so it stops moving, and its color changes to yellow. Based on (C) actions, robots #2 and #3 are still connected as with a good intersection area shown in (D), but the total number of robots in their range is less than the total operated robots. Also, robot #1 and #4 is disconnected. So, robots #2, #3 , and #4 decrease their quadrant numbers, and robot #1 gets a new random position in Q1. In (E), robots #3 and #4 are connected with a good intersection area, but robots #1 and #2 need to move back to adjust their intersection areas. In (F), all robots stop moving and establish one connected cluster with minimum intersections.

2.1.2 Reinforcement learning implementation

The states for the Q-Learning algorithm are determined based on a robot's point of view and how many neighbors are in its range. Besides, the robots have limited memory and computation capabilities; we tried to find the minimum possible representation for state and action space.

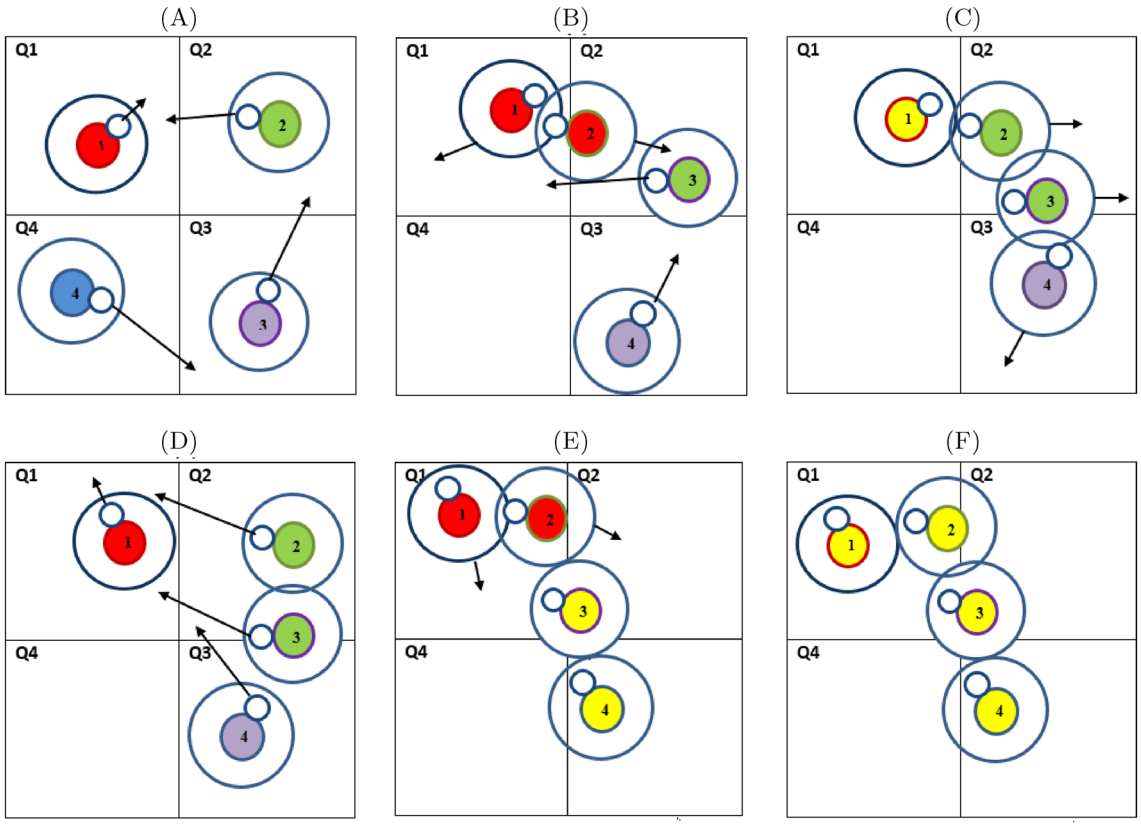

Fig. 2 The sequence of snapshots shows the hard-coded implementation in the presence of a low-resolution scalar field; the colored robots indicate which quadrants they are; the black arrows show the robots' next decision based on these situations

Therefore, we discretize the many states into four states $\{n = 0, n < \frac{2}{3}TR, \frac{2}{3}TR <= n < TR, n = TR\}$ where n is number of neighbors and TR is number of Total Robots.

The robot chooses between eight actions: normal movement in which it goes to a random spot in the lower integer quadrant; selecting any other quadrant from the environment (Q1, Q2, Q3, Q4); or increasing or decreasing its forward speed, or stop.

In the reward function, we focus on maximizing the number of connected robots and increasing the coverage area as much as possible by minimizing the intersection area between neighbors.

$$R_i(t) = \frac{Connected\ Neighbors\ (t)}{Operated\ Robots} + \frac{Coverage\ Area\ (t)}{Max\ Coverage\ Area} - 1 \tag{2}$$

As shown on Eq. (2), in the worst scenario, when a robot moves alone and does not belong to any cluster, the reward function approximately equals -1. And, when all robots are connected, the reward value is directly proportional to the coverage area.

