LPMARL: LINEAR PROGRAMMING BASED IMPLICIT TASK ASSIGNMENT FOR HIEARCHICAL MULTI-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Training a multi-agent reinforcement learning (MARL) model with a sparse-reward is notoriously difficult because the final outcome (i.e., success or failure) is induced by numerous combinations of interactions among agents. Earlier studies have tried to resolve this issue by using hierarchical MARL to decompose the main task into subproblems or employing an intrinsic reward to induce interactions for learning an effective policy. However, none of the methodologies have shown significant success. In this study, we employ a hierarchically structured policy to induce effective coordination among agents. At every step, LPMARL conducts the two hierarchical decision-makings: (1) solving an agent-task assignment problem and (2) solving a local cooperative game among agents assigned to the same task. For the first step, LPMARL formulates the agent-task assignment problem into a resource assignment problem, a type of linear programming (LP). For this, LPMARL uses a graph neural network to generate state-dependent cost coefficients for the LP problem. The solution of the formulated LP is the assignments of agents to tasks, which decompose agents into tasks to accomplish the sub-goals among the agents in the group. For the lower-level decision, LPMARL employs a general MARL strategy to solve each sub-task. We train the GNN generating the state-dependent LP for high-level decisions and the low-level cooperative MARL policy together end-to-end using implicit function theorem. We empirically demonstrate that our algorithm outperforms existing algorithms in various mixed cooperative-competitive environments.

1 INTRODUCTION

As engineering systems become highly complicated and distributed, it becomes challenging to understand the collective and interactive behavior of such multi-agent systems. Thus, multi-agent reinforcement learning has recently drawn much attention due to the practical and potential applications for controlling highly complicated and distributed multi-agent systems in intelligent transportation, smart grid, smart factory, etc. Although numerous multi-agent reinforcement learning (MARL) training strategies have been successful, these successes have been limited to simple ideal tasks where the problem-specific immediate reward can be well defined and used to induce the cooperative policy to achieve the system-level goal. There are practical problems where the success or failure of the task only reveals after a long sequence of strategic interactions among agents.

The main research objective of the current study is to develop a practical MARL framework for deriving the cooperative decision-making policy using only a spare and delayed reward. Training a MARL model with a sparse-reward is notoriously difficult because the final outcome (i.e., success or failure) is induced by numerous combinations of interactions among agents and stochatic environment. To resolve these challenges, one needs to develop an algorithm that can learn how the interaction among agents over long-term episodes entails the outcome of the target tasks, a delayed and sparse episodic reward, and deduce this understanding into an effective sequential decision-making policy.

Earlier studies have tried to resolve this issue by introducing structured decision-making schemes or devising an effective exploration strategy. As an example of a structured decision-making scheme,

hierarchical MARL, composed of high-level policy for assigning sub-goals to agents and the low-level policy for conducting independent goal-dependent primitive action, has been proposed (Tang et al., 2018). As an example of an exploration strategy, utilizing intrinsic reward to induce interactions among agents (Wang et al., 2019) or sharing experience among agents (Christianos et al., 2020) have been proposed. Although each of these algorithms was successful for specific tasks, these approaches require task-specific design choices. For example, one needs to define the high-level and low-level actions for hierarchical MARL. In addition, one needs to design a task-specific intrinsic reward, making these approaches difficult to use as a general scheme for solving cooperative tasks.

In this study, we propose a hierarchically structured decision-making scheme to induce effective coordination among agents. At every step, LPMARL conducts the two hierarchical decision-makings: (1) solving an agent-task assignment problem and (2) solving a local cooperative game among agents assigned to the same sub-task. For the first step, LPMARL formulates the agent-task assignment problem into a constrained resource assignment problem, a type of linear programming (LP). For this, LPMARL uses a graph neural network to generate state-dependent cost coefficients for the LP problem. The solution of the formulated LP is the assignments of agents to tasks, which decompose agents into tasks to accomplish the sub-goals among the agents in the group. For the lower-level decision, LPMARL employs a general MARL strategy to solve each sub-task. We train the GNN generating the state-dependent LP for high-level decisions and the low-level cooperative MARL policy together end-to-end using implicit function theorem.

The technical contributions and novelties of the proposed method are as follows:

- **Interpretability.** In the case of hierarchical MARL, it is sometimes necessary to discover a latent representation of the goal, but it is difficult to grasp the meaning, and it is difficult to assign a constraint. On the other hand, LPMARL can impose the behavior inductive bias, e.g., requiring accomplishing particular objectives or imposing constrained behavior, when formulating LP for decomposing tasks, making our approach more interpretable.
- **Exploration.** Because LPMARL employs high-level actions regarding decomposing the agents into target tasks, we can conduct exploration over decomposing strategies. This is similar to explore over the goal space for typical hierarchical MARL (Liu et al., 2021; Tang et al., 2018). The advantage of LPMARL over hierarchical MARL is that we can effectively control the exploration region by adding the constraints on the decomposing strategy by LP formulation. For example, when assigning agents to tasks, we can add the constraint of assigning at least one agent to each task or vice versa. Furthermore, this exploration strategy is a lot effective than the one conducting exploration over the primitive action (lower-level action) because this single-level MARL strategy lacks the temporal abstraction to consistently induce the cohesive and consistent actions to achieve the meaningful sparse-reward.

2 Related works

Hierarchical MARL. Earlier studies have tried to resolve the sparse-reward MARL problem by adding a hierarchical structure to decompose the main problem into task-dependent subproblems. Tang et al. (2018) proposed a hierarchical MARL framework with temporal abstraction to solve cooperative MARL tasks. In their work, agents cooperatively select high-level actions to achieve goals, while low-level policies are independently trained based on the selected high-level action. Although the proposed hierarchical MARL framework enables agents to learn cooperation in temporally abstracted high-level policy, it is difficult to induce cooperation within sub-tasks only by parameter sharing. Vezhnevets et al. (2019); Yang et al. (2019); Wang et al. (2020a;b) also presented methods for learning the latent goal representation without predefining a set of high-level tasks in a complex multi-agent system. The latent goal indirectly limits the state space or action space so agents can cooperate by learning only primitive low-level policy conditioned on the latent goal. However, to learn the relationship between the global state, the latent goal, and the primitive action, a lot of training loop and a sparse-reward signal are required. Also, the it is not interpretable because the decomposes the problem indirectly.

