Nested Markov Decision Process

Shuo Jiang* and Jinkun Zhang*

*Northeastern University

November 4, 2020

1 Introduction

There are many scenarios require multiple agents engagement. Though they share the same goal for cooperative task (e.g. teammates in soccer game), such task can be divided into different roles (striker, fullback, goalkeeper) and each role focuses a specific aspect of the whole task.

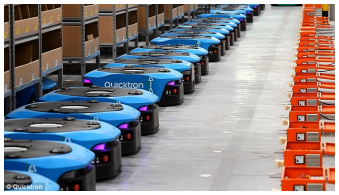
Previous work [18] has discussed the idea of different level of coordination as:

tight coordination: We say that robot A coordinates with robot B if it considers the state of B when selecting its own actions. Further, we say that this coordination is tight if A considers B's state at a high frequency throughout execution.

planned coordination: We say that a robot A plans its coordination with robot B if at some time t it anticipates the interactions it will have with B at a later time t' .

Apparently, tight coordination would take place more locally and frequently, usually several agents are working a a specific task that requires constant cooperation. Planned coordination happens occasionally, and usually between

(a) Amazon Kiva warehouse management system



(b) Alibaba Quicktron warehouse management system

(a) AutoStore warehouse management system

(b) TERMES robot construction system

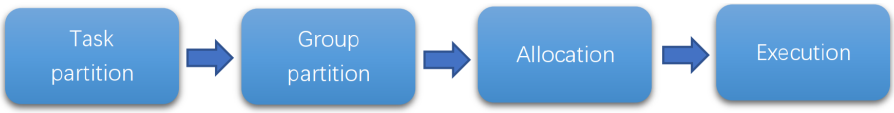


Figure 3: Decision making

groups that the execution of different groups interfere each other that requires high level planning to solve.

Usually, the way for a team to make decision goes through the following steps:

(1) **Task partition**: The team has a consensus of how a task can be partitioned into (quasi) uncorrelated sub-tasks. (Group partition)

(2) **Group partition**: The team are partitioned into groups that each group is responsible for a certain sub-task. (partition in agent space). Usually in such phase, local communication is essential.

(3) **Allocation**: Find a matching between the set of sub-tasks and groups. Usually in such phase, global communication (inter-group) is essential.

(4) **Execution**: Each group execute its own sub-task, and only try to maximize the reward in sub-task. In this phase, other groups are only seen as environment dynamics. May not or only need week coordination.

2 Literature Review

2.1 Multi-agent Reinforcement Learning

Decentralization
Scalability
Coordination

DecDQN [31]
COMA [13]
MADDPG [25]
QMIX [35]
VDN [41]
CM3 [50]
Attention based [1, 16, 17]
Graph based [44]
Mean Field based [51]

Multi-agent Reinforcement Learning suffers from severe dimension explosion problem. Previous works is limited in coordination of several agents (most less than 10). Methods have been developed to extend scalability of MARL only works in homogeneous agent setting and with nearly reactive behavior.

2.2 Coordination

[18] discussed a multi-agent system performs in different level of coordination. Each agent generates a local plan and broadcast to others, and re-plan based on received plans, such recursive planning iteratively incorporates intentions of teammates.

2.3 Hierarchical Multi-agent Reinforcement Learning

Most of previous works in hierarchical multi-agent learning is under option framework, or called macro-action. The definition inherits from MDP and added a intermediate level of control. An option is defined as a tuple $m = \{\beta_m, I_m, \pi_m\}$, β_m is termination condition, I_m is initiation set of states, and π_m is low level policy. The high level policy selects option to execute is Ψ. The goal of option learning is find Ψ to maximize $V^{\Psi}(s_0) = \mathbb{E}\left[\sum_{t=0}^{h-1} \gamma^t \cdot r(s_t, a_t) | s_0, \pi, \Psi\right]$.

[48] implemented macro-action with Deep Q learning framework and proposed concurrent experience replay for multi-agent learning. Result shows the introduction of macro-action solved some multi-agent domains which are hard to be solved by methods with primitive actions.

[30] proposed to learn high level policy using a set of finite state automata and low level policy using feedback controller. Macro actions are automatically decomposed to a set of local feedback controllers by Feedback-based Information Roadmaps(FIRM) [2], and learns the high level finite state automata by G-DICE algorithm [29]. Such method extends badly while the search space for G-DICE algorithm grows exponentially.

[45] extended hierarchical Deep Q-Networks (hDQN) [22] to hierarchical independent MARL and proposed an Augmented Experience Replay (ACER). The low level policy is homogeneous with Parameter Sharing. However they neglected some local cooperative behaviors which preclude the application of algorithm.

[3] extended feudal Reinforcement learning [**?**] to multi-agent setting. They proposed to use a manager agent to give sub-goals and reward signal to other worker agents. Only the manager perceives the environmental reward and the workers are only directed by manager.

[49] proposed a latent variable based hierarchical MARL system. The high level policies' action space is latent variable. The low level policy receive the latent variable and select action conditioning on that. Only the high level policy receives extrinsic reward and low level policy's reward is to correctly infer trajectory and latent variable. However they assume the macro-actions defined with fixed duration and run synchronously, which limits the generality of their algorithm.

[26] used Parameter-Sharing TRPO to execute decentralized policies.

