Designing Self-Assembly Systems with Deep Multiagent Reinforcement Learning

Hao Ji and Yan Jin
University of Southern California, USA

Self-organizing systems (SOS) are developed to perform complex tasks in unforeseen situations with adaptability. Previous work has introduced task fields and rule-based social structuring for individual agents to comprehend the task situations and follow social rules to accomplish their tasks without a centralized controller. Although the task fields and social rules can be predefined for relatively simple tasks, for complex tasks and changing environments, acquiring *a priori* knowledge about the fields and rules can be challenging. In this paper, a multiagent reinforcement learning (RL) model is introduced as a design approach to solving the rule generation problem for complex tasks. A deep multiagent RL algorithm was devised to train agents to acquire the task and self-organizing knowledge, and a behavior regulation reward was introduced to regulate agent behaviors during training. Scalability of the training algorithm to different team sizes was investigated. The results show that SOS design based on deep multi-agent RL with behavior regulation reward performs better than training without behavior regulation. Scaling up to larger team sizes has better performance than scaling downwards.

Key words: Deep Q-learning, self-organizing system, behavior reward

Introduction

Self-organizing systems can consist of simple agents that work together to complete complex tasks. Design of complex systems by applying a self-organizing approach has many advantages, such as scalability, adaptability, and reliability [1,2]. Various approaches have been proposed to support the design of SOS. The field-based behavior regulation (FBR) approach [3]

Design Computing and Cognition DCC'20. J.S. Gero (ed), pp. xx-yy.

models the task environment with a field function and the behavior of the agents is regulated based on the positions of these agents in the field by applying a field transformation function. With this approach the agents need little knowledge to perform tasks since their behavior is governed by the task field. However, the limitation of the task field transformation leads to the inadequacy of applying the FBR approach to more complex task domains. The social structuring approach [4] attempts to simplify the task field by introducing a social field modeled by social rules [4-5]. A major issue with this social rule based approach is that a designer must know *a priori* what rules should be applied, which is often difficult to attain, especially in collaborative self-assembly tasks where the ending positions and angles of the objects are crucial.

In multiagent RL, each agent can be trained using its own neural network, such trained neural network can be a generalization of rules. Therefore, in this research, a *RL* approach is taken to capture the self-organizing knowledge in SOS design. We focus on how to design appropriate rewards to train multi-agents to learn efficiently and what are the key parameters that contribute to the successful training? Specifically, a multiagent Q-learning algorithm with behavior regulation rewards is explored to address two research questions: *What factors impact the stability of learning dynamics in SOS*? *Will the RL captured knowledge be robust enough to be applied in a wide range of task situations*?

In the following, the relevant work is reviewed in Section 2, and an independent Q-learning framework is introduced in Section 3. Section 4 presents a self-assembly case study, and the results are discussed in Section 5. Section 6 draws the conclusions and points to future research directions.

Related Work

An artificial self-organizing system is a system that is designed by human and has emergent behavior and adaptability like Nature [6]. Werfel developed a system of homogenous robots to build a pre-determined shape using square bricks [7]. Khani et al. developed a social rule-based regulation approach in enforcing the agents to self-organize and push a box toward the target area [4,8]. Jin and Chen used a field-based regulation (FBR) approach and guides self-organizing agents to perform complex tasks such as approaching long-distance targets while avoiding obstacles [3]. Price investigated into the use of genetic algorithm (GA) in optimizing Self-organizing multi-UAV swarm behavior [9].

Multiagent RL (MARL) applies to multiagent settings and is based largely on the single agent RL such as Q-learning, policy gradient and actor-

critic [10-11]. One natural approach for MARL is to optimize the policy or value functions of individuals. The most used value-function based multiagent learning is independent Q-learning [12]. It trains individual's state-action values using Q-learning [12-13] and is served as a common benchmark in the literature. Tampuu [14] extended previous Q-learning to deep neural networks and applied DQN [15] to train two independent agents playing the game Pong [14]. Foerster used a centralized critic to evaluate decentralized agents and estimated a counterfactual advantage function based on each agent and allocated credit among agents [16]. He also analyzed his replay stabilization methods for independent Q-learning based on StarCraft game combat scenarios [17].