Exploration-based MARL. Another vein of multi-agent policy training with sparse-reward signal is to train policy with efficient exploration strategies. CMAE (Liu et al., 2021) introduced an explo-

ration policy instead of random exploration, and expressed the agent's behavior policy as a mixture of target policy and exploration policy. Instead of exploring the whole state space, the exploration policy plays a role in exploring only within the low-dimensional restricted space in which the state space is projected. SEAC (Christianos et al., 2020) presented a technical way to learning through shared experience so that agents can do diverse exploration. In SEAC, individual agents consider the other agent's trajectory as off-policy data, and learn individual policies through off-policy gradient optimization with importance sampling. REMAX (Ryu et al., 2020) also proposes a technical strategy to accelerate the training of the agent by generating an initial state similar to the goal state. Although the above methodologies are useful techniques that can accelerate learning in MARL, they do not solve the fundamental credit assignment problem of the sparse-reward problem.

Optimization-based agent-task decomposition. Our work draw inspiration from Carion et al. (2019), a centralized optimization-based decomposition approach. In this paper, authors formulate a high-level agent-task allocation problem into a linear programming where the optimization parameters are learnable. By learning the state-dependent allocation coefficient and constraint coefficient, agents can optimally decomposed into tasks every step in a centralized point of view. However, they use rule-based low-level policy to train the high-level allocation coefficient, and thus the agents may not behave cooperatively in a decomposed subproblem. Also, the optimization problems and low-level policy training are not end-to-end differentiable when trying to train two policies together. For this reasons, we formulate the high-level policy as LP to solve agent-task assignment problem with optimization layer, and low-level policy as agent-action selection policy.

3 BACKGROUND

3.1 IMPLICIT DEEP LEARNING

Implicit deep learning is a framework for incorporating an implicit rules (e.g., ODE, Chen et al., 2018; fixed point iterations, Bai et al., 2019; optimization, Amos & Kolter, 2017) into a feed-forward neural network. Specifically, differentiable optimization is a framework for incorporating an optimization problem into the layer. A differentiable optimization layer indexed with l takes the optimization variables h^{l-1} as an input, where h^{l-1} is the the output of the previous layer. The input variable h^{l-1} works as the optimization variable corresponding to the differentiable optimization layer. The layer then solves the optimization problem and outputs the optimal solution, $h^l := \mathrm{argmin}_{x \in g(h^{l-1})} f(x, h^{l-1})$, where $f(x, h^{l-1})$ is the objective function and $g(h^{l-1})$ is the feasible solution space defined by h^{l-1}. The output of the layer h^l is then fed into the next layer.

By using this approach, one can infuse optimization inductive bias in a layer. OptNet (Amos & Kolter, 2017) propose an differentiable optimization layer, specifically for a quadratic programming (QP). The backpropagation of this optimization layer requires the computation of the derivative of the QP solution with respect to the input parameters, which is derived by taking the matrix differentials of the KKT conditions of the QP. Ferber et al. (2020) and Wilder et al. (2019) extended the idea of OptNet to general linear programming (LP) and mixed-integer linear programming (MILP). To compute the gradient of the optimal solution on discontinuous solution space, they considered the continuous surrogate formulation of the original integer programming problem based on the cutting plane method. To further develop the computation of the gradient of the optimal solution of the combinatorial optimization, Vlastelica et al. (2019) provides a way to construct a continuous interpolation of the loss function. In combinatorial optimization problems, the optimal solution may not vary in small change on the input parameters; thus, the loss function may in the form of the piecewise constant. To solve this problem, Vlastelica et al. (2019) presented a piecewise linear surrogate loss function that can approximate the original loss function of the combinatorial optimization layer.

3.2 HIERARCHICAL MULTI-AGENT REINFORCEMENT LEARNING

A decentralized partially observable Markov decision process (Dec-POMDP) is an extension of the partially observable Markov decision process for a game with multiple agents. Dec-POMDP consists of $< \mathcal{N}, \boldsymbol{\mathcal{S}}, \{\mathcal{O}_i\}_{i \in \mathcal{N}}, \{\mathcal{A}_i\}_{i \in \mathcal{N}}, \mathcal{R}, T >$, where $\mathcal{N}$ is a set of agents, $\boldsymbol{\mathcal{S}}$ is state space, $\{\mathcal{O}_i\}_{i \in \mathcal{N}}$ is the observation space, $\mathcal{A} = \{\mathcal{A}\}_{i \in \mathcal{N}}$ is an action space, $\mathcal{R}$ is a reward function, and T is transition probability. At each timestep, each agent i obtains a partial observation $o_{t,i} : \boldsymbol{\mathcal{S}} \rightarrow \mathcal{O}_i$ from a global state $\boldsymbol{s}_t \in \boldsymbol{\mathcal{S}}$ and selects action $a_{t,i}$ through its policy $\pi_i : \mathcal{O}_i \rightarrow \mathcal{A}_i$. Agents obtain a global reward,

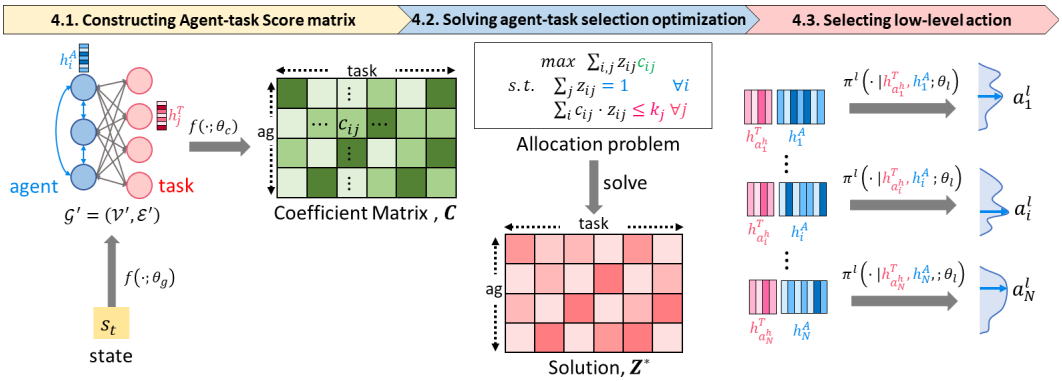


Figure 1: Overall architecture of LPMARL.

$r_t : \boldsymbol{\mathcal{S}} \times \{\mathcal{A}_i\}_{i \in \mathcal{N}} \rightarrow \mathbb{R}$ from the environment. The objective of Dec-POMDP is to find a set of individual policies, $\boldsymbol{\pi}(\boldsymbol{a}_t|\boldsymbol{s}_t) \sim \prod_i \pi_i(a_{t,i}|o_{t,i})$, that maximizes the expected sum of cumulative high-level reward, $\mathbb{E}_{\boldsymbol{a}_t \sim \boldsymbol{\pi}} \left[\sum_{t=0}^{T} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})\right]$.

