PCRL: Priority Convention Reinforcement Learning for Microscopically Sequencable Multi-agent Problems

Anonymous Author(s)
Affiliation
Address
email

Abstract

Reinforcement learning (RL) has played an important role in tackling the decision problems emerging from agent (robotic) fields. However, RL still has challenges in tackling multi-agent large-discrete-action-space (LDAS) problems, possibly resulting from large agent numbers. At each decision step, a multi-agent LDAS problem is often faced with an unaffordable number of candidate actions. Existing work mainly tackled these challenges utilizing indirect approaches such as continuation relaxation and sub-sampling, which may lack solution quality guarantees from continuation to discretization. In this work, we propose to embed agreed priority conventions into reinforcement learning (PCRL) to directly tackle the microscopically sequencable multi-agent LDAS problems. Priority conventions include position-based agent priority to break symmetries and prescribed action priority to break ties. In a microscopically sequencable multi-agent problem, the centralized planner, at each decision step of the whole system, generates an action vector (each component of the vector is for an agent and is generated in a micro-step) by considering the conventions. The action vector is generated sequentially when microscopically viewed, and such generation will not miss the optimal action vector, and can help exploitation around the lexicographic-smallest optimal action vector. Proper learning schemes and action-selection schemes have been designed to make the embedding reality. The effectiveness and superiority of PCRL have been validated by experiments on multi-agent applications, including the multi-robot complete coverage planning application (involving up to $4^{18} > 6.8 \times 10^{10}$ candidate actions at each decision step) and the cooperative pong game (state-based and pixel-based, respectively), showing PCRL's LADS dealing ability and high optimality-finding ability than the joint-action RL methods and heuristic algorithms.

1 Introduction

Agent (robot) systems[1] and reinforcement learning have experienced great progresses these years [1], and multi-agent (multi-robot) systems have been applied to various civil (see Fig. 1), military and entertainment applications [2, 3]. With the applications becoming broader and deeper, the agent number in a multi-agent system has become larger and larger [4, 5], which results in the large discrete action space (LDAS) problem. A multi-agent problem with even a dozen of agents is hard to be solved for its explosive action combinations [6].

[1] We will use agent and robot interchangeably through this paper though they have certain differences in the real world.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

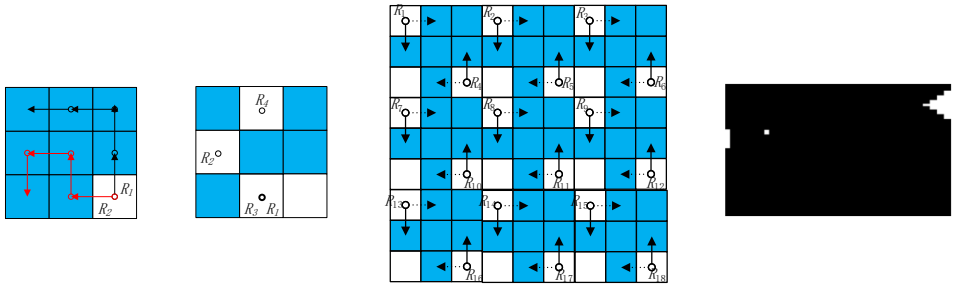

Figure 1: Examples of microscopically sequencable multi-agent applications: (a)∼(c) for multi-robot complete coverage path planning (CCPP), cleaning robots cooperate to visit the shaded cells in the fewest steps (min-max); (d) for cooperative pong (CP). (a) A smaller-scale situation needs at least 4 steps for the two-robot system to cover all uncovered cells. The action vectors (left,up) and (up,left) for R_1, R_2 will be tie optimal at current state, but (up,left) is the smallest lexicographic optimal action vector. (b) The priorities (can be relative-position determined) for the agents from high to low, at this state, are $R_4 \prec R_2 \prec R_1 \prec R_3$. The state representation includes map state and agent state, and can be "101011100010100020" where the '2' means 2 agents. (c) At this decision step, the system has $|\mathcal{A}^N| = 4^{18} > 6.8 \times 10^{10}$ feasible action vectors ($|\mathcal{A}|$ is the action set size for one agent and N is the number of agents), where the solid and dashed arrows are two tie optimal action vectors among many others. The PCRL will learn to select certain optimal action vector regarding priority conventions. (d) Cooperative Pong (detailed descriptions see experiment section). CP can be established as a microscopically sequencable problem.

To tackle the multi-agent LDAS problems, some researchers resorted to independent reinforcement learning (RL) regardless of other agents, but that will form loose cooperation and yield sub-optimal solutions; some researchers resorted to distributed reinforcement learning and communication to cooperate , but that lacks a global knowledge and is hard to design cooperation mechanisms [7]; many other researchers [8, 9] turned to centralized RL for high performance. For a centralized RL method, a trivial joint-action representation can be applied to a small-scale multi-agent system. However, when the agent number increases to a dozen, it becomes unaffordable because the action space is exponential to the number of agents (see Fig. 1(a)∼(c)). Previous centralized RL [8] has tried relaxing large discrete action space to continuous action space and generating a few discrete samples near the optimal continuous action vector to find the locally optimal discrete action vector. However, this subspace may miss the optimal discrete action vector leading to sub-optimal, and lack a solution quality guarantee from continuation to discretization.

Therefore, it remains challenging for a centralized RL to effectively and efficiently tackle the multi-agent LDAS problems, difficult in representing the actions and finding the optimal action from 10^{10} or more candidate actions. Moreover, a multi-agent LDAS problem often has many tie optimal actions; see Fig. 1 for an example. With ties like this, RL for LDAS is also difficult to train because of multiple return-maximum peak landscapes, so new action selection and training schemes are required to avoid divergence to multiple peaks.

We considered the microscopically sequencable multi-agent problems (including LDAS and non-LDAS) like in Fig. 1, and tackled them by borrowing inspirations from social conventions (see Fig. 2(b)(c) for more details) of human society. "**microscopically sequencable**" means that at each decision step of the whole system, the planner (can be set to R_1) selects out from $\mathcal{A}^N$ an action vector $\vec{a}$ (*macroscopically*-viewed), but each component of $\vec{a}$ **can be** sequentially selected out (**microscopically**-viewed) according to convention without losing optimality. That is, the planner inputs the map state and the positions of the agents, and outputs a minimum lexicographic $\vec{a}$ based on the agreed priority conventions. This decision process is like a Stackelberg leadership model [10]:

The decision flow at a decision step has been shown in Fig. 2(c) and can be expressed by natural language (formal description in later paragraphs) as follows. For the highest priority agent, the planner chooses one action (from $\mathcal{A}$) that the planner has learned to produce the largest Q value for the whole system, with the following agents optimally supporting the highest priority agent. And if the first agent has tie actions, the action priority will be obeyed. The planner refers to the first agent's action and then chooses the action for the second agent that will bring about the largest Q value for the whole system. The procedure continues until the last agents.