[7] employed manager-worker hierarchy, each worker has local critic and the manager has a global critic. The workers update their policy by incorporating local and globla critics' information.

2.4 Task Allocation

Some successful (non-)commercial multi-task multi-robot systems [36, 14, 37, 19] have been developed and validated in past decade. Previous methods mainly focused on solutions in "one to many" assumption that each robot can solve many tasks sequentially and independently. However many cases are under "many to one" assumption that one task requires multiple robot to engage.

Time is important in exploration tasks such as search and rescue activities and patrolling. There are 4 items to evaluate the time efficiency [38] of a allocation strategy.

(1) **Run time:** Time between the start of all robot and the point the last

robot completes the tasks allocated to it.
(2) **Deliberation time:** Time between start and finish of the allocation phase.
(3) **Execution time:** Time that the robots take to execute the set of tasks allocated to it.
(4) **Idle time:** The amount of time that robots sit idly during the execution of a set of tasks.

Auction based task allocation is widely adopted by robot community and can be implemented centrally[28, 38] or locally[5]. However such mechanism usually produces sub-optimal solution [8] and needs many iterations to fully allocate all the tasks, which introduces high communication load. Valuation on tasks in task pool is necessary to determine the bid, however such valuation is usually unknown, and agent has unlimited budget for bid. Also, bidding in auction always has one winner that precludes cooperation.

[52] addressed multi-robot multi-task allocation problem in warehouse scenario. They assumed all robots and tasks are homogeneous and proposed balanced heuristic mechanism which is a heuristic function minimizing the travel cost as well as balancing the individual travel cost of each robot individual travel cost. Such mechanism is implemented with a auction based method and a clustering based method to improve the performance.

[9] discussed scheduling and path planning problem jointly. They employed genetic algorithm [21] for solving scheduling and Q-learning [42] for path planning problem. However there method has obvious defect when executing in more complicated environment. First, evolutionary strategy such as genetic algorithm is inefficient under problem with many constrains. In such case, samples can be hard to generate. Second, the path planning is running under independent single agent case and employed simple "stop and wait" strategy to deal with collision, which can be inefficient in many cases.

Clustering method was also used in [11], where the task pool is divided into several clusters by k-means algorithm. Robots are allocated to different clusters and try to planned a optimal path to solve all the tasks sequentially in the cluster as a travelling salesman problem and can be efficiently solved by evolutionary algorithms. Where similar solution as been implemented in simulation environment for autonomous underwater vehicles (AUVs) [46].

[47] designed a bounty hunter mechanism that tasks are published as bounties and can be committed by many agents. Bounty model does not assume that agents have accurate valuation of the task. Once agent finishes the bounty, it will get the reward based on bounty. Uncompleted bounty will have increasing reward. Initially, agents are encouraged to explore all bounties, and based on which evaluate the estimated time and probability of finishing it. Later commissions will be based on such evaluation and adapt to optimal allocation. This approach assumes no prior of tasks where takes time for agents to explore. Also,

the allocation is not exclusive that many agents can commit to the same task that increases redundancy.

[24] formulated the task allocation problem as a 0–1 quadratic integer programming problem, and solved by artificial bee colony algorithm [20]. However they still assumed all the tasks are atomized and neglect local cooperation.

The problem of previous methods is assuming all tasks are atomized and homogeneous which can be solved by single robot. In such case, the requirement of cooperation behavior will be less considered or neglected. Such atomized task can be possible in certain scenario but not general. In most cases task might require composition of skills from individuals and hard to be solved solely. Local cooperation and coordination becomes essential also brings new challenge for task allocation.

2.5 Transfer Learning

The knowledge solving a particular task can be effective transferred when dealing with a unseen but similar task. Such knowledge is used as prior and cuts down the area of exploration, which may take long time to learn from scratch. knowledge can be transferred can be policy as initial exploratory trajectories or value as initial guess of value function.

In [33], the author proposed a method to transfer the knowledge of some expert DQN in to a mimic policy network. The method generates a target Boltzmann policy from expert DQN and tries to minimize the cross entropy between target policy and mimic policy. Furthermore, the hidden layer of mimic policy network has auxiliary task to predict the output of hidden layer of expert DQN to ensure useful feature to be transferred.

[10] addressed the problem by adding a recurrent layer to the policy network and train it on a distribution of MDPs which share some similarity but possess randomness. In such setting, the recurrent layer learns in which MDP it is, and make the agent to act differently in different MDPs. Such knowledge of general MDPs can be transferred to new MDP which is previously unseen as prior.

[34] assume the solution for the target task can be expressed as a weighted linear combination of solutions of source tasks. An additional attention network is employed to assign importance to different source solution. They implemented their solution on policy transferring and value transferring, both of which have shown accelerated learning speed.

[39] leverages the distributed policy learnt by [4] as policy prior in target domain and showed improved learning speed compared with learning from scratch.

2.6 Curriculum Learning

The concept of curriculum learning is first proposed by [6] in machine learning. The idea is to pre-train the learner on a 'simpler' task compared with target 'hard' task, then use the trained model as the prior of learner on target task. This decomposition of such 'hard' task in to a series of 'simpler' tasks with gradually increased difficulty is denoted as curriculum strategy. Curriculum learning is a specific implementation of transfer learning. General transfer learning focuses on transfer knowledge from a certain domain to another similar domain. However curriculum learning focuses on learning from a simplified domain and transfer to harder tasks.