Tang et al. (2018) considered a hierarchical Dec-POMDP to introduce a temporal abstraction. A high-level of hierarchy can be modeled as a Semi-Markov game, a multi-agent extension of Semi-MDP (Sutton et al., 1999). Formally, each agent i receives observation $o_{t,i}$ and chooses a high-level action $a^h_{t,i} \in \mathcal{A}^{i,h}$, where $\mathcal{A}^{i,h}$ denotes a set of possible high-level actions (goals). The high-level action may last for τ timesteps with low-level action execution until current high-level action $a^h_{t,i}$ is terminated. After the high-level action ends, a next high-level action $a^{i,h}_{t+\tau}$ is selected based on the observation, $o_{t+\tau,i}$.

To train high-level policy and low-level policy, rewards for each level of hierarchy are defined. Agent receives the high-level reward $r^h(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ whenever the agent accomplishes the whole task, i.e., $\boldsymbol{s}_{t+1}$ corresponds is the success state. Agent receives a low-level reward (intrinsic reward) for reaching sub-goal, denoted by $r^l(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}|a^h_{t,i})$ depending to its own high-level action, $a^h_{t,i}$.

4 Methodology

At every step, LPMARL conducts two forward decision-makings steps:(1) solving an agent-task assignment problem and (2) solving a local cooperative game among agents assigned to the same sub-task. For the first step, we represent a global state $\boldsymbol{s}$ as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and use a graph neural network to generate an encoded graph $\mathcal{G} = (\mathcal{V}', \mathcal{E}')$. Using the encoded graph embeddings, we compute a state-dependent cost coefficient $C_{\theta_c}(\mathcal{G}') \in \mathbb{R}^{N \times M}$ for the constrained resource assignment problem. The solution of the formulated LP $Z^*(C_{\theta_c})$ is the assignments of agents to tasks, which decompose agents into tasks to accomplish the sub-goals among the agents in the group. For the lower-level decision, LPMARL employs a general MARL strategy to solve each sub-task. Figure 1 illustrates an decision-making process of LPMARL.

4.1 Constructing agent-task score matrix

We generate a state-dependent cost coefficient matrix $C \in \mathbb{R}^{N \times M}$ for the LP problem. First, we represent a global state into a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to explicitly define the relationship between agents and tasks. The graph nodes are composed of agent nodes $\mathcal{N}$ and task nodes $\mathcal{M}$, $\mathcal{V} = \mathcal{N} \cup \mathcal{M}$. The agent node embedding and task node embedding are initialized as the local state of agent and task, respectively. Two sets of edges are defined; agent-agent edges, $\mathcal{E}_{\mathcal{N}} = \{(i,j) : i \in \mathcal{N}, j \in \mathcal{N}\}$ and task-agent edges, $\mathcal{E}_{\mathcal{M}} = \{(i,j) : i \in \mathcal{M}, j \in \mathcal{N}\}$.

With the constructed graph, we use message-passing graph neural network (Battaglia et al., 2018) to encode $\mathcal{G} = (\mathcal{V}, \mathcal{E})$:

$$m_{ij}^{\mathcal{N}} \leftarrow f(s_i, s_j; \theta_g^{\mathcal{N}}), ij \in \mathcal{E}_{\mathcal{N}}; \quad m_{ij}^{\mathcal{M}} \leftarrow f(s_i, s_j; \theta_g^{\mathcal{M}}), ij \in \mathcal{E}_{\mathcal{M}} \tag{1}$$

$$h_i \leftarrow f([\sum_{ji \in \mathcal{E}_{\mathcal{M}}} m_{ji} \| \sum_{ji \in \mathcal{E}_{\mathcal{N}}} m_{ji} \| s_i]; \theta_g^v), \forall i \in \mathcal{V} \tag{2}$$

where $\theta_g = (\theta_g^{\mathcal{N}}, \theta_g^{\mathcal{M}}, \theta_g^v)$ are a parameters of the GNN. After the graph encoding process, the updated node embedding $\{h_i^A\}_{i \in \mathcal{N}}$ and $\{h_j^T\}_{j \in \mathcal{M}}$ constructs an encoded graph, $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$.

Using the encoded graph node embeddings $\mathcal{V}'$, we can compute a coefficient matrix as follows.

$$c_{ij} := [C_{\theta_c}(\mathcal{G}')]_{i,j} = f(h_i^A, h_j^T; \theta_c) \tag{3}$$

where θ_c is a parameter computing agent-task score, shared among all of agent-task pairs. The coefficient c_{ij} is an allocation score for agent i to finish task j. The constructed cost coefficient will be used to construct the objective function of the agent-task allocation problem.

4.2 High-level policy: Solving Agent-task allocation optimization

We formulate the high-level agent-task assignment problem into an LP, especially resource-constrained allocation problem (Katoh & Ibaraki, 1998). The high-level policy aims to find the optimal agent-task allocation action in a centralized way. Formally, we consider the optimal solution of LP $Z^* := Z^*(C_{\theta_c}(\mathcal{G}'))$ as a function the input coefficients $C_{\theta_c}(\mathcal{G}') \in \mathbb{R}^{N \times M}$ given resource constraints. The LP formulation is as follows:

$$\textit{maximize} \quad \sum_{i,j} c_{ij} \cdot z_{ij} \tag{4}$$

$$\textit{s.t.} \quad \sum_{i} z_{ij} \leq k_i \qquad \forall j \in \mathcal{M} \tag{5}$$

$$\sum_{j} z_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{6}$$

$$0 \leq z_{ij} \leq 1 \qquad \forall i \in \mathcal{N}, j \in \mathcal{M} \tag{7}$$

where $\{z_{ij}\}_{i \in \mathcal{N}, j \in \mathcal{M}}$ is decision variable, equation 4 is objective function to maximize, equation 5 is capacity constraint to restrict maximal number of agents per goal, equation 6 is allocation constraint, and equation 7 is boundary constraint.

