Off-Beat Multi-Agent Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

We investigate model-free multi-agent reinforcement learning (MARL) in environments where *off-beat* actions are prevalent, *i.e.*, all actions have pre-set execution durations. During execution durations, the environment changes are influenced by, but not synchronised with, action execution. Such a setting is ubiquitous in many real-world problems. However, most MARL methods assume actions are executed immediately after inference, which is often unrealistic and can lead to catastrophic failure for multi-agent coordination with off-beat actions. In order to fill this gap, we develop an algorithmic framework for MARL with off-beat actions. We then propose a novel episodic memory, LeGEM, for model-free MARL algorithms. LeGEM builds agents' episodic memories by utilizing agents' individual experiences. It boosts multi-agent learning by addressing the challenging temporal credit assignment problem raised by the off-beat actions via our novel reward redistribution scheme, alleviating the issue of non-Markovian reward. We evaluate LeGEM on various multi-agent scenarios with off-beat actions, including Stag-Hunter Game, Quarry Game, Afforestation Game, and StarCraft II micromanagement tasks. Empirical results show that LeGEM significantly boosts multi-agent coordination and achieves leading performance and improved sample efficiency.

1 Introduction

In Multi-Agent Reinforcement Learning (MARL), multiple agents act interactively and complete tasks in a sequential decision-making manner with Reinforcement Learning (RL). It has made remarkable advances in many domains, including autonomous systems [8, 19, 72] and real-time strategy (RTS) video games [58]. By the virtue of the *centralised training with decentralised execution* (CTDE) [33] paradigm, which aims to tackle the scalability and partial observability challenges in MARL, many CTDE-based MARL methods are proposed [13, 49, 41, 62, 47, 63, 23, 35]. With these methods, an agent executes actions only via feeding its individual observations independently and optimizes its own policy with access to global trajectories centrally.

Despite the recent successes of MARL, learning effective multi-agent coordination policies for complex multi-agent systems remains challenging. One key challenge is the *off-beat* actions, *i.e.*, all actions have pre-set execution durations[1] and during the execution durations, the environment changes are influenced by, but not synchronised with, action execution (an illustrative scenario is shown in Fig. 1). However, Dec-POMDP [32], which underpins many CTDE-based MARL methods, hinges on the assumption that actions are executed momentarily after inference, leading to catastrophic failure for *centralized training* on various off-beat multi-agent scenarios (OBMAS). To fill this gap, we study MARL in settings where off-beat actions are prevalent. Such setting is very common in many real-world problems. For example, in the traffic light control problem, traffic lights in the conjunctions of the road network have pre-set execution time which is set asynchronously.

[1] In the RL literature [39, 6], action execution durations are called *delays of actions*. In this paper, we use the term *execution durations*, which is self-consistent with off-beat actions defined in Sec. 3.

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

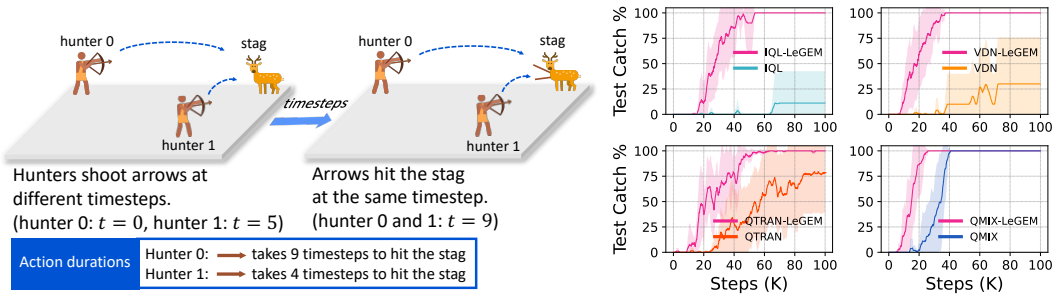

Figure 1: **An illustrative scenario:** two-agent stag-hunter game, where two agents (hunters) have only partial observations, different durations of the shoot action, and cannot communicate. The goal is to catch the stag and they are rewarded when their shot hits – as in, completion of the action is synchronised, the stag at the same time. Both agents can see the stag. As the shoot action durations of the two agents are different, to catch the stag, the two agents should shoot the arrow at different timesteps given the distances. Though the scenario is easy for human beings, it is hard for MARL agents due to the action duration. **Experiment results:** in this scenario, the optimal policy for agent 0 is to shoot the arrow at timestep 0 while the optimal policy for agent 1 is to shoot the arrow at timestep 5. Such asynchronous property of OBMAS motivates agents to learn tacit policies. The curves show that VDN and IQL fail to learn coordination policies even in this simple scenario. With LeGEM, MARL methods gain enhanced performances as well as improved sample efficiency.

The problem of off-beat actions in MARL has yet to be investigated and tackled. Training MARL policies in OBMAS is challenging: (i) Each agent's actions can have a variety of execution durations, which augments the order of complexity of OBMAS during decentralized execution, resulting in failure of the coordination; (ii) The action durations are unknown to agents during individual executions, and communication is constrained and not always feasible, making it non-trivial to model the environment; (iii) During training, both the temporal credit assignment with TD-learning [51] and the *inter-agent* credit assignment with value decomposition methods [41] cannot perform well due to the displaced rewards in multi-agent replay. With off-beat actions, the nonstationarity issue, which mainly stems from rewards' time dependency on the agents' past actions, is exacerbated.

While actions durations are ubiquitous, existing works only focus on single-agent settings, *i.e.*, delay, in RL. Many approaches [59, 39, 66] augment the state space with the queuing actions to be executed into the environment. However, such state-augmentation trick leads to exponentially increasing training samples with the growing action duration, making training intractable [11]. Chen et al. [10] extend the delayed MDP [39] and propose Delayed Markov Game for MARL. However, on one hand, such state-augmentation treatment is confined to short delays, *e.g.*, one timestep delay; on the other hand, the delayed timestep of the actions is privileged information, which is not available in many scenarios. Recent works on macro-actions [67, 68] introduce asynchronous actions by designing macro-actions with prior environment knowledge. Macro-actions are different from options in hierarchical RL (HRL) [52, 3] in that the later is not manually designed but learned. The key difference between macro-actions and off-beat actions is that macro-actions are high-level actions while off-beat actions are primitive actions. Unfortunately, the *inter-agent* credit assignment is still a challenge of HRL in OBMAS and the asynchronous [2] nature of off-beat actions undermines the temporal credit assignment of *centralized training*, causing poor sample efficiency and unsatisfactory performance (more discussions can be found in the related works section in Sec. 7).

We aim to address the aforementioned issues. We first propose off-beat Dec-POMDP. We then instantiate a new class of episodic memory, LeGEM, for model-free MARL algorithms. LeGEM boosts multi-agent learning by addressing the challenging temporal credit assignment problem raised by the off-beat actions via our novel levelled graph-based temporal recency reward redistribution scheme. Specifically, each agent maintains LeGEM and during centralized training, each agent searches the pivot timestep given observations from its graph. The pivot timestep is the timestep wherein the off-beat reward relates to the given node. The pivot timesteps of each agent are ranked, in which the final pivot timestep will be chosen by recency and later used for reward redistribution and target estimation in TD-learning. We evaluate our method on Stag-Hunter Game, Quarry Game, Afforestation Game, and StarCraft II micromanagement tasks. Empirical results show that our method significantly boosts multi-agent coordination and achieves leading performance as well as improved sample efficiency.

[2] We clarify the term *asynchronous*: actions that simultaneously committed into the environment by all agents in MARL will not complete their respective action durations at the same time in future timesteps.

2 Preliminaries

Dec-POMDP. A cooperative MARL problem can be modeled as a *decentralised partially observable Markov decision process* (Dec-POMDP) which can be formulated as a tuple $\langle \mathcal{S}, \mathcal{U}, \mathcal{P}, R, O, \mathcal{N}, \gamma \rangle$, where $\boldsymbol{s} \in \mathcal{S}$ denotes the state of the environment. Each agent $i \in \mathcal{N} := \{1, ..., N\}$ chooses an action $u_i \in \mathcal{U}$ at each timestep, forming a joint action vector, $\boldsymbol{u} := [u_i]_{i=1}^N \in \mathcal{U}^N$. The Markovian transition function can be defined as $\mathcal{P}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{u}) : \mathcal{S} \times \mathcal{U}^N \times \mathcal{S} \mapsto [0, 1]$, transiting one state of current timestep to the state of next timestep conditioned on current state and joint action. Every agent shares the reward and the reward function is $R(\boldsymbol{s}, \boldsymbol{u}) : \mathcal{S} \times \mathcal{U}^N \mapsto \mathcal{R}$. $\gamma \in [0, 1)$ is the discount factor. Due to *partial observability*, each agent has individual partial observation $o \in \mathcal{O}$, according to the observation function $O(\boldsymbol{s}, i) : \mathcal{S} \times \mathcal{N} \mapsto \mathcal{O}$. The goal of each agent is to optimize its own policy $\pi_i(u_i|\tau_i) : \mathcal{T} \times \mathcal{U} \mapsto [0, 1]$ given its action-observation-reward history $\tau_i \in \mathcal{T} := (\mathcal{O} \times \mathcal{U})$.

Multi-Agent Reinforcement Learning. MARL aims to learn optimal policies for all the agents in the team. With TD-learning and a global Q value proxy Q^{tot} for the optimal Q^*, $\{Q_i\}_{i=1}^N$ are optimized via minimizing the loss [65, 31]: $\theta^* = \arg\min_{\theta^*} \mathcal{L}(\theta) := \mathbb{E}_{D' \sim \mathcal{D}}[(y_t^{\text{tot}} - Q_\theta^{\text{tot}}(\boldsymbol{s}_t, \boldsymbol{u}_t))^2]$, where $y_t^{\text{tot}} = r_t + \gamma \max_{\boldsymbol{u}'} Q_{\bar{\theta}}^{\text{tot}}(\boldsymbol{s}_{t+1}, \boldsymbol{u}')$ and θ is the parameters of the agents. $\bar{\theta}$ is the parameter of the target Q^{tot} and is periodically copied from θ. D' is a sample from the replay buffer $\mathcal{D}$.

3 Off-Beat Dec-POMDP

We introduce our formulation for OBMAS. We first define the off-beat actions[3] for multi-agent scenarios; then we propose the Off-Beat Dec-POMDP. All the proofs can be found in Appx. A.

Definition 1 (Off-Beat Actions)**.** Off-beat action $\tilde{u} \in \mathcal{U}$ characterizes OBMAS where the action $\tilde{u}_i$ taken by agent i has execution duration $m_{\tilde{u}_i} \sim A(m|\tilde{u}_i, i)$, $A \in \mathcal{A}$, $m \in \{0, 1, 2, \cdots, M\}$ and $M \leq T$, where T is the maximum duration and A is the action duration distribution. It is a distribution and takes $\tilde{u}_i$ and the index of the agent as parameters. A can be either stochastic or deterministic. The joint off-beat action is $\tilde{\boldsymbol{u}} = [\tilde{u}_i]_{i=1}^N$. The execution duration is decided at the time the action was committed to the environment. Thus, the execution duration of an action $\tilde{\boldsymbol{u}}_t$ initiated at timestep t is $\boldsymbol{m}_t = \{m_{\tilde{u}_i^t}^t\}_{i=1}^N$.

Note that for each agent, $m_{\tilde{u}_i^t}^t$[4] can be different. At timestep t, there are at least 1 action [5] and at most N actions being initiated (committed to the environment for execution), leading to asynchronicity of the joint actions. Next, we propose the Off-Beat Dec-POMDP for OBMAS and discuss its properties.

