Artificial Intelligence for Engineering Design, Analysis and Manufacturing

cambridge.org/aie

Evaluating the learning and performance characteristics of self-organizing systems with different task features

Hao Ji and Yan Jin

Department of Aerospace and Mechanical Engineering, University of Southern California, 3650 McClintock Avenue, OHE 400, Los Angeles, CA 90089-1453, USA

Research Article

Cite this article: Ji H, Jin Y (2021). Evaluating the learning and performance characteristics of self-organizing systems with different task features. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **35**, 404–422. https://doi.org/10.1017/S089006042100024X

Received: 14 May 2021
Revised: 30 August 2021
Accepted: 31 August 2021
First published online: 27 December 2021

Key words:
Complex system; deep q-learning; robustness; scalability; self-organizing system

Author for correspondence:
Yan Jin, E-mail: yjin@usc.edu

Abstract

Self-organizing systems (SOS) are developed to perform complex tasks in unforeseen situations with adaptability. Predefining rules for self-organizing agents can be challenging, especially in tasks with high complexity and changing environments. Our previous work has introduced a multiagent reinforcement learning (RL) model as a design approach to solving the rule generation problem of SOS. A deep multiagent RL algorithm was devised to train agents to acquire the task and self-organizing knowledge. However, the simulation was based on one specific task environment. Sensitivity of SOS to reward functions and systematic evaluation of SOS designed with multiagent RL remain an issue. In this paper, we introduced a rotation reward function to regulate agent behaviors during training and tested different weights of such reward on SOS performance in two case studies: box-pushing and T-shape assembly. Additionally, we proposed three metrics to evaluate the SOS: learning stability, quality of learned knowledge, and scalability. Results show that depending on the type of tasks; designers may choose appropriate weights of rotation reward to obtain the full potential of agents' learning capability. Good learning stability and quality of knowledge can be achieved with an optimal range of team sizes. Scaling up to larger team sizes has better performance than scaling downwards.

Introduction

Self-organizing systems (SOS) consist of simple agents that work cooperatively to accomplish complex tasks. Through agents' local interactions, high-level system complexity can be achieved by a bottom-up approach (Reynolds, 1987; Ashby, 1991). Designing complex systems through a self-organizing approach has many advantages, such as adaptability, scalability, and robustness in comparison to traditional engineering systems with centralized controllers (Chiang and Jin, 2012; Humann *et al.*, 2014; Khani and Jin, 2015; Khani *et al.*, 2016; Ji and Jin, 2018). A swarm of robots is often homogeneous, with compact size and limited functionality, and is an example of such SOS (Kennedy, 2006). Through simple rules of interaction between robots, the collective behavior emerges, and such emergent phenomena can be utilized in real-life situations, such as unmanned aerial vehicle patrolling, traffic control, distributed sensing, search and rescue, and box-pushing (Pippin, 2013; Pippin and Christensen, 2014; Khani *et al.*, 2016).

Numerous approaches have been proposed to aid SOS design. The field-based behavior regulation (FBR) approach (Chen and Jin, 2011) uses a field function to model the task environment, and the agents' behaviors are regulated based on their positions in the field with a field transformation function. The advantage of this approach is that the behaviors of agents are simple, and hence little knowledge is required to accomplish tasks as agents only need to move toward a higher or lower position. However, this behavioral simplicity limits its capability of solving more complex domain problems because field representation cannot capture all features of the task domain, and inter-agent relationships are ignored in this approach.

An evolutionary design method and the social structuring approach have been developed to overcome the limitations of the FBR approach. The design of SOS can be more parametric and optimizable by considering both task and social field modeled by social rules (Khani *et al.*, 2016). It was shown that applying social rules help increase the coherence among agents by avoiding potential conflicts and promoting cooperation opportunities (Khani *et al.*, 2016). A critical issue with the rule-based approach is that designers must know the rules *a priori* and how they should be applied, which might not be possible with more complex and unpredictable tasks such as autonomous physical structure assembly, space structure construction, and disaster rescue situations.

To address the issues in the rule-based design approach, our previous work (Ji and Jin, 2019; Ji and Jin, 2020) introduced a multiagent reinforcement learning (RL) approach to

© The Author(s), 2021. Published by Cambridge University Press

CAMBRIDGE UNIVERSITY PRESS

capture the self-organizing knowledge and regulate agent behavior in SOS design. In our approach, each agent is trained using its own independent neural network, which solves the problem of the high dimensionality of action space of using single-agent RL in multiagent settings. Our work shows that state-of-the-art independent multiagent RL is a promising approach in tackling the existing problems faced by SOS design. However, the simulation was conducted on one specific task environment, and it still remains a question how sensitive SOS designed with multiagent RL is to the reward function and how to systematically evaluate the SOS.

In this paper, we specifically introduced a rotation reward function to regulate agent behaviors during training and tested different weights of such reward on SOS performance in two different case studies: box-pushing and T-shape assembly. Three metrics were incorporated to evaluate the SOS: learning stability, quality of knowledge, and scalability. With the introduction of reward functions and evaluation metrics in two different case studies, we are trying to address three research questions: Q1: *How sensitive is SOS to specific reward functions, and how to decide on the weight of such reward functions?* Q2: *What factors impact the stability of learning dynamics in self-organizing systems with different task scenarios?* Q3: *How to systematically evaluate SOS performance with different task features?*

In the rest of this paper, "Related work" provides a review of the relevant work in complex system design, SOS, and RL. After that, a multiagent independent Q-learning framework is presented as a complex system design approach in "A Deep multiagent reinforcement learning model". In "Case study box-pushing" and "Case study self-assembly", box-pushing and T-shape assembly case studies are introduced that apply the proposed Q-learning model, followed by a detailed analysis and discussion of the simulation results. Finally, in "Conclusions and future work", the conclusions are drawn from the case study, and future work directions are pointed out.

Related work

Design of complex systems

A complex system consists of many components, and the collective behavior of the system can be difficult to model because of the interactions or interdependencies between its parts (Bar-Yam, 2002). Much research has been done so far to study ways of designing complex systems. For instance, Arroyo *et al.* (2018) came up with a bio-inspired, binary-tree-based method for complex system design with fault adaptiveness. The binary-tree branches indicate how organisms achieve adaptive behaviors. Their results show that a strategy-based method can aid designers with innovative analogies, which is useful in providing needed details in design applications (Arroyo *et al.*, 2018). Königseder and Shea (2016) analyzed various strategies of rule applications in two design synthesis case studies: bicycle frame synthesis and gearbox synthesis, and they found that the effectiveness of each strategy depends on the task scenario. Meluso and Austin-Breneman (2018) build an agent-based model which simulates the parameter estimation strategy in complex engineering systems on a large scale. They found that accuracy and uncertainty of estimation depend mainly on subsystem behavior but not much on the "gaming" strategy of system engineers' (Meluso and Austin-Breneman, 2018). McComb *et al.* (2017) developed a computational model that analyzed how to use the characteristics of configuration design problems into choosing optimal values of teams, such as team size and frequency of interaction. Equations were generated using regression analysis, which was used to predict optimized team values given problem properties (McComb *et al.*, 2017). Min *et al.* (2016) calculated the structured complexity of complex engineering systems with different levels of decomposition. Their analysis found that the structural complexity of the system is mainly dependent on the lowest level of architectural configurations. Ferguson and Lewis (2006) developed a method for effective reconfigurable systems design, such as vehicles. Their method focused on identifying changes in system design variables and analyzing the stability of the reconfigurable system with a state-feedback controller. Martin and Ishii (1997) proposed a design-for-variety approach, which can help companies quickly adapt their products and address product variation at different generations to reduce the time from products to market. Both qualitative design and quantitative indices have been developed, and the effectiveness of such tools was presented in examples of electronics and automotive industries. Research in previous complex system design requires extensive domain knowledge to build models or draw inspiration from nature, which can be time-consuming and hardly generalizable to different scenarios. Moreover, it still remains a challenging task to generate design rules from global system requirements to local agents' interactions.

Artificial SOS

Artificial SOS are designed by humans and can have emergent behaviors and adaptability like nature (Reynolds, 1987). Much research has been conducted in terms of the design of artificial SOS. Werfel (2012) devised a system of homogeneous robots that can build a pre-determined shape with square bricks. Beckers *et al.* (2000) analyzed a robotic gathering task where multiple robots need to patrol a given area in order to collect pucks. As robots have a preference for dropping pucks in high-density areas, the collective positive feedback loop leads to a dense group of available pucks (Ashby, 1991; Beckers *et al.*, 2000). Khani and Jin (2015) and Khani *et al.* (2016) introduced a social rule-based behavior regulation approach, which can enforce the agents to self-organize and push a rectangle box towards the goal area. Swarms of UAVs can self-organize and accomplish complex tasks, such as shape formation, target detection, and collaborative patrolling, based on a set of cooperation rules (Lamont *et al.*, 2007; Dasgupta, 2008; Ruini and Cangelosi, 2009; Wei *et al.*, 2013). Chen and Jin (2011) adopted an FBR approach, which can assist self-organizing agents in accomplishing complex tasks such as moving towards long-distance targets while avoiding obstacles. Price and Lamont (2006) tested the effectiveness of the genetic algorithm (GA) in optimizing multi-UAV swarm behaviors. He analyzed the performance of using the GA algorithm in the "destroying retaliating target" task with both homogenous and heterogenous UAVs. The robotic applications mentioned above have demonstrated great potentials for building SOS and the self-organizing system design methods (Chen and Jin, 2011; Khani and Jin, 2015; Khani *et al.*, 2016) have had their drawbacks, as discussed in "Introduction".

Multiagent RL

Multiagent RL applies to multiagent situations. It is based on the basic concept of single-agent RL, such as Q-learning, policy

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

gradient, and actor-critic (Busoniu *et al.*, 2008; Sutton and Barto, 2018). In comparison to single-agent RL, multiagent learning has a non-stationary learning environment as multiple agents are simultaneously learning.

In order to resolve the issues such as high-dimensionality of state and action spaces in multiagent environments and to approximate state-action values, there has been a move from tabular-based approach to deep RL-based approach in the past several years (Peng *et al.*, 2017; Tampuu *et al.*, 2017; Foerster *et al.*, 2018; Rashid *et al.*, 2018). Multiagent systems can be classified into several categories: cooperative, competitive, and mixed cooperative and competitive settings (Tampuu *et al.*, 2017). Cooperative agents share the same rewards, competitive agents (usually in two-agent settings) have opposite rewards, and the mix of cooperative and competitive environments assume agents are not only cooperating but also have individual preferences. In the design of SOS, the focus is on cooperative settings since agents share the same goals.

A natural approach for multiagent RL is to optimize the value functions or the policy of each individual agent. One commonly used value function-based multiagent learning algorithm is independent Q-learning (Tan, 1993). In independent Q-learning, each individual's state-action values are trained using Q-learning (Watkins, 1989; Tan, 1993), and it is served as a common benchmark in the literature. Tampuu *et al.* (2017) expanded previous Q-learning using deep neural networks and applied DQN (Mnih *et al.*, 2015) to train two independent agents learning how to play the game, Pong. In his simulation, he shows how the cooperative and competitive phenomenon emerges depending on an individual's different reward schemes (Tampuu *et al.*, 2017). Foerster developed a framework called "COMA" and trained multiple agents learning how to cooperate and play StarCraft games (Foerster *et al.*, 2018). He utilized a centralized critic network to evaluate decentralized actors and estimated a counterfactual advantage function for each agent and finally allocated credit among them (Foerster *et al.*, 2018). In another work by Foerster, he further analyzed the replay stabilization methods he proposed for independent Q-learning in various StarCraft combat scenarios (Foerster *et al.*, 2017).

As a multiagent environment is often partially observable, Hausknecht and Stone (2015) applied deep recurrent networks such as LSTM (Hochreiter and Schmidhuber, 1997) or GRU (Chung *et al.*, 2014) to facilitate agents' learning over long time periods. Lowe *et al.* (2017) proposed Multiagent Deep Deterministic Policy Gradient (MADDPG) algorithm, which uses centralized training and decentralized execution. They developed an extension of the actor-critic policy gradient method, which augmented critics with extra information about the policies of other agents and then tested their algorithm in cooperative navigation, predator–pre,y and other environments. Results of their training algorithm show good convergence properties (Lowe *et al.*, 2017). Drogoul and Zucker (1998) and Collinot and Drogoul (1998) introduced a framework for multiagent system design named "Andromeda," which integrates a machine learning approach with an agent-oriented, role-based approach named "Cassiopeia." The main idea is to let learning take place within different layers of "Cassiopeia" framework, such as individual roles, relational roles, and organizational roles, which can make the design of multiagent system more systematic and modular (Collinot and Drogoul, 1998; Drogoul and Zucker, 1998). However, as a real multiagent environment can be rather complex, the actions of agents are affected by not only their own roles but also by other agents and the group. Dividing learning into different layers of abstraction may not be feasible. Abramson *et al.* (2020) studied how interactive intelligence can be achieved by artificial agents. They used another learned agent to approximate the role of the human and adopted ideas from inverse RL to reduce the differences between human–human and agent–agent interactive behavior. Results showed that interactive training and auxiliary losses could improve agent behavior beyond what can be achieved by supervised learning (Abramson *et al.*, 2020).