2.2 High-resolution scalar field (HRSF)

We projected the scalar field on the working environment with a central light source. The grid value is between one and zero based on the intensity of the projected light on the floor. The robots use the three floor sensors readings, which measure the scalar field value projected on the environment. With the scalar field's guidance, the robots move towards the lower scalar values. Once the robots reach the lowest scalar value, they save this value τ as the desired contour line to use in their circular movement. The robots follow the orbiting algorithm illustrated in [5], so all robots are aligned and

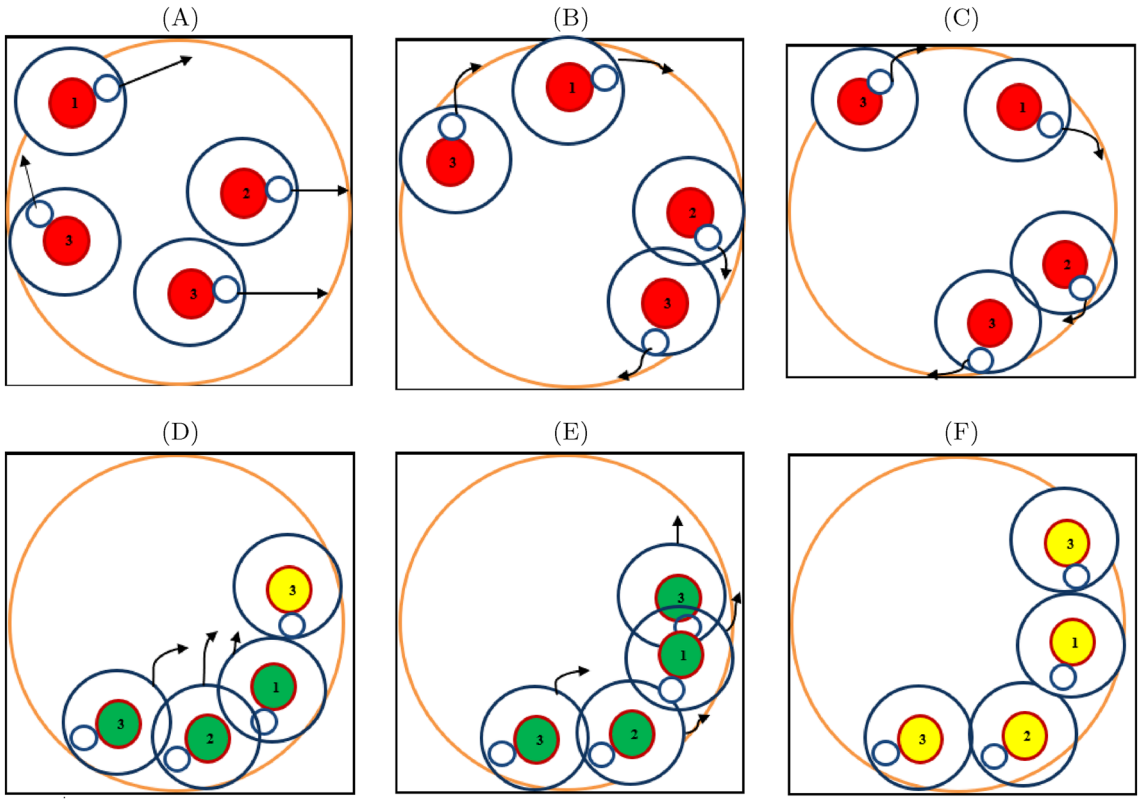

Fig. 3 The sequence of snapshots illustrates the hard-coded implementation in the presence of a high-resolution scalar field. The orange ring shows the lowest scalar field in the environment. The colored robots indicate the motion of the robots, either forward, backward, or stop; the black arrows show the robots' next decision based on the intersection between robots' communication range

orbited in a clockwise direction in the same contour line. In Rl and hard-coded implementation, the initial value of the forwarding speed is calculated based on the number of connected robots in its range Eq. (3).

$$\text{Forward Speed}\,(t) = \text{Max Forward Speed}\left(1 - \frac{\text{Robots In Range}\,(t)}{\text{Operated Robots}}\right) \tag{3}$$

2.2.1 Hard-coded implementation

If any robot detects another robot in its range using the communication range sensor, the robot's forward speed is reduced based on the number of robots in its range as shown in Eq. (3). The more robots in a cluster, the less forward speed for the robots, which helps the other robots catch them.