Definition 2 (Off-Beat Dec-POMDP)**.** Off-Beat Dec-POMDP extends Dec-POMDP, such that
(1) state space is $\mathcal{S}$; (2) joint action space is $\mathcal{U}^N$; (3) action duration space is $\mathcal{A}^N$;
(4) transition function is $\mathcal{P}(\boldsymbol{s}'|\boldsymbol{s}, \tilde{\boldsymbol{u}}, \boldsymbol{m}) : \mathcal{S} \times \mathcal{U}^N \times \mathcal{S} \times \mathcal{A}^N \mapsto [0, 1]$, and $\boldsymbol{m}$ is the action durations of the joint action;
(5) the reward function is $R(\boldsymbol{s}, \tilde{\boldsymbol{u}}, \boldsymbol{m}) : \mathcal{S} \times \mathcal{U}^N \times \mathcal{A}^N \mapsto \mathcal{R}$;
(6) we call a reward r as off-beat reward when any its $m_{\tilde{u}_i} \geq 1$, $m_{\tilde{u}_i} \in \boldsymbol{m}$, and $r \neq 0$.

In OBMAS, at each timestep t, the environment receives actions that agent initiates for execution in the environment. The initiated actions $\tilde{\boldsymbol{u}}_t$ are instantaneous actions inferred by agents' policies given individuals' observations. The joint reward is the consequence of the committed joint actions of current timestep and previous timesteps, depending on the actions' duration. The asynchronicity is an inherent feature of the environment, which is different from asynchronicity incurred by communication delays in many video games (asynchronous gameplay[6]). We discuss some properties of Off-Beat Dec-POMDP below.

[3] Asynchronicity is prevalent in real-world multi-agent scenarios, including asynchronicity in observations, actions and communication, etc. In this paper, we focus on the asynchronicity of actions in multi-agent scenarios. For brevity, we name the asynchronicity of actions in MARL as *off-beat*.

[4] We will omit t in the rest of the paper for brevity.

[5] We note that agents have a special NO-OP action available.

[6] `https://www.whatgamesare.com/2011/08/synchronous-or-asynchronous-definitions.html`

Remark 1. *When the durations for all actions are identical, off-beat Dec-POMDP reduces to Delayed Dec-POMDP and there is no off-beat actions in it.*

Remark 2. *There exists $\tilde{\boldsymbol{u}}$ that is synchronous since duration of agents' actions can be $m = 0$. When m of all actions is $zero$, off-beat Dec-POMDP reduces to Dec-POMDP.*

In Delayed Dec-POMDP, actions have the same delayed timesteps, which is different from off-beat actions where actions have different action durations or delays. In order to investigate the problem, we consider the deterministic setting of the transition function and the reward function.

Remark 3 (Non-episodic Reward)**.** *In our formulation, the reward is not episodic reward [16].*

Remark 4 (Non-Markovian Reward)**.** *With off-beat actions, the Markovian property of the reward function $R(\boldsymbol{s}, \tilde{\boldsymbol{u}}, \boldsymbol{m})$ does not hold.*

With off-beat actions, the shared rewards can be readily displaced, causing non-Markovian rewards. Solving Off-Beat Dec-POMDP is challenging as discussed in Sec. 1. We propose our methods to tackle aforementioned challenges.

4 The Journey is the Reward: A Collective Mental Time Travel Method

We propose two methodological elements for Off-Beat MARL. The first, LeGEM, presented in this section, is a form of episodic memory that facilitates discovery of a pivotal timestep for off-beat rewards; and the second, presented in Sec. 5, is redistribution of the off-beat reward to the pivot timestep when the relevant off-beat actions were initialised.

4.1 LeGEM: A Levelled Graph Episodic Memory for Off-Beat MARL

Human learning relies on retrospecting our detailed memory of the past [55, 48]. For example, while exploring a new scenic area, we do not just remember a multitude of specific spots there, but can recall the paths that connect them with junctions and turns. However, there is no MARL method that can explicitly recall the past and identify key states that lead to future rewards. Such "*mental time travel*" [24] ability is vital for tackling the challenges in OBMAS. Inspired by the recent progress in RL with episodic memory [18, 5, 17] that is based on the memory prosthesis proposed by neuroscientists [55, 48], we propose our method of episodic memory representation for MARL. Unlike previous episodic memory methods that train a parameterized memory by either augmenting the policy inputs for execution [18] or regularizing the TD learning [17] for RL, our method utilizes the levelled graph data structure [4], a well established structure for data storage and retrieval, to represent an agent's individual episodic memory.

We propose our novel episodic memory, Levelled Graph Episodic Memory method (LeGEM), via the levelled graph data structure. LeGEM memorizes each agent's past trajectories which are partial observations and the unilateral action of the agent. During training, each agent i collects its individual trajectories τ_i. We then define τ_i of agent i as $\tau_i = [(o_i^0, \tilde{u}_i^0, r^0), \cdots, (o_i^{T-1}, \tilde{u}_i^{T-1}, r^{T-1})]$, where T is the length of the trajectory and the triplet $(o_i^t, \tilde{u}_i^t, r^t)$ represents the observation, action and reward of timestep t. Note that r^t is globally shared between agents. We define agent i's LeGEM as a directed graph $\phi_i^t \in \Phi_i$ where Φ_i is the set of graphs of agent i and ϕ_i^t is the t-th graph of Φ_i, $t \in \{0, \cdots, T-1\}$. Each ϕ_i^t consists of a tuple of (Ψ, Ξ) where Ψ is the set of nodes and Ξ represents the set of edges that connect nodes in the graph. To model an agent's behaviour explicitly and make the trajectories of agents easy to represent, we create T graphs for each agent and let $\Phi_i = \{\phi_i^t\}_{t=0}^{T-1}$ where T is the maximum level of all graphs and the maximum length of the episode as well. The maximum level of ϕ_i^t is $t+1$. The node contains key, visit count and pointers connecting the precursors (node at the previous level) and the successors (node at the next level). Unlike many parameterized episodic memory using state/observation as the key [18, 24], we resort to *afterstate* [36]. That is, we use agent i's observation o_i^t and action at timestep t, $\tilde{u}_i^t$, to define the key $(o_i^t, \tilde{u}_i^t)$. We provide an example to showcase the relationship between sub-graph and the graph in Fig. 2. For complex and continuous state scenarios, for example StarCraft II scenarios, we

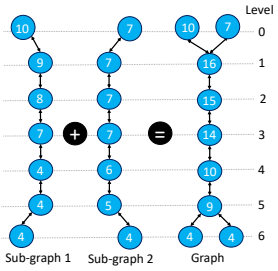

Figure 2: The maximum level of the graph is 7. Circles indicate the nodes and numbers indicate the visit count.

Algorithm 1: `SearchPivotTimesteps` (ρ)

Input: τ, Φ, Υ and `Search` (scheme I or II);
Initialize: κ: an empty list to store pivot timesteps;
`// Length of` τ `and` τ_i `are equal.`
$l \leftarrow$ `length(`τ_i`)-1`;
for $t \leftarrow 0$ ***to*** `length(`τ`)-1` **do**
  if $r^t \neq 0$ ($r^t \in \tau$) **then**
    `// Off-beat reward`
    for $i \leftarrow 1$ ***to*** N **do**
      Get τ_i from τ;
      $\phi_i^l \leftarrow \Phi_i$`[`$l$`]`;
      $\psi \leftarrow \phi_i^l$`.getNode(`$o_i^t, \tilde{u}_i^t$`)`;
      Find all the paths $\Lambda_i^{t,l}$ from node ψ to the node at level 0;
      Get the discretized episode return $\boldsymbol{r}^{l,i}$;
      Get the index ω from Υ with $\boldsymbol{r}^{l,i}$;
      $e_t^i \leftarrow$ `Search(`$\omega, \Lambda_i^t, \tau_i, \boldsymbol{r}^{l,i}, \Upsilon, \Phi_i$`)`;
    Get e_t (Eqn. 1) and append e_t to κ;
Return: κ.

Algorithm 2: Search Scheme I

Input: ω, $\Lambda_i^{t,l}$, τ_i, $\boldsymbol{r}^{l,i}$, Υ and Φ_i;
Initialize: $\boldsymbol{e}_t^i$: a list whose values are all t and its size is the number of paths in $\Lambda_i^{t,l}$;
$\phi_i^{l,\omega} \leftarrow \Phi_i^{l,\Omega}$`[`$\omega$`]`;
vc $\leftarrow$ `VisitCount(`$\Lambda_i^{t,l}$`)` (Alg. 4);
foreach *path* $\Lambda_i^{t,l}$`[`j`]` $\in \Lambda_i^{t,l}$ **do**
  $e_t^{i,j,\downarrow} \leftarrow$ `UL(`$\Lambda_i^{t,l}$`[`j`]`, vc, τ_i`)` (Alg. 5);
  $e_t^{i,j,\uparrow} \leftarrow$ `LU(`$\Lambda_i^{t,l}$`[`j`]`, vc, τ_i`)` (Alg. 6);
  if $e_t^{i,j,\downarrow} \neq -1$ **then**
    $\boldsymbol{e}_t^i$`[`j`]` $\leftarrow e_t^{i,j,\downarrow}$;
  else if $e_t^{i,j,\uparrow} \neq -1$ **then**
    $\boldsymbol{e}_t^i$`[`j`]` $\leftarrow e_t^{i,j,\uparrow}$;
  else
    $\boldsymbol{e}_t^i$`[`j`]` $\leftarrow t$;
$e_t^i \leftarrow$ `Summarize(`$\boldsymbol{e}_t^i$`)` (Alg. 7) ;
Return: e_t^i.

use SimHash [9] to discretize the key $(o_i^t, \tilde{u}_i^t)$. This technique has been widely used in commercial search engines and RL [54]. Visit count indicates the total visits made by agent i to the node. It initial value is 1. Note that nodes are bidirectional since it is helpful for searching (see Sec. 4.2).

Given a τ_i with the length of T, if the node is already in the graph at level t, we then increase the visit count by 1. Otherwise, we create a new node for level t of the graph and update its pointers. Meanwhile, sub-graphs will be also created and updated. The process of updating LeGEM is in Alg. 3. We provide an example of Alg. 3 in Fig. 9, Appx. B.1. It is worth noting that τ_i is generated via the interaction of the agent with the environment, and there is no extra interaction needed to collect τ_i. The generated trajectories are saved in the experience replay and later sampled for MARL training.

4.2 Multi-Agent Collective Mental Time Travel with LeGEM

With structured agent's past experiences, it can be used to search the pivot timestep when actions that triggered the rewarded state were executed. For example, with LeGEM, we can find the pivot timestep, $e_t = 5$, when agent 1 shoots the arrow in Fig. 1.

Fact 1. (Action-Reward Association) *When an off-beat reward r_t exists in the trajectory τ_i ($i \in \{1, \cdots, N\}$), $r_t \in \tau_i$, off-beat action $\boldsymbol{u}_{t'}$ exists ($t' < t$) in the trajectory set $\{\tau_j\}_{j=1}^N$, where $\{\tau_j\}_{j=1}^N$ constitutes the global trajectory of all agents.*

As the reward function and transition function are deterministic in our setting, Fact 1 holds. Intuitively, once we find an off-beat reward in a trajectory, we are sure that the action which triggered the reward can be found in the trajectory. With more experiences collected by the agents, such pattern is obvious and significant. It motivates us to propose a method to leverage the association property of the off-beat action-reward data and search the pivot timestep for timesteps when off-beat rewards occur, which can further help to redistribute the reward backward to mitigate the temporal credit assignment issue (c.f. Sec. 5). Therefore, we first propose a search method to search the pivot timestep and then propose a proximal ranking method to estimate the pivot timestep that invokes the future reward.