Most multiagent RL algorithms emphasize achieving optimal cumulative rewards or desirable convergence properties by proposing new training techniques or neural network structures. Many approaches are based on fully observable states, such as the entire game screen. Training of the multiagent reinforcement model is often conducted in prespecified environments with fixed team sizes. Moreover, besides terminal reward often proposed in the literature, it is crucial to identify other important reward functions and test how such reward functions affect the performance of the systems with different task features. For instance, having a rotation reward in multiagent training can play a key role in regulating the system dynamics and facilitating agents' learning. In addition, the sensitivity of the trained network to different weights of such reward function should be analyzed, which are crucial factors in SOS design. Finally, it is important to develop an evaluation metric of SOS and offer guidelines on how design should be implemented and analyzed in various task situations. Such areas are often ignored in the literature and are the focus of this paper.

A deep multiagent RL model

Single-agent RL

Multiagent RL is based on single-agent RL, which is used to optimize system performance based on training so that the system can automatically learn to solve complex tasks from the raw sensory input and the reward signal. In single-agent RL, learning is based on the Markov Decision Process defined by a tuple of $<S, A, P, R, \gamma>$. S is the state space, composed of the agent's all possible sensing information of the environment. A is the action space, including the agent's all possible actions. P is the transition matrix, which is usually unknown in a model-free learning environment. R is the reward function, and γ is the discount factor for the future rewards. At any given time t, the agent's goal is to maximize its expected future discounted return, $R_t = \sum_{t'}^{T} \gamma^{t'-t} r_t$, where T is the time when the game ends. Also, agents estimate the action-value function $Q(s, a)$ at each time step using Bellman Eq. (1) as an update. E represents the expected value. Eventually, such a value iteration algorithm converges to the optimal value function.

$$Q_{i+1}(s, a) = E[r + \gamma \max_{a'} Q_i(s', a')|s, a]. \quad (1)$$

In recent years, deep neural networks are introduced as functional approximators to replace the old Q-table for estimating Q-values; such learning methods are called deep Q-learning (Mnih *et al.*, 2015). A Q-network with weights θ_i can be trained by minimizing the loss function at each iteration i, illustrated in Eq. (2),

$$L_i(\theta_i) = E[(y_i - Q(s, a; \theta_i))^2], \quad (2)$$

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

where

$$y_i = E[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})] \quad (3)$$

is the target value for iteration i. The gradient can be calculated with the following equation:

$$\nabla_{\theta_i} L_i(\theta_i) = E_{s,a,r,s'}[\{r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a, \theta_i)\} \nabla_{\theta_i} Q(s, a, \theta_i)]. \quad (4)$$

Multiagent RL

There are generally two approaches in multiagent training. One is to train the agents as a team, treating the entire multiagent system as "one agent." It has good convergence property but can hardly scale up or down. For increasing learning efficiency and maintaining scalability, a multiagent independent deep Q-learning (IQL) approach is adopted. In this approach, A_i, $i = 1, \ldots, n$ (n: number of agents) are the discrete sets of actions available to the agents, yielding the joint action set $A = A_1 \times \cdots \times A_n$. All agents share the same state space and the same reward function as the task is cooperative. During training, each agent has its own neural network; they perceive and learn independently except that they share the same reward function and hence the reward value. As the agents are homogeneous and share the same action space, the trained neural networks can be reused and applied in different team sizes. In our multiagent RL mechanism described above, each agent $i(i = 1, 2, \ldots, n)$ engages in learning as if it is in the single-agent RL situation. The only difference is that the next state of the environment, S_{t+1}, is updated in response to the joint action $a_t = \{a_1, a_2, \ldots, a_n\}$, instead of its own action a_i, in addition to the current state S_t.

In this research, we first introduced a rotation reward for regulating system dynamics and agents' behaviors, as shown in Eq. (5), where $\alpha_2 - \alpha_1$ represents the change of angle of the moving object under evaluation, the constant K is between 0 and 1 and is used to discourage the agents' behaviors that result in rotation of the moving object with more than a certain degree threshold. For example, if $K = 0.98$, the simulation discourages the rotation of more than 11° as rotation reward R_{rot} would be negative. This way, the object under control can be rotated constantly to a small degree, which can facilitate the learning of agents by controlling the search space of the agents during exploration with the environment.

$$R_{\text{rot}} = \text{Cos}(\alpha_2 - \alpha_1) - K. \quad (5)$$

After the introduction of the rotation reward, we evaluated the sensitivity of the learning process to the rotation reward – that is, whether good knowledge can be acquired through RL with different weights of rotation reward. Then we looked into the knowledge evaluation issue – that is, how to systematically evaluate SOS performance in different tasks through a set of proposed evaluation metrics: learning stability, quality of knowledge, and scalability. Learning stability measures cumulative reward of agents with increasing training episodes; quality of knowledge is a greedy approach to assess the system performance under the best state-action policy after training; scalability is measured by reusing the trained neural networks to different team sizes, and it tests whether SOS is robust enough to addition or loss of team members.

Case study box-pushing

To test the concepts and explore the multiagent RL algorithm discussed above, we carried out a box-pushing case study.

The box-pushing problem

The box-pushing problem is often categorized as a trajectory planning or piano mover's problem (LaValle, 2006). Many topological and numerical solutions have been developed in the past. In our paper, a self-organizing multiagent deep Q-learning approach is taken to solve the box-pushing problem. During the self-organizing process, each agent acts based on its trained neural network, and collectively all agents can push the box towards a goal without any system-level global control.

In this research, the box-pushing case study was implemented in pygame, a multiagent game package in the Python environment. In the box-pushing case study, each individual agent is trained with its IQL neural network as a member of a team. After the training, the resulting neural networks are applied to the testing situations with various team sizes between 1 and 10. Both training and testing results are analyzed to elicit the learning properties, the quality of learned knowledge, scalability, and robustness of the learned neural networks.

A graphical illustration of the box-pushing case study is shown in Figure 1. The game screen has a width x of 600 pixels and a height y of 480 pixels. Numerous agents (the green squares) with limited pushing and sensing capabilities need to self-organize in order to push and rotate the brown rectangle towards the goal (the white dot with a "+" mark). As there is an obstacle (the red dot) on the path and walls (the white solid lines) along the side, the agents cannot just simply push the box but must rotate the box when necessary (Khani and Jin, 2015; Khani *et al.*, 2016). This adds complexity to the task. The box has sensors deployed at its outside boundary. When the outside perimeter of the box reaches the horizontal x-coordinate of the goal, represented as a white vertical line, the simulation is deemed a success.

There are four major tasks of box-pushing, as summarized below. Agents need to move, rotate the box, and keep the box away from potential collisions with walls and obstacles.

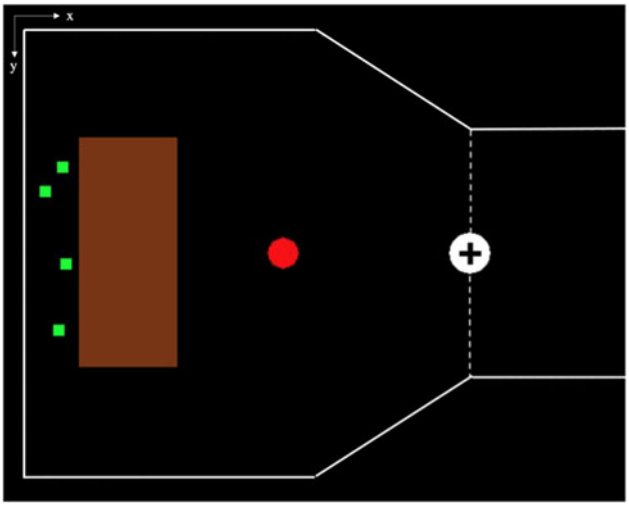


Fig. 1. A graphical illustration of the box-pushing task.

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

T1 = <Move><Box> to <Goal>
T2 = <Rotate><Box> to <Goal>
T3 = <Move><Box> away from <Walls>
T4 = <Move><Box> away from <Obstacle>

In pygame, the distance is measured by pixels. Each pixel is a single square area in the simulation environment. In this case study, a brown box, which is 90 pixels wide and 225 pixels long, is shown in Figure 1.

In box-pushing, agents have *limited sensing and communication capabilities*. They can receive information from the sensor on the box, which measures the orientation of the box and senses the obstacles at a range of distances. They have limited storage of observation information: their experiences such as state, action, reward, and next state. They possess a neural network that can transform the perceived state information into action. These assumptions are in line with the definition of the "minimalist" robot (Jones and Mataric, 2003) and are reasonable with the current applications of physical robot hardware (Groß *et al.*, 2006).

State space and action space

Task state space: Based on the task decomposition and constraint analysis mentioned above, the state space of the box-pushing task is defined as shown in Figure 2. For gathering relevant environment information, a sensor is deployed in the center of the box, which can sense nearby obstacles. The radius of the sensor range is 150 pixels, and the entire circular sensor coverage is split into 8 sectors of equal size, as shown in Figure 2. In this case study, part of the state space captures the state of each sector at any given time: whether the sector is occupied by an obstacle, represented as 1, or not, represented as 0. Furthermore, we assume the sensor can also detect the orientation of the box's x-axis with respect to the location of the goal (Wang and De Silva, 2006; Rahimi *et al.*, 2018). Denoting the sectors with s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, and the box orientation with s_9, respectively, the state space can be defined as:

$$S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}. \quad (6)$$

For the example of Figure 2, sectors 3, 5, and 7 are occupied by obstacles. Therefore, the corresponding state attributes s_3, s_5, s_7 are having value 1, and s_1, s_2, s_4, s_6, s_8 are 0. In Figure 2, the box angle θ is about 30°. And such degree information can be shaped into the range of $[-1,1]$ by applying $s_9 = (\theta - 180)/180 = -0.83$. This shaped angle method can facilitate deep Q-network training and is used commonly in practice (Wang and De Silva, 2006; Rahimi *et al.*, 2018).

Given the above, the state representation of Figure 2 can be expressed as a 9-item tuple <0,0,1,0,1,0,1,0, −0.83>.

During training, each agent is close to the vicinity of the box center, and it can receive the sensor information broadcasted locally among agents. It is assumed that the sensor can also sense the distance from the center of the box to the location of the goal, analogous to real-world radar sensor, and is also like the gradient-based approach in literature where the task field is assumed (Khani and Jin, 2015; Khani *et al.*, 2016). Agents can also receive such distance information from the sensor.

Box neighborhood: The box neighborhood is defined as six regions (Khani and Jin, 2015; Humann *et al.*, 2016), as shown in Figure 3. During each simulation, an individual agent can move to one of the six regions of the box neighborhood, and that specific neighborhood is the position of the agent. As the individual agent is relatively small, we assume there can be multiple agents in the same region at the same time. This is in line with the definition of the "minimalist" robot (Jones and Mataric, 2003).

Box dynamics: The box dynamics are based on a simplified physical model. The box movement depends on the simulated force and torque. Forces equal to the sum of vector forces of each pushing agent. Every push carries the same amount of force, which acts from an agent towards the box in the normal direction. The sum of two pushes moves the box 10 pixels in a given direction. Torque is assumed to be exerted on the centroid of the box, and 2 pushes on the diagonal neighborhood of the box (such as positions 2 and 5 each) rotate the box 10°. We assume the box carries a large moment of inertia, and when it hits the obstacle, which is considered rather small, it continues its movement until its expected end position is reached.