The solution of the above optimization problem, $\{z_{ij}^*\}_{i \in \mathcal{N}, j \in \mathcal{M}}$, can be interpreted as the contribution of the agent i to a given task j, and will be used as the policy of all agents, i.e., $\pi_i^h(a_i^h = j | \boldsymbol{s}) = z_{ij}$. This high-level decision decomposes the global task into sub-tasks, inducing a sub-group dedicated to solving the sub-problem per task. A high-level action is obtained only from events that require reallocation, such as a change in the number of agents or tasks. Once a high-level action is calculated at timestep t, the high-level action is maintained for timestep τ until the next event at $t + \tau$.

4.3 Low-level policy: Selecting low-level action

Since the global problem is decomposed into the task-dependent sub-problems, agents only need to cooperate with other agents assigned to the same task to accomplish the sub-problem. Given the selected high-level action a_i^h, agent i select the low-level action a_i^l using the selected task embedding $h_{a_i^h}^T$ as follows:

$$a_i^l \sim \pi^l(\cdot | \boldsymbol{s}, a_i^h) = f(h_i^A, h_{a_i^h}^T; \theta_l). \tag{8}$$

The low-level policy may either continuous or discrete, and we validated two different policy on experiment (continuous policy in section 6.2 and discrete policy in section 6.3).

5 TRAINING

We have the following parameters to train:

- Parameters $\theta_g = (\theta_g^{\mathcal{N}}, \theta_g^{\mathcal{M}}, \theta_g^v)$ of GNN modules $f(\cdot;\theta_g) : \mathcal{G} \to \mathcal{G}'$
- Parameters θ_c of coefficient generating function $f(\cdot;\theta_c) : \mathcal{G}' \to \mathcal{C}$
- Parameters $\theta_h = (\theta_g, \theta_c)$ for high-level policy, $a_i^h \sim \pi^h(\cdot|\boldsymbol{s};\theta_h)$
- Parameters ϕ_h for high-level critic, $Q^h(\boldsymbol{s}, a_i^h;\phi_h)$,
- Parameters θ_l for low-level policy, $a_i^l \sim \pi^l(\cdot|\boldsymbol{s}, a_i^h;\theta_l)$,
- Parameters ϕ_l for low-level critic, $Q^l(\boldsymbol{s}, a_i^l|a_i^h;\phi_l)$.

The objective of the high-level policy is to maximize a sum of the discounted global high-level return, $\mathcal{J}(\theta_h) = \mathbb{E}[\sum_t \gamma^t r_t^h] = \mathbb{E}_{\boldsymbol{a}_t\sim\pi}[Q^h(\boldsymbol{s}_t, \boldsymbol{a}_t)] \approx \mathbb{E}_{(\boldsymbol{s},a_i^h,r^h,\boldsymbol{s}')\sim D}[\sum_i Q^h(\boldsymbol{s}_t, a_{t,i}^h;\phi_h)]$. Similarly, the objective of the low-level policy is to maximize a sum of discounted individual low-level return when assigned to a task, $\mathcal{J}(\theta_l) = \mathbb{E}[\sum_t \sum_i \gamma^t r_{t,i}^l] \approx \mathbb{E}_{(\boldsymbol{s},a_{t,i}^h,a_{t,i}^l,r_i^l,\boldsymbol{s}')\sim D}[\sum_i Q^l(\boldsymbol{s}_t, a_{t,i}^l|a_{t,i}^h;\phi_l)]$.

Since the same high-level action may continuously repeated over a period, the target of the high-level actor-critic is discounted proportionally to the length of timestep the high-level action lasted on. The high-level critic $Q^h(\cdot;\phi_h)$ is trained to minimize the mean squared error between predicted critic value and target $\mathcal{L}(\phi_h)$, defined as

$$\mathcal{L}(\phi_h) = \mathbb{E}_{(\boldsymbol{s},a_i^h,r^h,\boldsymbol{s}')\sim D}\left[\left(\sum_i Q^h(\boldsymbol{s}_t, a_{t,i}^h;\phi_h) - y\right)^2\right] \tag{9}$$

where y is the target computed as $y = r_t^h + \gamma^\tau \cdot \sum_i \max_{a_{t,i}^h} Q(\boldsymbol{s}_{t+\tau}, a_{t,i}^h;\bar{\phi}_h)$ with $\bar{\phi}_h$ being the target parameter. The parameters of high-level actor θ_h is optimized using the partial gradient of the objective, computed as:

$$\nabla_{\theta_h}\mathcal{J}(\theta_h) = \mathbb{E}_{(\boldsymbol{s},a_i^h,r^h,\boldsymbol{s}')\sim D}\left[\sum_i \{Q^h(\boldsymbol{s}_t, a_{t,i}^h;\bar{\phi}_h) - V(\boldsymbol{s}_t)\}\nabla_{\theta_h}\log\pi^h(a_{t,i}^h|\boldsymbol{s}_t;\theta_h)\right] \tag{10}$$

The combined procedures of (1) computing the node embedding, (2) constructing the agent-task score matrix C, and (3) solving the agent-task allocation problem can serve as the high-level policy ($\pi^h(\cdot;\theta_h) : \mathcal{G} \to Z$). This decision making pipeline then construct the following forward propagation chain:

$$\pi^h(\cdot;\theta_h) : \mathcal{G} \xrightarrow{f(\cdot,\theta_g)} \mathcal{G}' \xrightarrow{f(\cdot,\theta_c)} C \xrightarrow{\text{solver}} Z^* \tag{11}$$

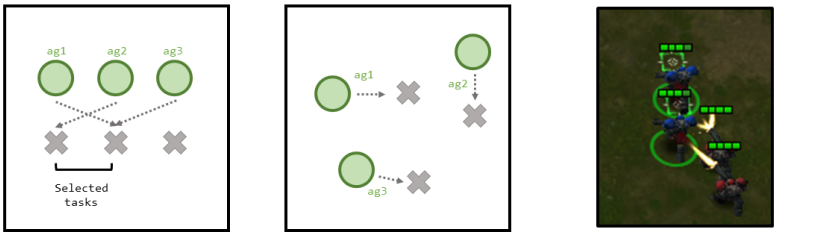
where $\theta_h = (\theta_g, \theta_c)$.