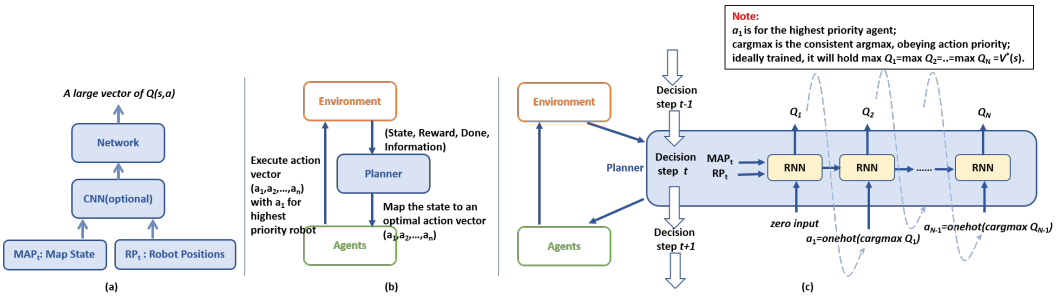

Figure 2: The diagram explanation of methods. (a) trivial joint-action RL for multi-agent LDAS problems, the length of Q will be $|\mathcal{A}|^N$. (b) *macroscopic* view of PCRL. The planner maps the current state (including map state + agent state) to an optimal action vector whose first component is for the highest priority agent and so forth. (c) **microscopic** view of PCRL: at this decision step (Decision step t), the planner outputs in N micro-steps the best action for the highest priority agent and then sequentially for other agents. The length of Q_i is $|\mathcal{A}|$.

The agreed priority conventions are prescribed before the learning starts and mainly consist of (1) **Agent priority convention**: Dynamically, at each decision step of the whole system, the agents' priorities are determined by current states, say, the relative positions of agents. For example, in Fig. 1(c), the agent's priorities can be determined from left to right and upper to lower position, so the priorities of robots are $R_1 \prec R_2 ... \prec R_{16}$. Here, $x \prec y$ means x has a higher priority than y. (2) **Agent ID convention**: Statically assign each agent a unique ID before learning to identify itself from others. When at least two agents coincide in the same position during the application, the smaller id agent has a higher priority. (3) **Action priority convention**: The actions of an agent have prescribed priority, say, up action$\prec$left action$\prec$down action$\prec$right action, to break action ties if two actions both will bring about optimality, and to converge to one minimum lexicographic action vector. Hereafter, we define that two action vector $\vec{\alpha} \prec \vec{\beta}$ (or called "$\vec{\alpha}$ is lexicographic smaller than $\vec{\beta}$") if and only if $\exists i \in [1, N],\ \alpha_i \prec \beta_i$ and $\forall j \in [1, i-1],\ \alpha_j = \beta_j$.

The contribution of this paper lies in (1) Priority conventions and proofs: this paper proposes the concept of "microscopically sequencable" to tackle LDAS multi-agent problems without missing the optimality, and can help exploitation around the lexicographic-smallest optimal action vector. (2) New schemes: it proposes new learning schemes that fully exploit the agreed conventions, such as auxiliary equality constraint, and neural network action selection schemes for PCRL. (3) Proof-of-concept practices: applying PCRL to tackle seemingly different problems, including 10^{10} magnitude LDAS multi-robot path planning problems and the pixel cooperative task, and achieves 20% fewer steps and competitive performance.

2 Related Work

Nowadays, deep reinforcement learning has been successful in large state problems [1, 11]. However, it is still very challenging to deal with the LDAS problems, especially in the action representation and selection. We will summarize previous work on dealing with this.

For the action representation, previous work has followed several lines to circumvent the large-discrete-action-space obstructions:

One line is to parameterize actions into continuous spaces and then finds discrete actions near the optimal continuous action [12] because continuous actions are differentiable. While doing the approximation, an ability to generalize over the set of actions and sub-linear complexity relative to the action set's size are necessary to handle such tasks. For example, [8] exploited the k-nearest-neighbor to find the optimal policy $\pi^*(\mathrm{s}) \approx KNN(\arg\max_{\mathrm{a}\in\mathcal{D}=\mathcal{A}^N} Q(\mathrm{s}, \mathrm{a}))$ in a subspace near the optimal continuous actions. It has been validated in discrete cartpole problems, recommending systems and multi-agent systems. One step further, work [13] exploitedly utilized the action space structure and without assuming the structure is provided a priori. This line is likely to miss the optimal discrete solutions for inappropriate continuation and to lead to suboptimality. Also, it lacks a quality guarantee from continuation to discretization, which also has been encountered and studied in the randomized rounding theory of integer programming [14].

Another line is to factor or eliminate action spaces [15–17], using binary coding and coarse coding to eliminate actions that are not optimal with high probability. However, the probability is hard to model and eliminations are not always possible. Some researchers employed new factor and combination methods [18–20] for action representation, when the problem satisfies Individual-Global Maximization (IGM) Constraint. For example, QMIX has a good performance on real-time decision-making under partial observability conditions (e.g. SMAC), which can decompose the total Q-value function into each agents' local Q-value functions. However, QMIX is suitable for monotonic property multi-agent problems.

Action selection usually has deep entanglement with action representation. [21–23] uses DQN for action selection and has achieved success in Atari games and human-level control. [24] applied a recurrent neural network (RNN) to select actions for POMDP. [25] used bidirectional RNN to represent the actions and select actions, successfully dealing with the continuous multi-actions and have achieved success in SC2 problem and other problems. Some work resorted to social conventions for selecting actions but not for LDAS problems, for example, [26, 27]. And some work relied on sophisticated communications for exchanging selected action [28–30], which may be vulnerable to be impacted by unstable communication in practical cooperation/coordination.