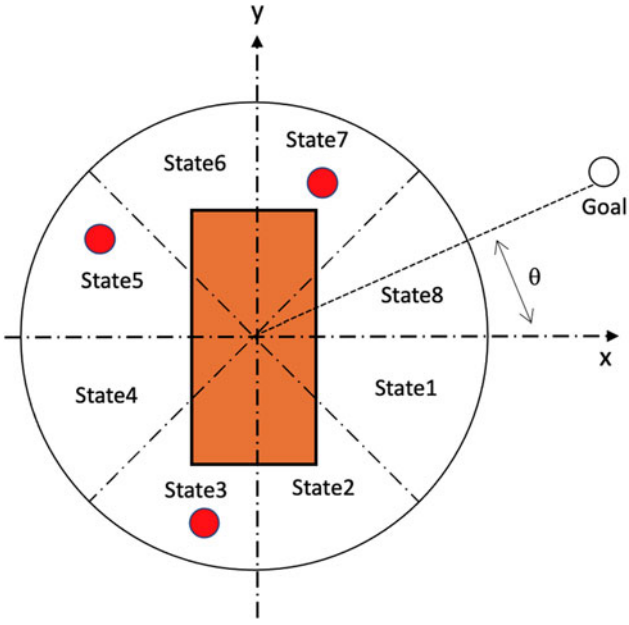


Fig. 2. Box-pushing task state representation.

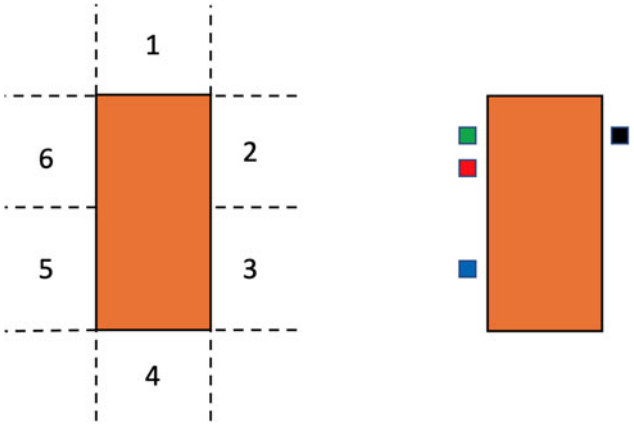


Fig. 3. The six regions of the box neighborhood.

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

Agent action space: The *agent action space* is defined based on the box neighborhood and simulated box dynamics. At each time step, an agent can choose a place in one of the six regions of the box neighborhoods to push the box. Therefore, the agents share the same action space of $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$, as shown in Figure 3. For instance, if an agent chooses action a_1, it *moves* to box region "1" and *push* the box from there, the box moves downwards along the box's *y*-axis based on the simulated box dynamics, and the same logic applies to other agent actions.

Reward schema and training model

In order to train multiple agents to self-organize and push the box to the final goal area, which is the group level function, we need to design a proper reward schema to facilitate agent training. Adapted from the previous Q-table-based box-pushing reward schema (Wang and De Silva, 2006; Rahimi *et al.*, 2018), we designed a new reward schema for agents' box-pushing training. The total reward is composed of four parts: *distance*, *rotation*, *collision*, and *goal*.

Distance Reward: The reward for pushing the box closer to the goal position is represented as R_{dis}, and is shown in Eq. (7). The previous distance D_{old} represents the distance, measured in pixels, between the center of the box and the goal position in the previous time step. D_{new} represents such distance at the current time step. C_d is a constant, called *distance coefficient* in our simulation, and is set to 2.5. At each simulation time step, agents calculate the change of distance between the current distance and previous distance based on Eq. (7) and draw its distance reward.

$$R_{\text{dis}} = (D_{\text{old}} - D_{\text{new}}) * C_d. \tag{7}$$

Rotation Reward: The reward for rotation R_{rot} is represented in Eq. (8). where α_1 is the previous time step angle of the box's *x*-axis with respect to goal position and α_2, the current angle. The rotation reward is given to discourage the rotation of more than 11°. This way, the box can be rotated constantly with small degrees and avoid large rotation momentum, which can result in a collision with obstacles. The rotation reward is relatively small as it is used only for rotation of the box rather than pushing the box towards the goal, which is the task's goal.

$$R_{\text{rot}} = \text{Cos}(\alpha_2 - \alpha_1) - 0.98. \tag{8}$$

Collision Reward: The collision reward is analogous to the reward schema in common collision avoidance tasks (Liu and Jin, 2018) and is represented in Eq. (9) with R_{col}. During each simulation step, if there is no collision for the box with either the obstacle or the wall, $R_{\text{col}} = 0$. If a collision occurs, a −900 reward is given to all the agents as a penalty.

$$R_{\text{col}} = \begin{cases} -900 & \text{if collision occurs} \\ 0 & \text{if no collision occurs.} \end{cases} \tag{9}$$

Goal Reward: The reward for reaching the goal R_{goal} is represented in Eq. (10). At each simulation step, if the box reaches the goal position, each agent receives a positive 900 reward; if the goal is not reached, the agents do not receive any reward.

$$R_{\text{goal}} = \begin{cases} 900 & \text{if reaching goal} \\ 0 & \text{if not reaching goal.} \end{cases} \tag{10}$$

The total reward is a weighted sum of all these rewards, as shown in Eq. (11).

$$R_{\text{tot}} = w_1 * R_{\text{dis}} + w_2 * R_{\text{rot}} + w_3 * R_{\text{col}} + w_4 * R_{\text{goal}}. \tag{11}$$

In our simulations, after repeated testing, the weights were set as $w_1 = 0.6$, $w_2 = 0.1$, $w_3 = 0.1$, $w_4 = 0.2$, with the sum of these weights equal to 1, shown in Eq. (12). The weights are chosen so that during each step in training: $w_1 = 0.6$, which means agents can have a more immediate reward in terms of whether or not they are closer to the goal; $w_2 = 0.1$, which gives a little incentive for agents to rotate the box a small angle; $w_3 = 0.1$, to offer some penalty reward if box collides with an obstacle; $w_4 = 0.2$, since the agents' final goal is to reach the target zone, if their goal is achieved, they should be given more rewards than when box collides with an obstacle. The weight for four different rewards is adapted and based on previous research on multiagent box-pushing (Wang and De Silva, 2006; Rahimi *et al.*, 2018).

$$w_1 + w_2 + w_3 + w_4 = 1. \tag{12}$$

During training, as the agents are homogenous and are cooperating to push the box, they should receive the same rewards. This is the reason why the reward Eqs (7)–(10) are defined based only on the box's position and orientation. In this way, each agent's neural network considers other agents' actions as part of its environment and learns to explore its action space and also its best policy based on an ϵ-*greedy* action selection strategy. Gradually, the agent grasps how to differentiate its actions from other agents to collaboratively push the box towards the goal position, which is the characteristic of multiagent independent deep Q-leaning neural networks.

Experiment design

As aforementioned, the three issues of this research are (1) *the stability of the learning dynamics* for agents to acquire knowledge for self-organizing behavior regulation, (2) sensitivity of SOS performance to specific reward functions, (3) evaluation of *the SOS performance with different task features*. To address these issues, we conducted a set of multiagent training and testing experiments, as described below.

Multiagent RL-based agent training

As illustrated in Figure 4, for the multiagent RL-based training process, the Deep Q-learning algorithm described above was applied. The independent variable of the training is the *Number of Agents* involved in the box-pushing task, denoted by *#Tr*, varying between 1 and 10. The dependent variables of the training process are the *Cumulative Reward* signifying the learning behavior of the agents, and a set of *#Tr* neural networks trained, denoted as, $NN_{\#Tr}$, representing the learned knowledge.

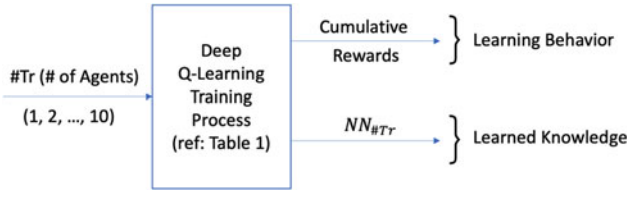


Fig. 4. Agent training experiment design.

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

During training, each episode is defined by a complete simulation run, from the starting point to the ending point. The starting point is arranged as shown in Figure 1, where the agent positions are randomly assigned while the box's initial position and the goal position are both fixed. For the ending point, there are three different situations: (1) the box is pushed into the target position, (2) the box collides with the obstacle or any sidewall, and (3) the simulation reaches the maximum number of 500 training steps (each time an agent chooses its action, it is considered one training step). When any one of the three situations happens, a simulation run is completed, and the episode finishes.

One training process in the experiment goes through 16,000 episodes. The numbers are chosen because they are the thresholds for achieving training stabilization at the minimum training cost. In order to maintain statistical stability, each training was run 100 times with different random seeds. The mean value of the 100 cumulative rewards of a corresponding episode is used as the reward value of that episode and plotted as the solid colored lines in the cumulative reward plots in the subsequent part of "Case study box-pushing". At the same time, the standard errors are plotted as green spreads around the mean values.

As shown in Figure 4, the two dependent variables of the training process are *cumulative rewards* and the resulting *neural networks,* representing the team learning behavior and the agents' learned knowledge, respectively. Details of these variables are discussed subsequent part of "Case study box-pushing".

Table 1 summarizes the training parameters used in the simulations. Replay memory size is 1000, and 32 mini-batches are randomly sampled from the replay memory during each training step. The discount factor is 0.99, and the learning rate is 0.001. Total training episodes are 16,000. The ϵ-greedy action selection algorithm was followed where ϵ is annealed from 1.0 to 0.005 over 100,000 training steps during training. Agents choose actions from entirely random (i.e., $\epsilon = 1.0$) to nearly greedy ($\epsilon = 0.005$). The target neural networks are updated at every 200 training steps to stabilize the training. Training the network of each agent consists of an input layer of 9 states, a hidden layer of 16 neurons, a dueling layer (Wang *et al.*, 2016), and output 6 state-action values.

Table 1. Simulation parameters

Replay memory size	1,000
Mini-batch size	32
Discount factor	0.99
Learning rate α	0.001
Total training episodes	16,000
ϵ	1.0 → 0.005
Annealing steps	100,000
Target network update frequency	200
Training network size	(9, 16, dueling layer, 6)

Testing and performance evaluation

The neural networks that are acquired from the MARL-based training are evaluated through testing simulations. Figure 5 illustrates the design of the testing experiment, in which *#Tr* indicates the number of agents trained and *#Ts* is the number of agents being tested.

There are three independent variables in the testing experiment. They are: *learned neural networks* $NN_{\#Tr}$, *agent team size* (1, …, 10), and *noise level* (10%). The noise level is defined by the percentage in which the agents choose *random* actions instead of *greedy* ones during testing runs. This variable is introduced to model system malfunctions, such as sensor errors and agent disabilities. By correlating the system response to the noise levels, the robustness of the trained neural networks can be evaluated.

When the number of trained agents *#Tr* is not the same as the number of testing agents *#Ts* (i.e., $\#Tr \neq \#Ts$), there exists a mismatch for deploying the neural networks. To resolve this mismatch, for cases of $\#Tr < \#Ts$, one or more randomly selected networks are repeatedly deployed in the test case. When $\#Tr > \#Ts$, then *#Ts* networks are randomly selected for the testing team agents.

There is one dependent variable *performance score* in the testing experiment design, as shown in Figure 5, which indicates the *quality of the learned knowledge* and *scalability* as it is *trained* ($\#Tr = \#Ts$) and when it is applied in *knowledge transfer* testing situations ($\#Tr \neq \#Ts$), respectively. Furthermore, correlating the *performance score* with the *noise level* helps assess the *robustness to noise*.

As mentioned above, for each training situation, that is, given *#Tr*, there are 100 training runs. At the end of each training run, a *greedy testing simulation* is carried out to evaluate whether the box-pushing is successful. The *training performance score,* in this case, is defined by the percentage of the number of successful runs. Usually, the training performance score is greater than 60%, except for the team of one agent, which is close to 30%, as will be discussed below.

From the more than 60 successful runs of each training situation (*#Tr*), 50 sets of neural networks are randomly selected for

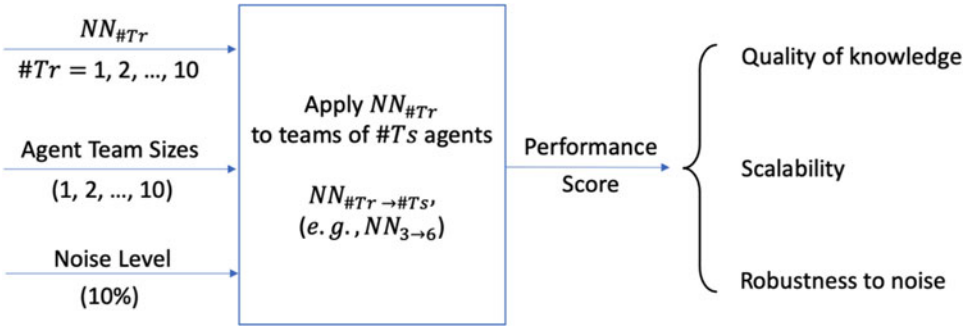


Fig. 5. Testing experiment design and performance evaluation.