Then, the gradient of the loss function $\mathcal{J}(\theta_h)$ with respect to the policy parameter $\theta_h = (\theta_g, \theta_c)$ can be computed as the following chain rule:

$$\frac{\partial\mathcal{J}(\theta_h)}{\partial\theta_h} = \frac{\partial\mathcal{J}}{\partial Z^*}\cdot\frac{\partial Z^*(C)}{\partial C}\cdot\frac{\partial C}{\partial\theta_h} \tag{12}$$

In our optimization problem, there exists some extreme points generated only by boundary constraints (equation 7). Thus, optimal solution may integer-valued on this extreme point (please refer to Appendix A.1 for more description). In this case, the optimal solution may not vary continuously with respect to the input; a small change can induce an abrupt change in the objective value. Thus, the $\frac{\partial Z^*(C)}{\partial C}$ be in the form of the piecewise constant, making it difficult to estimate the gradient. To solve this issue, Vlastelica et al. (2019) presented a piecewise linear surrogate loss surface that can approximate the original piecewise constant surface $\frac{\partial\mathcal{J}}{\partial C}$ of the combinatorial optimization layer. To obtain a meaningful gradient for this integer-solution, we modify the gradient $\frac{\partial\mathcal{J}}{\partial C} = \frac{\partial\mathcal{J}}{\partial Z^*}\cdot\frac{\partial Z^*(C)}{\partial C}$ by of solution gap between the original solution of the optimization problem and the solution of the perturbed optimization problem, as:

$$\frac{\partial\mathcal{J}}{\partial C} \approx -\frac{1}{\lambda}(Z^* - Z_\lambda^*) \tag{13}$$



(a) Single-step Matching (b) Cooperative Navigation (c) StarCraft Micromanagement

Figure 2: Visual description on the experiment environments

where Z^* is the optimal solution used in forward computation and Z^*_λ is the optimal solution of LP with perturbed coefficient. In a backward pass, equation 13 replaces $\frac{\partial \mathcal{J}}{\partial Z^*} \cdot \frac{\partial Z^*(C)}{\partial C}$ in equation 12. We refer to Appendix A.2 for more details of computation of high-level policy.

5.1 Training low-level actor-critic

The low-level parameters, ϕ_l and θ_l are trained using an independent actor-critic algorithm with the task embedding corresponding to the high-level action is argument with. The training of the low-level parameters can be found in Appendix A.3.

6 Experiment

Figure 2 shows the environments we use to evaluate the performances of the proposed LPMARL algorithm. These environments are designed to measure how efficiently the overall work is well-divided into sub-tasks, and sub-tasks are assigned to agents.

6.1 Single-Step, Static assignment: Single-step grouping environment

The first environment is a single-stage task assignment problem. The environment consists of N agents and M destinations. The high-level policy assigns each agent an index of groups as an action, i.e., $a_i \in \{1, \ldots, M\}, \forall i \in \mathcal{N}$. The goal of the policy is to divide agents into exactly K groups (with $K \leq M$). Agents receive the maximum high-level reward $(+10)$ when they are divided into exactly K groups. Otherwise, agents receive a penalty proportional to the difference between the number of groups selected and K. We implemented the task with $N = 10$, $M = 10$. This environment is designed to evaluate the capability of the high-level policy module in LPMARL in optimally allocating tasks to agents.

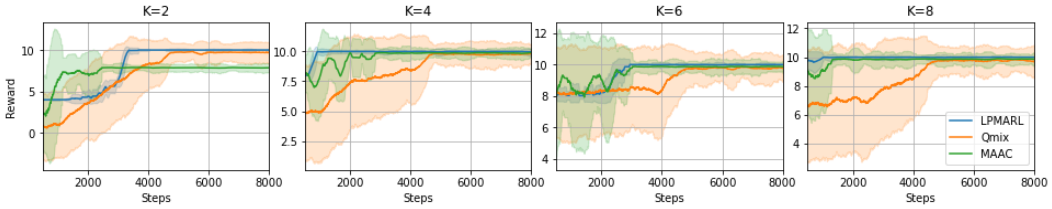


Figure 3: Training curve on single-step grouping environment.

The baseline algorithms we use to compare the performance with LPMARL are MAAC (Diddigi et al., 2019) and QMIX (Rashid et al., 2018), both of which are designed to learn a cooperative policy for the discrete-action space. Figure 3 compares the training curves for all models on the single-stage task allocation problems with the different numbers $K = \{2, 4, 6, 8\}$ of groups required to form. Table 1 reports the winning ratio of different algorithms on each experiment.

Table 1: Winning ratio (%) of MARL algorithms on single-step grouping environment.

#. optimal groups	QMIX	MAAC	LPMARL
$K = 2$	97.2±1.7	0.0±0.0	99.9±0.1
$K = 4$	91.0±2.5	96.2±0.4	100.0±0.0
$K = 6$	95.7±1.5	98.8±0.9	100.0±0.0
$K = 8$	96.8±1.4	99.5±0.8	99.9±0.1

Figure 3 shows that LPMARL effectively learns how to optimally cluster agents into the specified number of groups. LPMARL has the highest average wining ration and the lowest standard deviation of it. Table 1 compares the performance of the algorithms in more detail. It can be seen from the result that LPMARL shows consistent performance even when K is large.

6.2 Multi-Step, Static assignment: Cooperative navigation

The second environment is cooperative navigation, whose objective is to have all agents reach and occupy distinct destinations (Lowe et al., 2017). In this game, each agent is allowed to make s series of navigation actions; therefore, consistent coordination among agents is vital to solving the problem. LPMALR solves this problem by allocating agents to the destination (high-level action) and having each agent navigate itself to the assigned destination (allocation). Thus, this environment is used to mainly verify the effectiveness of the task allocation module of LPMARL in solving the dynamic cooperative game. We consider both dense-reward and sparse-reward settings. In a dense-reward setting, the global reward the agents receive is the combination of the distance between agent and landmarks, a collision between agents, and the number of occupied landmarks. In a sparse-reward setting, agents receive a high-level reward when agents occupy all the landmarks $(+10)$ and receives a low-level reward when the agent occupies the landmark selected by high-level policy $(+10)$. To compare algorithms fairly, individual agents of algorithms other than LPMARL receive low-level (individual) rewards plus high-level (global) rewards.

Table 2: Winning ratio (%) and the percentage of the landmarks occupied (%) on cooperative navigation. In the table, the left column corresponds to the winning ratio, and the right column corresponds to the landmark occupied ratio for each algorithm. The mean and standard deviation of 1,000 repeated experiments is reported.