Our way of action selection resembles Stackelberg model and the sequence-to-sequence paradigm (for example image-to-caption [31]). [32] also proposed seq-to-seq and on-policy policy gradient to tackle LDAS problems, along with others [33] but we offer a large agent number off-policy and optimality-preserved one. Meanfield reinforcement learning [5] also has LDAS dealing ability. It iteratively learns each agent's best response to the mean effect from its neighbors and is proved to converge to Nash Q-value. However, the Nash Q-value is not always the global optimal value in scenarios like the prisoner's dilemma.

3 Methodology

This section introduces the basics of PCRL; then followed by the mathematics and details of how to learn the agreed priority conventions in PCRL.

Our concern problems are markovian and can be established as a classic MDP $M =< S, A, R, P, \gamma >$, where the letters are as usually defined, and $A = \mathcal{A}^N$ where N is the agent number and $\mathcal{A}$ is the action set of one agent. PCRL is aimed to learn the optimal policies to maximize the accumulated discounted return of an MDP.

We treat the microscopically sequencable multi-agent problem as a discrete-time decision system. The planner (can be played by R_1) decides the optimal action vector at each decision step.Without losing generality, we suppose agents $R_1, R_2, ..., R_N$ are priority-ordered according to the agreed priority, where N is the total agent number. The action vector for agents is $\vec{a} = (a_1, a_2, ..., a_N)$ and each component of this action vector is assigned for R_1 to the corresponding agent and executed. The overall reward after agents' simultaneous execution of $\vec{a}$ is r.

Corresponding to the above basics and Fig. 2, the overall pseudocode of PCRL is in Algorithm 1. It is modified from the DQN [1], differing in aspects of action representation-selection, neural network architecture and learning schemes. We will explain the critical aspects of the algorithm in the following paragraphs.

State representation When *macroscopically* viewed as in Fig. 2(b), the planner (can be played by agent ID 1) maps current state s (consisting of current physical map state and robot state, i.e., $s_t = concat(MAP_t, RP_t)$) to an action vector:

$$\vec{a} = f(s), \tag{1}$$

where f is a trainable RNN-based function and is to be optimized. For cleanness, we will omit the subscription of t for the current state.

The state s can be convoluted features or simply the concatenation of raw map state and agent state. Using the CCPP (in Fig. 1) as an example, $s = S_{grid_map}|S_{robot_positions}$, where "|" means concatenation; grid_map can be onehot vector indicating whether each grid has been covered (i.e., visited), and robot_position are an integer vector with length equal to grid_map indicating the robot number upon each cell (the studied robots are homogeneous).

Algorithm 1 PCRL: Priority convention RL

Initialize replay memory D
Initialize RNN-based action-value function Q with random weights θ
Initialize target RNN-based action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
for episode=1 to E **do**
  Observe initial state s
  while unterminated **do**
    Select an action $\vec{a}$: with probability ε select a random action vector by uniform sampling; otherwise, generate $\vec{a}$ as Fig. 2 and (2) to (4)
    Carry out action $\vec{a}$
    Observe reward r and new state s'
    Store experience $< s, \vec{a}, r, s' >$ into replay memory D
    Sample random transitions $< ss, \overrightarrow{aa}, rr, ss' >$ from replay memory D
    Calculate target for each minibatch transition:
    if ss' is terminal state, then $tt = rr$
    otherwise $tt = rr + \gamma * \max_{\vec{a'}} Q\left(ss', \vec{a'}\right)$ where $\vec{a'}$ is generated as Fig. 2(c) and (2) to (4)
    Train the evaluation Q-network using $(tt - Q(ss, \overrightarrow{aa}))^2 + L_{aux}(ss, \overrightarrow{aa}, rr, ss')$ as loss with respect to θ, where L_{aux} is the auxiliary convergence loss in (11) and (13)
    Update target network as $\theta' = (1-\tau)\theta' + \tau\theta$
    $s = s'$
  end while
end for

For simplicity, the state s will be featured and supplied as the initial hidden state (other usages are also possible), i.e., the h and c of the RNN of the planner.

Action representation and selection If the action space is not large enough, then a joint-action will be the best. However, large action brings in intractability. The action in PCRL is represented using recurrent neural networks. When generating $\vec{a} = (a_1, a_2, ..., a_n)$, our method resembles the image-to-caption [31] technology in that when given a state s, PCRL produces a sequence of $a_i,\ i \in [1, N]$ with a_1 assigned to the highest priority agent, and this sequence aims to bring about the maximum return. Mathematically, the action for the highest priority agent is generated as below:

$$a_1 = \text{cargmax}\ RNN(\vec{0}, (h_0, c_0))), \tag{2}$$

where $h_0 = C_0$ is the featured and supplied state s. $\vec{0}$ means no agent has an action (can refer to Fig. 2(c)). cargmax (stands for consistent argmax) function means selecting the action for R_1 according to the agreed action priority convention if ties exist. $RNN()$ function is a one-step forward function of RNNs, which is with only $|\mathcal{A}|$ entries. For the $mathrmcargmax$ function, say, if $RNN()$ outputs a value vector of $(5.1, 5.32, 5.2, 5.32)$, then we consitently select the first "5.32" (index=1) in case of ties whereas trivial argmax in pytorch may return ungaurenteed and inconsitent index.

Remark 1: Using CCPP as an illustration. Denote $RNN(\vec{0}, (h_0, c_0)))$ as $\overrightarrow{Q_1}$, meaning the expected Q value for the whole system when agent 1 selects each action from $\mathcal{A}$, so $\overrightarrow{Q_1}[j],\ j \in |\mathcal{A}|$ tries to approximate

$$\max_{a_2 \times a_3 \cdots \times a_N \in A^{N-1}} Q(s, (a_1 = action_j, a_2, \cdots, a_N)), \tag{3}$$

which is possible since s contains all information of current state including the total number of agents, and the following agents reference to and fully cooperate $a_1 = action_j$. (2) can also be viewed from state mapping, i.e., $\overrightarrow{Q_1}$: (grid states, agent state, total=N, priority=1 (or highest)) $\rightarrow$ 4 real values $\rightarrow$ one index. This mapping can be learned by reinforcement learning.