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

testing cases. Again, for statistical stability, each selected set of neural networks is simulated 100 times, and the performance score is the percentage of the successful ones among the 100 simulations. This way, after 50 × 100 testing runs, the average performance score of the 50 sets of neural networks is used as the *performance score* of the testing situation (*#Tr, NoiseLevel*).

Results and discussion

Box-pushing trajectory

Figure 6 shows one successful testing run with *6 agents* pushing the box by following max state-action values with motion trace examples. Although the final optimized trajectory is not strictly perfect, through applying the multiagent deep Q-networks, agents can approximate their actions and push the box towards the goal area.

Training stability

As the number of agents trained varies from 1 to 10, details of the training stability plots of each agent are included in the appendix. When the number of agents is smaller than 5, it is hard for the system to reach a stable and high reward level. The standard deviation of these training cases is significant. This implies that the box-pushing task requires the cooperative effort of agents, and the size of agents needs to be at least 5 agents to reach a reasonable performance level. This "magic" number *5* appears to be correspondent to the complexity level of the box-pushing task (Ashby, 1961), which will be mentioned again below.

When the agent size is between 6 and 10, the final converged reward tends to decrease as team size increases. This is a strong indication that adding more agents to the system does not necessarily lead to better performance; having more agents involved only makes the learning less effective. Finding a good team size is crucial in self-organizing design, especially in constrained environment training cases.

Quality of learned knowledge

The quality of the learned knowledge is evaluated based on the performance score. The "performance score" (see Fig. 5) of a specific team is defined by the percentage of the successful simulation runs based on the learned knowledge, as discussed in "Experiment Design". Because the multiagent RL is only partially observable, not all training cases can be successfully trained. By "being successfully trained," it is meant that after training, the trained agents can successfully push the box towards the target by taking actions greedily suggested by their own neural networks. Figure 7 illustrates the results of the performance score of the trained teams of different sizes. When the number of agents is small, the performance score of training is low as it is hard to train a small number of agents to push the box. When the number of agents reaches the "magic" number 5, the performance score of training becomes 97%. When the number of agents further increases, the performance score drops. This means that the quality of the learned knowledge depends on the team size: higher-quality knowledge can be acquired for the team sizes at the "magic" number 5; for smaller and larger teams, the learned knowledge is less reliable. Identifying the "magic" number – that is, the matching point between the task complexity and team complexity – is the key to acquiring high-quality knowledge through multiagent RL.

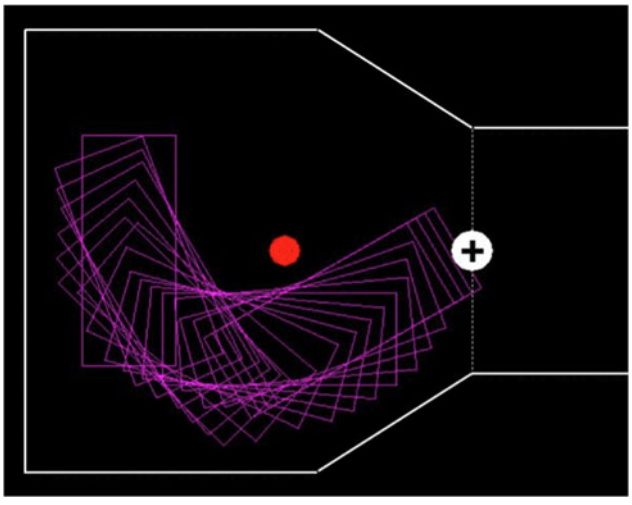


Fig. 6. A successful box-pushing trajectory with motion traces of a 6-agent team.

As rotation reward is a critical element in regulating agents' behaviors during training, we tested the sensitivity of the quality of knowledge with respect to different weights of rotation reward in different team sizes.

During training, the rotation reward is multiplied by a weight factor W_f in addition to its original weight W_2, as shown in Eq. (13). A list of values was chosen for W_f: 0, 0.1, 1, 10, 100. When $W_f = 0$, agents are trained with no rotation reward, and when $W_f = 100$, agents are trained with a very big rotation reward. After training, with the new trained neural networks $NN_{\#Tr}$, we evaluated the quality of learned knowledge using the same method as discussed before.

$$R_{\text{tot}} = w_1 * R_{\text{dis}} + w_2 * R_{\text{rot}} * w_f + w_3 * R_{\text{col}} + w_4 * R_{\text{goal}}. \quad (13)$$

The results of the sensitivity analysis are shown in Figure 8. As the number of agents increases, there is a general trend that the quality of knowledge improves with higher performance scores and plateaus at the "magic" team size of 5 agents. And after 5 agents, the performance score drops, which is similar to the trend shown in Figure 7. The finding shows that "magic" team size is insensitive to the weights of rotation reward and is related to the complexity of the system and task.

Also, for each specific team size, there is not a big difference between the quality of knowledge when agents are trained with various weight factors of rotation reward. This is because in a box-pushing environment, as agents' goal is merely reaching the goal area, the final orientation of the box is not crucial. As rotation reward regulates box-pushing angles, it is not very important with respect to its terminal goal. Also, as there is an obstacle between the initial box position and the goal area, it serves as an additional constraint during box-pushing. This makes learning more robust as it provides more state information for agents during learning and also prevents the box from rotating too much before hitting the obstacle, thus eliminating the need for incorporating rotation reward in regulating box rotation.

Scalability and robustness

The neural networks learned from a given team size are tested in teams of different sizes for scalability evaluation. In addition, 10% of random agent actions are introduced to evaluate the robustness of the trained networks to environmental changes. The results are

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

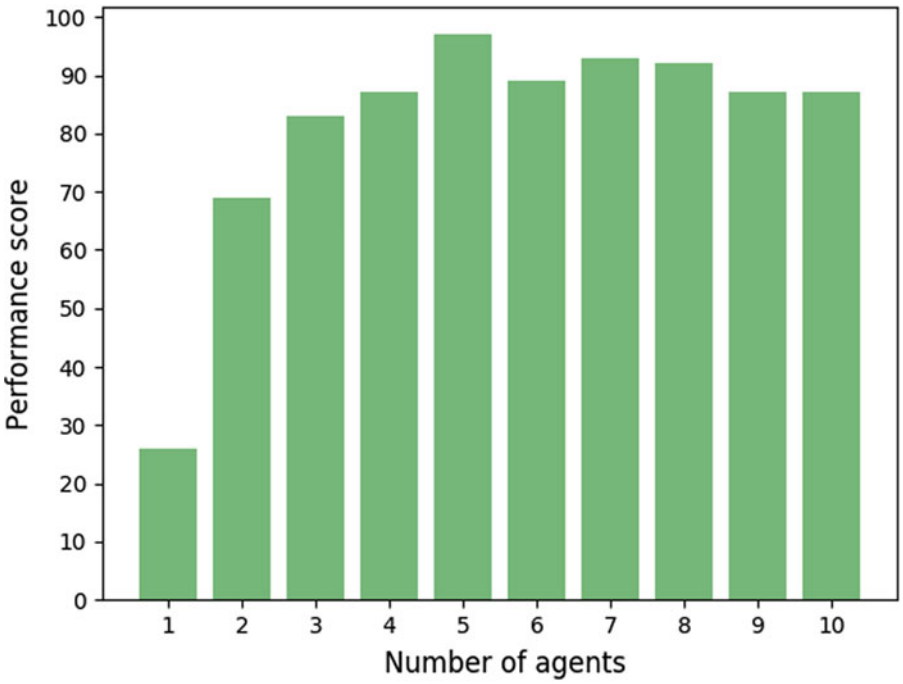


Fig. 7. Performance score with a different number of agents in box-pushing task.

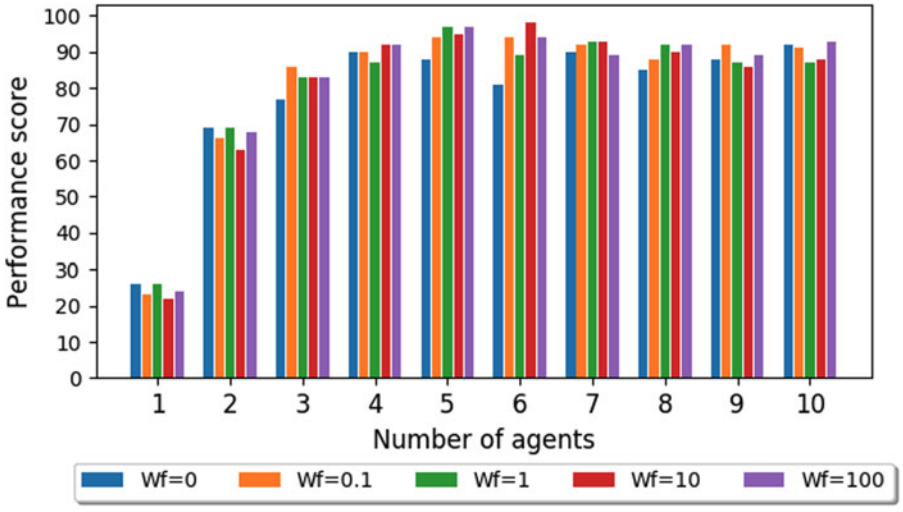


Fig. 8. Sensitivity of performance score with different weight factors of rotation reward in different number of agents in box-pushing task.

shown in Figure 9. The colors of the curves in the figures represent the different sets of neural networks obtained from the training of different numbers of agents. For example, "Tr:3" means the neural networks used by the agents are obtained from the training of 3 agents. The horizontal axis indicates the number of agents that engaged in the testing of the box-pushing task, that is, *#Ts*. The vertical axis shows the performance score of simulation runs.

Scalability for knowledge transfer

As shown in Figure 9, when the trained network is applied to the same team size, the performance score of testing runs first increases up to team size 5 and then decreases. This is consistent with the performance score of quality of knowledge of each team size.

Scaling downward: Transferring $NN_{\#Tr}$ downward to the teams of a smaller size *#Ts* leads to a much inferior performance score as transferring downward loses some vital information/knowledge within the system. Also, this can be considered due to the mismatch between the task complexity and the team complexity (Mnih *et al.*, 2013).

Scaling upward: When the trained agent team size *#Tr* is small and the learned networks are applied to a larger team, that is, $\#Tr < \#Ts$, scaling upward has led to reasonably good performances within neighboring 3–4 agents. This shows that system performance is more robust to adding more agents. Introducing additional agents can be considered as adding extra noise to the system; since neural networks are trained with random noise, it is robust to such "noise" generated by the additional agents.

Case study self-assembly

Self-assembly problem

A graphical illustration of a self-assembly case study is shown in Figure 10. The game screen has 800×800 pixels. Numerous

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

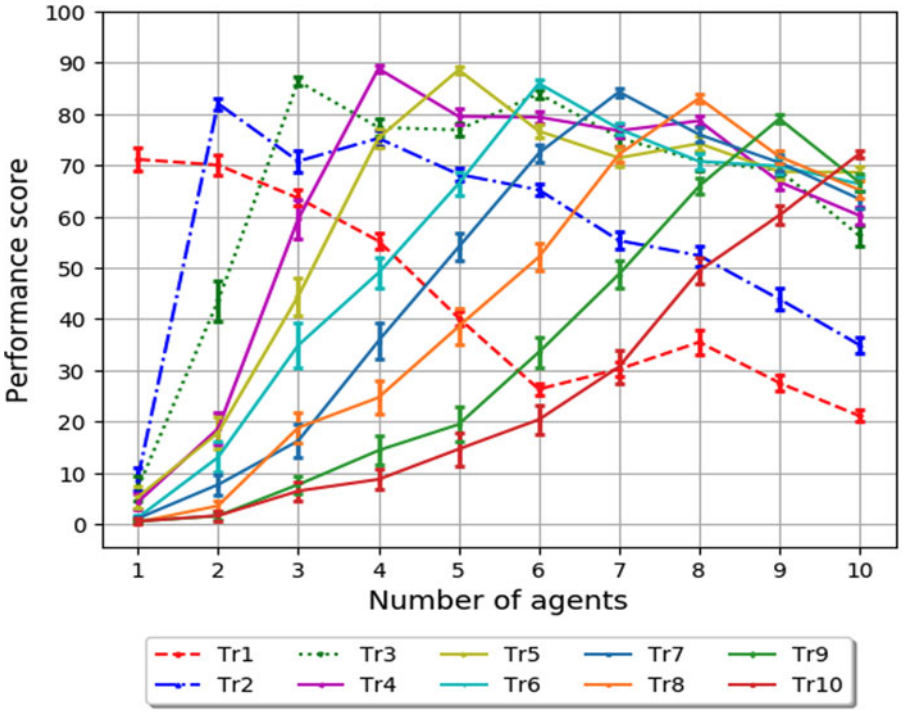


Fig. 9. Performance score of running with 10% individual random actions of a varying number of trained agents in box-pushing task.