[27] formally defined curriculum problem in reinforcement learning and proposed several methods for finding the task sequence of increasing difficulty.

[1] employed attention mechanism to extend the scalability of multi-agent system. They set a series of curricula by gradually increasing the number of agents that the skill learnt in a small group can be transferred to a larger group.

[43] proposed an automatic curricula generating method that the generated curriculum is represented as a Directed Acyclic Graph. The limitation of their method is that they assume countable and disentangled domain features can be used to generate domains of different difficulties, however it is not always available.

[12] proposed a curricula generating mechanism with specific goal state and sparse reward. The distribution of initial state is initially near goal state and gradually expand to true initial state distribution, and each initial distribution can be seen as a curriculum.

[40] implemented curriculum learning in Starcraft game with homogeneous agents and showed improved performance.
[15] introduced a curriculum learning mechanism in decentralized homogeneous multi-agent learning by parameter sharing.

3 Example

Considering there is a classroom to be cleaned as shown in Figure 4, and several desks (need two student to move) and chairs (need one student to move) need to be firstly moved out of the classroom.

Task partition: Apparently the whole task is to move all the furniture out out classroom, and can be partitioned into 6 sub-tasks (move two desks out and move four chairs out), and a additional sub-task is "doing nothing".

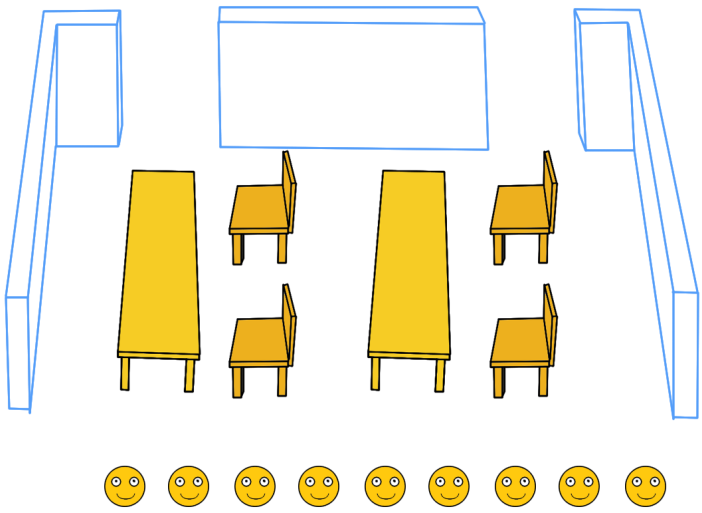


Figure 4: Classroom example

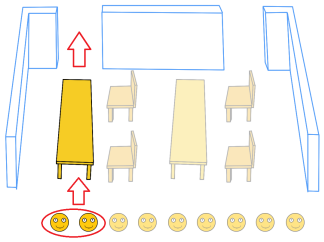
Group partition: Assuming we have nine students, we can partition them as:

Group 1: student 1 + student 2
Group 2: student 3 + student 4
Group 3: student 5
Group 4: student 6
Group 5: student 7
Group 6: student 8
Group 7: student 9

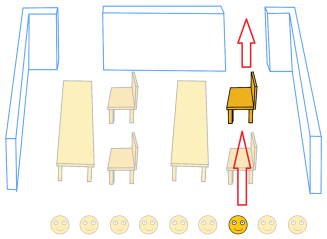
This is just a random partition not something optimal, because we did not optimize any score function to get such partition.

Allocation: In such case, we can assign group 1 and 2 responsible for the two desks and group 3-6 to move the chairs. Group 7 has no tasks available, then it can choose "doing nothing".

Execution: Each group can choose their optimal path to move the desks and chairs out of classroom. If there's no path collision, we can assume each group executes independently. Otherwise, the coordination between groups should be considered.



(a) Group 1 (student 1 + 2) is assigned to move a desk



(b) Group 5 (student 7) is assigned to move a chair

4 Nested MDP

We propose a new framework to model such kind of problems, which we name as Nested Markov Decision Process (NMDP), and we can see it is a effective and solvable model for multi-agent multi-task scenario.

Definition A Nested Markov Decision Process is defined as a tuple $\langle \mathbf{N}, \mathbf{M}, \mathbf{K}, \mathbf{T}, \mathbf{R}, \mathbf{S}, \mathbf{A}, \mathbf{C}, h \rangle$.

(1) $\mathbf{N}$ is the set of all agents n
(2) $\mathbf{M}$ is the set of all tasks m
(3) $\mathbf{K}$ is the set of configuration k of each task
(4) $(T_{m,k}, R_{m,k})$ are transition probability and reward function defining a nested MDPs, thus the system is segmented to $|M| \times |K|$ nested MDPs as $MDP_{m,k}$
(5) $\mathbf{S}$ is the joint state space of all agents
(6) $\mathbf{A}$ is the joint action space of all agents
(7) $\mathbf{C}$ is the allocation function that segment joint state space and joint action space into $|M|$ sub-spaces each to be the state space and action space of $MDP_{m,k}$.
(8) h is the horizon of the problem.