Fig. 3 shows how the hard-coded algorithm works. The red, green, and yellow robots indicate the robots move clockwise, anticlockwise, or stopped, respectively. The black arrows show the actions that the robots decide to take based on the current situation. In (A), all robots are distributed randomly in the environment and look to decrease the sensing scalar field projected on the ground. So that it helps them to go to the orange outer ring (just for demonstration). In (B) and (C), the robots orbit clockwise and try to align with the lowest scalar value. The robots' speeds are based on the number of robots in their range, as shown in Eq. (3). In (D), robot #3 connects to all other robots, and its intersection area is smaller than or equal to 2% of its communication range, so it stops moving. However, robots #1, #2, and #3 connect with a big intersection area, so they try to reduce this intersection by orbiting anticlockwise with speed, as shown in the Eq. 1. In (E), All robots should move anticlockwise to reduce their intersection area. In (F), all robots are connected and stop moving.

2.2.2 Reinforcement learning implementation

The robots share the same orbital so that the robots can be connected at any time because they move with different forward speeds based on the number of their connected neighbors. Therefore, the states for Q-learning are defined based on the maximum intersection areas between the robots and their connected neighbors. The states are discretized into

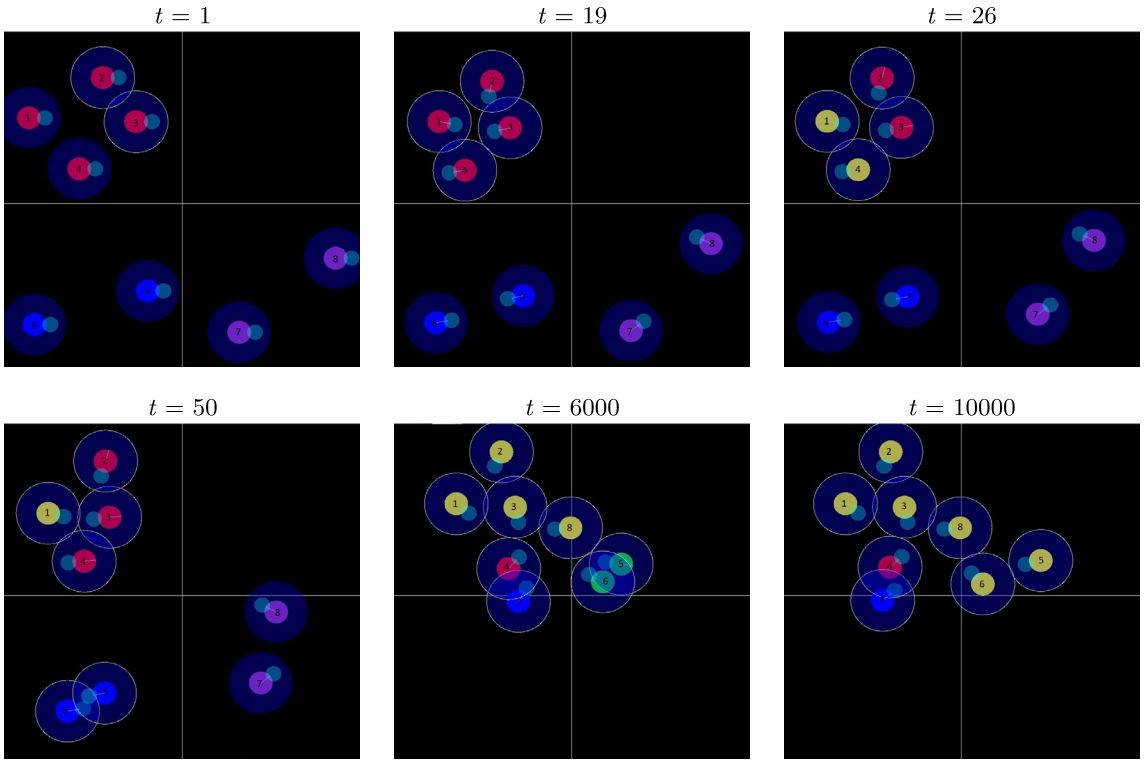

Fig. 4 Screenshots from the CWaggle simulation in different time steps. The working environment is divided into four quadrants. The colored robots indicate the destination quadrant the robots choose to go there. The colors are red, green, purple, and blue for the four quadrants respectively starting from the upper left corner. The yellow color indicates the robot chooses to stop

four states: (1) there is no intersection between this robot and the others; (2) the maximum intersection area between this robot and its neighbors is greater than two-thirds of its coverage range; (3) the maximum intersection is greater than one-third; or (4) less than one-third of its coverage range.

According to the actions, the robots choose between three actions: rotating with the same forward speed, increasing it by 0.3 to catch the other robots, or stopping their movement. We used the same reward function as defined in Eq. (2) in the LRSF environment.

3 Experiments and results

We used the C++ open-source simulator *CWaggle*[1]. This simulation is inspired by the Javascript robot simulator *Waggle*[2]. *CWaggle* was used before in [6] and [10].

The circular robots move in a rectangle space 900×900 and we use two performance metrics to evaluate the proposed methods. Firstly, the number of clusters constructed by the robots. Secondly, calculating the axis-aligned bounding box around the coverage area. We simulated all experiments over ten trials; the thick lines are the average performance for each proposed method, and the shaded areas are the confidence interval for the 90% confidence level. For all RL experiments in LRSF and HRSF, we use a discount factor equal to 0.9 with learning rate varying between experiments.