Similarly, for the remaining agents, the planner conditions at previous agents' actions and the priority conventions as well, so

$$a_i = cargmax\ RNN(onehot(a_{i-1}), (h_{i-1}, c_{i-1})), \tag{4}$$

Remark 2: Assume now $i = 2$, then now this $RNN(.)$ is an approximation to

$$\max_{a_3 \times a_4 \cdots \times a_N \in A^{N-2}} Q(s, (a_1, a_2 = action\ j, a_2, \cdots, a_N)), \tag{5}$$

where a_1 has been determined at previous micro-step and conditioned at this micro-step.

The approximation in each micro-step has physical meaning. If ideally trained

$$\overrightarrow{Q_1^*} \triangleq RNN^*(s, \vec{0}),\ length(\overrightarrow{Q_1^*}) = |\mathcal{A}| \tag{6}$$

$$V^*(s) = \max_{a_1 \in \mathcal{A}} \overrightarrow{Q_1}^* \tag{7}$$

$$\overrightarrow{Q_2^*} \triangleq RNN^*(s, (\text{cargmax}\overrightarrow{Q_1})),\ length(\overrightarrow{Q_2^*}) = |\mathcal{A}| \tag{8}$$

$$V^*(s) = \max_{a_2 \in \mathcal{A}} \overrightarrow{Q_2^*} \tag{9}$$

$$\cdots\cdots \tag{10}$$

$$\max \overrightarrow{Q_1^*} = \max \overrightarrow{Q_2^*} = ... = \max \overrightarrow{Q_n^*}. \tag{11}$$

The last line is important because $V^*(s)$ links all the $\overrightarrow{Q_i^*}$ and can be exploited to achieve better learning.

Reward The reward for each decision step is the total reward that all robots earn. In Fig. 1(c), the reward is the number of uncovered grids that the whole system visits at this step. An uncovered grid will give a reward of 1 only for the first time it was visited. If $R_1 \sim R_{16}$ executed the solid arrow action, the reward would be 16. The total reward is beneficial for learning and regards the system as a monolithic system. Sometimes, a step penalty is added to the step reward so that the hypothesis reward in RL holds[2], i.e., maximizing the return equals to minimizing steps to cover all shaded cells.

Training As Fig. 2 and Algorithm 1 depict, the training strategy needs specially designation. When in state s and takes action $\vec{a}$, it will transfer to some s' and get a reward r, then $Q(s, \vec{a})$ approximates to the target Q_{target},

$$Q_{target} = r + \gamma \max_{\vec{a'}} Q'(s', \vec{a'}) \tag{12}$$

where $\vec{a'}$ can be selected out on the target network according to (2) to (4), rather enumerating all $\mathcal{A}^N$ action combinations.

The $Q(s, \vec{a}) = Q(s, (a_1, a_2, ..., a_N)$ can be calculated by feeding $\vec{0}$, $onehot(a_1)$, ..., $onehot(a_{N-1})$ to the RNN in N micro-steps and gathering the a_N-th value in $\overrightarrow{Q_N}$ (see Fig. 2(c)).

Moreover, considering the uniqueness of our PCRL decision flow and (11), we can use an auxiliary loss to accelerate and smoothen the training. The auxiliary loss is equalities among agents' Q_i values (as explained in (11) and that paragraph), i.e.,

$$L_{aux}(ss, \overleftrightarrow{aa}, rr, ss) = (\max Q(ss, \vec{0}) - Q_{target})^2 + (\max Q(ss, onehot(aa[1])) - Q_{target})^2 + \cdots + (\max Q(ss, onehot(aa[N-1])) - Q_{target})^2, \tag{13}$$

where ss and $\overrightarrow{aa}$ are batch data from the experience replay buffer. L_{aux} guides the mean-squared error of Qs of each agents to target Q_{target} of ss' and rr. Other type of (**??**) is also possible.

We conjecture the following theorem and the proofs are in the appendix.

Theorem 1. *The action representation scheme and selection process will not lose the optimality of the action space and can select out the lexicographic-smallest optimal action vector.*

Theorem 2. *The action selection process breaks the exponential action space into linearly expressible space complexity.*

[2] http://incompleteideas.net/rlai.cs.ualberta.ca/RLAI/rewardhypothesis.html

Table 1: CCPP illustration of PCRL and consistent argmax

State	Q value at two micro-steps after sufficiently trained	Converged Optimal action	Reward	Explaination
1 1 1 1 1 1 1 1 R,R	[[5.7079, 5.5537, 4.9969, 5.1314], [4.9332, 5.7386, 5.1010, 5.0878]]	(up, left)	2+(-0.5)=1.5 (number of newly covered cells plus the step penalty)	Initital state. The highest priority robot has tie optimal actions (up or left). It selects lexicographic smaller "up". The second robot selects left can get the whole system largest Q.
1 1 1 1 1 R 1 R 0	[[4.2621, 3.5443, 3.3271, 3.6442], [4.2466, 3.9317, 4.1653, 3.4481]]	(up, up)	2+(-0.5)=1.5	At this step, the second priority robot has near up or left action and "up" is selected.
1 1 R 1 R 0 1 0 0	[[2.1108, 2.7424, 1.9205, 2.1509], [2.0065, 2.7412, 2.4293, 2.0047]]	(left, left)	2+(-0.5)=1.5	This action vector is far beyond other action vector.
1 R 0 R 0 0 1 0 0	[[0.6524, 1.5007, 0.3517, 0.4291], [0.5143, 0.9889, 1.5010, 0.4947]]	(left, down)	2+(-0.5)=1.5 ($V^*(s)$ value quite close to this reward)	After executing (left, down), mission completed (done).

4 Experiments and Discussions

We evaluate the performance of PCRL on two seemingly different microscopically sequencable multi-agent applications: multi-robot complete coverage path planning (CCPP) problem and Cooperative Pong (CP). We compared our method with joint-action RL or heuristics on their efficacy and efficiency. All experiments were carried out on a computer of amd ryzen 9 5950x, with a single 3090 RTX GPU. Every method has run 10 times with different random seeds at each experiment setting and reports the averages and the standard deviations (STD), to gain the reliability of the evaluation. The hyperparameters are listed in appendix, also in the source code that is supplemented.

Our experiments consist of (1) The LDAS dealing ability and optimality-finding ability of PCRL on CCPP, compared to joint-action DQN. (2) the optimality-finding ability of PCRL on CCPP, compared to the heuristic algorithm. (3) the optimality-finding and end-to-end ability of PCRL on CP, compared to joint-action DQN.