Most approaches to MARL focus on achieving optimal system reward or desirable convergence properties, often based on fully observable states. Training of MARL models is usually in prespecified environments and the scalability of the trained network to different agent team sizes is not analyzed, that is crucial for SOS design. One needs a MARL approach that is scalable to various team sizes and can guide SOS system design.

Method: Deep Multiagent Reinforcement Learning

Single Agent Reinforcement Learning: MARL is based on single agent RL, which is used to optimize system performance based on training so that the trained system can solve complex tasks from the raw sensory inputs. In single agent RL, learning is based on the Markov Decision Process (MDP) 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 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 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 equation (1) below as an update. E represents the expected value. Eventually, such a value iteration will converge to optimal value function.

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

Recently, deep neural networks are introduced as functional approximator to replace the old Q-table for estimating Q values, such learning methods are called deep Q-learning [15]. A Q-network with weights θ_i can be trained by minimizing the loss function at each iteration i, shown in equation (2),

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

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 (4):

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

Multiagent Reinforcement Learning: 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. To increase learning efficiency and maintain scalability, a multiagent independent deep Q-learning 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 $A = A_1 \times \bullet \bullet \bullet \times A_n$. All agents share the same state space and the same reward function as task is cooperative. During training, each agent has its own neural network, they perceive and learn independently but 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 MARL mechanism described above, each agent i (i = 1, 2, …, 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 only its own action a_i, in addition to the current state S_t. In this research, we explore the *stability issue* of the learning process—i.e., whether the knowledge can be acquired in the form of neural networks through RL with behavior regulation, and the *adaptability issue*—i.e., whether the learned neural networks can be effectively applied to the situations of similar tasks but different agent team sizes.

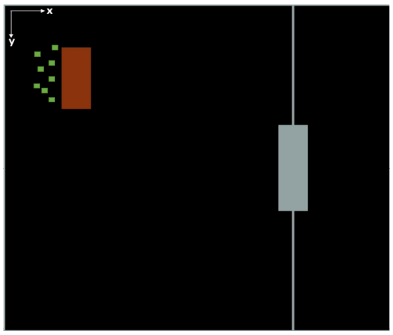


Figure 1. Graphical illustration of Box-pushing task

Case studies

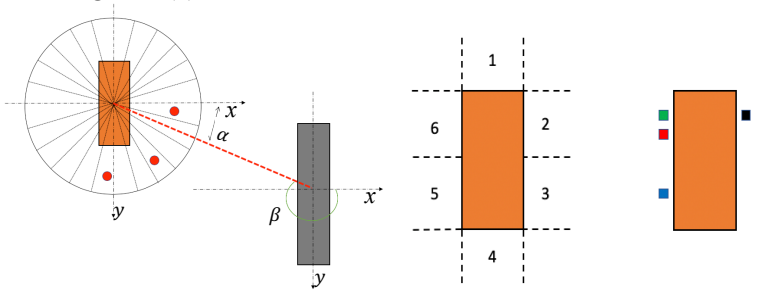
A graphical illustration of a self-assembly case study is shown in Figure 1. The game screen has a 800X800 pixels. Numerous agents (the green squares) with limited pushing and sensing capabilities need to self-organize in order

to push and rotate the box (the brown rectangle) towards the static target box (the grey rectangle box) and form a T shape structure centered at target. The agents cannot just simply push the box but have to rotate the box when necessary [4,8]. 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. Each pixel is a single square area in the simulation environment. As an example, the brown box is 60 x 150 pixels and grey box is 60 x 210 pixels. In the self-assembly task, agents have *limited sensing capabilities*. They can receive information from the sensor of the box, which measures orientation of the box and senses obstacles at a range of distance, implying the "minimalist" [18] and complying with physical robot hardware [19].