Low-Resolution Scalar Field, The sequence of screenshots from the hard-coded algorithm with the presence of LRSF can be seen in the Fig. 4, at $t = 1$, robots are distributed randomly over the four quadrants. At $t = 19$, robots reach their positions in their quadrants; robots #1, #2, #3, and #4 are connected with some intersection areas between them, so they move back to reduce these intersections. At $t = 26$, robots #1 and #4 have a good intersection area, so they stop moving, and their colors turn to yellow; however, robots #2 and #3 are still moving back, aiming to reduce their intersection and keep connecting to that cluster. As a consequence of robot #3 's movements, it has a big intersection with robot #4, so both start moving back to reduce this intersection. After a while at $t = 6000$, robots #2 and #3 fix

[1] https://github.com/davechurchill/cwaggle

[2] https://github.com/BOTSlab/waggle

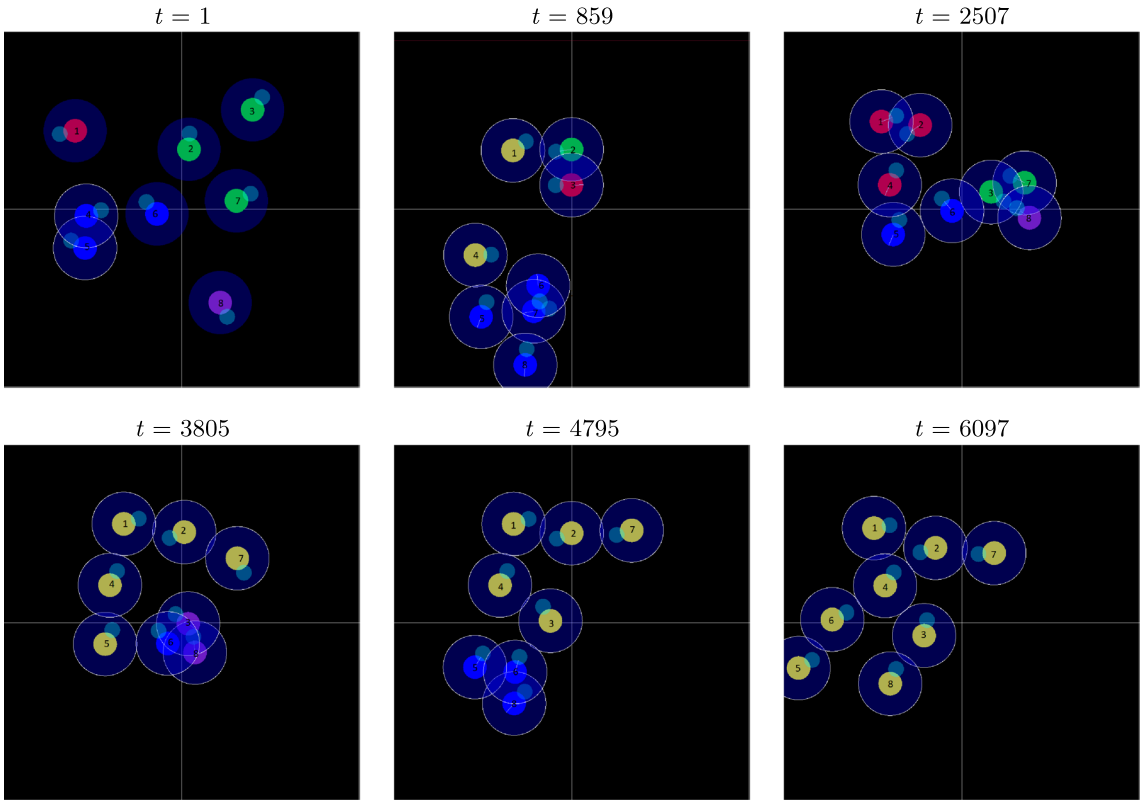

Fig. 5 Screenshots in different time steps show how the robots perform in the Cwaggle simulation using RL

there intersections. Robots #5 and #6 move from their initial fourth quadrant to the third one and then reach the second quadrant. After that, they connect with the cluster, but they still have an intersection that needs to be reduced. Robot #8 changes its quadrant to the second quadrant and connects with the cluster with the acceptable intersection, so it stops moving. Robot #7 reaches the second quadrant but with a big intersection with robot #8, so it moves back and disconnects from the cluster, so it moves to the first quadrant and intersects with robot #4. Finally, at $t = 10000$, all robots are connected with the smallest intersection except robots #4 and #7; they still move backward and forward to reduce the intersection and keep connecting with the cluster.