Collective Mental Time Travel. The displaced rewards in the replay buffer hinder multi-agent learning. It is essential for each agent to search the pivot timestep when the potential off-beat action that triggered the rewarded state was committed to the environment. Therefore, we propose two search schemes to find the pivot timestep for all agents given an off-beat reward.

Scheme I: For agent i, given $r_t \in \tau_i$, episode return $\boldsymbol{r}^{l,i}$ of τ_i, $\phi_i^l = \Phi_i$`[`l`]` and $\phi_i^{l,\omega} = \Phi_i^{l,\Omega}$`[`$\omega$`]` , agent i searches from the node (the key is $(o_i^t, \tilde{u}_i^t)$ and $o_i^t \in \tau_i$, $u_i^t \in \tau_i$) at level t in sub-graph $\phi_i^{l,\omega}$ to find the pivot timestep e_t for r_t. Concretely, we propose our bi-directional search method. The first one is called Low-Up (LU) search, which traverses from the given node at level t upwards to the node at level 0. The second one is named Up-Low (UL) search which traverses from the node at level 0 downwards to the given node at level t. LU traversing ends when the pattern of increasing visit count

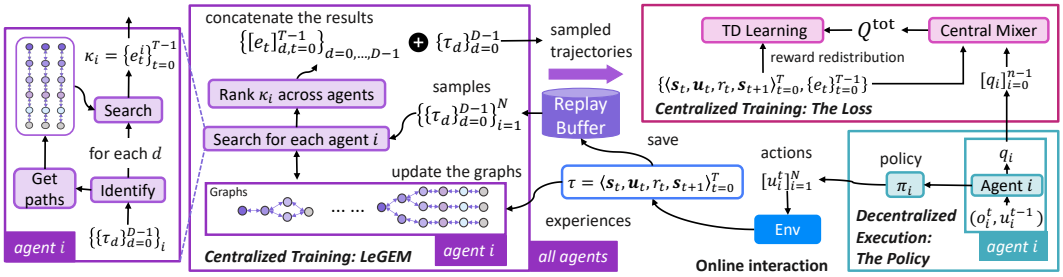

Figure 3: Our framework: LeGEM, the loss and the agent's policy.

ends and the corresponding level is the candidate pivot timestep. On the contrary, UL traversing ends when the pattern of decreasing visit count ends and the corresponding level is the candidate pivot timestep. In Alg. 2, we first get visit count (Line 4) and then apply UL traversing (Line 6) and LU traversing (Line 7). We summarize the results (Line 14) by select the pivot timestep that has the maximum count. UL traversing has a higher searching priority than its counterpart. The reason is that there exists pattern that the visit count is decreasing from the node at level 0 and such pattern ends at the pivot timestep. In practise, it works well in scenarios whose trajectories are single-off-beat-reward trajectories (there is only one off-beat reward) and the accuracy of Scheme I is over 90% in grid world scenarios. For scenarios, especially complex scenarios, whose trajectories are multiple-off-beat-reward trajectories, we apply Scheme II. We put Alg. 4, Alg. 5, Alg. 6 and Alg. 7 in Appx. B.1 as these algorithms are intuitive and easy to understand literally. The time complexity is $\mathcal{O}(n \cdot m)$ (a slight notation abuse) where n is the size of each $\Lambda_i^{t,l}$ and m ($1 \leq m \leq n$) is the average distance between the level of the given node to the level of the node at the pivot timestep.

Scheme II: Scheme II is a simplified version of scheme I for scenarios that have multiple-off-beat-reward trajectories, which searches the pivot timestep by finding the nearest timestep in the most visited path. The node of the nearest timestep has the maximum visitcount in that path. Despite the simplicity, it works effective and the time complexity is $\mathcal{O}(n)$ where n is the number of paths in $\Lambda_i^{t,l}$. The pseudo code is shown in Alg. 8 in Appx. B.1.

Given a node at level t, agents collectively search from the node to find the pivot time step (Line 13 in Alg. 1). The visit count is vital for search methods. In MARL, we use ϵ-greedy [31] for agents to explore the environment and collect individual trajectories. The collected trajectories will be used to build the memory and train the policy. We apply annealing to ϵ (in Appx. E).

Ranking the Pivot Timesteps. With our two search schemes, we can search the pivot timesteps for each global trajectory $\tau = \{(\boldsymbol{s}^t, \tilde{\boldsymbol{u}}^t, r^t, \boldsymbol{s}^{t+1})\}_{t=0}^{T-1}$. We define the pivot timesteps κ of each global trajectory τ as $\kappa = \{e_t\}_{t=0}^{T-1}$, $0 \leq e_t \leq t$, where e_t indicates the pivot timestep of t when r_t is the consequence of actions committed before timestep t. We first get e_t by aggregating all the searching outcomes (Line 13 in Alg. 1). Then, each agent gets $\kappa_i = \{e_t^i\}_{t=0}^{T-1}$. In order to subserve the inter-agent credit assignment [13, 41], κ can be collectively calculated via proximity:

$$e_t = \min_{e_t^i} \left[t - e_t^1, \cdots, t - e_t^N \right], \ i \in \{1, \cdots, N\} \tag{1}$$

The pseudo code is shown in Alg. 1. For each sampled global trajectory τ, we extract τ_i for each agent in Line 7; then we get e_t for each agent and aggregate κ in line 14 and line 15, respectively.

5 Reward Redistribution for Off-Beat Multi-Agent Reinforcement Learning

Searching in LeGEM leverages the collective intelligence [25, 15] in OBMAS. We utilize TD learning to train MARL policies. The TD error is the difference between the TD target and the prediction. TD targets can be estimated with n-step target, TD(λ) and other techniques [12, 56]. Unfortunately, current n-step target and TD(λ) methods are far from accurate estimating TD targets. They even incur underestimation with off-beat trajectories. In essence, to train MARL policies in OBMAS, one should accurately estimate the TD target where the reward plays the key role [46, 70]. We resolve the aforementioned conundrum by redistributing rewards to their pivot timesteps. The key idea is that we can pull the outcome of one joint off-beat action back to the timestep when it was committed to the environment, which can dramatically enhance learning despite the long-term reward delays incurred by off-beat actions. We utilize e_t to update the reward of the transit $(\boldsymbol{s}^{e_t}, \tilde{\boldsymbol{u}}^{e_t}, r^{e_t}, \boldsymbol{s}^{e_t+1})$:

$$\hat{r}^{e_t} = \mathbb{1}(e_t \geq t) \cdot r^{e_t} + \mathbb{1}(e_t < t) \cdot r_t, \tag{2}$$

where $\mathbb{1}(\cdot)$ is the indicator function.Such update rule is conducted iteratively from $t = 0$ to $t = T - 1$. β is a very small positive hyperparameter. To stabilize learning and circumvent the overestimation of the TD target, r_t is also updated after Eqn. 2 via $r_t = (1 - \mathbb{1}(e_t < t) \cdot (1 - \beta)) \cdot r_t$. It also avoids aggregated biased/wrong estimation of TD target being back propagated in Bellman Equation. Formally, we define the reward redistribution operator as Π_Φ, *i.e.*, $e_t = \Pi_\Phi \rho(r^t, \boldsymbol{s}, \tilde{\boldsymbol{u}})$, and then define the Off-Beat Bellman operator Γ:

$$(\Gamma Q^{\text{tot}})(\boldsymbol{s}, \tilde{\boldsymbol{u}}) := \mathbb{E}[\Pi_\Phi R(\boldsymbol{s}, \tilde{\boldsymbol{u}}, \boldsymbol{m}) + \gamma \max_{\tilde{\boldsymbol{u}}'} Q^{\text{tot}}(\boldsymbol{s}', \tilde{\boldsymbol{u}}')] \tag{3}$$

With the Off-Beat Bellman operator Γ, we propose its contraction property.

Proposition 1. *$\Gamma : \mathcal{Q} \mapsto \mathcal{Q}$ is a γ-contraction.*

Therefore, we can utilize $\hat{r}_{e_t}$ for *centralized training* in TD-learning:

$$\mathcal{L}^{\text{TD}}(\theta) := \mathbb{E}_{\mathcal{D}' \sim \mathcal{D}}[(\hat{y}^{\text{tot}}_{e_t} - Q^{\text{tot}}_\theta(\boldsymbol{s}^{e_t}, \tilde{\boldsymbol{u}}^{e_t}))^2], \quad \text{where } \hat{y}^{\text{tot}}_{e_t} = \hat{r}^{e_t} + \gamma \max_{\tilde{\boldsymbol{u}}'} Q^{\text{tot}}_{\tilde{\theta}}(\boldsymbol{s}^{e_t+1}, \tilde{\boldsymbol{u}}'). \tag{4}$$

Our method can be easily incorporated into any model-free MARL method for OBMAS. We present the pseudo code of incorporating our method into model-free MARL methods in Alg. 9, Appx. E. We also provide a pictorial view of our framework in Fig. 3 to show the whole pipeline.

6 Experiments

We perform experiments on various multi-agent scenarios with off-beat actions. We introduce off-beat actions in Stag-Hunter Game, Quarry Game, Afforestation Game and StarCraft II micromanagement tasks [44] and use them as testbeds in our experiments. We aim to answer the following questions: **Q1:** *Can our LeGEM improve the multi-agent coordination of many MARL methods in OBMAS?* **Q2:** *Can our LeGEM outperform previous parameterized episodic memory (EM) for MARL?* **Q3:** *Can bootstrapping method of RL help?* **Q4:** *Can our LeGEM outperform the multi-agent exploration and multi-agent risk-sensitive (Ex-Risk) methods?*

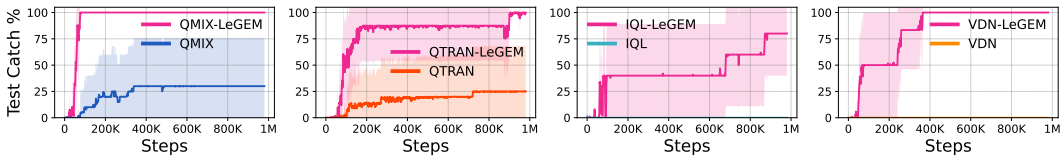

Figure 4: The test catch rate of the stag on the Stag-Hunter Game with off-beat actions.

6.1 Experiment Setup

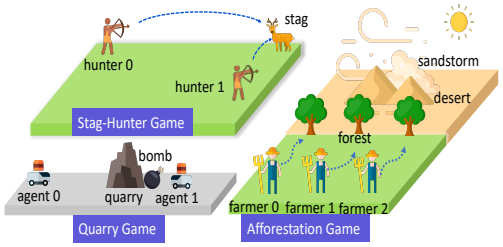

Figure 5: Stag-Hunter Game, Quarry Game and Afforestation Game. More information can be found in Appx. C.

Categories	Methods
MARL (**Q1**)	QMIX [41], VDN [49] IQL [53], QTRAN [47] QPLEX [60]
EM (**Q2**)	EMC [71]
Bootstrap (**Q3**)	N-step &λ-Return [51]
Ex-Risk (**Q4**)	MAVEN [28], EMC [71] RMIX [38]

Table 1: Baseline algorithms.

Baselines and scenarios. We list all baselines in table 1, including the corresponding research questions to be answered. We implement our method on PyMARL [44] and use 10 random seeds to train each method on all environments. We do not use macro-action methods [67, 68] as the baseline because it is hard to make a fair comparison between macro-actions methods and our method. As discussed in Sec. 1, macro-actions rely on manually designed macro-actions, *i.e.*, designing the macro-actions by utilizing the simulator settings and domain knowledge, which is different from learning options [52, 3]. Designing macro-actions is not feasible in scenarios where domain knowledge and simulator settings are unknown, such as the OBMAS scenarios. In OBMAS, the agent has no idea of the durations of other agents' actions, which is challenging for designing macro-actions. We conduct experiments on Stag-Hunter Game, Quarry Game, Afforestation Game (Fig. 5) and StarCraft II micromanagement tasks [44] where off-beat action are introduced.