Here we clarify some properties of NMDP.

(1) For a certain m, there are k different MDPs. We use an example to show how it works.

At high level, the scenario can be modelled as a directed acyclic graph $\mathbf{G} = \langle \mathbf{N}, \mathbf{M}, \mathbf{C} \rangle$, each agent can be represented as a node in Node set $\mathbf{N}$ and each task can be represented as a node in Node set $\mathbf{M}$ as shown in Figure (6) left. An assignment $\mathbf{C}$ is the set of edges emit from each node in $\mathbf{N}$ to each node in $\mathbf{M}$. The assignment (if we use hard assignment) can be represented as a $|\mathbf{N}| \times |\mathbf{M}|$ shape matrix that each element $c_{n,m}$ is 0 or 1 for agent n is not assigned or assigned to task m. In the example, the current allocation is a 5×4 matrix, which is:

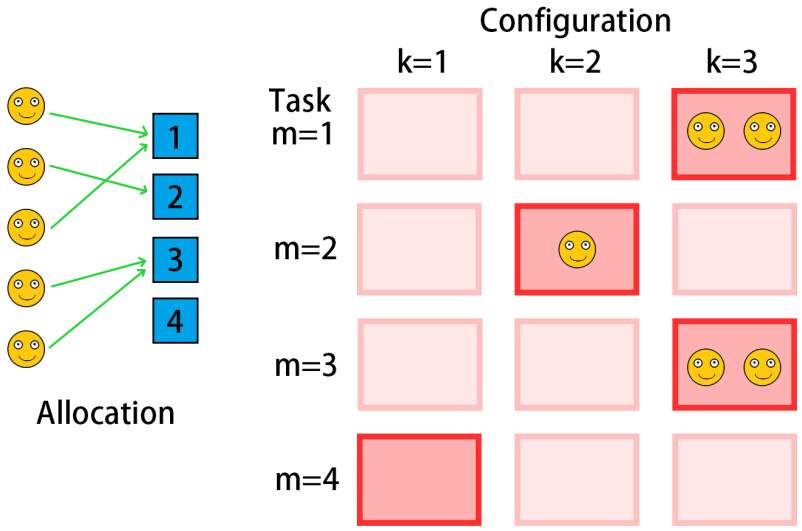


Figure 6

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1}$$

Each row is the assignment of an agent, each column is the assignment of a task.

At low level, if each task can accept at most 2 agents, there will be three different configurations for each task. $k = 1$ indicates there is no agent assigned to task m. $k = 2$ indicates there is one agent assigned to task m. $k = 3$ indicates there are two agents assigned to task m. Each time there is only one MDP for each m is activated as shown in Figure (6) right side. Each configuration is a nested MDP. When $k = 1$, no agent participates in the task thus the state space and action space of $MDP_{m,1}$ is $\varnothing$. When $k = 2$, the state space of $MDP_{m,2}$ is the individual state space of single agent as S and the action space is A. When $k = 3$, the state space of $MDP_{m,3}$ is the joint state space of two agents as $S \times S$ and the action space is $A \times A$.

We assume all $MDP_{m,k}$ run simultaneously, however for the $MDP_{m,k}$ which are not assigned any agent, the state space and action space are all empty set $\varnothing_s$ and $\varnothing_a$. The transition probability is $T_{m,k}\left(\varnothing_{s'}|\varnothing_s, \varnothing_a\right)$ and reward function as always zero $R_{m,k}\left(\varnothing_s, \varnothing_a\right) = 0$.

(2) An allocation c uniquely defines the transition probability and reward func-

tion for each task m and each configuration k, and uniquely defines the nested state space $S_{m,k}$ and action space $A_{m,k}$ so

$$\bigcup_{m,k} S_{m,k} = \mathbf{S} \tag{2}$$

$$\bigcup_{m,k} A_{m,k} = \mathbf{A} \tag{3}$$

Also the transition probability and reward function take the parameters as

$$T_{m,k}\left(s'_{m,k}|s_{m,k}, a_{m,k}\right) = \bar{T}_{m,k}\left(\mathbf{s}'|\mathbf{s}, \mathbf{s}, c\right) \tag{4}$$

$$R_{m,k}\left(s_{m,k}, a_{m,k}\right) = \bar{R}_{m,k}\left(\mathbf{s}, \mathbf{a}, c\right) \tag{5}$$

Where $s_{m,k} \in S_{m,k}$ and $a_{m,k} \in A_{m,k}$. Notice if in current configuration k, if more than one agent is assigned to the $MDP_{m,k}$, $s_{m,k}$ and $a_{m,k}$ can still be joint state and action, but locally. If we want to specifically indicate the individual state and action, they should be $s_{m,k,n}$ and $a_{m,k,n}$. The equality holds because c uniquely defines the nested state space $S_{m,k}$ and action space $A_{m,k}$ so we can use $S_{m,k}$ and $(\mathbf{S}, c)$ in mixed way, so as $A_{m,k}$ and $(\mathbf{A}, c)$.