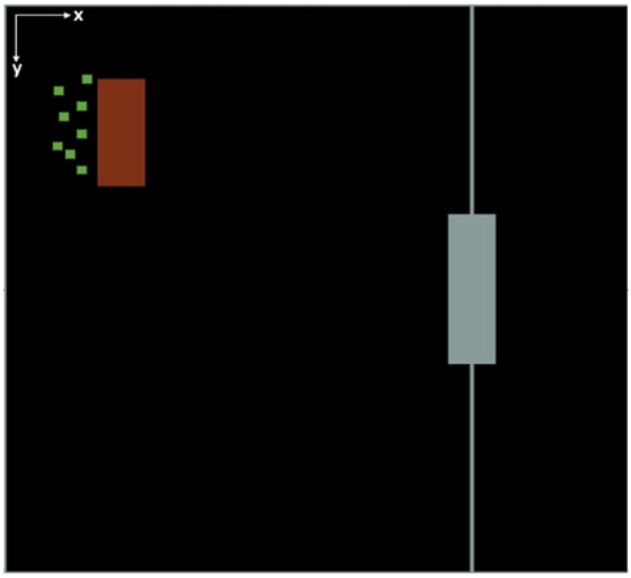


Fig. 10. Graphical illustration of self-assembly task.

agents (the green squares) with limited pushing and sensing capabilities need to self-organize in order to push and rotate the box (the brown rectangle) towards the static target box (the gray rectangle box) and form a T-shape structure centered at the target. The agents cannot just simply push the box but have to rotate the box when necessary (Khani and Jin, 2015; Khani *et al.*, 2016). This adds complexity to the task. Pymunk, a physics simulation module, was used to build the case study model. In pymunk, the distance is measured by pixels. As an example, the brown box is 60×150 pixels, and the gray box is 60×210 pixels.

State space and action space

The state space of the self-assembly task is defined as $S = < s, \alpha, \beta, \omega, v >$, as shown in Figure 11a and Table 2.

For gathering the state information, a sensor is deployed in the center of the moving box, which can sense nearby obstacles or targets with a radius of 200 pixels, and the entire circular sensor coverage is equally split into 24 sectors. The vicinity situation is modeled by $\boldsymbol{s} = \{s_1, s_2 \ldots, s_{24}\}$ with s_i representing the corresponding sector situation: has *nothing* or *obstacle* or *target*. In the state of Figure 11a, three obstacles are detected.

The box neighborhood is defined as six regions (Humann *et al.*, 2014; Khani and Jin, 2015), as shown in Figure 11b. The box dynamics are based on the pymunk physics model. The mass of the box is 1 kg. An agent can push the box from its position. Every push carries the same amount of impulse acting on the center of one of the box regions from an agent towards the box. The magnitude of an impulse is $1(N \cdot \mathrm{s})$, and the box will have a change of velocity of 1 m/s in the pushing direction and a change of angular velocity of around 1°/s if agents push on one of the longer sides of the box neighborhood. For every step in the simulation, agents perform an action and wait for 1 s until the next push is carried out. The *agent action space* is the same as the box-pushing case study, as shown in Figure 11b.

Reward schema

The reward schema follows the same format as the previous case study, and the total reward is composed of four parts: *distance*, *rotation*, *collision*, and *goal*.

Distance Reward: The reward for pushing the box closer to the goal position is represented as R_{dis}, and is shown in Eq. (14). *Distance coefficient* C_d is set to 0.02.

$$R_{\mathrm{dis}} = (D_{\mathrm{old}} - D_{\mathrm{new}}) * C_d. \tag{14}$$

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

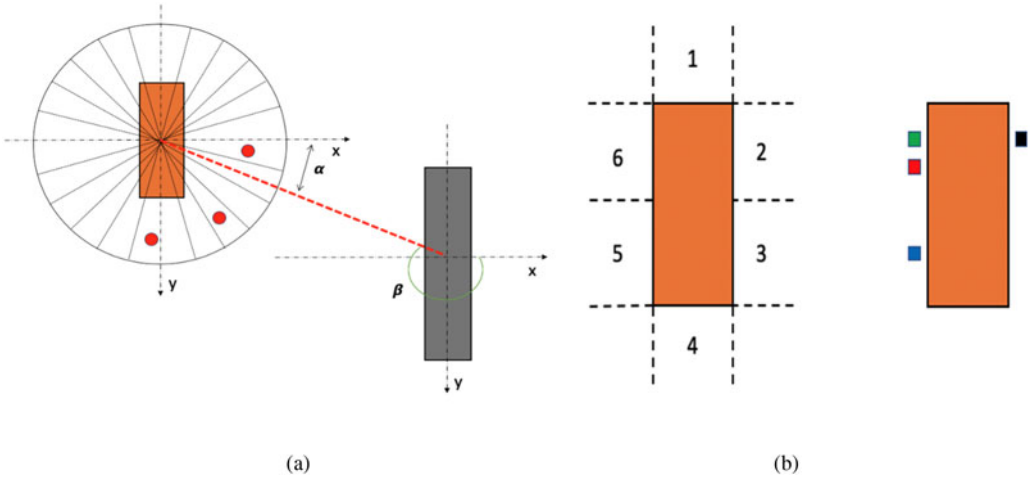


Fig. 11. Box state and neighborhood: (a) box state representation, (b) six regions of box neighborhood.

Table 2. State space definition

State	Description
$\boldsymbol{s}$	The vicinity situation (has nothing/obstacle/target) of the 24 equal sectors around the moving box within 200-pixel range.
α	Angle between moving box x-axis and the target vector [20–21].
β	Angle between fixed box x-axis and the moving box center
ω	Angular velocity of the moving box.
$\boldsymbol{v}$	Velocity of the moving box in global x-axis and y-axis directions

Table 3. Training simulation parameters and values

Replay memory size	1,000	ϵ	1.0 → 0.01
Mini-batch size	32	Annealing steps	250,000
Discount factor	0.99	Target network update frequency	200
Learning rate	0.001		
Total training episodes	16,000	Dueling DQN network size	(29, 64, 128,6)

Rotation Reward: The reward for rotation R_{rot} is represented in Eq. (15), where α_1 is the previous time step angle between the center of the target box with respect to the moving box's x-axis, and α_2 is the current angle. C_r is called rotation coefficient and is set to 0.1.

$$R_{\text{rot}} = (\text{Cos}(\alpha_2 - \alpha_1) - 0.98) * C_r. \tag{15}$$

Collision Reward: The collision reward is represented in Eq. (16) as R_{col},

$$R_{\text{col}} = \begin{cases} -10 & \text{if collision occurs} \\ 0 & \text{if no collision occurs.} \end{cases} \tag{16}$$

During each simulation step, if there is no collision for the box with the wall, $R_{\text{col}} = 0$. If a collision occurs, a -10 reward is given to all the agents as a penalty.

Goal Reward: The reward for reaching the goal R_{goal} is represented in Eq. (17),

$$R_{\text{goal}} = \begin{cases} 100 * |\text{Sin}(\alpha)| * |\text{Cos}(\beta)| & \text{if reached goal} \\ 0 & \text{if no reaching goal.} \end{cases} \tag{17}$$

α and β are shown in Table 2. During simulation, if the box reaches the target box and form a perfect T-shape, each agent receives a 100 reward; if reached the target box but slightly off angle, the reward is between 0 and 100; if the target is not reached, the agents do not receive any reward. The total reward is the sum of all the rewards, as shown in Eq. (18). The simulation parameters of our algorithm can be seen in Table 3.

$$R_{\text{tot}} = R_{\text{dis}} + R_{\text{rot}} + R_{\text{col}} + R_{\text{goal}}. \tag{18}$$

Issue and experiment setup

The case study is focused on three issues: (1) achieving *stable learning dynamics* by investigating the impact of rotation reward, (2) sensitivity of SOS to different weights of rotation reward function, (3) evaluation of the SOS performance by analyzing the quality of learned knowledge and assessing *the robustness* of the learned neural network knowledge in the context of varying team sizes and environmental noise.

The number of agents i during training was varied between 4 and 40, with 4 agents in between, that is, $4 \le i \le 40$. The training yields i different neural networks denoted as N_i which is a set of i networks. To evaluate the robustness of the acquired network knowledge, each N_i was applied to the testing cases with j number of agents, and j was varied between 4 and 40, with 4 agents in between, that is, $4 \le j \le 40$. The notation N_i^j ($4 \le i \le 40$; $4 \le j \le 40$) describes the situation where the network knowledge obtained from the training of i agents is applied to the testing case of j agents. It is conceivable that there are 10 training cases but 100 testing cases. When the testing case agent number j is larger than the training agent number i, that is, $i < j$, four or more randomly selected networks are repeatedly deployed in the test case. When $i > j$, then j networks are randomly selected from N_i for testing simulations.

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

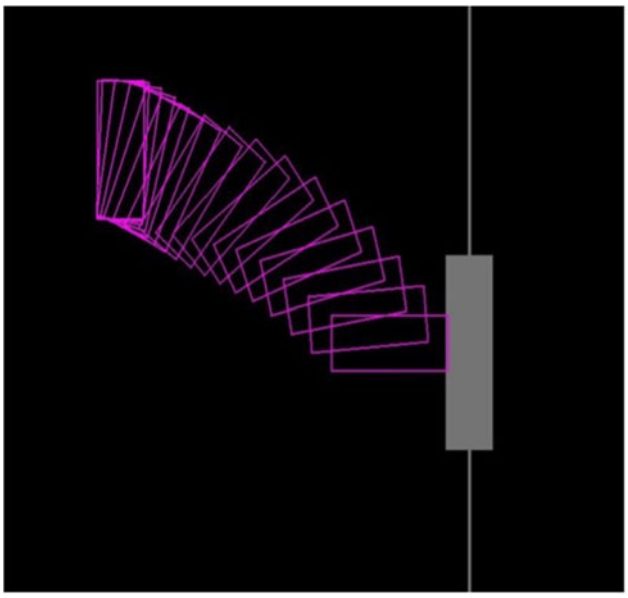


Fig. 12. Successful self-assembly trajectory with motion traces.

For maintaining statistic stability, for each i agents team, the agents are trained 100 times with different random seeds, resulting in neural networks N_{il}, $4 \leq i \leq 40$; $l = 1, 2, \ldots, 100$. To evaluate the performance of a specific N_{il} after training, the goal reward of Eq. (17) is used as the performance score. The maximum score is 100, which is when the perfect T-shape is formed in the middle of the target box. For a given i, all 100 networks N_{il} were collected as the network dataset from which a specific N_{il} is randomly selected to test j-agent teams, N_{il}^{j}. The testing simulation for a given j-agent team is also run 100 times, and the mean cumulative reward values and performance scores are used in the plots shown in Figures 13–16.

Results and discussion

Box-pushing trajectory

Figure 12 shows one successful training run with an 8-agent team by following a policy that maximizes state-action values. Although the final optimized trajectory is not strictly perfect, through multiagent deep Q-networks, agents can approximate its actions and push the moving box towards the target box and form a T-shape structure.