Training settings. We use opensourced code of baselines publicly by the corresponding authors on Github in all experiments. We resort to mean-std values as our performance evaluation measurement in all figures where the bold line and the shaded area indicate the mean value and one standard deviation of the episode return, respectively. Readers can refer to Appx. C, D, E and F for more information on our environment, baselines, training method, training platform and empirical results.

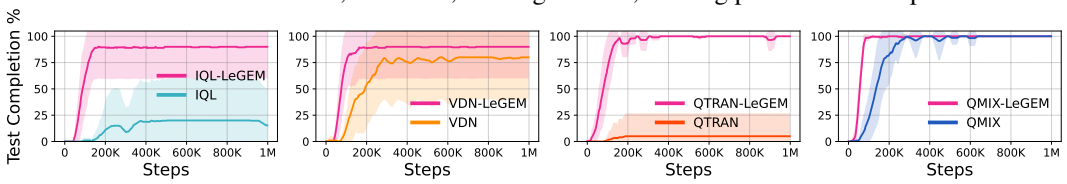

Figure 6: The test task completion rate of the Quarry Game with off-beat actions.

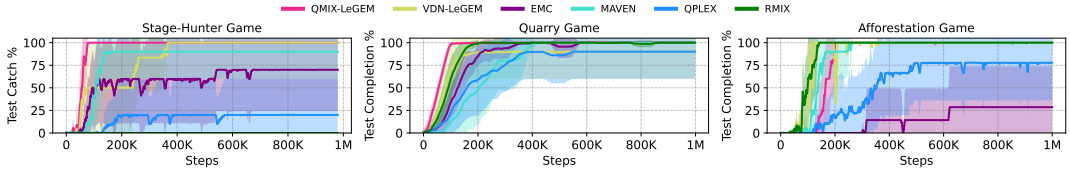

Figure 7: Performance of MARL methods

6.2 Experiment Results

The Effectiveness of LeGEM. We answer **Q1**. With LeGEM, MARL methods get enhanced performance as shown in Fig. 4. Without LeGEM, all methods perform poorly in Stag-Hunter Game; IQL and VDN's final final results are even 0. By incorporating LeGEM, all of them can get converged performance and improved sample efficiency. We are also interested in finding if LeGEM could reinforce the performance of simple methods. As depicted in Fig. 7, with LeGEM, both VDN and QMIX outperforms QPLEX, which is a state-of-the-art MARL method armed with various advanced techniques, including attention network [57], dueling network [64] and advantage function.

Performance of Episodic Memory method. We answer **Q2** by presenting the performance curves of EMC in Fig. 7. EMC is an episodic memory MARL method with curiosity-driven exploration. It utilizes the episodic memory from RL [74, 17].With LeGEM, QMIX outperforms EMC. EMC even fails to converge in Stag-Hunter Game.

Table 2: Results (mean and std) of n-step return (left) and TD(λ) (right) on Stag-Hunter Game.

n	1	5	10	15	λ	0.8	0.9	0.99	1
QMIX	$60.0 \pm 40\%$	0 ± 0	0 ± 0	0 ± 0	QMIX	$100 \pm 0\%$	$100 \pm 0\%$	$89 \pm 10\%$	$61 \pm 37\%$
VDN	0 ± 0	0 ± 0	0 ± 0	0 ± 0	VDN	0 ± 0	0 ± 0	0 ± 0	0 ± 0

Performance of n-step return and TD(λ) methods. To answer **Q3**, we use n-step return and TD(λ) to estimate the TD-target. As shown in Table. 2, with n-step return, both QMIX and VDN fail to learn good policies even with $n = 15$. Surprisingly, with TD(λ), QMIX can achieve good performance with $\lambda \in \{0.8, 0.9, 0.99, 1\}$. However, we cannot find such outcome on VDN and there is no guarantee of good results on using TD(λ).

Performance of Multi-Agent Exploration and Risk-Sensitive MARL methods. We also provide results of exploration methods for MARL and risk-sensitive MARL method to answer **Q4**. MAVEN utilizes mutual information to learn latent space for exploration and RMIX aims to learning risk-sensitive policies for MARL. In Fig. 7, RMIX even fails to learn. Mainly because the potential loss of reward is displaced by off-beat actions. Overall, MAVEN is stabler than EMC and RMIX. QMIX-LeGEM is stable in all scenarios and outperforms MAVEN. With LeGEM, even simple method such VDN can perform well and outperforms many MARL methods with complex and advanced components. Indeed, exploration in OBMAS is beneficial for multi-agent learning. However, the key challenge of temporal credit assignment can not be easily addressed merely with exploration.

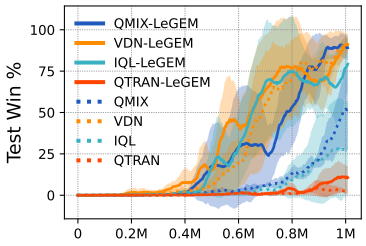

Figure 8: The performance of MARL methods on 2m_vs_1z.

SMAC. We also conduct experiments on SMAC [44]. We train MARL methods and our method on 2m_vs_1z where are two agents combating with one opponent. To overcome the issue of

high dimension continuous state space, We utilize simhash [9] to calculate the hash value of the key. We only select the attack action and set the action duration with 9. As illustrated in Fig. 8, incorporated with our novel episodic memory, QMIX, IQL and VDN illustrate enhanced performance, demonstrating the superiority of our method on complex multi-agent scenarios.

7 Related Works

Action Delay in RL. Conventionally, the execution of actions in RL is instantaneous and the execution duration is neglected. Katsikopoulos et al. [20] propose the Delayed MDP where actions have delays and Walsh et al [59] propose a model-based method for the Delayed MDP. To optimize the delayed MDP, many RL approaches [59, 39, 66, 69] augment the state space with the queuing actions to be executed into the environment. However, this state-augmentation trick is intractable [11]. Chen et al. [10] extend the delayed MDP [39] and propose a Delayed Markov Game. However, the state-augmentation treatment is confined to short delays and neglects the off-beat actions in multi-agent scenarios. Recently, Bouteiller et al. [6] apply replay buffer correction method. However, the delayed timestep is privileged information. It is not available for agents in many scenarios. Simply applying this single-agent trajectory correction in MARL cannot attain satisfactory performance due to off-beat actions; devising inter-agent trajectory correction methods for OBMAS is non-trivial.

Credit Assignment in RL. Credit assignment [50, 52] tackles long-horizon sequential decision-making problem by distributing the contribution of each single step over the temporal interval. TD learning [51] is the most established credit assignment method, which is the basis of many RL methods. RUDDER [2] redistributes the episodic return to key timesteps in the episode [14, 42, 40]. Klissarov et al. [22] propose a reward propogation method via graph convolutional neural network [21]. Another line of works utilize episodic memory (EM) [37, 5, 73, 27, 74] to recall key events and aggregate information of the past for decision-making or learning. However, simply applying EM of RL to MARL cannot perform well in OBMAS due to the non-stationarity and the displaced rewards.

Multi-Agent RL. Many MARL methods focus on factorizing the global Q value to train agents' policies via CTDE [13, 49, 41, 47, 60, 63, 35]. However, these existing works assume actions are executed synchronously. Messias et al. [30] propose an event-driven, asynchronous formulation of the multi-agent POMDP. However, the assumption of free communication [61] is limited and the asynchronous execution [34] in the paper is confined to the design of events and did not propose methods on solving challenging credit assignment issue in OBMAS. Recently, Amato et al. [1] and Xiao et al. [67, 68] propose macro-action methods, which are similar to hierarchical methods. Macro-actions are manually designed via abstracting primitive actions. However, macro-action methods mainly focus on macro-action selection during multi-timestep decision-making and assume the environment can use manually pre-defined methods for state transition. Unfortunately, the above works either focus on synchronous actions or defining specific asynchronous execution components with human knowledge. Learning coordination in OBMAS remains a challenge.

8 Conclusion

In this paper, we investigate model-free MARL with off-beat actions. To address challenges in OBMAS, we first propose Off-Beat Dec-POMDP. Then, we propose a new class of episodic memory, LeGEM, for model-free MARL algorithms. LeGEM addresses the challenging temporal credit assignment problem raised by off-beat actions in TD-learning via the novel reward redistribution scheme. We evaluate our method on various OBMAS scenarios. Empirical results show that our method significantly boosts the multi-agent coordination and achieves leading performance as well as improved sample efficiency.

Limitations and Future Work. Searching from a graph-structured episodic memory takes much overhead in LeGEM. Scaling up LeGEM to complex OBMAS is our future direction. Recently, there is a growing interest in model-based planing [45]. Leveraging LeGEM for model-based planning is also our future work. Our paper focuses on Dec-POMDP-based MARL methods. We leave it to future work for investigating off-beat actions in frameworks like Markov Game [26] and MMDP [7]. We are also interested in finding the merit of our method in real-world problem in our future work, such as scheduling [29] with off-beat settings.

References

[1] C. Amato, G. Konidaris, L. P. Kaelbling, and J. P. How. Modeling and planning with macro-actions in decentralized pomdps. *Journal of Artificial Intelligence Research*, 64:817–859, 2019.

[2] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter. RUDDER: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[3] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[4] N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[5] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis. Reinforcement learning, fast and slow. *Trends in Cognitive Sciences*, 23(5):408–422, 2019.

[6] Y. Bouteiller, S. Ramstedt, G. Beltrame, C. Pal, and J. Binas. Reinforcement learning with random delays. In *International Conference on Learning Representations*, 2020.

[7] C. Boutilier. Planning learning and coordination in multiagent decision processes. In *Proceedings of the 6th Conference on the Theoretical Aspects of Rationality and Knowledge*, pages 195–210, 1996.

[8] Y. Cao, W. Yu, W. Ren, and G. Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial Informatics*, 9(1):427–438, 2012.

[9] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth Annual ACM Symposium on Theory of Computing*, pages 380–388, 2002.

[10] B. Chen, M. Xu, Z. Liu, L. Li, and D. Zhao. Delay-aware multi-agent reinforcement learning for cooperative and competitive environments. *arXiv e-prints*, pages arXiv–2005, 2020.

[11] E. Derman, G. Dalal, and S. Mannor. Acting in delayed environments with non-stationary Markov policies. In *International Conference on Learning Representations*, 2020.

[12] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416, 2018.

[13] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2017.

[14] T. Gangwani, Y. Zhou, and J. Peng. Learning guidance rewards with trajectory-space smoothing. *Advances in Neural Information Processing Systems*, 33:822–832, 2020.

[15] D. Ha and Y. Tang. Collective intelligence for deep learning: A survey of recent developments. *arXiv preprint arXiv:2111.14377*, 2021.

[16] B. Han, Z. Ren, Z. Wu, Y. Zhou, and J. Peng. Off-policy reinforcement learning with delayed rewards. *arXiv preprint arXiv:2106.11854*, 2021.

[17] H. Hu, J. Ye, G. Zhu, Z. Ren, and C. Zhang. Generalizable episodic memory for deep reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning*, pages 4380–4390, 18–24 Jul 2021.

[18] C.-C. Hung, T. Lillicrap, J. Abramson, Y. Wu, M. Mirza, F. Carnevale, A. Ahuja, and G. Wayne. Optimizing agent behavior over long time scales by transporting value. *Nature Communications*, 10(1):1–12, 2019.