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 2(a) and Table 1.



(a) (b)

Figure 2. Box state and neighborhood: (a) box state representation (b) six regions of box neighborhood

Table 1: 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 direction

To gather the state information, a sensor is deployed in the center of the moving box, which can sense nearby obstacles/target with the 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 represents the corresponding sector situation: has *nothing* or *obstacle* or *target*. In the state of Figure 2(a), three obstacles are detected.

In order to define the action space for the assembly task, the concepts of *box neighborhood* and *box dynamics* are introduced. The *box neighborhood* is defined as six regions [2,8], as shown in Figure 2(b). During simulation, individual agent can move to one of the six regions and that specific region is the *position* of the agent. As agents are relatively small, there can be multiple agents in the same region at the same time.

The *box dynamics* is based on the *pymunk* physics model. The mass of box is 1kg. An agent can push the box from its position. Every push carries the same amount of impulse acting on the center of one of box regions, from an agent towards the box. The magnitude of an impulse is 1($N \cdot$ s) and the box will have change of velocity of 1m/s in the pushing direction and change of angular velocity of around 1 degree/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 second until next push is carried out.

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 actions space of $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$, as shown in Figure 2(b). For instance, if an agent chooses action a_1, it will move to box region "1" and push from there, the box will move downwards along the box's *y*-axis based on the simulated box dynamics and the same logic applies to other agent actions.

Reward Schema

Adapted from the Q-table based box-pushing reward schema [20], we designed a new reward schema for agents' self-assembly 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 equation (6). Design with distance reward gives incentive for the agents to move closer to the target. The previous distance D_{old} represents how far the center of the box is away from the target position (measured in pixels) and can be evaluated by the box sensor and stored into an agent's memory. D_{new} represents such distance at the current time step. C_d is a constant, called *distance coefficient*, and is set to 0.02, which gives a little incentive for agents to move closer to target box. At each simulation time step, agents calculate the change of distance between the current distance and previous distance and draw its distance reward based on Equation (6).

$$R_{dis} = (D_{old} - D_{new}) * C_d \tag{6}$$

Rotation Reward: The reward for rotation R_{rot} is represented in equation (7).

$$R_{rot} = (Cos(\alpha_2 - \alpha_1) - 0.98) * C_r \quad (7)$$

where α_1 is the previous time step angle between center of target box with respect to the moving box's *x*-axis and α_2 is the current angle. The rotation reward is given to discourage the rotation of more than 11 degrees; this way, the box can be rotated constantly with small degrees and avoid large rotation momentum. C_r is called *rotation coefficient* and is set to 0.1. The rotation reward is relatively small as it is used only for box rotation rather than pushing towards the target. It is a behavior regulation reward as it regulates agents' behavior during training by punishing agents' misbehaviors to obtain full potential within the agents.

Collision Reward: The collision reward is analogous to the reward schema in common collision avoidance tasks [21] and is represented in equation (8) as R_{col},

$$R_{col} = \begin{cases} -10 \; if \; collision \; occurs \\ 0 \; if \; no \; collision \; occurs \end{cases} \quad (8)$$

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

Goal Reward: The reward for reaching the goal R_{goal} is represented in equation (9),

$$R_{goal} = \begin{cases} 100 * |Sin(\alpha)| * |Cos(\beta)| & if \; reached \; goal \\ 0 & if \; no \; reaching \; goal \end{cases} \quad (9)$$

α and β are shown in Table 1. 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 will be between 100 and 0; if the target is not reached, the agents do not receive any reward. Design with a large positive goal reward can motivate agents to reach their ultimate goal.