4.1 Multi-robot complete coverage planning

PCRL illustration and consistent argmax Table 1 depicts the sufficiently trained (for 1 hour) and converged results of the two-robot complete coverage path planning example on 3×3 gridworld. The first column is the current state, where "1" stands for shaded grid cell and R stands for a robot. The robots are homogeneous and the priorities are position-determined, so we ignore the ID of them. The second column shows the Q values of the RNN in two micro-steps. The first micro-step has the Q value for the highest priority agent. It is the expectation for the first agent to optimally choose an action and the following robot to choose its corresponding action. And the second micro-step is the second robot's Q value referring to the first robot's action. As can be seen, the two value has a near Q value, which is almost the $V^*(s)$ (The ground truth $V^*(s)$ can be easily calculated). The third column is the optimal action for each agent, obeying to the priority. The fourth column is the reward for the whole two-robot system. We can see that the total return received in the first state is six, which is eight plus the penalty multiply minimal steps. The last line column is some explanation for data interpretation. Due to cargmax function, this RNN has converged to the smallest lexicographic action vector at each state.

PCRL and DQN in deterministic transition environment CCPP We compared the final performance of PCRL, joint-action DQN on grid world on various numbers of robots. The details of the results are shown in the Fig. 3.

Fig. 3(left) shows that when trained enough steps, PCRL can get a close final performance to joint-action DQN. That means the PCRL network can successfully learn the conventions and the proper Q values, and has an ability simlilar to the monothetic system. Moreover, PCRL has the potential to deal with over 10^{10} actions: the needed parameters of DQN is exponentially increasing while PCRL have a linearly increasing number of parameters, about 562176 for 12 robots (see the middle of Fig. 3). The performance of DQN is slightly better than PCRL, however the performance decreases quite rapidly and soon fails to tackle 12 or more robots scenarios (see the right of Fig. 3, where the averaged return of PCRL is normalized as 100%).

Large space and non-deterministic transition environment The Fig. 4 depicts 18 robots on a 9×9 grid world the same as Fig. 1(c), with 54 uncovered cells in total. This case, at each step, owns over 10^{10} action vector candidates. The experiment is further made tougher to a non-deterministic

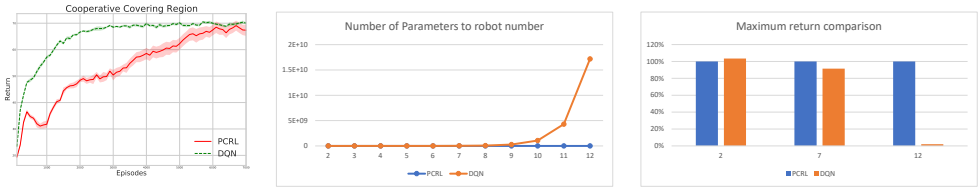

Figure 3: Comparisons of PCRL and joint-action DQN. left: performance on CCPP of 2 agents and 10×10 map size with 99 cells except the down-rightmost to visit in deterministic transition environment; middle: the total number of needed parameters; right: performance of DQN dives comparing to PCRL's.

(stochastic) environment. In the non-deterministic transition environment experiment, there is a non-zero probability p of "leftwards slipping" when agents take actions (say agent takes "up" then it has p probability being moved to the "left" cell), where agents' do not have any prior knowledge about the environment transition. Fig. 4 shows the averaged score and averaged steps with sliding-window width 100 in the training episode. As shown, the training is converging gradually and smoothly. The averaged steps is approaching 20 and the averaged return is about 44.

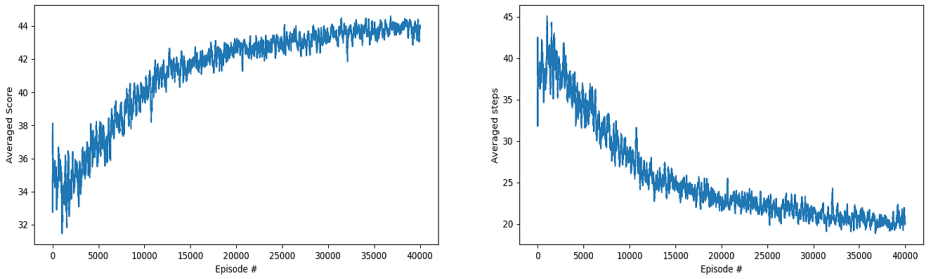

Figure 4: The training of 18 robots in stochastic environment, where the joint-action DQN is unable to tackle.

To further compare the steps, which is the main focus of CCPP, we employ a heuristic search algorithm where the agent selects the minimum Manhattan distance cell and goes towards the cell. The Violin plot is depicted in Fig. 5, where the first "vilion" is the performance of RNN in mid-course of training, while the second one is of the final RNN after training. The third one is for the heuristic algorithm. PCRL can learn to optimize and use 20% smaller steps (19.7VS24.5) to cover all the shaded cells, and is more stable.

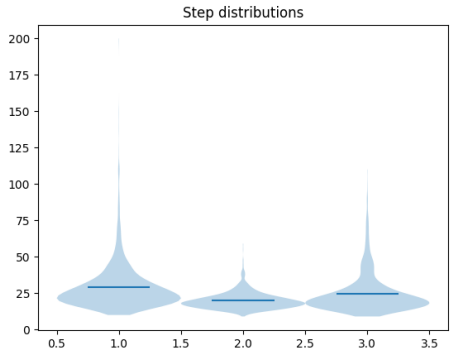

Figure 5: 18 robot performance of PCRL (mid-course and final) compared to heuristic. Mean/Stdev of steps are 28.9/16, 19.7/5.6, 24.5/13.3, respectively.

4.2 Cooperative pong

Cooperative pong Fig. 1(d) is a game where the objective is to keep the ball in play for the longest time. The game is over when the ball goes out of bounds from either the left or right edge of the screen. All collisions of the ball are elastic. Cooperative Pong has 480 pixels $\times$ 280 pixels, so a ball with speed 9 will cost about 60 steps to go from the left paddle to the right paddle. Algorithms that can keep the ball for 400 steps are rare [34]. The CP can be established as a joint-action DQN with and a microscopically sequencable multi-agent problem. To show the applicability of PCRL in CP task, we applied PCRL on the state-based and pixel-based CP, respectively.