[19] M. Hüttenrauch, A. Šošić, and G. Neumann. Guided deep reinforcement learning for swarm systems. *In AAMAS 2017 Autonomous Robots and Multirobot Systems (ARMS) Workshop*, 2017.

[20] K. V. Katsikopoulos and S. E. Engelbrecht. Markov decision processes with delays and asynchronous cost collection. *IEEE Transactions on Automatic Control*, 48(4):568–574, 2003.

[21] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[22] M. Klissarov and D. Precup. Reward propagation using graph convolutional networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[23] J. G. Kuba, R. Chen, M. Wen, Y. Wen, F. Sun, J. Wang, and Y. Yang. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*, 2021.

[24] A. K. Lampinen, S. C. Chan, A. Banino, and F. Hill. Towards mental time travel: a hierarchical memory for reinforcement learning agents. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[25] J. Z. Leibo, E. D. nez Guzmán, A. S. Vezhnevets, J. P. Agapiou, P. Sunehag, R. Koster, J. Matyas, C. Beattie, I. Mordatch, and T. Graepel. Scalable evaluation of multi-agent reinforcement learning with melting pot. PMLR, 2021.

[26] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Elsevier, 1994.

[27] X. Ma and W.-J. Li. State-based episodic memory for multi-agent reinforcement learning. *arXiv preprint arXiv:2110.09817*, 2021.

[28] A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson. MAVEN: Multi-agent variational exploration. In *Advances in Neural Information Processing Systems*, pages 7613–7624, 2019.

[29] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM special interest group on data communication*, pages 270–288. 2019.

[30] J. V. Messias, M. T. Spaan, and P. U. Lima. Multiagent pomdps with asynchronous execution. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, pages 1273–1274, 2013.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[32] F. A. Oliehoek, C. Amato, et al. *A Concise Introduction to Decentralized POMDPs*, volume 1. Springer, 2016.

[33] F. A. Oliehoek, M. T. Spaan, and N. Vlassis. Optimal and approximate q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

[34] S. Omidshafiei, A.-A. Agha-Mohammadi, C. Amato, and J. P. How. Decentralized control of partially observable markov decision processes using belief space macro-actions. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5962–5969, 2015.

[35] L. Pan, T. Rashid, B. Peng, L. Huang, and S. Whiteson. Regularized softmax deep multi-agent q-learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[36] W. B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[37] A. Pritzel, B. Uria, S. Srinivasan, A. P. Badia, O. Vinyals, D. Hassabis, D. Wierstra, and C. Blundell. Neural episodic control. In *International Conference on Machine Learning*, pages 2827–2836, 2017.

[38] W. Qiu, X. Wang, R. Yu, R. Wang, X. He, B. An, S. Obraztsova, and Z. Rabinovich. RMIX: Learning risk-sensitive policies for cooperative reinforcement learning agents. In *Advances in Neural Information Processing Systems*, 2021.

[39] S. Ramstedt and C. Pal. Real-time reinforcement learning. *Advances in Neural Information Processing Systems*, 32:3073–3082, 2019.

[40] D. Raposo, S. Ritter, A. Santoro, G. Wayne, T. Weber, M. Botvinick, H. van Hasselt, and F. Song. Synthetic returns for long-term credit assignment. *arXiv preprint arXiv:2102.12425*, 2021.

[41] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304, 2018.

[42] Z. Ren, R. Guo, Y. Zhou, and J. Peng. Learning long-term reward redistribution via randomized return decomposition. *arXiv e-prints*, pages arXiv–2111, 2021.

[43] R. T. Rockafellar, S. Uryasev, et al. Optimization of conditional value-at-risk. *Journal of Risk*, 2:21–42, 2000.

[44] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

[45] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[46] D. Silver, S. Singh, D. Precup, and R. S. Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, 2021.

[47] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5887–5896, 2019.

[48] T. Suddendorf, D. R. Addis, and M. C. Corballis. Mental time travel and the shaping of the human mind. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1521):1317–1324, 2009.

[49] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[50] R. S. Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 1984.

[51] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

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

[53] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PLoS ONE*, 12(4), 2017.

[54] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[55] E. Tulving. Memory and consciousness. *Canadian Psychology/Psychologie Canadienne*, 26(1):1, 1985.

[56] H. van Hasselt, S. Madjiheurem, M. Hessel, D. Silver, A. Barreto, and D. Borsa. Expected eligibility traces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, number 11, pages 9997–10005, 2021.

[57] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[58] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[59] T. J. Walsh, A. Nouri, L. Li, and M. L. Littman. Learning and planning in environments with delayed feedback. *Autonomous Agents and Multi-Agent Systems*, 18(1):83–105, 2009.

[60] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang. QPLEX: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062*, 2020.

[61] R. Wang, X. He, R. Yu, W. Qiu, B. An, and Z. Rabinovich. Learning efficient multi-agent communication: An information bottleneck approach. In *International Conference on Machine Learning*, pages 9908–9918. PMLR, 2020.

[62] W. Wang, T. Yang, Y. Liu, J. Hao, X. Hao, Y. Hu, Y. Chen, C. Fan, and Y. Gao. Action semantics network: Considering the effects of actions in multiagent systems. *arXiv preprint arXiv:1907.11461*, 2019.

[63] Y. Wang, B. Han, T. Wang, H. Dong, and C. Zhang. DOP: Off-policy multi-agent decomposed policy gradients. In *International Conference on Learning Representations*, 2021.

[64] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003, 2016.

[65] C. J. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8(3-4):279–292, 1992.

[66] T. Xiao, E. Jang, D. Kalashnikov, S. Levine, J. Ibarz, K. Hausman, and A. Herzog. Thinking while moving: Deep reinforcement learning with concurrent control. In *International Conference on Learning Representations*, 2019.

[67] Y. Xiao, J. Hoffman, and C. Amato. Macro-action-based deep multi-agent reinforcement learning. In *Conference on Robot Learning*, pages 1146–1161. PMLR, 2020.

[68] Y. Xiao, J. Hoffman, T. Xia, and C. Amato. Multi-agent/robot deep reinforcement learning with macro-actions (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13965–13966, 2020.

[69] Y. Yuan and R. Mahmood. Asynchronous reinforcement learning for real-time control of physical robots. *arXiv preprint arXiv:2203.12759*, 2022.

[70] T. Zahavy, B. O'Donoghue, G. Desjardins, and S. Singh. Reward is enough for convex mdps. *Advances in Neural Information Processing Systems*, 34, 2021.

[71] L. Zheng, J. Chen, J. Wang, J. He, Y. Hu, Y. Chen, C. Fan, Y. Gao, and C. Zhang. Episodic multi-agent reinforcement learning with curiosity-driven exploration. *Advances in Neural Information Processing Systems*, 34, 2021.

[72] M. Zhou, J. Luo, J. Villella, Y. Yang, D. Rusu, J. Miao, W. Zhang, M. Alban, I. Fadakar, Z. Chen, et al. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving. *arXiv preprint arXiv:2010.09776*, 2020.

[73] Y. Zhou, D. E. Asher, N. R. Waytowich, and J. A. Shah. On memory mechanism in multi-agent reinforcement learning. *arXiv e-prints*, pages arXiv–1909, 2019.

[74] G. Zhu, Z. Lin, G. Yang, and C. Zhang. Episodic reinforcement learning with associative memory. In *International Conference on Learning Representations*, 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 Sec. 4.
   (b) Did you describe the limitations of your work? [Yes] See Sec. 8.
   (c) Did you discuss any potential negative societal impacts of your work? [No] Our method does not have negative social impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See the statements.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appx. A.
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] See the supplementary files.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appx. E.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Sec. 6 and Appx. E.
   (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] See Appx. E.
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? [Yes] We cited, see Sec. 6.
   (b) Did you mention the license of the assets? [Yes] See Appx. E.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See the supplementary files.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
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? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proofs

Proposition 1. *$\Gamma : \mathcal{Q} \mapsto \mathcal{Q}$ is a γ-contraction.*

Proof. Recall that the Off-Beat Bellman operator Γ is defined as:

$$(\Gamma Q^{\text{tot}})(\boldsymbol{s}, \tilde{\boldsymbol{u}}) := \mathbb{E}[\Pi_\Phi R(\boldsymbol{s}, \tilde{\boldsymbol{u}}, \boldsymbol{m}) + \gamma \max_{\tilde{\boldsymbol{u}}'} Q^{\text{tot}}(\boldsymbol{s}', \tilde{\boldsymbol{u}}')] \tag{5}$$

The sup-norm is defined as $\|Q\|_\infty = \sup_{\boldsymbol{s}\in\mathcal{S}, \tilde{\boldsymbol{u}}\in\mathcal{U}} |Q(\boldsymbol{s}, \tilde{\boldsymbol{u}})|$. We consider the sup-norm contraction:

$$\left\|(\Gamma Q^{\text{tot}}_{(1)})(\boldsymbol{s}, \tilde{\boldsymbol{u}}) - (\Gamma Q^{\text{tot}}_{(2)})(\boldsymbol{s}, \tilde{\boldsymbol{u}})\right\|_\infty \leq \gamma \left\|Q^{\text{tot}}_{(1)}(\boldsymbol{s}, \tilde{\boldsymbol{u}}) - Q^{\text{tot}}_{(2)}(\boldsymbol{s}, \tilde{\boldsymbol{u}})\right\|_\infty \tag{6}$$

We prove:

$$\begin{aligned}
\left\|(\Gamma Q^{\text{tot}}_{(1)})(\boldsymbol{s}, \tilde{\boldsymbol{u}}) - (\Gamma Q^{\text{tot}}_{(2)})(\boldsymbol{s}, \tilde{\boldsymbol{u}})\right\|_\infty &= \max_{\boldsymbol{s}, \tilde{\boldsymbol{u}}} \left|\gamma \sum\nolimits_{\boldsymbol{s}'} \mathcal{P}(\boldsymbol{s}'|\boldsymbol{s}, \tilde{\boldsymbol{u}})(\max_{\tilde{\boldsymbol{u}}'} Q^{\text{tot}}_{(1)}(\boldsymbol{s}', \tilde{\boldsymbol{u}}') - \max_{\tilde{\boldsymbol{u}}'} Q^{\text{tot}}_{(2)}(\boldsymbol{s}', \tilde{\boldsymbol{u}}'))\right| \\
&\leq \max_{\boldsymbol{s}, \tilde{\boldsymbol{u}}} \gamma \sum\nolimits_{\boldsymbol{s}'} \mathcal{P}(\boldsymbol{s}'|\boldsymbol{s}, \tilde{\boldsymbol{u}}) \left|\max_{\tilde{\boldsymbol{u}}'}(Q^{\text{tot}}_{(1)}(\boldsymbol{s}', \tilde{\boldsymbol{u}}') - Q^{\text{tot}}_{(2)}(\boldsymbol{s}', \tilde{\boldsymbol{u}}'))\right| \\
&\leq \max_{\boldsymbol{s}, \tilde{\boldsymbol{u}}} \gamma \sum\nolimits_{\boldsymbol{s}'} \mathcal{P}(\boldsymbol{s}'|\boldsymbol{s}, \tilde{\boldsymbol{u}}) \max_{\boldsymbol{s}'', \tilde{\boldsymbol{u}}'} \left|Q^{\text{tot}}_{(1)}(\boldsymbol{s}'', \tilde{\boldsymbol{u}}') - Q^{\text{tot}}_{(2)}(\boldsymbol{s}'', \tilde{\boldsymbol{u}}')\right| \\
&= \max_{\boldsymbol{s}, \tilde{\boldsymbol{u}}} \gamma \sum\nolimits_{\boldsymbol{s}'} \mathcal{P}(\boldsymbol{s}'|\boldsymbol{s}, \tilde{\boldsymbol{u}}) \left\|Q^{\text{tot}}_{(1)} - Q^{\text{tot}}_{(2)}\right\|_\infty \\
&= \gamma \left\|Q^{\text{tot}}_{(1)} - Q^{\text{tot}}_{(2)}\right\|_\infty
\end{aligned} \tag{7}$$

□

Readers may find that with the reward redistribution operator Π_Φ, the reward is ordered. Consequently, Off-Beat Bellman equation is reduced to Bellman equation[7].