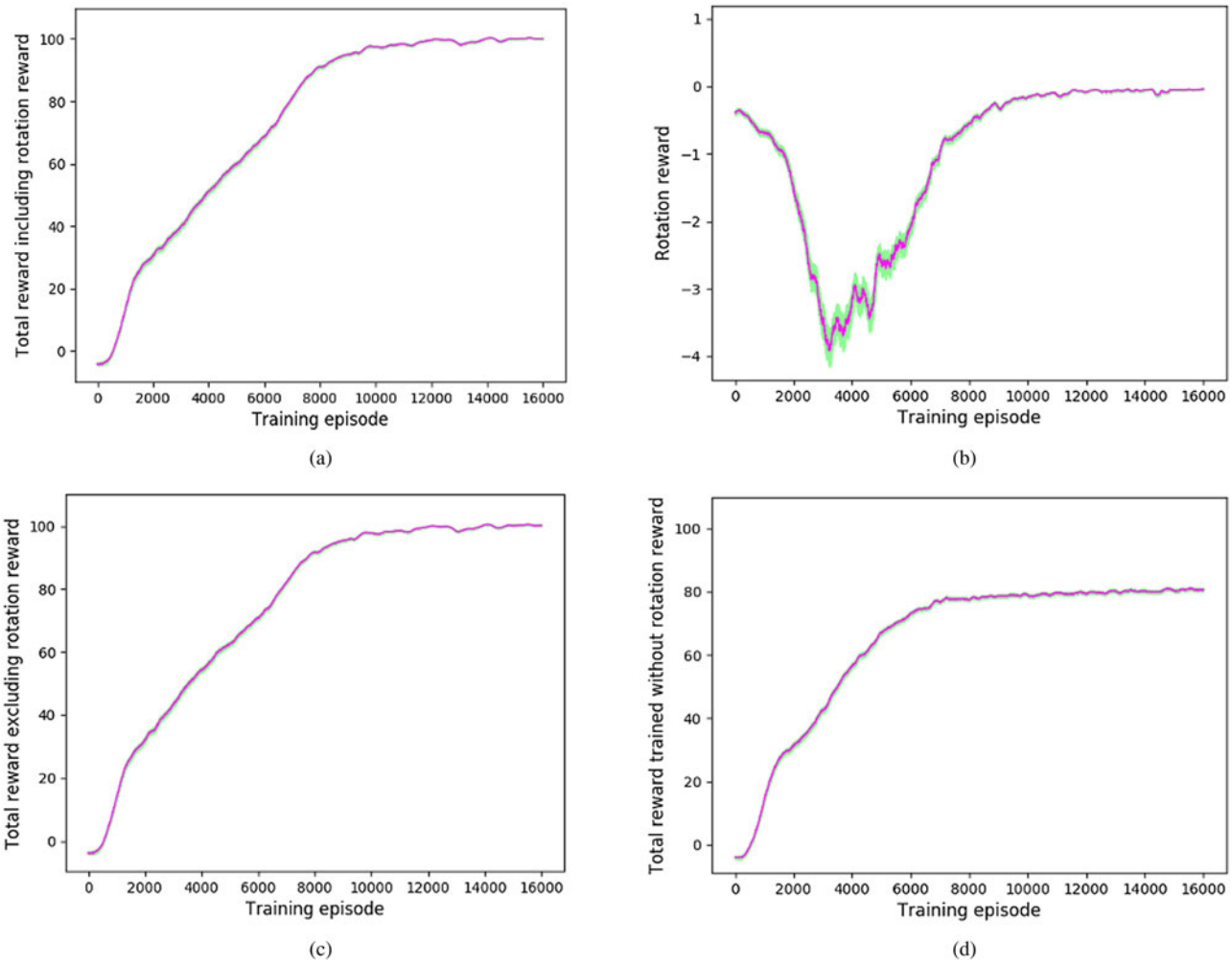


Fig. 13. Reward plot versus training episodes of 24-agent with standard error of the mean in green shaded region: (a) total reward including rotation reward, (b) rotation reward, (c) total reward excluding rotation reward, (d) total reward trained without rotation reward.

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

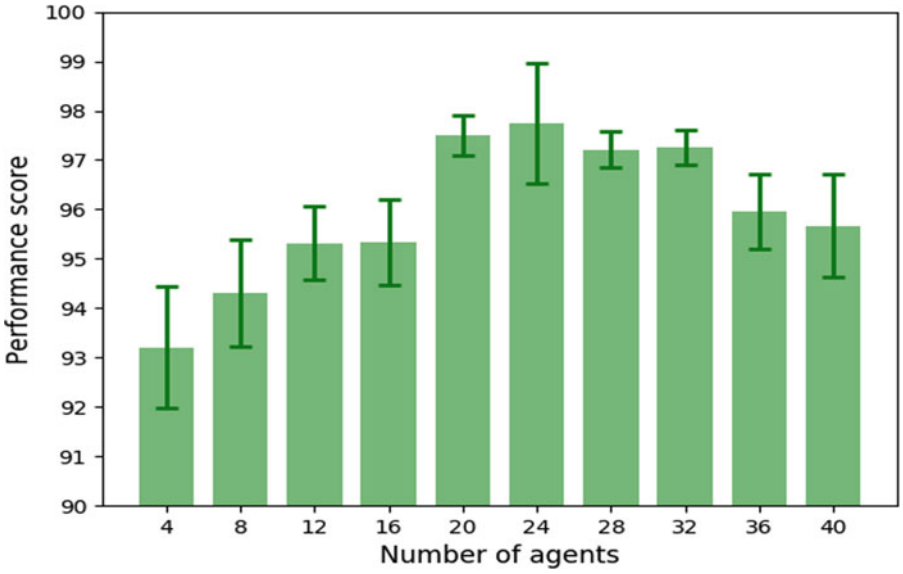


Fig. 14. Performance score with a different number of agents in self-assembly task.

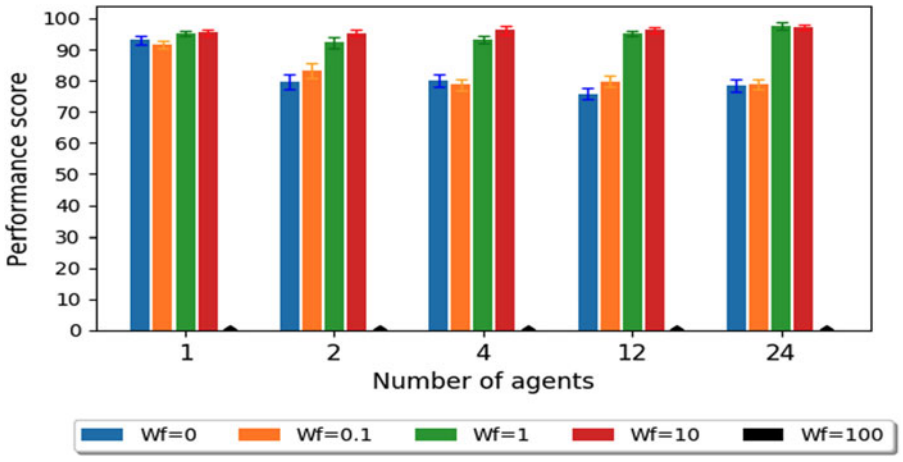


Fig. 15. Sensitivity analysis of performance score with a different number of agents with a different weight factor of rotation reward in self-assembly task.

Training stability

It was found that, unlike the first case study, the cumulative reward does not differ much between different team sizes. This is due to a rather small difference in reward values in self-assembly tasks among various team sizes. Therefore, in Figure 13, we only show the training convergence results of 24-agent teams, N_{24}. The reward plots are based on the mean values of 100 training simulation runs, *i.e.*, $l = 1, 2, \ldots, 100$, with different random seeds.

Figure 13a–c shows the reward plots for teams trained with the *rotation reward* [see Eq. (15)]. Figure 13d is the reward plot for teams trained without the rotation reward. As shown in Figure 13a, the cumulative reward of the 24-agent teams trained with the rotational reward converges to almost 100, and once the cumulative reward reaches the threshold, it stays the same without much oscillation. The green shaded region in the plots represents the standard error from the mean value of the reward. Figure 13b shows how the rotation reward changes with increasing episodes. As the episode number increases, the rotation reward first decreases and then increases to around 0. Figure 13c shows how the cumulative reward, excluding rotation reward increases with episodes. It is used as a comparison with Figure 13d, in which the agents are trained without the rotation reward, and training finally converged to only around 80 rewards.

This result indicates that introducing rotation reward had a significant impact during training in the self-assembly task, which further demonstrates that providing terminal rewards alone is not enough for agents to find optimal policies successfully. Regulating agents' behavior during the training process is essential to mitigate the effect of too much or too little rotation for arriving at the maximum potential of the agent team.

Quality of learned knowledge

The quality of the learned knowledge is evaluated by a performance score. For each i agents team, the agents are trained 100 times with different random seeds, resulting in neural networks N_{il}, $4 \leq i \leq 40$; $l = 1, 2, \ldots, 100$. The performance score of each N_{il} is evaluated using the goal reward of Eq. (17). The standard error of the mean is also calculated for each team size based on the 100 different performance scores of each random seed. Figure 14 illustrates the results of the performance score of the trained teams of different sizes. Similar to the box-pushing case study, the quality of the learned knowledge depends on the

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

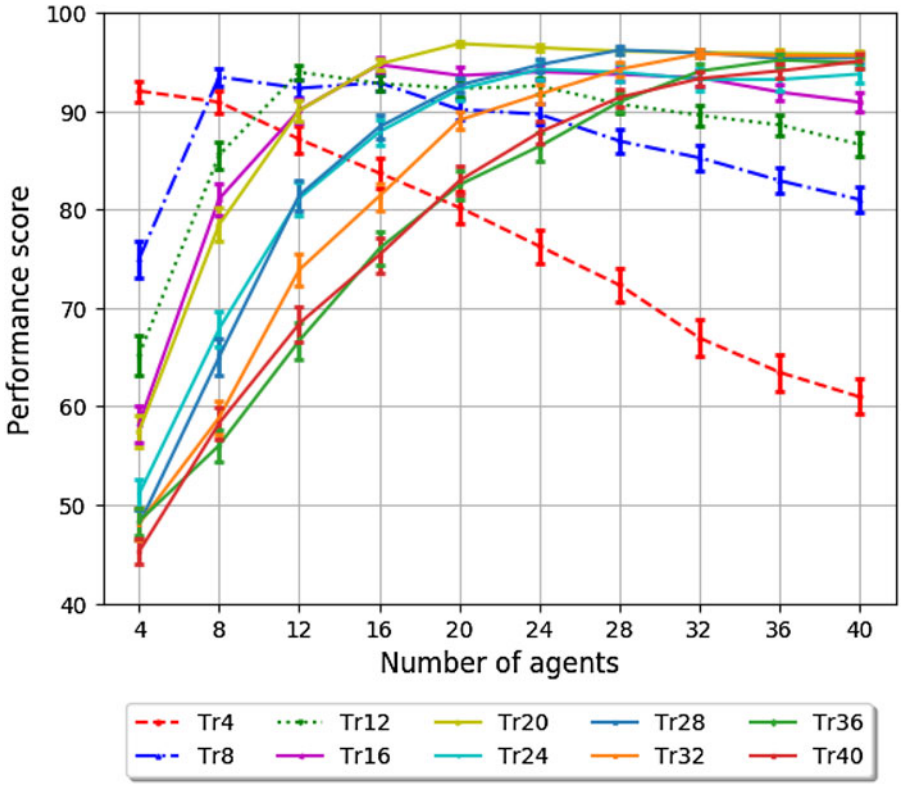


Fig. 16. Transfer performance score with a different number of transfer agents with error bars indicating the standard error.

team size: higher-quality knowledge can be acquired for the team sizes at the "magic" number 20 or 24 compared to smaller and larger teams. Finding the "magic" number – that is, the matching point between the task complexity and team complexity – is crucial in acquiring high-quality knowledge through multiagent RL.

We also tested the sensitivity of quality of knowledge with respect to different weights of rotation reward in various team sizes: 1, 2, 4, 12, and 24. During training, the rotation reward is multiplied by a weight factor W_f in addition to its original weight 1, as shown in Eq. (19). A list of values was chosen for W_f: 0, 0.1, 1, 10, 100. After training, with the new trained neural networks N_{il}, we evaluated the quality of learned knowledge using the same evaluation method: performance score.

$$R_{\text{tot}} = R_{\text{dis}} + R_{\text{rot}} * w_f + R_{\text{col}} + R_{\text{goal}}. \tag{19}$$

The results of the sensitivity analysis are shown in Figure 15. The error bar shows the standard error of the mean of the performance score with trained neural networks N_{il}. When agents of various team sizes are trained with $W_f = 100$, agents ultimately learned how to rotate the box for an extensive period of time in order to accumulate rotation reward instead of learning how to form the T-shape structure. The performance score is indicated with a black rectangle, which represents the inferior quality of knowledge learned when W_f is very big. Except $W_f = 100$, when the number of agents is 1, there is not a big difference between the quality of knowledge trained with various weight factors of rotation reward. This is because one agent can only move the box with a change of angular velocity of around 1°/s, which is not a very large angle. Thus, having a rotation reward as a penalty reward is not very useful in this case. However, when agent sizes reach 2, 4, and beyond, the quality of knowledge learned is largely dependent on the weights of rotation reward. When $W_f = 0$ and $W_f = 0.1$, the effect of regulating agents' behavior is not very strong, and the performance score is low. When $W_f = 1$ and $W_f = 10$, there is enough regulation, and agents can reach the full potential of learning, and the final performance score is most optimal. Thus, for the self-assembly task, it is always advised to introduce some rotation reward, and the weight of such rotation reward cannot be too small nor too big, especially when the agent team size is relatively large: greater than one.