The policies in NMDP are decentralized, each agent chooses action $a_{m,k,n}$ in the nested MDP which it is assigned to based on the nested state $s_{m,k}$

$$\pi_n\left(a_{m,k,n}|s_{m,k}, c\right) \tag{6}$$

So the the goal of a NMDP is to find a set of decentralized policy $\Pi = \{\pi_1, \pi_2, \cdots, \pi_n\}$ maximize the expected return as:

$$\Pi^*, c^* = \underset{\Pi,c}{argmax} \mathbb{E}_{a_{m,k} \sim \Pi}\left[\sum_{t=0}^{h-1} \sum_{m,k} \gamma^{T-h+1} \cdot R_{m,k}\left(s_{m,k}, a_{m,k}\right)|_c\right] \tag{7}$$

5 Curriculum Learning

Jointly learning Π^* and c^* can be hard as the hierarchical problem, at high level it learns allocation and at low level the agents learns how to solve each assigned $MDP_{m,k}$. A better solution is to separate the two levels of learning. At low level we can assume each $MDP_{m,k}$ as a curriculum, and learn a optimal policy for this curriculum, we denote such local joint policy as $\pi^*_{m,k}$. As we assume all agents have mastered the $\pi^*_{m,k}$ for all tasks m and all configurations k, and each agent only execute one of $\pi^*_{m,k}$ that it is assigned.

Then at low level, that task becomes

$$\pi^*_{m,k} = \underset{\pi^*_{m,k}}{argmax} \mathbb{E}_{a_{m,k} \sim \pi^*_{m,k}} \left[\sum_{t=0}^{h-1} \gamma^{T-h+1} \cdot R_{m,k} \left(s_{m,k}, a_{m,k} \right) \right] \tag{8}$$

as a traditional multi agent reinforcement learning problem.

6 Planning

When we have learned the optimal policy for each nested MDP, the byproduct is the corresponding optimal value function of certain pair of m, k is

$$V \left(s_{m,k}, a_{m,k} \right) = \mathbb{E}_{a_{m,k} \sim \pi^*_{m,k}} \left[\sum_{t=0}^{h-1} \gamma^{T-h+1} \cdot R_{m,k} \left(s_{m,k}, a_{m,k} \right) \right] \tag{9}$$

From previous section, we see $s_{m,k}$ and $a_{m,k}$ are uniquely determined by c from $\mathbf{s}$ and $\mathbf{a}$, so such value in Equation 9 can also be written as:

$$V \left(\mathbf{s}, \mathbf{a}, c \right) = \sum_{m,k} V \left(s_{m,k}, a_{m,k} \right) \tag{10}$$

Then the high level optimization can be formulated as

$$c^* = \underset{c}{argmax} V \left(\mathbf{s}, \mathbf{a}, c \right) \tag{11}$$

Where $V \left(\mathbf{s}, \mathbf{a}, c \right)$ are trained value function from individual nested MDPs, which is known.

As we illustrated in Example 1, c can be expressed as a $|\mathbf{N}| \times |\mathbf{M}|$ shape matrix with elements $c_{n,m}$. Depending on specific application, there are certain constrains on the choice of $c_{n,m}$. For example, in the classroom problem, one agent cannot be assigned to two tasks. Also, the maximum number of agents assigned to a task is maximal two (we can move desk or chair with two agents, but three agents doing one task is not allowed. I this example, we can write the optimization problem as:

$$\begin{aligned} \max_{c} \quad & V \left(\mathbf{s}, \mathbf{a}, c \right) = \sum_{m,k} V \left(s_{m,k}, a_{m,k} \right) \\ s.t. \quad & 0 \leqslant \sum_{m} c_{n,m} \leqslant 1 \\ & 0 \leqslant \sum_{n} c_{n,m} \leqslant 2 \\ & c_{n,m} \text{ are binary variables} \end{aligned} \tag{12}$$

Such problem is a binary linear programming problem, and can be solved analytically by *Branch and Bound* algorithm [23] or *Branch and Cut* algorithm [32]. However the problem is NP-hard which means when it is not efficient for application with large n or m.

We found a way to relax such problem into a linear programming problem which can be solved efficiently in polynomial time, and whose solution takes high possibility to be binary solution, or float solution can be efficiently rounded to binary solution.

7 Example

One example we give here is of N agents and M tasks, and each task allows maximally two agents to participate. There are two types of tasks, M_1 requires at least one agent and M_2 requires at least two agents. In total M tasks, there will be half to half chance to generate M_1 or M_2 task. M_1 task gives expected value uniformly between 0 to 10, M_2 task gives expected value uniformly between 10 to 20.

Directly optimize in binary space of c is difficult, so we relax such variable of $c_{n,m}$ to a real number in $[0, 1]$. Then we use linear programming solver to find the float solution in polynomial time and round it to binary matrix. One option is to relax each $c_{n,m}$ to $x_{n,m}$, but we use a different relax that produces results with higher quality.