[7] When all off-beat rewards are redistributed to the ground true pivot timestep, we can claim this finding.

B LeGEM for Off-Beat MARL

In this section, we list Alg. 4, Alg. 5, Alg. 6 and Alg. 7 in Sec. B.1 and present the training pipline for Off-Beat MARL in Alg. 9 in Sec. B.2. We also present lists of symbols for Dec-POMDP, Off-Beat Dec-POMDP, MARL, Off-Beat MARL and LeGEMin Tab. 3, 4 and 5. To make pesude code easy to read, we use Python-like[8] syntax to represent vectors and hashmaps (look-up tables).

B.1 LeGEM

We also define the sub-graph set of ϕ_i^t as $\Phi_i^{t,\Omega} = \{\phi_i^{t,\omega}\}_{\omega=0}^{\Omega-1}$ by using the discretized episode return and there are Ω sub-graphs. $\phi_i^{t,\omega}$ is the ω-th sub-graph whose episode return is $\Upsilon[\omega]$ ($\omega \in \{0, \cdots, \Omega-1\}, \Upsilon = [0, \cdots, \boldsymbol{r}^{t,i}]$) where $\boldsymbol{r}^{t,i}$ is the discretized maximum episode return of ϕ_i^t.

Algorithm 3: `UpdateLeGEM`

Input: Agent i's $\{\tau_i^d\}_{d=1}^D$ and Φ_i.
for $d \leftarrow 1$ ***to*** D **do**
  Get $\phi_i^l \leftarrow \Phi_i$`[length(`$\tau_i^d$`)-1]; // length(`$\tau_i^d$`)-1 equals` l
  Calculate the discretized episode reward $\boldsymbol{r}^{l,i}$;
  Get the index ω from Υ by using $\boldsymbol{r}^{l,i}$;
  Get $\phi_i^{l,\omega} \leftarrow \Phi_i^{l,\Omega}[\omega]$;
  for $t \leftarrow 0$ ***to*** $\text{length}(\tau_i^d) - 1$ **do**
    if $(o_i^t, u_i^t) \in \phi_i^l$ **then**
      `// There is no need to update the node of sub-graph` $\phi_i^{l,\omega}$ `as it shares the same node with` ϕ_i^l
      $\psi \leftarrow \phi_i^l$`.getNode(`o_i^t, u_i^t`);`
      ψ`.visitCount++;`
    else
      $\psi \leftarrow$ `newNode(`o_i^t, u_i^t, r^t`);`
      ϕ_i^l`.append(`ψ`);`
      ϕ_i^l`.updatePointers(`ψ`); // Sub-graph` $\phi_i^{l,\omega}$ `shares the same node with` ϕ_i^l
      $\phi_i^{l,\omega}$`.append(`ψ`);`
      $\phi_i^{l,\omega}$`.updatePointers(`ψ`);`
Return: Φ_i.

Alg. 3 shows the whole procedure to construct the graph. To illustrate it, we provide an example below (Fig. 9) to show how to construct the graph. Fig. 2 shows the relationship between sub-graphs and the graphs.

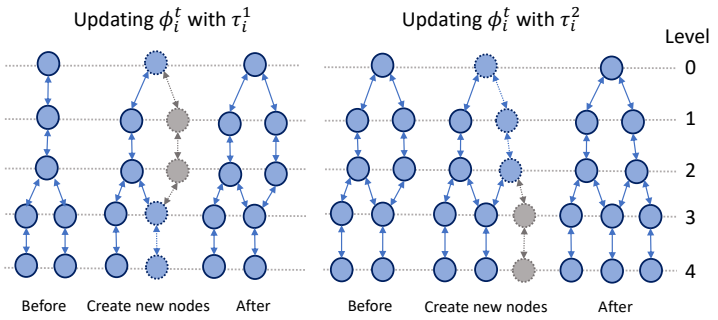

Figure 9: **Updating agent i's Φ_i:** Agent i's ϕ_i^t is updated with agent's trajectories τ_i^1 and then updated with τ_i^2. Solid arrows and circles indicate the pointers and nodes, respectively. Grey dotted lines indicate pointers to be created and grey circles with dotted outlines indicate nodes to be created. All the dotted elements (pointers and circles) consist of the a new path in τ_i^1. All the created pointers and nodes will be added to ϕ_i^t.

[8] `https://www.python.org/`

Algorithm 4: `VisitCount`

Input: $\Lambda_i^{t,l}$.
Initialize: vc $\leftarrow$ `[]`, an empty vector to store visit count mask for each path in $\Lambda_i^{t,l}$.
foreach *path* $\Lambda_i^{t,l}$`[`j`]` $\in \Lambda_i^{t,l}$ **do**
    pathVC $\leftarrow$ `sort(set([node.visitCount` for `node` in $\Lambda_i^{t,l}$`[`j`]]))`;
    Create an empty look-up table tb $\leftarrow$ `{}`;
    foreach *index* $k \in$ pathVC **do**
        tb`[`pathVC`[`k`]] `$\leftarrow k$;
    Create an empty vector ls $\leftarrow$ `[]`;
    foreach *node* $\in$ *trajectory* $\Lambda_i^{t,l}$`[`j`]` **do**
        ls`.append(`tb`[`node`.visitCount])`;
    vc`.append(`ls`)`;
Return: vc.

Algorithm 5: `UL`

Input: $\Lambda_i^{t,l}[j]$, vc, τ_i.
Initialize: $e_t^{i,j,\downarrow} \leftarrow -1$, res $\leftarrow$ []. `// Initialize the return value and an empty vector`
vc $\leftarrow$ vc[:t]; `// Slicing the visit count`
left $\leftarrow 0$, right $\leftarrow 1$; `// Create two pointers`
while right $<$ `size(vc)` **do**
  if vc[right] $=$ vc[left] **then**
    right++; `// Non-decreasing pattern`
  else if vc[right] $>$ vc[left] **then**
    res.`append`(right);
    left $\leftarrow$ right;
    right++;
  else
    `break`; `// Break the while loop`
if `size`(res) $= 0$ **then**
  $e_t^{i,j,\downarrow} \leftarrow -1$;
else
  $e_t^{i,j,\downarrow} \leftarrow$ res[-1]; `// Get the last value of res`
Return: $e_t^{i,j,\downarrow}$.

Algorithm 6: `LU`

Input: $\Lambda_i^{t,l}[j]$, vc, τ_i.
Initialize: $e_t^{i,j,\uparrow} \leftarrow -1$, res $\leftarrow$ []. `// Initialize the return value and an empty vector`
vc $\leftarrow$ `reverse(vc[:`t`])`; `// Slicing the visit count and then reverse`
left $\leftarrow 0$, right $\leftarrow 1$; `// Create two pointers`
while right $<$ `size(vc)` **do**
  if vc[right] $=$ vc[left] **then**
    right++; `// Non-increasing pattern`
  else if vc[right] $>$ vc[left] **then**
    res`.append(`right`)`;
    left $\leftarrow$ right;
    right++;
  else
    `break; // Break the while loop`
if `size(`res`)` $= 0$ **then**
  $e_t^{i,j,\uparrow} \leftarrow -1$;
else
  $e_t^{i,j,\uparrow} \leftarrow$ res`[-1]`; `// Get the last value of res`
  $e_t^{i,j,\uparrow} \leftarrow$ `size(vc)` $- e_t^{i,j,\uparrow} - 1$; `// Get the right timestep as vc is reversed`
Return: $e_t^{i,j,\uparrow}$.

Algorithm 7: `Summarize`

Input: e_t^i. `// Receives a vector of pivot timesteps`
`// Get the pivot timestep with the maximum count`
`// Since getValueOfMaxCount(·) is easy to implement, for brevity, we do not present its implementation here`
$e_t^i \leftarrow$ `getValueOfMaxCount(`e_t^i`);`
Return: e_t^i.

Algorithm 8: Search Scheme II

Input: ω, $\Lambda_i^{t,l}$, τ_i, $\boldsymbol{r}^{l,i}$, Υ and Φ_i.
Initialize: $e_t^i \leftarrow -1$; `// Initialize the results`
Initialize: tb $\leftarrow$ `{}`; `// Initialize a empty look-up table`
foreach *path* $\Lambda_i^{t,l}$`[`j`]` $\in \Lambda_i^{t,l}$ **do**
    `// Search backwards from` $t-1$ `to` 0
    foreach `node` $j = t-1 \in \Lambda_i^{t,l}$`[`$j$`]` to 0 **do**
        tb`[`j`]` $\leftarrow$ `{}`;
        if new `node` **then**
            tb`[`j`][node]=1`;
        else
            tb`[`j`][node]++`;
if the size of each tb`[`j`]` equals the size of $\Lambda_i^{t,l}$ **then**
    $e_t^i \leftarrow -1$;
else
    Get the timestep j' whose length of tb`[`j'`]` is the smallest one;
    $e_t^i \leftarrow j'$;
Return: e_t^i.

B.2 Off-Beat MARL Training

We present the pseudo code of incorporating LeGEM into model-free MARL method in Alg. 9. Lines 4-5 show that agents commit actions into the environment and then agents save the trajectories into the buffer. Agents update their individual episodic memory in line 6. In lines 7-13, agents' policies are trained with TD learning (line 10) by searching agent's episodic memories (line 9). We also provide a pictorial view of our framework in Fig. 3 to show the whole pipeline of MARL learning.

Algorithm 9: Off-Beat MARL Training

1 **Input**: initialize parameters $\bar{\theta}$ and θ of the network and the target network of agents, replay buffer $\mathcal{D}$ and Φ;
2 **for** $j \leftarrow 1$ ***to*** *max_episode* **do**
3 **while** episode_not_terminated **do**
4 All agents commit actions $\tilde{\boldsymbol{u}}_i^t$ into environment;
5 Collect $(\boldsymbol{s}^t, \{o_i^t\}_{i=1}^N, \tilde{\boldsymbol{u}}^t, r^t, \boldsymbol{s}')$; save it into $\mathcal{D}$;
6 Call `UpdateLeGEM` (Alg. 3);
7 **if** update_the_model **then**
8 Sample a min-batch $\mathcal{D}'$ from $\mathcal{D}$;
9 For each sample in $\mathcal{D}'$, get e_t by calling `SearchPivotTimesteps` (Alg. 1) ;
10 Calculate the TD target with Eqn. 2 and 4;
11 Update θ by minimizing the TD loss;
12 **if** update **then**
13 Update $\bar{\theta}$: $\bar{\theta} \leftarrow \theta$;
14 **Return**: A well-trained policy for each agent.