The total reward is the sum of all the rewards, as shown in Equation (10) below.

$$R_{tot} = R_{dis} + R_{rot} + R_{col} + R_{goal} \quad (10)$$

During training, as the agents are homogenous and cooperating to push the box, they should receive the same rewards. Thus, the reward equations (8) through (10) are defined based only on the box's position and orientation. As a result, each agent's neural network will consider other agents' actions as part of its environment and learn to explore its action space and its best policy based on an ε-*greedy* action selection strategy [15]. Gradually the agent grasps how to differentiate its actions from other agents to collaboratively push the box towards the target box and form a T-shape,

which is the characteristics of multiagent independent deep Q-leaning neural networks. The simulation parameters of our algorithm are in Table 2.

Table 2: Training simulation parameters and values

Replay memory size	1000	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	16000	Dueling DQN network size	(75, 64, 128,6)

Issues and Experiment Setup

The case study is focused on two issues: (1) achieving *stable learning dynamics* by investigating the impact of *behavior regulation reward* and (2) assessing *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 1 and 10, i.e., $1 \leq i \leq 10$. The training will yield *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 1 and 10, i.e., $1 \leq j \leq 10$. The notation N_i^j $(1 \leq i \leq 10; 1 \leq j \leq 10)$ describes the situation where the network knowledge obtained from 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*, i.e., $i < j$, one 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.

To maintain statistic stability, for each *i* agents team, the agents are trained 100 times with different random seeds, resulting in neural networks $N_{il}, 1 \leq i \leq 10\ ;\ l = 1, 2, \ldots, 100$. To evaluate the performance of a specific N_{il} after training, the goal reward of equation (9) 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*, the top 50 performers' 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 4&5.

Results and Discussion

Figure 3 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 strict perfect, through multiagent deep Q-

networks, agents can approximate its actions and push the box towards the target box and form a T-shape structure.

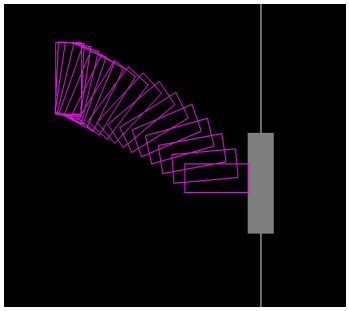


Figure 3: successful box-pushing trajectory with motion traces

Training stability

It was found that the cumulative reward does not differ much between different team sizes. Therefore, in Figure 4, we only show the training convergence results of 8-agent teams, N_8. The reward plots are based on the mean values of 100 training simulation runs, $i.e., l = 1, 2, \ldots, 10$, with different random seeds.

Figure 4(a) – 5(c) are reward plots for teams trained with the *behavior regulation rotation reward* (see Eq. (7)). Figure 4(d) is the reward plot for teams trained without the rotation reward. As shown in Figure 4 (a), the cumulative reward of the 8-agent teams trained with the rotational reward converges to almost 100, and once the reward reaches the threshold, it stays the same without much oscillation. The green shaded region in the plots indicates the standard error from the mean value of reward. Figure 4(b) shows how the behavior regulation rotation reward changes with increasing episodes. As the episode number increases, the rotation reward first decreases and then increases to around 0. Figure 4(c) shows how the cumulative reward excluding rotation reward increases with episodes. It is used to compare with Figure 4(d), in which the agents are trained without the shaping rotation reward and finally converged to only around 80.