As the maximum number of agents assigned to a task is two, we define a variable $x_{i,j,m}$ as real number in $[0, 1]$. When $x_{i,j,m} = 1$, it means agent i and agent j are assigned to task m, and $x_{i,j,m} = 0$ means agent i and agent j are not assigned to task m. Values in between are just for mathematical convenience, and $x_{i,j,m}$ is intrinsically valid when $x_{i,j,m} = 0/1$. $V\left(s_{m,k}, a_{m,k}\right)$ in Equation 10 is defined as a scalar $T^s_{i,j,m}$ for M_1 task and $T^p_{i,j,m}$ for M_2 task (we do not consider the effect of states of agents). The optimization problem can be defined as:

$$\begin{aligned} \max \sum_{m\in M_1}\sum_{i\in N} x_{i,i,m}\cdot T^s_{i,i,m} + \frac{1}{2}\sum_{m\in M_1}\sum_{i\in N, j\in N, j\neq i} x_{i,j,m}\cdot T^p_{i,j,m} \\ s.t. \quad x_{i,j,m} = x_{j,i,m} \\ \sum_{m\in M}\sum_{j\in N} x_{i,j,m} \leqslant 1 \\ \sum_{i\in N, j\in N} x_{i,j,m} \leqslant 2 \end{aligned} \tag{13}$$

Such problem can be solved by linear programming solver within polynomial time with $|N| \times |N| \times |M|$ constrains. When there is a non-integer solution,

we can round each task's allocation with non-integer value to integer to make $x_{i,j,m}$ valid.

Greedy	ours	linear
0.81	0.49	0.78

Table 1: Worst case ratio

Greedy	ours	linear
0.24	0.74	0.26

Table 2: Integer solution ratio

Greedy	ours	linear
0.57	0.85	0.55

Table 3: 95 percent performance ratio

8 Experiments

8.1 Environment Overview

9 Extensions

9.1 Perturbation

9.2 Decentralized Planning

9.3 Model Based Planning

9.4 Task Finding

10 Conclusion

References

[1] Akshat Agarwal, Sumit Kumar, and Katia Sycara. Learning transferable cooperative behavior in multi-agent teams. *arXiv preprint arXiv:1906.01202*, 2019.