State-based of CP We trained PCRL and the joint-action DQN on CP with a similar scale of network parameters, and the performance is illustrated in Fig. 6a. Set ball_speed=9 pixelsstep, left_paddle_speed=28, right_paddle_speed=28, cake_paddle=False, max_cycles=900, bounce_randomness=False, each frame reward=1, out boundary reward=-100. We can see that PCRL even outperforms the joint-action DQN which treats the two agents as a monothetic agent. The key reason for this result may be that PCRL has higher parameters efficiency than DQN of a similar scale of parameters.

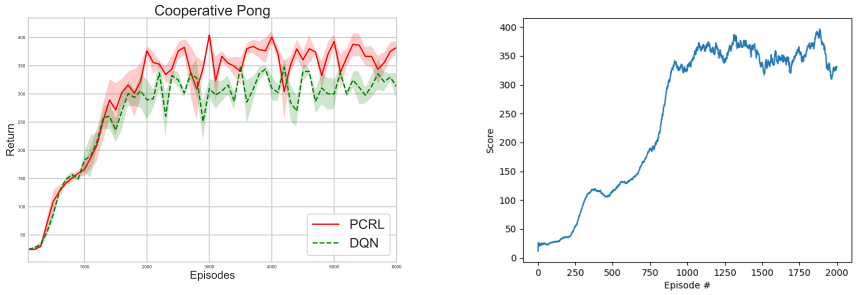

Figure 6: Convergence of Q_i. a: Training performance of state-based cooperative pong. b: training of pixel-based CP (playing video attached)

Pixel-based CP To better demonstrate the effectiveness of our approach, we also use the pixel states as the Q-net inputs. We resize the original pixel state to 84 * 84 * 3 tensor. We use a 3-layer convolution network transforming the pixel state to a vector state first, and then feed the vector state as the RNN hidden state. The performance in training with pixel states (in about four hours) is shown in Fig. 6b, from which we can see that our algorithm converges after about 1000 episodes and get a satisfactory performance. It indicates that PCRL is effective and efficient even in the end-to-end setting without losing much performance compared to the state-based pong performance.

5 Conclusion

This paper tackled the microscopically sequencable multi-agent problems via reinforcement learning that combines agreed priority conventions. The agreed priority conventions can successfully break the exponential scale actions into a sequence of actions (linear scale) without losing optimality. From the multi-robot complete coverage planning and cooperative pong experiments, the algorithm shows closeness or superiority in effectiveness and efficiency to the joint-action DQN and heuristics. The results suggest the potentials of PCRL for a larger number of multi-agent systems (say recommending systems) or swarm robots.

In the future, further work can be done, such as (1) larger agent number multi-agent systems to verify the ability of PCRL for larger action space problems, say swarm robots. (2) application of this algorithm to fully-distributed or centralized-training-and-decentralized-execution multi-agent systems. (3) studies of PCRL for heterogeneous agents.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[2] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

[3] Jing Zhang and Dacheng Tao. Empowering things with intelligence: a survey of the progress, challenges, and opportunities in artificial intelligence of things. *IEEE Internet of Things Journal*, 8(10):7789–7817, 2020.

[4] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *2012 IEEE International Conference on Robotics and Automation*, pages 3293–3298. IEEE, 2012.

[5] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5571–5580. PMLR, 2018.

[6] Wei Wu, Fengchun Yang, Fuhui Zhou, Han Hu, Qihui Wu, and Rose Qingyang Hu. Intelligent resource allocations for irs-assisted ofdm communications: A hybrid mdqn-ddpg approach. *arXiv preprint arXiv:2202.05017*, 2022.

[7] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. A survey on distributed machine learning. *ACM Computing Surveys (CSUR)*, 53(2):1–33, 2020.

[8] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

[9] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

[10] Heinrich Von Stackelberg. *Market structure and equilibrium*. Springer Science & Business Media, 2010.

[11] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[12] Jason Pazis and Ronald Parr. Generalized value functions for large action sets. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 1185–1192, 2011.

[13] Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip Thomas. Learning action representations for reinforcement learning. In *International Conference on Machine Learning*, pages 941–950. PMLR, 2019.

[14] Prabhakar Raghavan and Clark D Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[15] Brian Sallans and Geoffrey E Hinton. Reinforcement learning with factored states and actions. *The Journal of Machine Learning Research*, 5:1063–1088, 2004.

[16] Hajime Kimura. Reinforcement learning in multi-dimensional state-action space using random rectangular coarse coding and gibbs sampling. In *SICE Annual Conference 2007*, pages 2754–2761. IEEE, 2007.

[17] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. Learn what not to learn: action elimination with deep reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 3566–3577, 2018.

[18] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

[19] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. In *International Conference on Learning Representations*, 2020.

[20] Tonghan Wang, Heng Dong, Victor Lesser, and Chongjie Zhang. Roma: Multi-agent reinforcement learning with emergent roles. In *International Conference on Machine Learning*, pages 9876–9886. PMLR, 2020.

[21] Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov, and Anastasiia Ignateva. Deep attention recurrent q-network. *arXiv preprint arXiv:1512.01693*, 2015.

[22] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

[23] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[24] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[25] Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2, 2017.

[26] Adam Lerer and Alexander Peysakhovich. Learning social conventions in markov games. *arXiv preprint arXiv:1806.10071*, 2018.

[27] Bingcai Chen, Chao Yu, Qishuai Diao, Rui Liu, and Yuliang Wang. Social or individual learning? an aggregated solution for coordination in multiagent systems. *Journal of Systems Science and Systems Engineering*, 27(2):180–200, 2018.

[28] Jelle R Kok, Matthijs TJ Spaan, Nikos Vlassis, et al. An approach to noncommunicative multiagent coordination in continuous domains. In *Benelearn*, pages 46–52, 2002.

[29] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

[30] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[31] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[32] Yiming Zhang, Quan Ho Vuong, Kenny Song, Xiao Yue Gong, and Keith W Ross. Efficient entropy for policy gradient with multi-dimensional action space. In *6th International Conference on Learning Representations, ICLR 2018*, 2018.

[33] Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017.