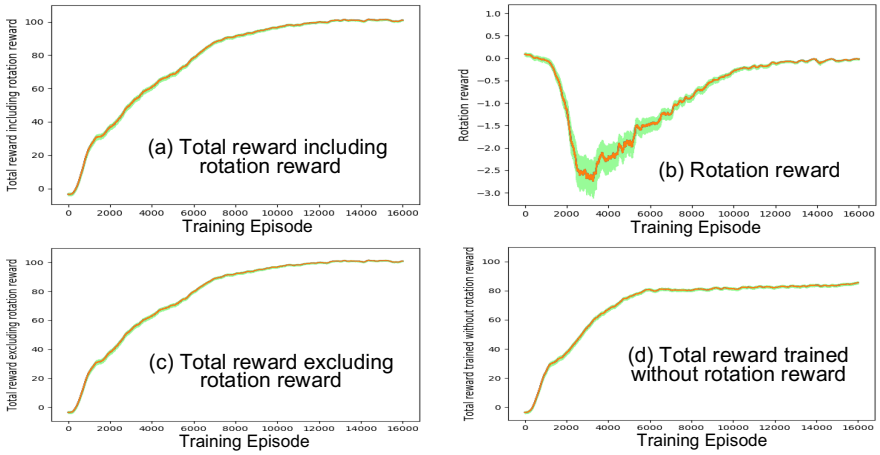


Figure 4. Reward plot vs training episodes of 8-agent teams with standard error in green shaded region

This result indicates that introducing rotation behavior regulation 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 successfully find optimal policies. Regulating agents' behavior during the training process is important to mitigate the effect of too much or too little rotation for arriving of the maximum potential of the agent team.

Scalability of training algorithm to varying team sizes

The *scalability* of the learned neural network knowledge to different team sizes and the *robustness* to various noises are two important issues for this research. Applying neural network trained with i-agent teams to test and assess the performance of j-agent teams, i.e., N_i^j $(1 \leq i \leq 10; 1 \leq j \leq 10)$, 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 5. The curves with different colors in the figure represent training of i-agent teams. For example, TO1 means training of 1-agent team and TO5 5-agent team. The horizonal axis indicates j-agent testing teams. The vertical axis shows the performance score of testing (Eq. (9)), with error bars showing standard error of the mean.

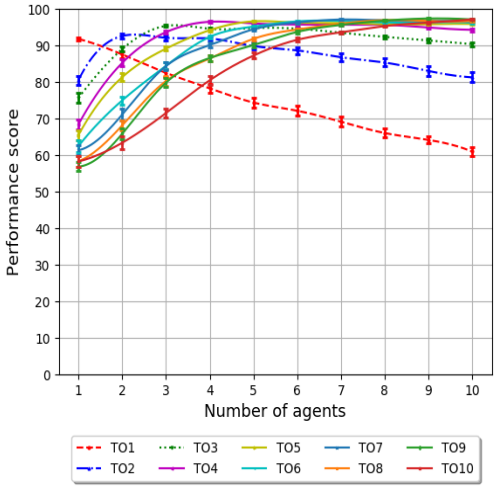


Figure 5. Transfer performance score with different number of transfer agents with error bars indicating standard error

As shown in Figure 5, 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 remains rather stable, indicating that there is a threshold of effect of adding more agents. In this case study, the threshold is around 5.

If we transfer the individual learned neural network to teams of larger size, i.e., $N_i^j, j > i,$ the performance score remains high as long as the number of agents of the trained team, $i,$ is equal to or larger than 4 ($i \geq 4$). For the cases of $i = 1$ and $i = 2$, both the learning dynamics and team dynamics are very different from other team sizes, hence the learned knowledge can hardly be scaled up.

When the transfer of learned knowledge is from a larger team to smaller teams, i.e., $N_i^j, j < i,$ the performance score maintains its value to a certain extent if starting number of agents i is between 5 to 10. When i is 2 to 5, the effect of transferring knowledge downward is not as good. As indicated in Figure 5, overall, transferring learned knowledge downward is less effective compared to doing so upward in terms of team size. The neural networks learned from team size of 5 to 8 are most scalable in both directions. It is considered that the team size of this range is most adequate for this specific task with the 8 having more flexibility and 5 on the border of needed capacity. When the task becomes more complex, the adequate team size may change.