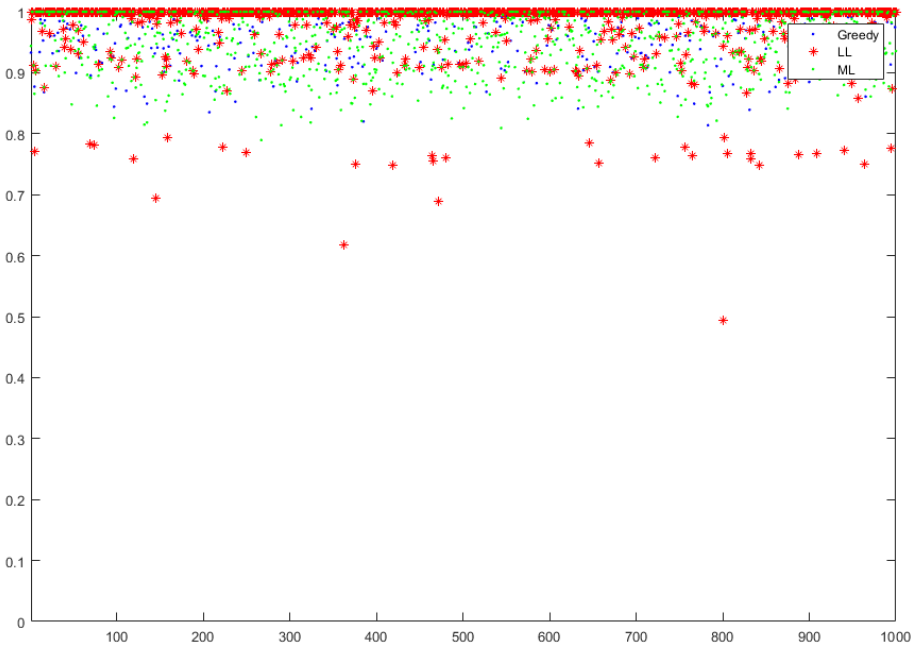


Figure 7: 8 agents, 5 tasks, 1000 trials, ratio to the global optimal solution

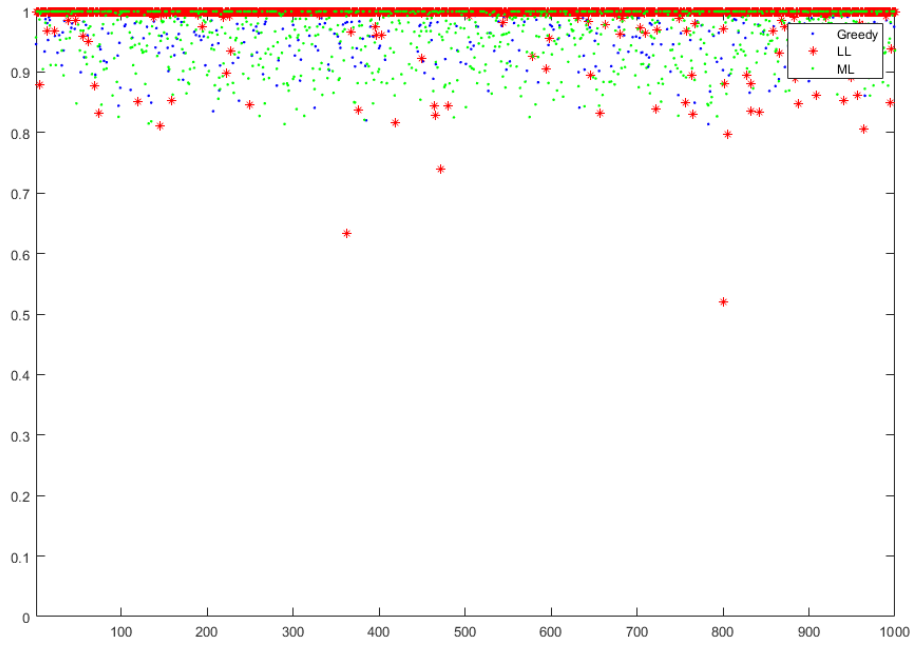


Figure 8: 8 agents, 5 tasks, 1000 trials, ratio to the maximum of all algorithms

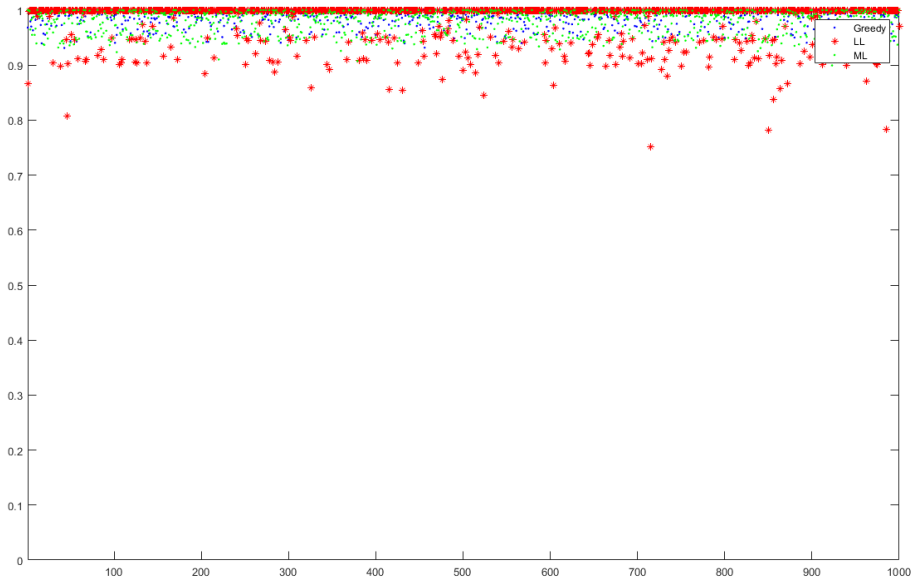


Figure 9: 20 agents, 20 tasks, 1000 trials, ratio to the maximum of all algorithms

x_RoundedLL =

0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Figure 10: Planning result

[2] Ali-Akbar Agha-Mohammadi, Suman Chakravorty, and Nancy M Amato. Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research*, 33(2):268–304, 2014.

[3] Sanjeevan Ahilan and Peter Dayan. Feudal multi-agent hierarchies for cooperative reinforcement learning. *arXiv preprint arXiv:1901.08492*, 2019.

[4] Philip Bachman, Riashat Islam, Alessandro Sordoni, and Zafarali Ahmed. Vfunc: a deep generative model for functions. *arXiv preprint arXiv:1807.04106*, 2018.

[5] Quentin Baert, Anne-Cécile Caron, Maxime Morge, Jean-Christophe Routier, and Kostas Stathis. Adaptive multi-agent system for situated task allocation. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1790–1792. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[7] Zehong Cao and Chin-Teng Lin. Hierarchical critics assignment for multi-agent reinforcement learning. *arXiv preprint arXiv:1902.03079*, 2019.

[8] Rongxin Cui, Ji Guo, and Bo Gao. Game theory-based negotiation for multiple robots task allocation. *Robotica*, 31(6):923–934, 2013.

[9] Jiajia Dou, Chunlin Chen, and Pei Yang. Genetic scheduling and reinforcement learning in multirobot systems for intelligent warehouses. *Mathematical Problems in Engineering*, 2015, 2015.

[10] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[11] Murugappan Elango, Subramanian Nachiappan, and Manoj Kumar Tiwari. Balancing task allocation in multi-robot systems using k-means clustering and auction based mechanisms. *Expert Systems with Applications*, 38(6):6486–6491, 2011.

[12] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017.

[13] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[14] Eric Guizzo. Three engineers, hundreds of robots, one warehouse. *IEEE spectrum*, 45(7):26–34, 2008.

[15] 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.

[16] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. *arXiv preprint arXiv:1810.02912*, 2018.

[17] Jiechuan Jiang, Chen Dun, and Zongqing Lu. Graph convolutional reinforcement learning for multi-agent cooperation. *arXiv preprint arXiv:1810.09202*, 2(3), 2018.

[18] Nidhi Kalra, Dave Ferguson, and Anthony Stentz. Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 1170–1177. IEEE, 2005.

[19] Andreas Kamagaew, Jonas Stenzel, Andreas Nettsträter, and Michael ten Hompel. Concept of cellular transport systems in facility logistics. In *The 5th International Conference on Automation, Robotics and Applications*, pages 40–45. IEEE, 2011.

[20] Dervis Karaboga and Bahriye Akay. A comparative study of artificial bee colony algorithm. *Applied mathematics and computation*, 214(1):108–132, 2009.

[21] John R Koza. Genetic programming. 1997.

[22] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[23] AH Land and AG Doig. An automatic method of solving discrete programming problems, . 497-520.