Scalability and robustness

The *scalability* of the learned neural network knowledge to different team sizes and the *robustness* to various noises are two important issues for this case study. Applying neural network trained with i-agent teams to test and assess the performance of j-agent teams, that is, N_i^j ($4 \le i \le 40; 4 \le j \le 40$), allows us to assess the scalability. Further, introducing 10% random agent actions helps evaluate the robustness of the trained networks. The results are shown in Figure 16. The curves with different colors in the figure represent the training of i-agent teams. For example, Tr4 means training of the 4-agent team and Tr8 8-agent team. The horizontal axis indicates j-agent testing teams. The vertical axis shows the performance score of testing [Eq. (17)], with error bars showing the standard error of the mean.

Similar findings can be drawn compared to the box-pushing case study. As shown in Figure 16, when the trained networks are applied to the same team size, N_i^i, agents have a performance score of above 90. As the team size i increases, the performance score of testing runs N_i^i first increases and then decreases, indicating that there is a threshold of the effect of adding more agents. In this case study, the threshold is around 20.

Like the box-pushing case study, transferring learned knowledge downward is less effective compared to doing so upward in terms of

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

team size. This is because during scaling upward, the extra agents can be considered as extra noise, and the system is robust to the impact of "noise." Scaling downward is more detrimental as the system loses some knowledge within its learned neural networks.

Implications for future SOS design

Although it is worth mentioning that the findings and insights described above are limited to the two case studies reported in this paper, some general guidelines for future SOS design can be derived as follows.

Adding more agents to the system increases system complexity. The "magic" team size is the point where system complexity matches task complexity and has optimal learning stability, quality of knowledge, and scalability. Magic team size is dependent on the type of tasks and system dynamics and is insensitive to the weights of rotation reward during training. Designers can test hypothetical "magic" team sizes using multiagent RL to obtain maximum system performance. For example, from the training stability perspective, at "magic" team size, training becomes most stable and reaches the maximum cumulative reward. From an evaluation perspective, magic team sizes should have a better quality of the learned knowledge and are most robust in response to environmental noises among all team sizes tested.

Learning stability, quality of knowledge, and scalability are three metrics that can be used to assess the SOS. Depending on the type of task, the results of each evaluation measurement can vary.

Learning stability: For tasks with fixed terminal rewards, for example, box-pushing, when agents successfully push the box to the goal, they can get a fixed terminal reward. Failure to push the box to the goal area leaves agents with zero terminal rewards. The variation between the learning stability of agents of various sizes can be manifested with cumulative reward plots as there is a big difference of reward between successful pushing and unsuccessful pushing. For the self-assembly task, the goal reward is based on the final orientation of the box; it is a variational reward depending on the terminal position and is in the range of 0–100. Thus, in self-assembly tasks, it is hard to differentiate between the learning stability of a different number of agents using cumulative reward plots.

Quality of knowledge: Unlike learning stability, which evaluates the learning behavior of agents during training, quality of knowledge assesses the learned knowledge of agents after training. The performance score is a numerical value that measures the quality of knowledge learned. In both box-pushing and self-assembly case studies, quality of knowledge measurement is an effective tool in assessing the learned knowledge of SOS because it can represent slight differences in values by using a performance score, which can be used to determine the "magic" team size of the SOS. In the box-pushing case study, for specific team size, the quality of knowledge is insensitive to the weight of rotation reward. This is because the final goal is not strictly related to the box orientation, and the presence of an obstacle along the pushing trajectory regulates agent behaviors and reduces the need for rotation reward in regulating agents' actions. However, in the self-assembly task, the inclusion of rotation reward is essential in obtaining the best quality of knowledge. This is because the self-assembly task does not have obstacles along the box's trajectory, and the final position and orientation of the box are critical in accomplishing the task.

Scalability: Scalability is another measurement that is used after training. In both case studies, scaling upward to a higher team size has less impact on system performance during the transfer of knowledge than scaling downward. This is because a neural network is more robust to the noise generated by extra agents than losing knowledge within its team.

Future SOS design should consider the features of the tasks, such as the ultimate goal of the task and whether there are physical constraints like obstacles presented, and introduce appropriate behavior regulation reward, for example, rotation reward during training. Finally, designers can selectively choose three metrics mentioned in our paper to evaluate their system performance depending on the type of task.

Conclusions and future work

In multiagent RL-based SOS design, the sensitivity of SOS performance to specific reward functions and systematic evaluation of SOS are essential issues to consider. In this paper, a rotation reward function has been introduced, and the sensitivity of SOS performance to such reward function was analyzed through two different case studies. Three evaluation metrics, including learning stability, quality of knowledge, and scalability, were investigated. Following conclusions can be drawn from the experiment results.

- The multiagent RL training process has good stability with a proper range of agent team sizes with different types of tasks.
- When the system complexity matches the task complexity, reaching the "magic" number, the learned knowledge has the best team performance, scalability, and robustness.
- The optimal team size and quality of knowledge are insensitive to the weights of rotation reward.
- Introducing rotation rewards has a significant impact on training performance.
- The multiagent RL-based self-organizing knowledge captured in the form of neural networks can scale upward better with respect to the team size.

The main contribution of this research to engineering design is twofold. First, the introduction of the rotation reward penalizes the excessive rotation movements and allows the agents to learn from making serious mistakes; the experiment results indicate that the landscape of shaping rewards reflects the essence of the tasks and should be carefully explored and designed. Second, three evaluation metrics, namely, learning stability, quality of learned knowledge, and scalability, are proposed to systematically measure the performance of SOS; the case studies have demonstrated the effectiveness of these measures. From a designing perspective, depending on the type of tasks, designers should choose appropriate weights of reward functions to obtain the full potential of agent learning capability. The evaluation metrics in our paper can be used to identify the optimal number of agents to achieve the desired system performance, and the selection of evaluation metrics should be based on the characteristics of the task.

It is worth mentioning that the insights obtained and the conclusions are limited by the case studies described. Future work includes adding more case studies to justify our results and findings. Also, introducing a more refined reward model could potentially benefit the learning of agents and improve the performance of the SOS. For instance, the collision reward in our paper only considers the situation when the box collides with the walls or obstacles and ignores the collision between agents as the assumption of minimal robots is made, and agent team size is relatively small. For large team sizes such as hundreds or thousands of agents, the collision between agents would become inevitable as

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

agents are likely to follow similar routes. Assigning a large collision reward as a penalty could be a generator of noise during training. Therefore, collision reward needs to be introduced with more precision in such situations, and additional measures need to be taken in the event of a collision to obtain better SOS performance. Finally, introducing mathematical measurement of the complexity of the task and the system is another direction for predicting the "magic" team size of SOS.

Acknowledgements. This paper was based on the work supported in part by the Autonomous Ship Consortium with member companies of MTI Co., Japan Marine United Corporation, Tokyo Keiki Inc., BEMAC Inc., ClassNK, and Japan National Maritime Research Institute. The authors are grateful to the sponsors for their support for this research.

References

Abramson J, Ahuja A, Barr I, Brussee A, Carnevale F, Cassin M, Chhaparia R, Clark S, Damoc B, Dudzik A, Georgiev P, Guy A, Harley T, Hill F, Hung A, Kenton Z, Landon J, Lillicrap T, Mathewson K, Mokrá S, Muldal A, Santoro A, Savinov N, Varma V, Wayne G, Williams D, Wong N, Yan C and Zhu R (2020) Imitating interactive intelligence. arXiv preprint arXiv:2012.05672.

Arroyo M, Huisman N and Jensen DC (2018) Exploring natural strategies for bio-inspired fault adaptive systems design. *Journal of Mechanical Design* **140**, 091101-1–091101-11.

Ashby WR (1961) *An Introduction to Cybernetics*. London, UK: Chapman & Hall Ltd.

Ashby WR (1991) Requisite variety and its implications for the control of complex systems. In Klir CJ (ed.), *Facets of Systems Science*. Boston, MA: Springer, pp. 405–417.

Bar-Yam Y (2002) General features of complex systems. In Kiel LD (ed.), *Encyclopedia of Life Support Systems (EOLSS)*. Oxford, UK: UNESCO, EOLSS Publishers.

Beckers R, Holland OE and Deneubourg JL (2000) From local actions to global tasks: stigmergy and collective robotics. In Cruse H, Dean J and Ritter H (eds), *Prerational Intelligence: Adaptive Behavior and Intelligent Systems Without Symbols and Logic, Volume 1, Volume 2 Prerational Intelligence: interdisciplinary Perspectives on the Behavior of Natural and Artificial Systems, Volume 3*. Dordrecht: Springer, pp. 1008–1022

Busoniu L, Babuska R and De Schutter B (2008) A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **38**, 156–172.

Chen C and Jin Y (2011) A behavior based approach to cellular self-organizing systems design. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. **54860**, pp. 95–107.

Chiang W and Jin Y (2012) Design of cellular self-organizing systems. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. **45028**. American Society of Mechanical Engineers, pp. 511–521.

Chung J, Gulcehre C, Cho K and Bengio Y (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.*3555.

Collinot A and Drogoul A (1998) Using the Cassiopeia method to design a robot soccer team. *Applied Artificial Intelligence* **12**, 127–147.

Dasgupta P (2008) A multiagent swarming system for distributed automatic target recognition using unmanned aerial vehicles. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* **38**, 549–563.

Drogoul A and Zucker JD (1998) *Methodological Issues for Designing Multiagent Systems with Machine Learning Techniques: Capitalizing Experiences from the Robocup Challenge* (Doctoral dissertation, LIP6).

Ferguson SM and Lewis K (2006) Effective development of reconfigurable systems using linear state-feedback control. *AIAA Journal* **44**, 868–878.

Foerster J, Nardelli N, Farquhar G, Afouras T, Torr PH, Kohli P and Whiteson S (2017) Stabilising experience replay for deep multiagent reinforcement learning. *International Conference on Machine Learning*. PMLR, pp. 1146–1155.

Foerster J, Farquhar G, Afouras T, Nardelli N and Whiteson S (2018) Counterfactual multiagent policy gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. **32**, No. 1.

Groß R, Bonani M, Mondada F and Dorigo M (2006) Autonomous self-assembly in swarm-bots. *IEEE Transactions on Robotics* **22**, 1115–1130.

Hausknecht M and Stone P (2015) Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*.

Hochreiter S and Schmidhuber J (1997) Long short-term memory. *Neural Computation* **9**, 1735–1780.

Humann J, Khani N and Jin Y (2014) Evolutionary computational synthesis of self-organizing systems. *AI EDAM* **28**, 259–275.

Humann J, Khani N and Jin Y (2016) Adaptability tradeoffs in the design of self-organizing systems. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. **50190**. American Society of Mechanical Engineers, p. V007T06A016.

Ji H and Jin Y (2018) Modeling trust in self-organizing systems with heterogeneity. *ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers Digital Collection.

Ji H and Jin Y (2019) Designing self-organizing systems with deep multi-agent reinforcement learning. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. **59278**. American Society of Mechanical Engineers, p. V007T06A019.

Ji H and Jin Y (2020) Designing self-assembly systems with deep multiagent reinforcement learning. *Design Computing and Cognition'14*. Springer, Cham, pp. xx–xx.

Jones C and Mataric MJ (2003) Adaptive division of labor in large-scale minimalist multi-robot systems. *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, Vol. **2**. IEEE, pp. 1969–1974.

Kennedy J (2006) Swarm intelligence. In Zomaya AY (ed.), *Handbook of Nature-Inspired and Innovative Computing*. Boston, MA: Springer, pp. 187–219.

Khani N and Jin Y (2015) Dynamic structuring in cellular self-organizing systems. In Gero JS (ed.), *Design Computing and Cognition'14*. Cham: Springer, pp. 3–20.

Khani N, Humann J and Jin Y (2016) Effect of social structuring in self-organizing systems. *Journal of Mechanical Design* **138**, 041101-1–041101-11.

Königseder C and Shea K (2016) Comparing strategies for topologic and parametric rule application in automated computational design synthesis. *Journal of Mechanical Design* **138**, 011102-1–011102-12.

Lamont GB, Slear JN and Melendez K (2007) UAV swarm mission planning and routing using multi-objective evolutionary algorithms. 2007 *IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making*. IEEE, pp. 10–20.

LaValle SM (2006) *Planning Algorithms*. Cambridge, UK: Cambridge University Press.

Liu X and Jin Y (2018) Design of transfer reinforcement learning mechanisms for autonomous collision avoidance. *International Conference on-Design Computing and Cognition*. Cham: Springer, pp. 303–319.

Lowe R, Wu Y, Tamar A, Harb J, Abbeel P and Mordatch I (2017) Multiagent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*.

Martin MV and Ishii K (1997.Design for variety: development of complexity indices and design charts. *Proceedings of ASME 1997 Design Engineering Technical Conferences, September 14–17, 1997, Sacramento, CA*, DFM-4359-1–DFM-4359-9.