[34] Justin K Terry, Benjamin Black, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Clemens Dieffendahl, Niall L Williams, Yashas Lokesh, Caroline Horsch, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See the abstract and introduction. Moreover, The abstract, introduction and experiment results are consistent.
   (b) Did you describe the limitations of your work? [Yes] See the conclusion part.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] The technique is neutral.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] Our paper is strictly comforms to them.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In the article, appendix and codes.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Each ran at least ten times.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In an ordinary desktop computer.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [No]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [No]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [No]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [No]

A Appendix

A.1 Microscopic generation proofs

Theorem: *The action representation scheme and selection process will not lose the optimality of the action space and can select out the lexicographic-smallest optimal action vector.*

Proof. Suppose $\vec{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_N)$ and $\vec{\beta} = (\beta_1, \beta_2, ..., \beta_N)$ are two (More than two tie optimal action vectors can also hold similar conclusion.) optimal tie actions for a state s, and in the article has deduced that $V^*(s) = Q(s, \vec{\alpha}) = Q(s, \vec{\beta})$.

Since the state s is the same, i.e., the map and robot states are the same, thus the agent priority orders are identical. Suppose $\vec{\alpha}$ is lexicographic smaller than $\vec{\beta}$ (this is possible, since each component of $\vec{\alpha}$ and $\vec{\beta}$ are comparable according to agreed action priority, so either $\vec{\alpha}$ is lexicographic smaller than $\vec{\beta}$ or $\vec{\beta}$ is lexicographic smaller than $\vec{\alpha}$. The latter can prove the following similarily).

Let $j = min_i(\alpha_i \prec \beta_i)$, the planner's procedure will select α_i instead of β_i for agent i.

Since the smallest-lexicographic action vector can be selected out and learned, PCRL will not lose the optimality for the concerned problem. □

Theorem: *The action selection process breaks the exponential action space into linearly expressible space complexity.*

Proof. The recurrent neural network has N micro-step outputs, each micro-step output consists of $|A|$ actions. Suppose that at each micro-step, the planner can select one action, so the possible total action combination space is still $|A|^N$, so RNN will not miss expressing the optimal vector. However, the optimum (maximum) action vector can be expressed by and selected from $|\mathcal{A}|$ per micro-step $\times$ N micro-steps. □

A.2 Common hyper-parameters

The common hyper-parameters for experiments are:

Table 2: Add caption

Tasks	CCPP	CP
learning rate	1.50E-04	1.50E-04
replay buffer size	2.00E+05	2.00E+05
batchsize	128	128
episode length	200	900
γ	1.0 with step penalty 0.5	0.995
τ	1.00E-03	1.00E-03
k(weight update frequency)	4	4
episode number	min(40000,H * W * agent_nums * 50)	2000
epsilon_start	1	1
epsilon_end	0.01	0.05
epsilon_decay	0.99992	0.95
LSTM hidden size	1024	512
auxiliary loss	type 1	type 2

The auxiliary loss type 1 is as (11), and type 2 are the mean squared error of $\max Q_1$ to $\max Q_2$, $\max Q_2$ to $\max Q_3$ etc. This two can get similar results.

The γ of CCPP is set as listed because of the reward hypothesis.

Reinforcement learning is based on the reward hypothesis, that is, RL is to maximize $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$. When $t = 0$, it is maximizing $G = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = R_1 + \gamma * r_2 + \gamma^2 * r_3 + ...$, i.e., the maximum expected return from the initial setting of the environment.

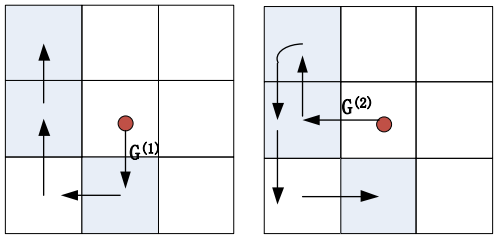

Figure 7: Example diagram of maximum reward hypothesis and shortest path

If one wants to maximize the reward G to get the fewest step coverage route of the robot, then one needs to set appropriate parameters such as γ so that G fits the aim. Generally, $0 \leq \gamma \leq 1$, when γ is 0, it is a single-step reflective and myopic behavior. In this appendix, we want to maximize the reward G to obtain the fewest steps of multi-robot complete coverage. Because the multi-robot is treated as a monothetic system, so one robot's proofs can naturally be extended to a multi-robot system. Through mathematical analysis, the following theorems and proofs can be obtained:

Theorem 3. *When $\gamma < 1$ and there is no single-step penalty for each step, maximizing the return does not always get the fewest step complete coverage. When $\gamma = 1$ and each step has a single-step penalty (step_penalty<0), maximizing the return is equivalent to the fewest step complete coverage.*

Before the proof, use Fig. 7 to illustrate the first sentence when $\gamma < 1$. The figure shows the same environment configuration (setting), including the same map state and robot position. The two paths of the robot are path (1) and path (2). Denote their rewards are respectively $G^{(1)}, G^{(2)}$. It can be seen in the figure that path (1) is shorter than path (2). However, when $\gamma < 1$ and each step has no penalty, the return of path (2) is larger, that is, $G^{(1)} < G^{(2)}$: According to the return formula, the return for paths (1) and (2) is $G^{(1)} = 1 + \gamma^2 + \gamma^3, G^{(2)} = 1 + \gamma^1 + \gamma^4$, at this time

$$G^{(1)} - G^{(2)} = (1 + \gamma^2 + \gamma^3) - (1 + \gamma^1 + \gamma^4) \tag{14}$$
$$= \gamma^2 + \gamma^3 - \gamma^1 - \gamma^4 \tag{15}$$
$$= (\gamma^2 - \gamma^1) - \gamma^2(\gamma^2 - \gamma^1) \tag{16}$$
$$= (\gamma^2 - \gamma^1)(1 - \gamma^2) < 0 \tag{17}$$

The last step is because when $0 < \gamma < 1$, $\gamma^2 < \gamma^1$ and $1 > \gamma^2$.

However, When $\gamma = 1$ and a single-step penalty $step_penalty = -0.2$, the return of path (1) is 2.2, and the return of path (2) is 2.0. This fits the requirements of CCPP: fewest steps and maximum return.

From the illustration of the figure, a mathematical proof of the theorem 3 can be obtained.

Proof. Because there are examples shown in Fig. 7, the first sentence of the theorem can be proved by contradiction. For the second sentence of the theorem, it can be equivalent to "When $\gamma = 1$, and each step has a single-step penalty (step_penalty<0), a greater return is equivalent to a fewer step complete coverage path".

The idea of the proof is to assume that there are path (1) and path (2), and the number of steps used are n and m, respectively. Assume $n < m, n > 0, m > 0$ and $step_penalty < 0$.