Number of agents	MADDPG		MAAC		CommNet		**LPMARL**	
$N = 3$ (dense)	85.7±1.1	74.7±10.7	74.2±3.2	69.8±15.1	**86.2**±3.9	85.1±13.2	**88.3**±5.1	**90.8**±7.1
$N = 5$ (dense)	21.3±5.2	54.8±6.4	16.7±3.1	51.3±7.5	28.3±5.9	61.2±8.0	**37.9**±2.8	**72.0**±5.7
$N = 7$ (dense)	5.7±0.1	21.8±4.8	4.1±1.9	31.7±3.0	**8.6**±0.7	43.0±5.1	**8.5**±2.8	**61.3**±10.1
$N = 3$ (sparse)	11.8±1.1	40.7±9.7	13.7±2.4	42.8±10.2	23.2±3.4	75.6±5.1	**30.1**±2.8	**81.8**±5.5
$N = 5$ (sparse)	0.0±0.0	25.8±5.9	1.8±0.0	28.0±9.1	5.2±1.6	35.6±7.8	**11.2**±3.2	**58.7**±8.1
$N = 7$ (sparse)	0.0±0.0	27.0±2.4	0.0±0.0	22.2±5.9	2.9±1.1	34.9±5.2	**5.1**±1.7	**49.8**±6.7

The baseline algorithms we use to compare the performance with LPMARL are MADDPG (Liu et al., 2021), MAAC (Iqbal & Sha, 2019), and CommNet (Sukhbaatar et al., 2016), which are designed to learn multi-agent policy in fully-cooperative setting. We compare algorithms with two performance metrics: winning ratio and the number of landmarks occupied.

Table 6.2 shows the average performance metric of 1,000 repeated evaluation in $N = \{3, 5, 7\}$. In a dense reward setting, all the algorithms have shown similar performance. As the number of the agent increases, the success ratio typically decreases due to the increasing complexity of the game. Nevertheless, LPMARL shows the best performance among the baseline models. In a sparse reward setting, most baseline algorithms except LPMARL have shown significantly reduced performance comparing the dense reward setting. In addition, when the number of agents is large, MADDPG and MAAC result in a zero percent of success ratio. On the other hand, LPMARL can still solve the game even when N is large. In addition, LPMARL has a similar landmark occupied ratio between dense-reward and sparse-reward settings.

6.3 MULTI-STEP, DYNAMIC ASSIGNMENT: STARCRAFT2 MICROMANAGEMENT

The last environment we consider is the StarCraft Multi-Agent Challange (SMAC, Samvelyan et al., 2019), where a team of agents battle against enemy agents. We use sparse reward setting for the game: agents get $r_i^h = 10, \forall i$ when they win a scenario (i.e., when all enemy units are killed), and get $r_i^l = 10$ when the selected enemy a_i^h is killed. When such spare reward is used, it is notoriously difficult to induce the coordinated policy. Thus, SMAC with a sparse reward setting is often used as a challenging environment to develop effective MARL strategies.

Table 3: Winning ratio (%) on Starcraft Micromanagement environment. The mean and standard deviation of 1,000 repeated experiments is reported.

	CMAE	SEAC	QMIX	LPMARL
3m (sparse)	38.2±23.2	0.0±0.0	0.0±0.0	**44.2**±4.7
2m vs 1z (sparse)	**44.3**±20.8	0.0±0.0	0.0±0.0	35.1±5.1
3s vs 5z (sparse)	0.0±0.0	0.0±0.0	0.0±0.0	**0.9**±0.1
3m (dense)	**98.7**±1.7	88.9±0.8	97.9±3.6	98.1±0.4
2m vs 1z (dense)	98.2±0.1	82.5±0.5	97.1±2.4	**100.0**±0.0
3s vs 5z (dense)	**81.3**±16.1	70.8±10.3	75.0±17.6	**81.1**±5.1

The baseline algorithms we consider for this environment are CMAE (Liu et al., 2021) and SEAC (Christianos et al., 2020), the MARL algorithms designed to solve cooperative tasks with sparse reward. We also consider QMIX algorithm that has shown great success in the various SMAC environment with dense rewards. Note that other than CMAE and SEAC, there were no MARL algorithms that have shown s meaningful success in the literature so far.

Table 3 shows the final winning ratio on SMAC tasks. In a dense-reward setting, LPMARL achieved at least similar or better performance than other learning algorithms. Note that in *2m vs. 1z* environment, LPMARL does not conduct the high-level action since there is only a single enemy agent (target); nevertheless, LPMARL shows a good performance, possibly due to the effective state representation module constructed based on GNN.

In a spare-reward setting, SEAC and QMIX show a zero percent winning ratio, which means that these two algorithms cannot learn a cooperative policy using an only spare rewards. On the other hand, SEAC and LPMARL achieve meaningful winning ratios in *3m* and *2m vs 1z* environments. Note that LPMARL has a lower variance of winning ratio than SEAC. In addition, although it is very low, LPMARL occasionally achieves a win for the difficult task *3s vs 5z*.

7 CONCLUSION

In this work, we proposed LPMARL, a linear programming-based hierarchical MARL approach. The high-level decision making of LPMARL is to assign agent into tasks, and formulated as linear programming. The low-level decision making is to solve a cooperative game for each sub-task. The high-level and low-level policy of LPMARL are trained end-to-end using differentiable optimization framework. We demonstrated that LPMARL can decompose the agents into sub-tasks in various MARL environment with sparse-reward setting.

REFERENCES

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, 2017.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Nicolas Carion, Nicolas Usunier, Gabriel Synnaeve, and Alessandro Lazaric. A structured prediction approach for generalization in cooperative multi-agent reinforcement learning. *Advances in neural information processing systems*, 32:8130–8140, 2019.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. *arXiv preprint arXiv:2006.07169*, 2020.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Raghuram Bharadwaj Diddigi, Sai Koti Reddy Danda, Shalabh Bhatnagar, et al. Actor-critic algorithms for constrained multi-agent reinforcement learning. *arXiv preprint arXiv:1905.02907*, 2019.

Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 1504–1511, 2020.

Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 2961–2970. PMLR, 2019.

Naoki Katoh and Toshihide Ibaraki. Resource allocation problems. In *Handbook of combinatorial optimization*, pp. 905–1006. Springer, 1998.

Iou-Jen Liu, Unnat Jain, Raymond A Yeh, and Alexander Schwing. Cooperative exploration for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*, pp. 6826–6836. PMLR, 2021.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

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*, pp. 4295–4304. PMLR, 2018.

Heechang Ryu, Hayong Shin, and Jinkyoo Park. Remax: Relational representation for multi-agent exploration. *arXiv preprint arXiv:2008.05214*, 2020.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29:2244–2252, 2016.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

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.

Alexander Sasha Vezhnevets, Yuhuai Wu, Remi Leblond, and Joel Z Leibo. Options as responses: Grounding behavioural hierarchies in multi-agent rl. *arXiv preprint arXiv:1906.01470*, 2019.

Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. *arXiv preprint arXiv:1912.02175*, 2019.

Tonghan Wang, Jianhao Wang, Yi Wu, and Chongjie Zhang. Influence-based multi-agent exploration. *arXiv preprint arXiv:1910.05512*, 2019.

Tonghan Wang, Heng Dong, Victor Lesser, and Chongjie Zhang. Roma: Multi-agent reinforcement learning with emergent roles. *arXiv preprint arXiv:2003.08039*, 2020a.

Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. Rode: Learning roles to decompose multi-agent tasks. *arXiv preprint arXiv:2010.01523*, 2020b.

Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1658–1665, 2019.

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

A APPENDIX

A.1 VISUALIZATION OF EXTREME POINT OF LP

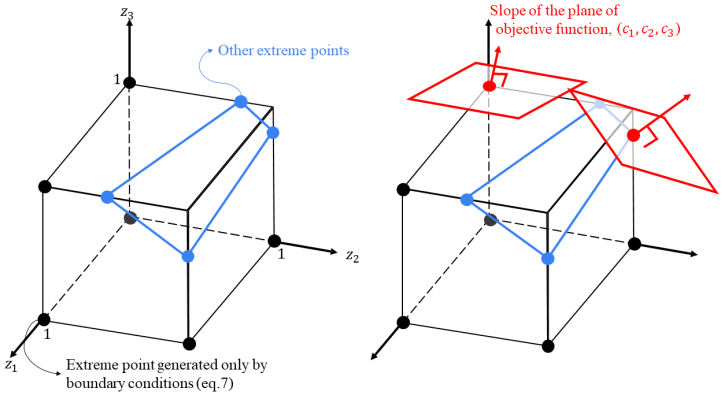


Figure 4: Visual description of extreme points

Figure 4 illustrates the extreme points of the linear programming in 3-dimensional case, where the decision variables are $(z_1, z_2, z_3) \in \mathbb{R}^3$. The boundary constraints are illustrated as the surface of the cube, and additional constraints other than boundary constraints are illustrated as the surface generated by blue lines. In figure 4 (left), extreme points corresponding only to boundary constraints are marked as black dots, and other extreme points are marked as blue dots.

The objective of the optimization is to maximize $f(\boldsymbol{z}) = c_1 z_1 + c_2 z_2 + c_3 z_3$. The objective coefficients, (c_1, c_2, c_3) determines the slope of the plane of the objective function, as illustrated as the red arrow on figure 4. In figure 4, we consider possible two optimization outcomes. For case (1), the optimal solution $z^* = (z_1^*, z_2^*, z_3^*)$ occurs on the extreme point only generated by boundary points, and for case (2): z^* occurs on the other extreme points. In case (1), the optimal solution z^* is integer-valued. In our LP formulation, we have $N \times M$ boundary constraints and $N + M$ additional constraints. Thus, depending on the value of the objective coefficient and constraint coefficients, there may exist some integer-valued solution. In addition, the solution of the LP changes discontinuously changes on the continuous change on the slope of the objective plane. For this reason, we need continuous interpolation of gradient of discontinuous surface $\frac{\partial \mathcal{J}}{\partial C}$.

A.2 TRAINING OF HIGH-LEVEL POLICY

The gradient of the high-level policy loss function $\mathcal{J}(\theta_h)$ with respect to the policy parameter $\theta_h = (\theta_g, \theta_c)$ can be computed as the following chain rule:

$$\frac{\partial \mathcal{J}(\theta_h)}{\partial \theta_h} = \frac{\partial \mathcal{J}}{\partial Z^*} \cdot \frac{\partial Z^*(C)}{\partial C} \cdot \frac{\partial C}{\partial \theta_h} \tag{14}$$

We can compute the gradient $\frac{\partial Z^*(C)}{\partial \theta_h} = \frac{\partial Z^*(C)}{\partial C} \cdot \frac{\partial C}{\partial \theta_h}$ by differentiating the equality of the KKT conditions. The previous works (Wilder et al., 2019; Ferber et al., 2020) proposed to add additional quadratic regularization term $\gamma||z||$ in the objective function (equation 4) to make LP as smooth QP and induce non-singular Jacobian matrix, to differentiate through the equality of the KKT conditions as in Amos & Kolter (2017).

However, although our high-level optimization problem is formulated as an LP, there exist extreme points generated only by boundary constraints (equation 7). Thus, optimal solution may integer-valued on this extreme point (please refer to Appendix A.1 for more description). In this case, the optimal solution may not vary continuously with respect to the input; a small change can induce an abrupt change in the objective value. Thus, the $\frac{\partial Z^*(C)}{\partial C}$ be in the form of the piecewise constant, making it difficult to estimate the gradient. To solve this issue, Vlastelica et al. (2019) presented a piecewise linear surrogate loss surface that can approximate the original piecewise constant surface $\frac{\partial \mathcal{J}}{\partial C}$ of the combinatorial optimization layer. To obtain a meaningful differential value for this

integer-solution, we modify the gradient $\frac{\partial \mathcal{J}}{\partial C} = \frac{\partial \mathcal{J}}{\partial Z^*} \cdot \frac{\partial Z^*(C)}{\partial C}$ by of solution gap between the original solution of the optimization problem and the solution of the perturbed optimization problem, as:

$$\frac{\partial \mathcal{J}}{\partial C} \approx -\frac{1}{\lambda}(Z^* - Z^*_\lambda) \tag{15}$$

where Z^* and C are the optimal solution and coefficient used on the forward pass, respectively. Z^*_λ is the approximate solution computed as

$$Z^*_\lambda = Z^*(C'_\lambda). \tag{16}$$

Here, C'_λ is the perturbed cost coefficient computed as

$$C'_\lambda := C + \lambda \cdot \frac{d\mathcal{J}}{dZ}(Z^*) \tag{17}$$

where λ is the hyperparemeter scaling the amount of perturbation considering the gradient respect to the solution Z^*. $\frac{d\mathcal{J}}{dZ}(Z^*)$ is the gradient of $\mathcal{J}$ with respect to solver output Z at given point Z^*. The amount of solution gap between original solution and perturbed solution $Z^* - Z^*_\lambda$ is the slope of piecewise linear function that will replace $\frac{\partial \mathcal{J}}{\partial C}$ evaluated at C. One can guarantee that the modified loss function is piecewise affine and similar to the original loss function (we refer to Vlastelica et al., 2019 for more detail). In a backward pass, equation 15 replaces $\frac{\partial \mathcal{J}}{\partial Z^*} \cdot \frac{\partial Z^*(C)}{\partial C}$ in equation 12.