Fig. 5, shows screenshots from RL implementation in the presence of LRSF in operation. At $t = 1$, the robots are on their way to their quadrants. At $t = 859$, some of the robots (yellow color) achieve their goal by connecting to others with minimum intersection areas. All other robots suffer from significant intersection areas, so they decide to continue searching for the best position for them. When these robots move away from that cluster, it reflects in the others' reward function, so all the robots continue working again, as shown at $t = 2507$. After some tries, they can form a cluster with a different shape at $t = 3805$. But there are still three robots that have a big intersection area. At $t = 4795$, based on their learning policy, they can take actions that increase their reward; in this case, they decided to move to Q4. Finally, all robots fix their intersections and connect to one cluster at $t = 6097$.

Fig. 6 shows the comparison between RL and hard-coded implementation with LRSF regarding the number of connected clusters and the maximum area achieved. The RL implementation can construct one cluster faster than the hard-coded implementation, and that cluster is more stable from t =5700. Unlike the hard-coded implementation, it can build one connected cluster, but due to its several backward movements to fix the intersections, this cluster is not stable like RL.

Regarding coverage area, the Rl can outperform the hard-coded implementation, and its average coverage area equals 290000. However, in hard-coded implementation, the average coverage area equals 270000 by the end of the simulation time.

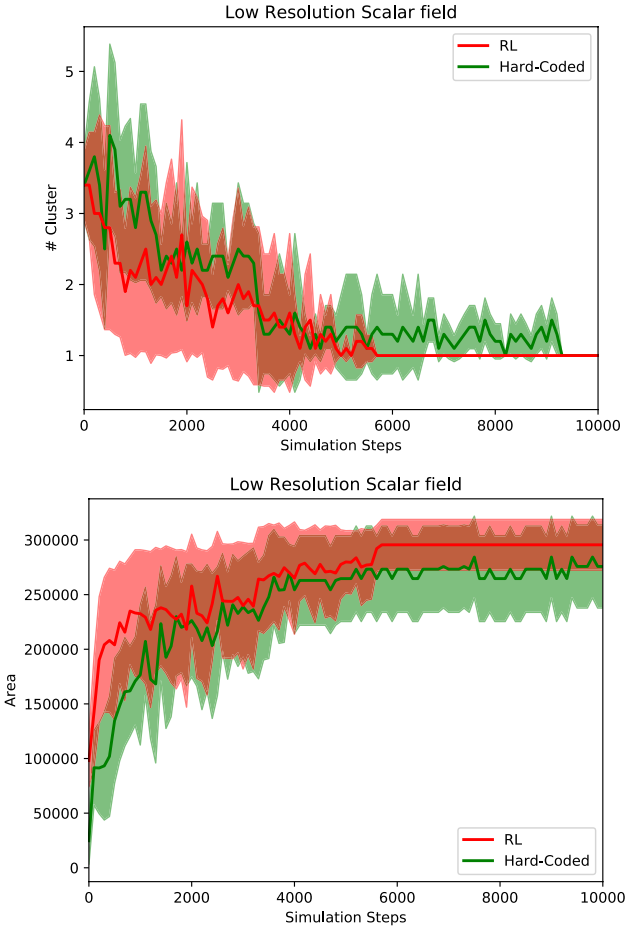

Fig. 6 Comparison between hard-coded and reinforcement learning implementation using low-resolution scalar field

High-resolution scalar field, Fig. 7 presents the screenshots of hard-coded implementation with the HRSF environment. At $t = 1$, the robots detect the scalar values and move with the guidance of the scalar field to reduce the sensing values. According to this motion, robots are aligned to the outer orbital near the border as shown in $t = 290$. The robots orbit in a clockwise motion, and some of them construct small clusters. So, robots in that cluster decrease their speed with the same ratio based on the number of robots in their cluster. At $t = 290$, the first group are consisting of robots #6, #7, and #8; and the second group has the robots #1, #3, and #4. Robots #5 and #2 are orbiting alone, so their speed is faster than the other robots so that they can catch them. After a while, robot #2 connect with robot #8, so their cluster speed is decreased, which allows robots #3, #4, and #1 to catch them as shown at $t = 1862$. At $t = 1868$, all robots in green color start to move anticlockwise to reduce their intersections, and robot #1 successfully gets a minimum intersection, so it stops moving. On the other hand, robot #5 in its moves clockwise to reach that cluster. At $t = 2498$, all robots construct one cluster, but they still fix the intersections among them by orbiting anticlockwise. Finally, at $t = 10{,}000$, robots in yellow color stop motion while the green robots keep moving anticlockwise, aiming to fix their intersection area.