First to prove the necessity, that is, under the same environment configuration, fewer steps will get a greater return, that is, one needs to prove $G^{(1)} > G^{(2)}$. Denote C_i the cell at step i and $C_i = 1$ if the cell has never been visited.

Because $G^{(1)} = (C_1 + step_penalty) * \gamma^0 + (C_2 + step_penalty) * \gamma^1 + ... + (C_n + step_penalty) * \gamma^{n-1} = (C_1 + step_penalty) + (C_2 + step_penalty) + ... + (C_n + step_penalty) = (C_1 + C_2 + ... + C_n) + n * step_penalty = number_of_uncovered_cells + n * step_penalty$.

Similarly, $G^{(2)} = (C_1 + C_2 + ... + C_m) + m * step_penalty = number_of_uncovered_cells + m * step_penalty$. Due to the penalty of each step, we get $G^{(1)} > G(2)$, that is, a smaller number of steps gets a higher reward.

Proof of adequacy, that is, greater returns can lead to fewer steps. Because the rewards of a scene can be written as $number_of_uncovered_cells + k * step_penalty$, where k is the number of steps visiting all shaded cells, a larger reward corresponds to a smaller number of steps. □

A.3 Planning difficulty for CCPP

CCPP can be established as a Euclidean Traveling Salesman Problem which is NP-hard. Here we establish it as an MDP. To realize the difficulty of the problem so as to set a proper problem scale, we first conducted a robot random wandering experiment to cover all grids and the experimental results are shown in Fig. 8: the x-axis represents the n of the square $n \times n$ gridworld, where the y-axis is the expected steps (averaged among 1000 runs) for a robot to visit all $n^2 - 1$ cells (the start cell is in down-rightmost and is empty). It can be seen that the number of steps required increases almost exponentially (the log linearly interpolation curve has been fit and has been plotted in the same figure). When n is 10, the number of random steps to completely cover all cells is more than 1000 steps, so we set it as the scale of the experiments. Roughly estimated, the total number of paths for a problem scale 10×10 map size and 2 agents can reach 100! (about 10^{158}). So, this pre-experiment has partitially shown the complexity of problems that PCRL can tackle and have strong optimization ability.

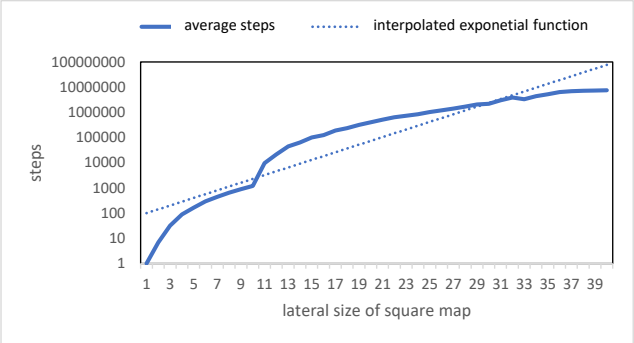

Figure 8: Expected steps to cover all cells with random wandering

A.4 Deterministic transition environment of 18 robots CCPP

Fig. 9 shows the settings of Fig. 1(c) and with a deterministic transition environment. It shows the return and steps of each training episode, the training cost and the sliding-window averaged steps. The ground truth return is 52.5 and PCRL sometimes can get 51.5 and an average near 50. This picture again shows the LDAS dealing ability and high optimality-finding ability of PCRL.

Gap of max Q_i in state-based CP As shown in (11), the maximum of $\overrightarrow{Q_1}$ will equal to the maximum of $\overrightarrow{Q_2}$ at the state s, theoretically. Respecting this, we evaluate our trained neural network to analyze the convergence of Q_is. Define the gap as $(\max \overrightarrow{Q_2} - \max \overrightarrow{Q_1}) / \max \overrightarrow{Q_1}$. The left part of Fig. 10 shows the gap at every state along one test episode, while the right part shows the averaged gap in the training history of an RNN. The former demonstrates an only 5% gap in the states. The latter demonstrates that as training goes, the gap from substantially large becomes narrow and narrow. Both parts proved the convergence of gaps between $\overrightarrow{Q_i}, i = 1, ..., N$.

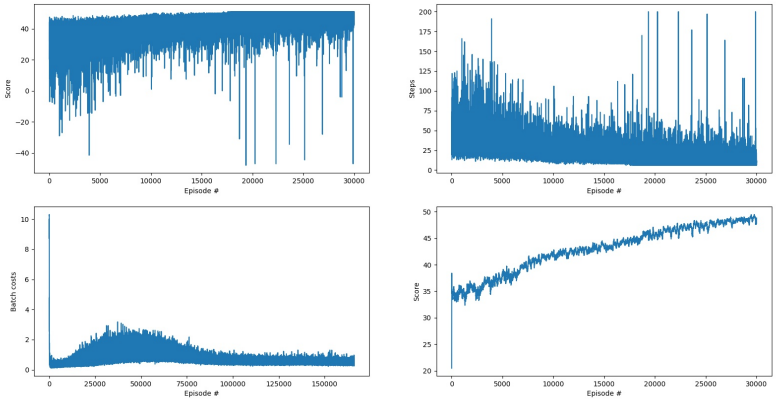

Figure 9: The training of 18 robots in deterministic environment.

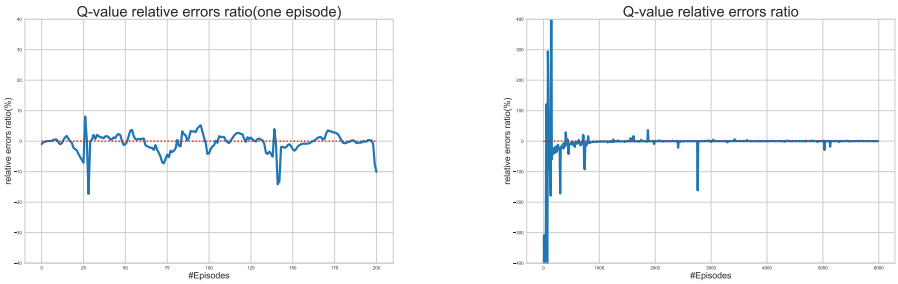

Figure 10: Convergence of Q_i. Left: The gap along a test episode. Right: The evolution of gap in training stage.