Conclusions

In multiagent RL based SOS design, learning stability and scalability of the system with various team sizes are two important issues to consider. In addition, the system needs to be robust enough to perform well in the face of noises. In this paper, we applied the multiagent independent Q-leaning algorithms to designing SOS and investigated the learning stability, scalability, and robustness characteristics of such design approach through a box-pushing self-assembly case study. The results have shown that

- MARL is an effective approach to *capture self-organizing knowledge through extensive training*. In our self-assembly case, the agents successfully accomplished the task with high performance scores based on the learned neural networks.
- The *behavior regulation reward* plays a central role for the MARL teams to reach its *full potential capacity* through learning. As shown in Figure 4(d), without such behavior regulation the self-organizing agents can only attain 80 percent of the full capacity.
- MARL based self-organizing knowledge captured in the form of neural networks can *scale upward better than downward* in terms of team size. There is a task specific *adequate team size range* in which the scalable and robust neural network knowledge can be obtained from training.
- The teams with more agents are more robust against random actions. But *too many agents* may lead to the learned network knowledge that can hardly be transferred to smaller teams.

It is worth mentioning that the above conclusions are limited to the case studies described in the paper. Further work is needed for generalization. Our future work includes adding individual heterogeneous rewards, testing the different reward functions, and applying to more complex task cases.

References

1. Chiang W, and Jin Y. "Design of Cellular Self-Organizing Systems" IDETC /CIE20120-71216.
2. Humann J, Khani N, & Jin Y. (2014). Evolutionary computational synthesis of self-organizing systems. AI EDAM, 28(3), 259-275.
3. Jin, Y. and Chen, C. (2014) Field Based Behavior Regulation for Self-Organization in Cellular Mechanical Systems, AIEDAM, 28(2), pp.115-128.
4. Khani, N., Humann, J., & Jin, Y. (2016). Effect of Social Structuring in Self-Organizing Systems. Journal of Mechanical Design, 138(4), 041101.
5. Ji H, and Jin Y (2018) "Modeling Trust in Self-Organizing Systems with Heterogeneity." ASME IDETC-2018-86006.
6. Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. ACM SIGGRAPH computer graphics, 21(4), 25-34.
7. Werfel, J. (2012). Collective construction with robot swarms. In Morpho-genetic Engineering (pp. 115-140). Springer Berlin Heidelberg.
8. Khani, N., & Jin, Y. (2015). Dynamic structuring in cellular self-organizing systems. In Design Computing and Cognition'14(pp. 3-20). Springer, Cham.
9. Price, I. C., & Lamont, G. B. (2006). GA directed self-organized search and attack UAV swarms. Proc. of the 38th Conf. on Winter Simulation 1307-1315
10. Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.
11. Bu, Lucian, Robert Babu, and Bart De Schutter. "A comprehensive survey of multiagent reinforcement learning." IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 38.2 (2008): 156-172.
12. Tan, Ming. "Multiagent reinforcement learning: Independent vs. cooperative agents." Proc. of the 10th international conference on machine learning. 1993.
13. Watkins, Christopher John Cornish Hellaby. Learning from delayed rewards. Diss. King's College, Cambridge, 1989.
14. Tampuu, Ardi, et al. "Multiagent cooperation and competition with deep reinforcement learning." PloS one 12.4 (2017): e0172395.
15. Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529.
16. Foerster, Jakob N., et al. "Counterfactual multiagent policy gradients." Thirty-Second AAAI Conference on Artificial Intelligence. 2018.
17. Foerster, Jakob, et al. "Stabilising experience replay for deep multiagent reinforcement learning." Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017.
18. Jones, Chris, and Maja J. Mataric. "Adaptive division of labor in large-scale minimalist multi-robot systems." Intelligent Robots and Systems, 2003.
19. Groß, Roderich, et al. "Autonomous self-assembly in swarm-bots." IEEE transactions on robotics 22.6 (2006): 1115-1130.
20. Wang, Y, and C W. De Silva. "Multi-robot box-pushing: Single-agent q-learning vs. team q-learning." 2006 IEEE/RSJ Int Conf on Int. Rob & Sys.
21. Liu, X, and Jin Y. "Design of Transfer Reinforcement Learning Mechanisms for Autonomous Collision Avoidance." *DCC'18*, Springer, Cham, 2018.