Fig. 8 shows screenshots of the RL algorithm with HRSF from the CWaggle simulator in different simulation time steps. At the beginning of the simulation $t = 1$, all robots are distributed randomly in the environment. After that, they reach the lowest scalar value. These robots choose between actions: some increase their forward speeds, moving with the calculated forward speed Eq. 3 or stop movement, and the robots are colored purple, red, or yellow, respectively. According to their learned policy, at $t = 1000$, the purple robots that move alone try to increase their speed to catch other robots. Also, the robots with the maximum intersection with its neighbor try to increase their speed to reduce the intersection area. At $t = 4100$, and $t = 4200$, the red robots have a minimum intersection, but the number of connected robots in their cluster is less than the total number of connected robots. So, they move with the calculated forward speed, while the purple robots increase their speed to reduce their intersection areas. At $t = 4500$, the yellow robots have a minimum intersection area, and the number of robots in their cluster equals operated robots, so they get a high reward and choose to stop their movement. Simultaneously, some robots are still moving to reduce their intersection areas, as shown in red. Finally, all robots stop and form one cluster with a minimum intersection area at $t = 4560$.

Fig. 9 presents the differences in the performance between both implementations. The top figure shows that most of the trials in the RL implementation can construct one cluster around $t = 3000$ simulation step, which is faster than the hard-code. However, In some trials, robots destroy this cluster while moving to minimize the intersection areas, and others keep it because it has minimum intersection areas among the robots. Regarding the average coverage area, the RL implementation got a larger coverage area than the hard-coded implementation.

The comparison between all the proposed solutions is shown in Fig. 10. Regarding number of clusters, all proposed algorithms start with high number of clusters and by the end of the simulation they can establish one cluster. However, the RL implementation with LRSF takes the lowest simulation steps to form one stable cluster. Also, in LRSF, every trial produces different shapes that depend on the initial position of the robots. Therefore, this solution will be preferable if

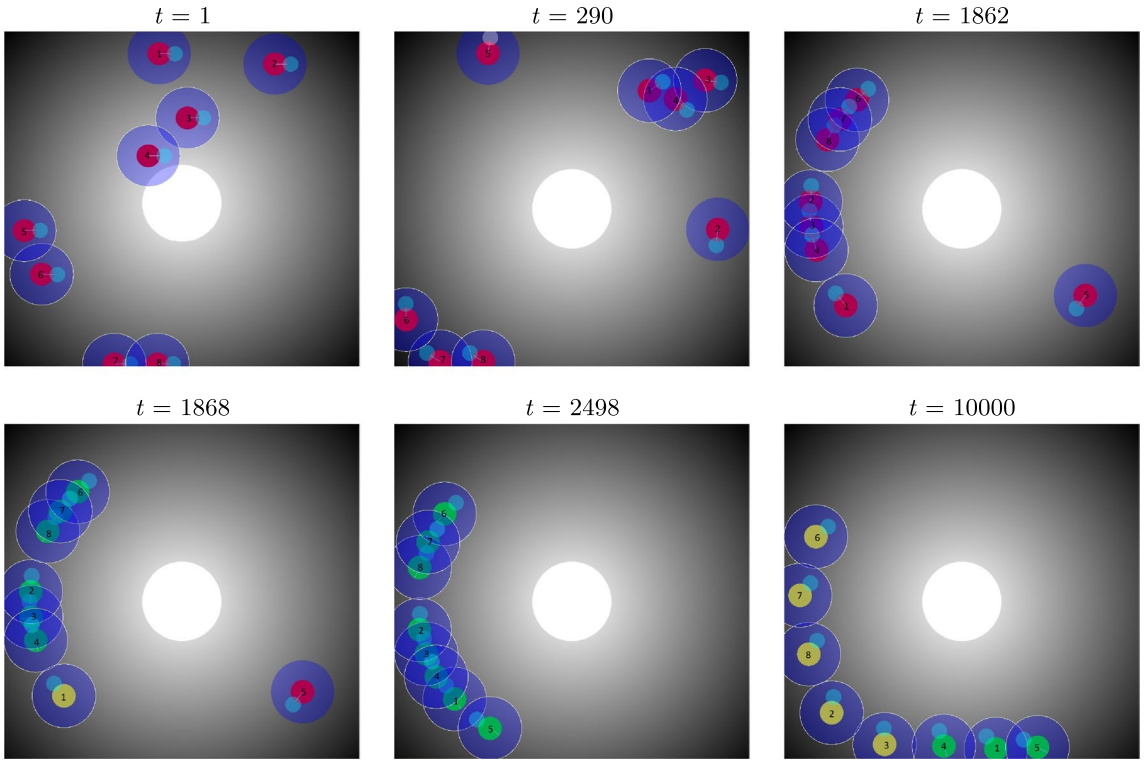

Fig. 7 Screenshots from a CWaggle simulation in different time steps. Eight robots are randomly distributed at $t = 1$, sharing the same orbital with different, forward speeds. The robots' colors are red, green, or yellow, indicating that robots orbit with forwarding, backward speed, or stop, respectively

the application wants to divide the working environment into arbitrary territories. In respect of the average coverage area, RL with HRSF achieved a higher coverage area compared to the other implementations.