B.3 List of Symbols

Table 3: List of Symbols for Dec-POMDP and Off-Beat Dec-POMDP

Symbol	Meaning
$\mathcal{S}$	The state space
$\mathcal{U}$	The action space
$\mathcal{P}$	The transition probability
R	The reward function
$\mathcal{R}$	The reward space
O	The observation function
$\mathcal{O}$	The observation space
$\mathcal{N}$	The index set agents
$\boldsymbol{s}$	The current global state
$\boldsymbol{u}$	The current action
$\boldsymbol{s}'$	The next global state
γ	The discount factor
i	The index of agent i
u_i	The action of agent i at current timestep
N	The number of agents
$\mathcal{T}$	The agent's action-observation-reward history space
τ_i	Agent's action-observation-reward history
$\tilde{u}$	Agent's off-beat action
$m_{\tilde{u}_i}$	The execution duration of agent i's action $\tilde{u}_i$
A	The action duration distribution
$\mathcal{A}$	The space of the action duration distribution
$\tilde{\boldsymbol{u}}$	The joint off-beat action
$\tilde{\boldsymbol{u}}_t$	The joint off-beat action at timestep t
$\boldsymbol{m}$	The execution duration of $\tilde{\boldsymbol{u}}$
$\boldsymbol{m}_t$	The execution duration of $\tilde{\boldsymbol{u}}_t$

Table 4: List of Symbols for MARL and Off-Beat MARL

Symbol	Meaning
Q_i	Agent i's Q value
Q^{tot}	The global Q value of all agents
Q^*	The optimal global Q value of all agents
θ	The the parameters of the agents (including agent's network, and networks for learning Q^{tot})
$\bar{\theta}$	The the parameter of the target network
D'	A sample from the replay buffer
$\mathcal{D}$	The replay buffer
e_t	The pivot timestep for r_t
Γ	The Off-Beat Bellman operator
$\hat{r}^{e_t}$	The redistributed reward
Π_Φ	The reward redistribution operator

Table 5: List of Symbols for LeGEM

Symbol	Meaning
τ_i	Agent i's observation-action-reward trajectory
$\overset{t}{i}$	Agent i's observation at timesetp t
$\tilde{u}_i^t$	Agent i's off-beat action at timesetp t
r^t	The global reward at timestep t
T	The length of the τ_i
Φ	The set of LeGEM
Φ_i	The set of agent i's LeGEM
ϕ_i^t	Agent i's LeGEMwhose maximum level is t
Ψ	The set of nodes
Ξ	The set of edges
ω	The index of episode return
Ω	The length of the episode return list Υ
$\phi_i^{t,\omega}$	Agent i's ω-th sub-graph of which maximum level is t
$\Phi_i^{t,\Omega}$	Agent i's set of sub-graphs of which maximum level is t
Υ	The episode return list
$\Upsilon[\omega]$	The ω-th episode return
$\boldsymbol{r}^{t,i}$	The discretized maximum episode return of ϕ_i^t
l	the index of the l-th level in the graph.
Λ_i^t	All the paths from node at level t to node at the level 0
e_t	The pivot timestep for r_t
e_t^i	Agent i's pivot timestep for r_t
$\boldsymbol{e}_t^i$	The vector of agent i's pivot timestep of each path in $\Lambda_i^{t,l}$ for r_t

B.4 Intuition Behind the Search Schemes

In this subsection, we provide the intuition behind the search schemes. It is worth noting that the search schemes in this paper aim to search the pivot time step where the key action was committed into the environment and the reward was returned after some time steps.

Node Co-Occurrence. Intuitively, when an off-beat reward is found in the trajectory, we can search backwards and find the actions committed by all agents. This is called Action-Reward Association. We restate it (Fact 1) below. In graph, we name it node co-occurrence since the node of the off-beat reward and nodes of off-beat actions exist in the graph. The key of the node in LeGEM is a tuple of observation and action.

Fact 1. (Action-Reward Association) *When an off-beat reward r_t exists in the trajectory τ_i ($i \in \{1, \cdots, N\}$), $r_t \in \tau_i$, off-beat action $\boldsymbol{u}_{t'}$ exists ($t' < t$) in the trajectory set $\{\tau_j\}_{j=1}^N$, where $\{\tau_j\}_{j=1}^N$ constitutes the global trajectory of all agents.*

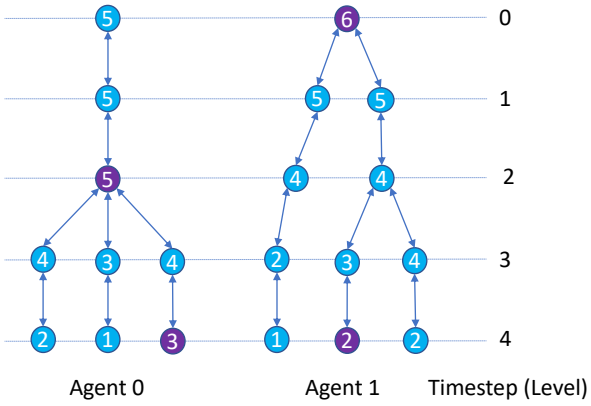

Figure 10: An example of node co-occurrence. There are two graphs. The circle indicates the nodes and the arrow stands for the edges. The number in each node is the visit count. Nodes in purple are nodes where off-beat actions are committed or off-beat rewards occur.

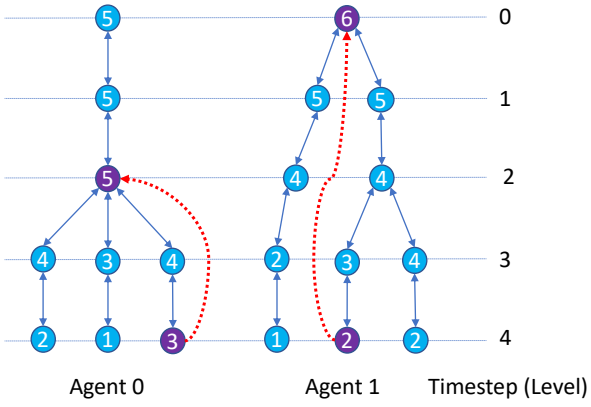

Figure 11: An example of search.

In Fig. 10, we present an example to showcase the node co-occurrence. The example is a simplified two-agent stag-hunter game as depicted in Fig. 1, Sec.1. There are two agents and each agent can only select `SHOOT` or `NOOP`. The `SHOOT` action of agent 0 takes the action duration of 2 to hit the stag while The `SHOOT` action of agent 1 takes 4 timesteps to hit the stag. The task is to hit the stag at the same time by 2 arrows shot by the two agents. By interacting with the environment, many trajectories are collected by the agents and agents' graphs are built. At timestep 0, agent 1 executes an off-beat action (`SHOOT`) and agent 0 commits an off-beat action (`SHOOT`) into the environment at timestep 2. At timestep 4, the stag is captured and an off-beat reward occurs due to the action duration. We can get the observation and the action at timestep 4 and then find the corresponding

nodes (in purple) for agent 0 and agent 1. Nodes that contain the off-beat action are also in purple. With the property of the node co-occurrence and Fact 1, we can search the nodes (i.e., timesteps) that are related to the off-beat rewards.

Searching. We can search the nodes (i.e., timesteps) that are related to the off-beat rewards via searching methods. As the visit count number provides the information of the occurrence between the node of the off-beat reward and the node of the off-beat action, we can utilise this pattern via searching backwards. As depicted in Fig.11, the searched timesteps are 2 and 0 for agent 0 and agent 1, respectively. The search processes are show with dotted red arrows. Now, there are two candidates of the pivot timestep and we can redistribute the reward (agents share a global reward) to the chosen pivot timestep. We select the candidate timestep that is closest to the timestep of the off-beat reward as the pivot timestep. Besides that, we tried two other ranking methods:

1. We redistribute the reward to the searched pivot time step t, which is the farthest time step from the time step of the reward. At time step t, an off-beat action was committed to the environment. However, the result is not as good as the one presented in our paper. The reasons are (i) the action taken at time step t is not the key action to the reward. In OBMAS, for example, in the scenario in Fig. 1 in our paper, action `SHOOT` is taken by agent one at time step 5 is the key action to the reward at time step 9; (ii) In Dec-POMDP MARL methods, we use RNN in the policy network to mitigate the issue raised by partial observation. The RNN can backpropagate the redistributed reward at the pivot timestep to the time steps before it. Besides that, the Bellman update can also backpropagate the redistributed reward to Q values of state-action pairs before the pivot timestep.
2. We redistribute reward at time step t (with LeGEM, the pivot time step is t') to all time steps where off-beat actions were taken. This scheme did not perform well either. The main reason is that the redistributed reward to time steps before the time step t' can overweigh the corresponding Q values.

Therefore, we use the ranking method as introduced in Eqn. 1, Sec. 4.2. Rewards are redistributed in Eqn. 2, Sec. 5 for TD-learning in multi-agent reinforcement learning. We also present the pseudo code of training in Alg. 9 and a pictorial view of our framework in Fig. 3 (in main text) to show the whole pipeline.

C Environments

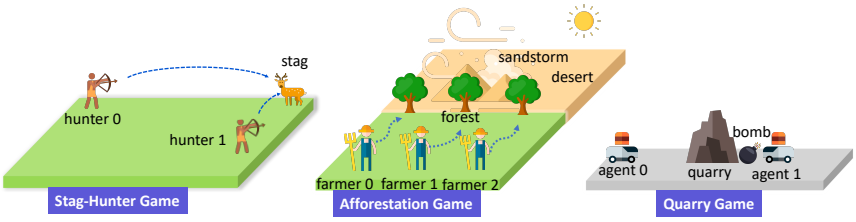

Figure 12: Stag-hunter Game, Quarry Game and Afforestation Game.

C.1 Stag-Hunter Game

As depicted in Fig. 12, there are n agents whose action durations are different; the task is to catch the stag by shooting it simultaneously for all agents. Agents cannot move and the distance between the agent and the stag is different. The stag will escape when hit by $j \in \{1, \cdots, n-1\}$ arrows. In this case, agents will receive a positive reward given the number of arrows that successfully shoot the stag. In Sec. 6, the environment dimension of the Stag-Hunter Game is 15×15 and maximum time steps is 14. At each time step, each agent can only observe its position and the position of the stag. It cannot observe the position of other agents. Agents can select `SHOOT` or `NOOP` actions. `SHOOT` means shooting the arrow and `NOOP` means no actions to be executed. For agent 0, the action duration of its `SHOOT` action is 14 while the action duration of agent 1 `SHOOT` action is 6. When agent 1 shoots the arrow at timestep 0, all agents will receive a positive reward at the end of the episode, making it challenging for TD learning to calculate the exact contribution of each agent.

C.2 Quarry Game

There are n agents in a quarry, as shown in Fig. 12. Agents' task is to complete the n-explosive installation task, and only when all the explosives detonate will agents receive the optimal positive reward. After the installation, agents should go to the safe zones. Otherwise, agents will die and receive a negative reward when the explosive detonates. Agents will receive a medium-level reward given the number of detonated explosives. The explosive has different period to detonate after the installation. At each time step, each agent can observe its position, the position of the quarry, the position of the explosive set by the agent (if any) and the time seconds left for the explosive set by the agent (if any). The agent can select `MOVE_LEFT`, `MOVE_RIGHT`, `NOOP` or `INSTALL_EXPLOSIVE` actions at each time step. Note that each agent cannot observe the status of the other agents and others' explosives. Episode ends after the maximum timesteps or the explosive detonates. To complete the task, agents should place the explosive at the right timestep and return to the safe zone.

C.3 Afforestation Game

In Fig. 12, there are n farmer agents in the farm. To the north of the farm, there is a desert. In the early spring, strong sandstorms may gust from the north and destroy the farm. In order to protect the farm, farm agents should plant trees in the north of the farm. Only when trees are tall enough can they protect the farm. Trees can have different durations to grow, making the off-beatness of the planting action. Agents receive the optimal positive reward when there are n trees can protect the farm before the sandstorm. Note that agents have partial observations and will receive a reward of -0.1 at each timestep in all scenarios shown in Fig. 12. At each time step, each agent can observe its position, the position of the trees planted by itself and the status (position and the age of the tree) of the sandstorm (if any). The agent cannot observe the other agents' positions and trees planted by other agents. Agents can take `MOVE_NORTH`, `MOVE_SOUTH`, `NOOP` or `PLANT_TREE` actions. Similar to Quarry Game and Stag-Hunter, agents should take the right action at the right timestep to complete the task. At the last timestep of the episode, the standstorm will gust and if there are enough trees to protect the farm, the task will be success. Note that agents should return to the safe zone when the standstorm comes. Otherwise, agents will receive a fraction of the punishment. Agents will receive rewards when they fail to return to the safe zone. The reward is proportional to the number of agents who fail to return to the safe zone.