[24] Hong Liu, Peng Zhang, Bin Hu, and Philip Moore. A novel approach to task assignment in a cooperative multi-agent design system. *Applied Intelligence*, 43(1):162–175, 2015.

[25] 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.

[26] Kunal Menda, Yi-Chun Chen, Justin Grana, James W Bono, Brendan D Tracey, Mykel J Kochenderfer, and David Wolpert. Deep reinforcement learning for event-driven multi-agent decision processes. *IEEE Transactions on Intelligent Transportation Systems*, 20(4):1259–1268, 2018.

[27] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source task creation for curriculum learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 566–574. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[28] Ernesto Nunes and Maria Gini. Multi-robot auctions for allocation of tasks with temporal constraints. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[29] Shayegan Omidshafiei, Ali-Akbar Agha-Mohammadi, Christopher Amato, Shih-Yuan Liu, Jonathan P How, and John Vian. Graph-based cross entropy method for solving multi-robot decentralized pomdps. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5395–5402. IEEE, 2016.

[30] Shayegan Omidshafiei, Ali-Akbar Agha-Mohammadi, Christopher Amato, Shih-Yuan Liu, Jonathan P How, and John Vian. Decentralized control of multi-robot partially observable markov decision processes using belief space macro-actions. *The International Journal of Robotics Research*, 36(2):231–258, 2017.

[31] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2681–2690. JMLR. org, 2017.

[32] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[33] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[34] Janarthanan Rajendran, Aravind S Lakshminarayanan, Mitesh M Khapra, P Prasanna, and Balaraman Ravindran. Attend, adapt and transfer: Attentive deep architecture for adaptive transfer from multiple sources in the same domain. *arXiv preprint arXiv:1510.02879*, 2015.

[35] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*, 2018.

[36] Guillaume Sartoretti, Yue Wu, William Paivine, TK Satish Kumar, Sven Koenig, and Howie Choset. Distributed reinforcement learning for multi-robot decentralized collective construction. In *Distributed Autonomous Robotic Systems*, pages 35–49. Springer, 2019.

[37] Jonas Lye Scheie, Jens Petter Røyseth, Preben Grøssereid, Lee Zheng, and Geir Fjermstad Rolandsen. Autostore. B.S. thesis, 2012.

[38] Eric Schneider, Elizabeth I Sklar, Simon Parsons, and A Tuna Özgelen. Auction-based task allocation for multi-robot teams in dynamic environments. In *Conference Towards Autonomous Robotic Systems*, pages 246–257. Springer, 2015.

[39] Disha Shrivastava, Eeshan Gunesh Dhekane, and Riashat Islam. Transfer learning by modeling a distribution over policies. *arXiv preprint arXiv:1906.03574*, 2019.

[40] Felipe Leno Da Silva and Anna Helena Reali Costa. Object-oriented curriculum generation for reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1026–1034. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[41] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[42] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

[43] Maxwell Svetlik, Matteo Leonetti, Jivko Sinapov, Rishi Shah, Nick Walker, and Peter Stone. Automatic curriculum graph generation for reinforcement learning agents. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[44] Andrea Tacchetti, H Francis Song, Pedro AM Mediano, Vinicius Zambaldi, Neil C Rabinowitz, Thore Graepel, Matthew Botvinick, and Peter W Battaglia. Relational forward models for multi-agent learning. *arXiv preprint arXiv:1809.11044*, 2018.

[45] Hongyao Tang, Jianye Hao, Tangjie Lv, Yingfeng Chen, Zongzhang Zhang, Hangtian Jia, Chunxu Ren, Yan Zheng, Zhaopeng Meng, Changjie Fan, et al. Hierarchical deep multiagent reinforcement learning with temporal abstraction. *arXiv preprint arXiv:1809.09332*, 2018.

[46] Nikolaos Tsiogkas, Georgios Papadimitriou, Zeyn Saigol, and David Lane. Efficient multi-auv cooperation using semantic knowledge representation for underwater archaeology missions. In *2014 Oceans-St. John's*, pages 1–6. IEEE, 2014.

[47] Drew Wicke, David Freelan, and Sean Luke. Bounty hunters and multiagent task allocation. In *Proceedings of the 2015 international conference on autonomous agents and multiagent systems*, pages 387–394. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[48] Yuchen Xiao, Joshua Hoffman, and Christopher Amato. Macro-action-based deep multi-agent reinforcement learning. In *3rd Annual Conference on Robot Learning*, 2019.

[49] Jiachen Yang, Igor Borovikov, and Hongyuan Zha. Hierarchical cooperative multi-agent reinforcement learning with skill discovery. *arXiv preprint arXiv:1912.03558*, 2019.

[50] Jiachen Yang, Alireza Nakhaei, David Isele, Kikuo Fujimura, and Hongyuan Zha. Cm3: Cooperative multi-goal multi-stage multi-agent reinforcement learning. *arXiv preprint arXiv:1809.05188*, 2018.

[51] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1802.05438*, 2018.

[52] Luowei Zhou, Yuanyuan Shi, Jiangliu Wang, and Pei Yang. A balanced heuristic mechanism for multirobot task allocation of intelligent warehouses. *Mathematical Problems in Engineering*, 2014, 2014.