4 Conclusion and future work

Methods to solve the largest coverage network for a robot swarm are presented in this paper and we propose four different approaches, either hard-coded or reinforcement learning using Low- and High-Resolution Scalar Fields. From the obtained simulation, the results show that using the benefits of reinforcement learning to find the best policy to maximize the coverage area is preferable to the hard-coded in both cases of low- and high-resolution scalar fields. Moreover, using the reinforcement learning combined with HRSF gives the largest coverage area , as HRSF guides the robots into alignment in the same orbital, making it much easier to construct one large, connected network. So, we show that increasing the scalar field resolution like HRSF will give us better results and make it easier for both hard-coded and RL, but it requires more sensing capabilities. On the other hand, in LRSF, when the robots have minimal sensing capabilities, it is also possible to construct one connected network, but that requires increasing the action space.

In future work, it would be interesting to investigate how the robots establish a connection with the presence of different sizes of obstacles and how these obstacles reflect on the formed shapes. Also, if the robots have a communication module, we would like to study how this may help them find the largest coverage area among the robots by exchanging information about the presence of obstacles, the number of robots in their range, and the maximum overlapping. So the communication among the robots could be helpful in the robot's decision if they destroy their structure or not. Also, we plan to study how the bigger size of robots can reflect on the proposed algorithms, does the increase in the number of robots directly proportional to the coverage area or may cause many intersection areas and collisions.

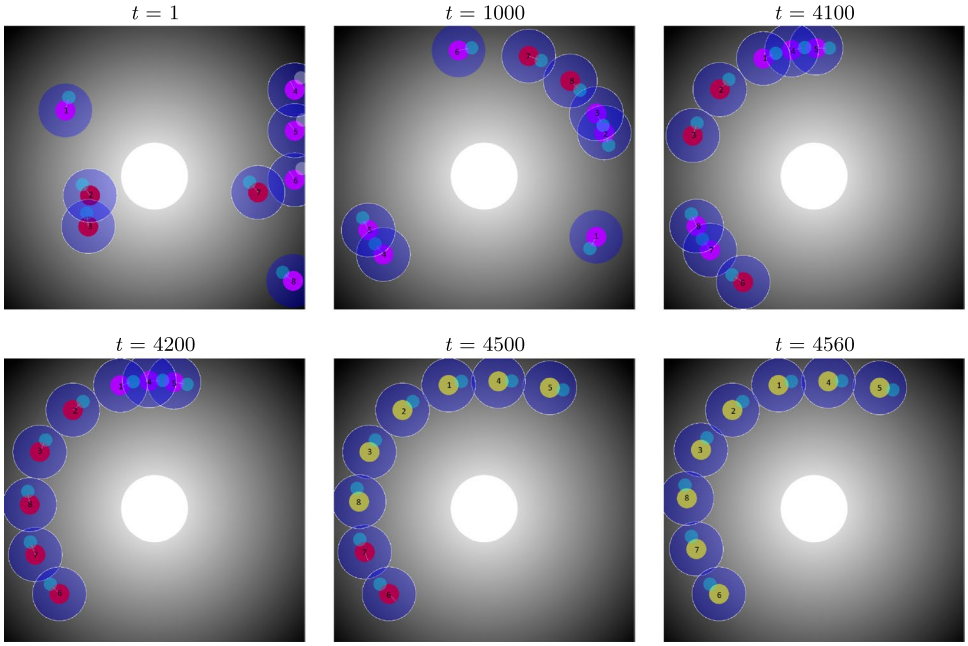

Fig. 8 Screenshots show a CWaggle simulation in different time steps. The eight robots are using RL to form LCN and working on a central source point scalar field. The colors of the robots indicate the action they choose; the red color means the robots move with the calculated forward speed; the purple color indicates that the robots increase their speed and the yellow color means the robot chooses to stop