A.3 Training of Low-level parameters

The objective of the low-level policy is to maximize a sum of discounted individual low-level return when assigned to a task, $\mathcal{J}(\theta_l) = \mathbb{E}[\sum_t \sum_i \gamma^t r^l_{t,i}] \approx \mathbb{E}_{(\boldsymbol{s}, a^h_{t,i}, a^l_{t,i}, r^l_i, \boldsymbol{s}') \sim D}[\sum_i Q^l(\boldsymbol{s}_t, a^l_{t,i} | a^h_{t,i}; \phi_l)]$.

The low-level critic is trained to minimize the mean squared error between predicted low-level critic value and target value,

$$\mathcal{L}(\phi_l) = \mathbb{E}_{(\boldsymbol{s}, a^h_i, a^l_i, r^l, \boldsymbol{s}') \sim D} \left[\sum_i \left(Q^l(h^A_{t,i}, h^T_{t,a^h_{t,i}}, a^l_{t,i}; \phi_l) - y \right)^2 \right] \tag{18}$$

where y is the target computed as $y = r^l_{t,i} + \gamma \cdot \max_{a^l_{t,i}} Q^l(h^A_{t+1}, h^T_{t+1,a^h_i}, a^l_{t+1,i}; \bar{\phi}_l)$. The low-level policy either can be continuous or discrete. For continuous policy, we use deep deterministic policy gradient method,

$$\nabla_{\theta_l} \mathcal{J}(\theta_l) = \mathbb{E}_{(\boldsymbol{s}, a^l_i, a^h_i, \boldsymbol{s}', r^l_i) \sim D} \left[\nabla_{a^l_i} Q^l(h^A_i, h^T_{a^h_i}, a^l_i; \phi_l) \nabla_{\theta_l} \pi^l(a^l_i | h^A_i, h^T_{a^h_i}; \theta_l) \right] \tag{19}$$

For discrete policy, we train policy using actor-critic,

$$\nabla_{\theta_l} \mathcal{J}(\theta_l) = \mathbb{E}_{(\boldsymbol{s}, a^l_i, a^h_i, \boldsymbol{s}', r^l_i) \sim D} \left[\nabla_{\theta_l} \{ Q^l(h^A_i, h^T_{a^h_i}, a^l_i; \bar{\phi}_l) - V^l(h^A_i, h^T_{a^h_i}) \} \cdot \nabla_{\theta_l} \pi^l(a^l_i | h^A_i, h^T_{a^h_i}; \theta_l) \right] \tag{20}$$

As both high-level actor and low-level actor takes as input the hidden embedding of the graph, we need to either freeze the parameters of GNN or train whole networks together. Therefore, we set the global loss as the weighted sum of low-level parameters and high-level parameters, $\mathcal{L} = w \cdot (\nabla_{\theta_l} \mathcal{J}(\theta_l) + \mathcal{L}(\theta_l)) + (1 - w) \cdot (\nabla_{\theta_h} \mathcal{J}(\theta_h) + \mathcal{L}(\theta_h))$. This also accelerates the policy training. At the beginning of the training, high-level policies are harder to train than low-level policies because high-level rewards are more sparse than low-level rewards. We set $w = 1$ initially and lineally decayed to $w = 0$ at the end of the training.

A.4 Hyperaparemeters

The hyperparameters of LPMARL used in the experiment are summarized in table 4. We used an CVXPY solver (Diamond & Boyd, 2016) to solve the linear programming in section 4.

For other baselines algorithms, we used 2-layered MLP with LeakyReLU activation for every policy network and actor network. Additionally , 4 attention heads are considered on MAAC. The optimizer, learning rate, and discounting rate γ are set to be the same as the LPMARL, as in table 4. As the batch size, training steps, and replay buffer size are different for each experiment, these are given in Appendix A.5.

Table 4: Hyperparemeters of LPMARL

Hyperparemter	Values
MLP units for GNN, $f(\cdot;\theta_g^{\mathcal{N}}), f(\cdot;\theta_g^{\mathcal{M}}), f(\cdot;\theta_g^{v})$	[32,32]
MLP units for coefficient matrix, $f(\cdot;\theta_c)$	[64,64]
MLP units for policy network, $f(\cdot;\theta_h), f(\cdot;\theta_l)$	[64,64]
MLP units for critic network $f(\cdot;\phi_h), f(\cdot;\phi_l)$	[64,64]
Nonlinear activation	LeakyReLU, negative slope=0.01
learning rate	10^{-3}
Discount rate, γ	0.99
λ	40
Optimizer	Adam

A.5 Details about Experiments

A.5.1 Single-step grouping environment

We run 10,000 episodes (1 step per episode) for training each algorithm. For every algorithms, we set the size of the replay buffer as 500, and set batch size as 100. Agents receive $R = 10$ When $K = |\{a_i\}_{i\in\mathcal{N}}|$ (agents are exactly divided into K groups), and get reward $R = 10 - 2 \cdot |K - |\{a_i\}_{i\in\mathcal{N}}||$.

A.5.2 Cooperative navigation

We used 50,000 episodes with 50 timesteps. For every algorithm, we set the size of the replay buffer as 5,000 with a batch size of 100 to train. For LPMARL, we set k_i, the capacity constraint coefficient of 5 as 1 at every scenario. In addition, we decayed the w by 5×10^{-5} at every episode.

The Ornstein-Uhlenbeck process with $\mu = 0.15$ and $\sigma = 0.3$ is employed for exploration. The initial location of agents and landmarks are randomly spawn on $[-1, 1] \times [-1, 1]$.

A.5.3 StarCraft Multi-Agent Challenge

We train all models over 10,000 episodes. For every algorithm, we set the size of the replay buffer as 5,000 with a batch size of 100 to train. For LPMARL, we set k_i, the capacity constraint coefficient of equation 5 same to $\mathcal{M}$ at every scenario. In addition, we decayed the w by 2×10^{-4} at every episode.

We considered three SMAC scenarios:

- *3m*: 3 marines ($N = 3$) versus 3 marines ($M = 3$). The episode limit is 60 timesteps.
- *2m vs 1z*: 2 marines ($N = 2$) versus 1 zealot ($M = 1$). The episode limit is 150 timesteps.
- *3s vs 5z*: 3 stalker ($N = 3$) versus 5 zealots ($M = 5$). The episode limit is 150 timesteps.

Difficulty levels of scenarios are all set to be *very difficult (7)*.