McComb C, Cagan J and Kotovsky K (2017) Optimizing design teams based on problem properties: computational team simulations and an applied empirical test. *Journal of Mechanical Design* **139**, 041101-1–041101-12.

Meluso J and Austin-Breneman J (2018) Gaming the system: an agent-based model of estimation strategies and their effects on system performance. *Journal of Mechanical Design* **140**, 121101-1–121101-9.

Min G, Suh ES and Hölttä-Otto K (2016) System architecture, level of decomposition, and structural complexity: analysis and observations. *Journal of Mechanical Design* **138**, 021102-1–021102-11.

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

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.

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S and Hassabis D (2015) Human-level control through deep reinforcement learning. *Nature* **518**, 529–533.

Peng XB, Berseth G, Yin K and Van De Panne M (2017) Deeploco: dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)* **36**, 1–13.

Pippin CE (2013) *Trust and Reputation for Formation and Evolution of Multi-robot Teams* (Doctoral dissertation). Georgia Institute of Technology.

Pippin C and Christensen H (2014) Trust modeling in multi-robot patrolling. 2014 *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 59–66.

Price IC and Lamont GB (2006) GA directed self-organized search and attack UAV swarms. *Proceedings of the 2006 Winter Simulation Conference*. IEEE, pp. 1307–1315.

Rahimi M, Gibb S, Shen Y and La HM (2018) A comparison of various approaches to reinforcement learning algorithms for multi-robot box pushing. *International Conference on Engineering Research and Applications*. Cham: Springer, pp. 16–30.

Rashid T, Samvelyan M, Schroeder C, Farquhar G, Foerster J and Whiteson S (2018) Qmix: monotonic value function factorisation for deep multiagent reinforcement learning. *International Conference on Machine Learning*. PMLR, pp. 4295–4304.

Reynolds CW (1987) Flocks, herds and schools: a distributed behavioral model. *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. pp. 25–34.

Ruini F and Cangelosi A (2009) Extending the evolutionary robotics approach to flying machines: an application to MAV teams. *Neural Networks* **22**, 812–821.

Sutton RS and Barto AG (2018) *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Tampuu A, Matiisen T, Kodelja D, Kuzovkin I, Korjus K, Aru J, Aru J and Vicente R (2017) Multiagent cooperation and competition with deep reinforcement learning. *PLoS One* **12**, e0172395.

Tan M (1993) Multiagent reinforcement learning: Independent vs. cooperative agents. *Proceedings of the Tenth International Conference on Machine Learning*. pp. 330–337.

Wang Y and De Silva CW (2006) Multi-robot box-pushing: single-agent q-learning vs. team q-learning.*2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 3694–3699.

Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M and Freitas N (2016) Dueling network architectures for deep reinforcement learning. *International Conference on Machine Learning*. PMLR, pp. 1995–2003.

Watkins CJCH (1989) Learning from delayed rewards.

Wei Y, Madey GR and Blake MB (2013) Agent-based simulation for uav swarm mission planning and execution. *Proceedings of the Agent-Directed Simulation Symposium*, pp. 1–8.

Werfel J (2012) Collective construction with robot swarms. In Doursat R, Sayama H and Michel O (eds), *Morphogenetic Engineering*. Berlin, Heidelberg: Springer, pp. 115–140.

Appendix

Figure A1 shows the convergence results of box-pushing simulations with a different numbers of agents. The figures show the average cumulative reward of 100 random training seeds with respect to the training episodes. The standard error of the mean is plotted with the green shaded region.

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

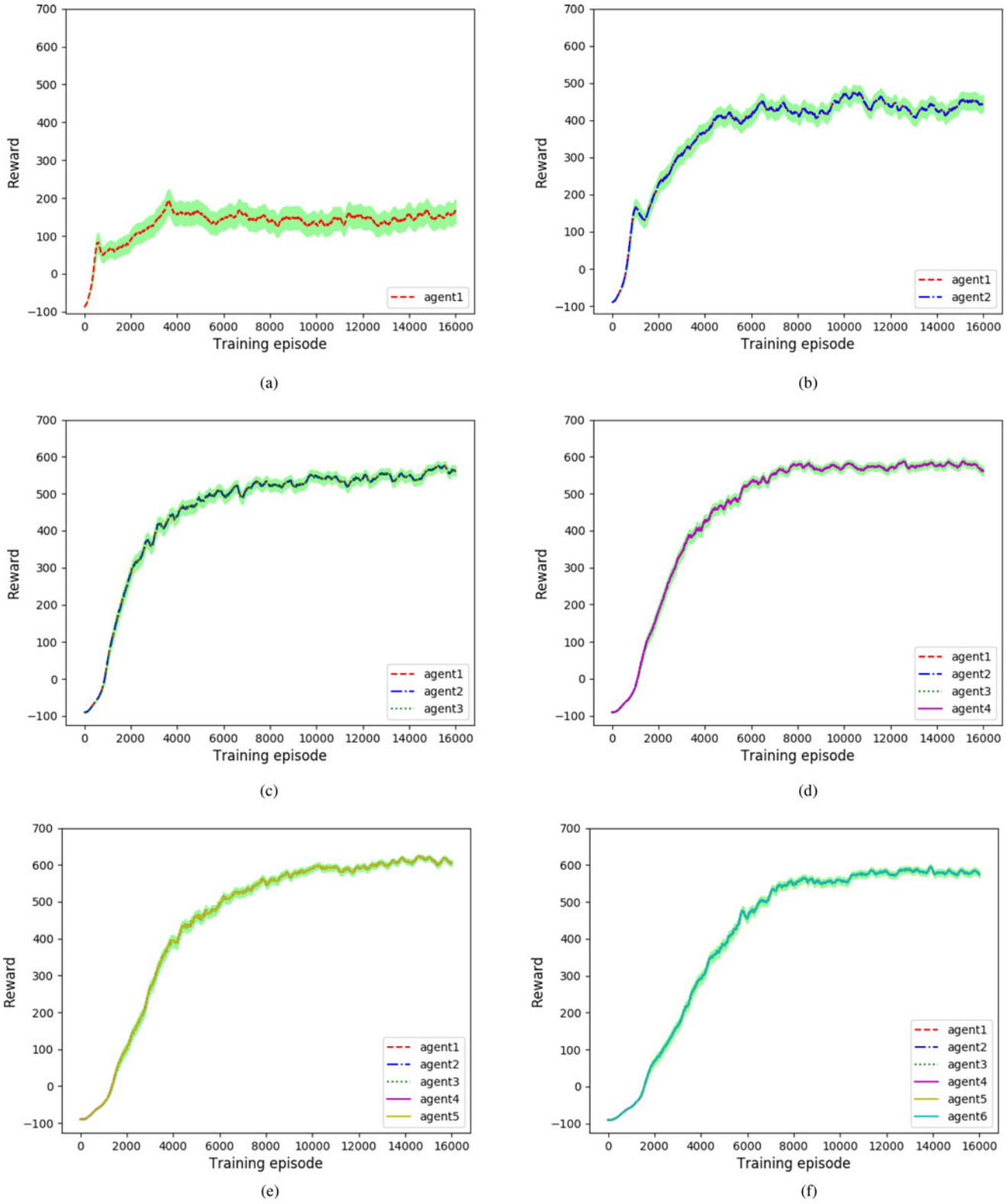


Fig. A1. Cumulative reward versus training episode plots with varying sizes of teams. (a) team size = 1, (b) team size = 2, (c) team size = 3, (d) team size = 4, (e) team size = 5, (f) team size = 6, (g) team size = 7, (h) team size = 8, (i) teams size = 9, (j) team size = 10.

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press

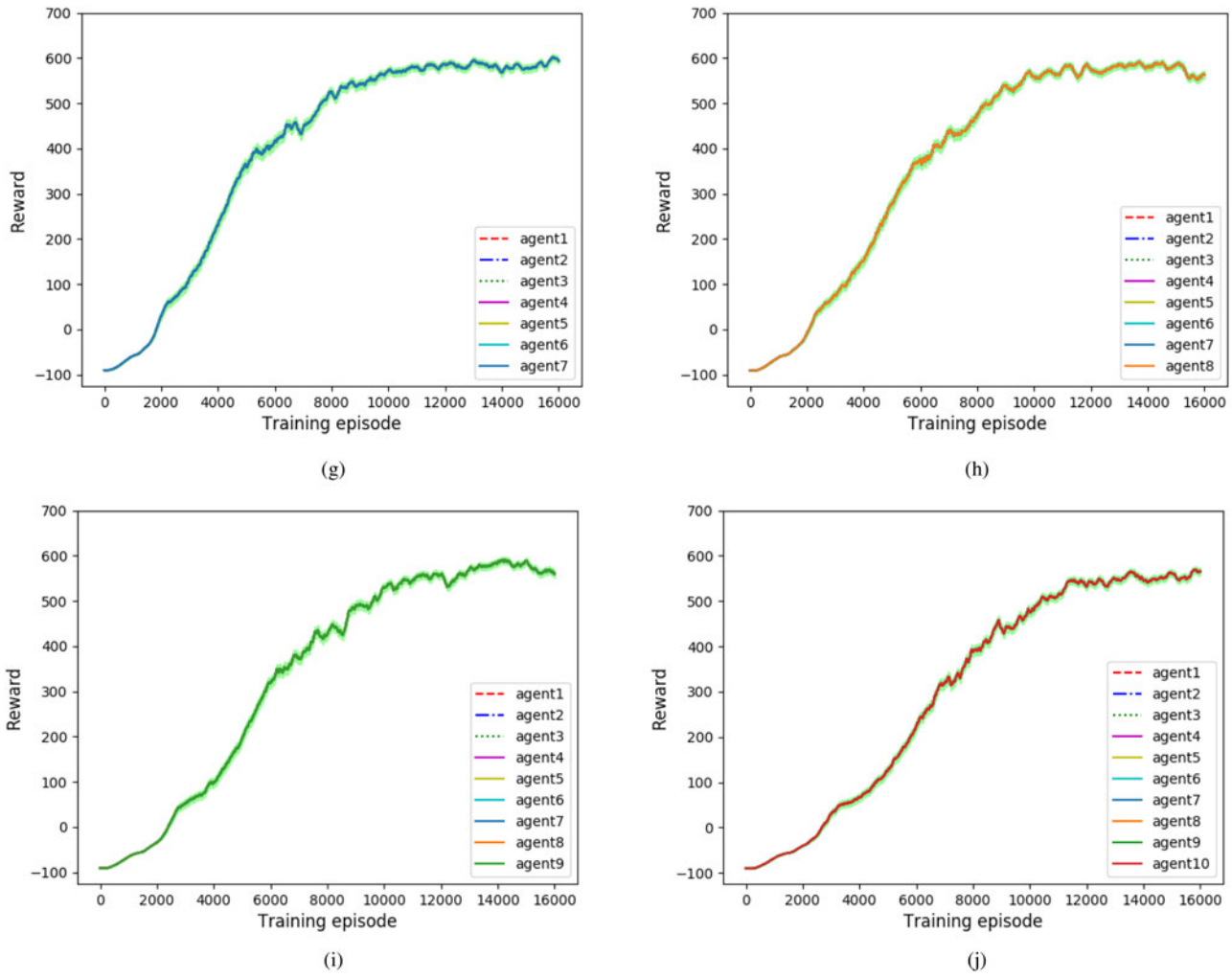


Fig. A1. Continued.

Hao Ji received his PhD degree in Mechanical Engineering from the University of Southern California in the summer of 2021. His research was focused on developing multiagent reinforcement learning algorithms for self-organizing systems. Dr. Ji received his BS degree from Donghua University and his Master's degree from the University of California at Berkeley. He worked as an affiliate researcher at the University of California at Berkeley before joining the University of Southern California for his PhD study. He is currently working as a data scientist in machine learning-related areas.

Yan Jin is Professor of Aerospace and Mechanical Engineering at the University of Southern California. He received his PhD degree in Naval Engineering from the University of Tokyo and did his post-doctoral research at Stanford University. Dr. Jin's research covers design theory & methods, multiagent, and self-organizing systems. His current research interests include design cognition, machine learning and its applications in engineering design, knowledge capturing, self-organizing, and adaptive systems. Dr. Jin is a recipient of the National Science Foundation CAREER Award and TRW Excellence in Teaching Award. He served as Editor-in-Chief of AIEDAM and Associate Editor of JMD and is currently Associate Editor of Design Science Journal. Dr. Jin is a Fellow of ASME.

https://doi.org/10.1017/S089006042100024X Published online by Cambridge University Press