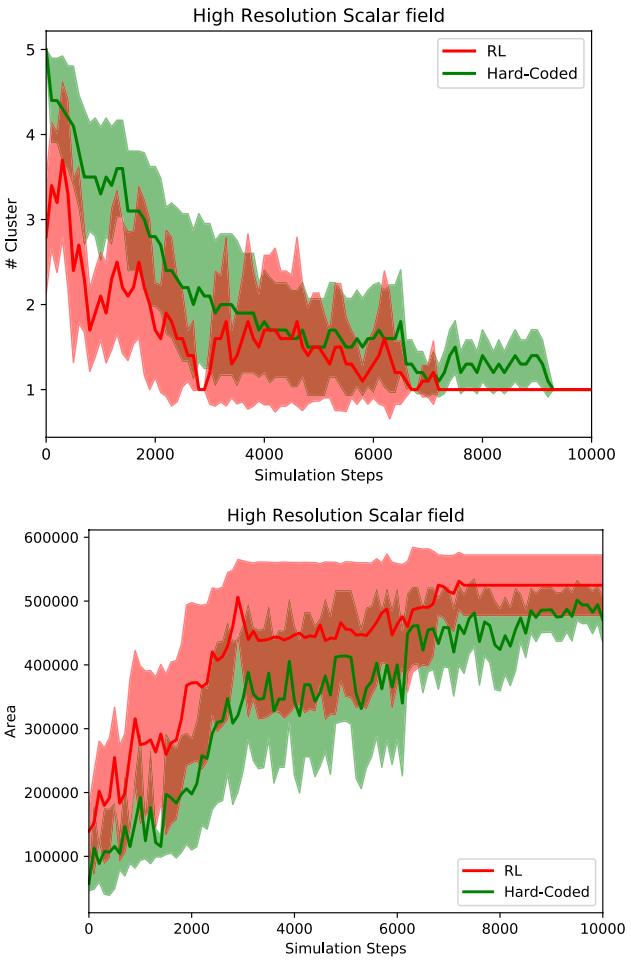

Fig. 9 Comparison between hard-coded and RL using high-resolution scalar field

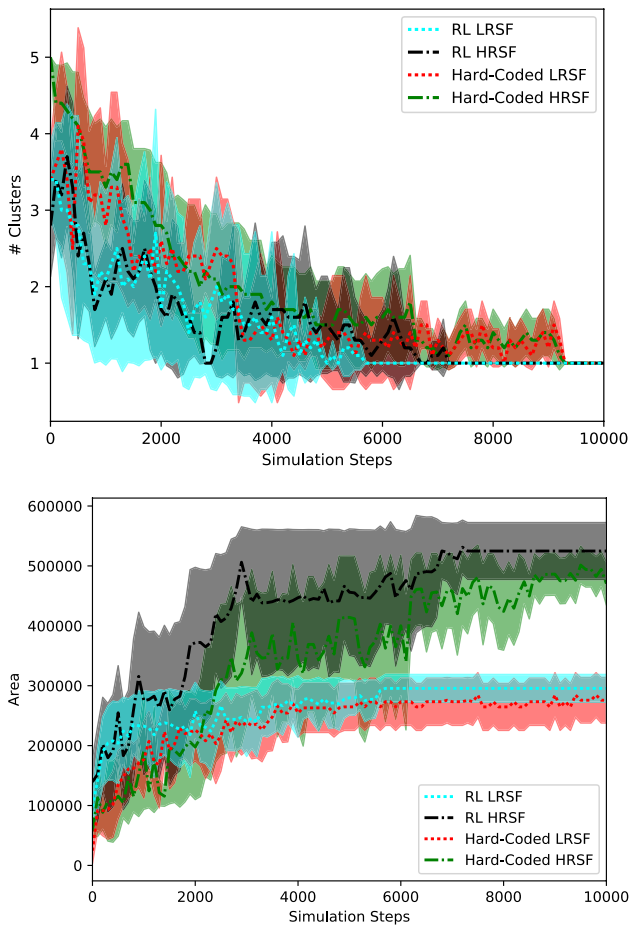

Fig. 10 RL Vs hard-coded implementations

References

1. Panerati J, Gianoli L, Pinciroli C, Shabah A, Nicolescu G, Beltrame G (2018) From Swarms to Stars: Task Coverage in Robot Swarms with Connectivity Constraints, 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 7674-7681
2. Francesca Gianpiero, Brambilla Manuele, Brutschy Arne, Garattoni Lorenzo, Miletitch Roman, Podevijn Gaëtan, Reina Andreagiovanni et al (2015) AutoMoDe-Chocolate: automatic design of control software for robot swarms. Swarm Intell 9(2):125–152
3. Mitaka Y, Akino T (2021) A review of termite pheromones: multifaceted, context-dependent, and rational chemical communications. Front Ecol Evol. https://doi.org/10.3389/fevo.2020.595614
4. Eric B, Marco D, Guy T (2020) Swarm intelligence: from natural to artificial systems swarm intelligence: from natural to artificial systems self-organization and templates: application to data analysis and graph partitioning. In: Swarm intelligence: from natural to artificial systems. Oxford University Press (**Oxford Scholarship Online (1999)**)
5. Vardy Andrew (2018) Orbital construction: Swarms of simple robots building enclosures. In 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W), IEEE, pp. 147-153
6. Strickland Caroline, Churchill David, Vardy Andrew (2019) A reinforcement learning approach to multi-robot planar construction. In 2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS),IEEE, pp. 238-244
7. Bajaj D Manju (2014) Maximum coverage heuristics (MCH) for target coverage problem in Wireless Sensor Network, 2014 IEEE International Advance Computing Conference (IACC), pp. 300-305
8. Cardei M, Du DZ (2005) Improving wireless sensor network lifetime through power aware organization. ACM Wirel Netw 11(3):333–40
9. Szepesvári Csaba (2010) Algorithms for reinforcement learning. Synth Lect Artif Intell Mach Learn 4(1):1–103
10. Vardy Andrew, Ibrahim Dalia S (2020) A swarm of simple robots constructing planar shapes. arXiv preprint arXiv:2004.13888