C.4 SMAC

We introduce the off-beat action into SMAC [44]. To make reasonable changes of the environment, we select the `ATTACK` actions as off-beat actions for SMAC scenarios. The rest actions, including `MOVE`, `NOOP` will be executed immediately into the environment after inference. At each time step, agents get local observations within their field of view, which contains information (relative x, relative y, distance, health, shield, and unit type) about the map within a circular area for both allied and enemy units and makes the environment partially observable for each agent. All features, both in the global state and in individual observations of agents, are normalized by their maximum values. Actions are in the discrete space: move[direction], attack[enemy id], stop and no-op. The no-op action is the only legal action for dead agents. Agents can only move in four directions: north, south, east, or west.

D Baselines

We introduce the baselines evaluated in the experimental section. All baselines are summarized in Table 6.

IQL [53]: IQL is an independent Q-learning method for multi-agent RL. Each agent learns its Q values independent with Q-learning [65].

VDN [49]: VDN uses a linear combination of individual Q values to approximate the $Q^{\text{tot}}(\boldsymbol{\tau}, \boldsymbol{u})$ as $Q^{\text{tot}} = \sum_{i=1}^{N} Q_i(\tau_i, u_i)$.

QMIX [41]: QMIX introduces the monotonic constraint on the relationship between Q^{tot} and Q_i:

$$\frac{\partial Q^{\text{tot}}(\boldsymbol{\tau}, \boldsymbol{u})}{\partial Q_i(\tau_i, u_i)} \geq 0, \forall i \in \{1, 2, \ldots, N\}$$

where $Q^{\text{tot}(\boldsymbol{\tau}, \boldsymbol{u})} = f_m(Q_1(\tau_1, u_1), \ldots, Q_N(\tau_N, u_N))$ and f_m is a mixing network used to approximate the Q^{tot}.

QTRAN [47]: QTRAN factorize the $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ with transformation:

$$\sum_{i=1}^{N} Q_i\left(\tau_i, u_i\right) - Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) + V_{\text{jt}}(\boldsymbol{\tau}) = \begin{cases} 0 & \boldsymbol{u} = \overline{\boldsymbol{u}} \\ \geq 0 & \boldsymbol{u} \neq \overline{\boldsymbol{u}} \end{cases}$$

where $V_{\text{jt}}(\boldsymbol{\tau}) = \max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - \sum_{i=1}^{N} Q_i\left(\tau_i, u_i\right)$.

QPLEX[60]: Wang et al. [60] utilizes the established dueling structure $Q = V + A$ [64], advantage function and attention network [57] and introduces the following factorization:

$$\begin{array}{l} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) = V_{tot}(\boldsymbol{\tau}) + A_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) \\ V_{tot}(\boldsymbol{\tau}) = \max_{\boldsymbol{u}'} Q_{tot}\left(\boldsymbol{\tau}, \boldsymbol{u}'\right) \\ Q_i\left(\tau_i, a_i\right) = V_i\left(\tau_i\right) + A_i\left(\tau_i, a_i\right) \\ V_i\left(\tau_i\right) = \max_{a_i'} Q_i\left(\tau_i, a_i'\right) \end{array}$$

EMC [71]: EMC is MARL episodic memory method that utilizes episodic memory of RL [74, 17] in MARL curiosity-driven exploration.

N-step Return and TD(λ) methods [50, 51]: N-step Return and TD(λ) are methods for Q value prediction.

MAVEN [28]: MAVEN builds a latent space with multual information for multi-agent exploration.

RMIX [38]: RMIX aims to learnig risk-sensitive policies for MARL. It replaces the Q value policy with CVaR [43] for risk-sensitive policy learning.

Table 6: Baseline algorithms.

Categories	Methods
MARL Baselines (**Q1**)	QMIX[41] VDN [49] IQL [53] QTRAN [47] QPLEX [60]
EM (**Q2**)	EMC [71]
Bootstrap (**Q3**)	N-step Return & λ-Return [51]
Ex-Risk (**Q4**)	MAVEN [28] EMC [71] RMIX [38]

E Experiment Settings

We implement our method on PyMARL [44] and use 10 random seeds to train each method on all environments. We use opensourced code of baselines publicly by the corresponding authors on Github in all experiments. We use the default settings of PyMARL in our research, including the relay buffer, the mixing network, the training hyperparameters. In order to explore, we use ϵ-greedy with ϵ annealed linearly from 1.0 to 0.05 over 50K time steps from the start of training and keep it constant for the rest of the training for all methods. The discount factor $\gamma = 0.99$ and we follow the default hyper-parameters used in the original papers of all methods in our research. We carry out experiments on NVIDIA A100 Tensor Core GPU and NVIDIA GeForce RTX 3090 24G. We resort to mean-std values as our performance evaluation measurement. We use $\beta = 0.00001$ in Eqn. 2. To create sub-graphs in LeGEM, we first calculate the episode return and keep 1 decimal of it. We then use this episode return to create each sub-graph. We list some important hyper-parameters in Tab. 7.

Table 7: Hyper-parameters

hyper-parameter	Value
Optimizer	RMSProp
Learning rate	5e-4
RMSProp alpha	0.99
RMSProp epsilon	0.00001
Gradient norm clip	10
Batch size	32
Replay buffer size	5,000
Exploration method	ϵ-greedy
ϵ-start	1.0
ϵ-finish	0.05
ϵ-anneal time	50,000 steps
γ	0.99
β	0.00001
Evaluation interval	10,000
Target update interval	200

F Experiment Results

F.1 n-step return and TD(λ)

We provide additional experiment results on n-step return and TD(λ). As illustrated in Fig. 13 and 14, QMIX can attain acceptable performance with some specific values of n and λ in Stag-Hunter Game. However, there is no convincing improvements of performance of VDN, QTRAN and IQL. On Quarry Game (Fig. 15 and 16) and Afforestation Game (Fig. 17 and 18), we can find that TD(λ) cannot help to improve the performance of MARL methods. We can conclude that n-step and TD(λ) have limited ability on improving the performance of MARL methods on OBMAS.

In addition to the empirical results of n-step and TD(λ) returns, we present the results of MARL methods on Afforestation Game. In Fig. 19, we can find that with LeGEM, all four methods get improved performance. We also compare QMIX-LeGEM and VDN-LeGEM with EMC, MAVEN, QPLEX, and RMIX. Despite the simple structure of VDN, VDN-LeGEM performs well and even outperforms QMIX-LeGEM, demonstrating comparable performance with RMIX as depicted in Fig. 7. In Afforestation Game, agents will receive reward when they fail to return to the safe zone. The reward is proportional to the number of agents who fail to return to the safe zone. Such a clear and simple reward rule (*i.e.*, a "hint" for agents) makes learning much easier than that on Quarry and Stag-Hunter Game. This is the main reason why RMIX performs well. We can also find that MAVEN is also showing good performance due to its latent space learning model, which can efficiently learn the environment dynamics of Afforestation Game. QMIX-LeGEM also shows good performance and it outperforms EMC and QPLEX.

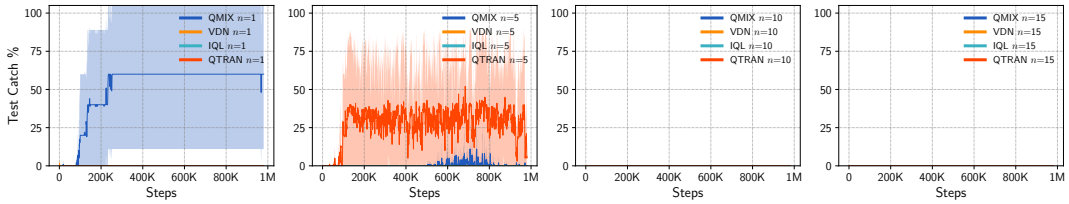

Figure 13: Results of n-step return Stag-Hunter Game.

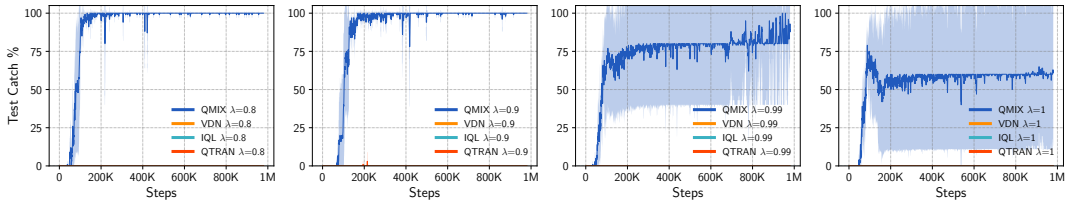

Figure 14: Results of TD(λ) Stag-Hunter Game.

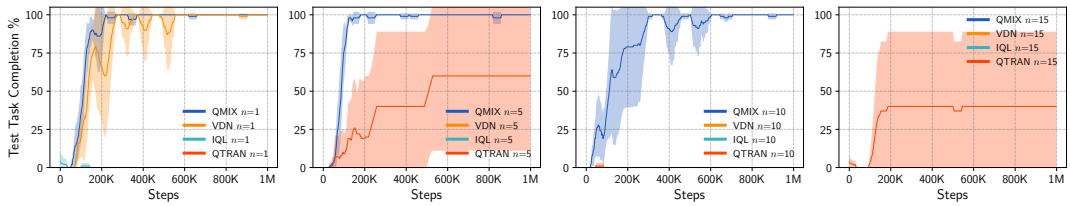

Figure 15: Results of n-step return Quarry Game.

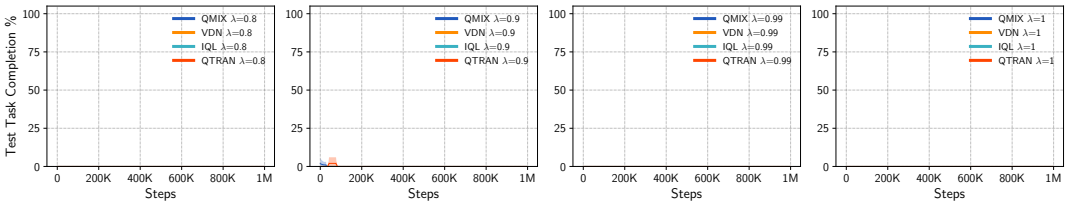

Figure 16: Results of TD(λ) Quarry Game.

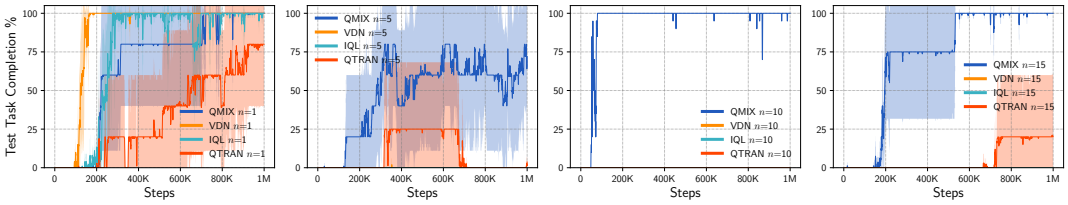

Figure 17: Results of n-step return Afforestation Game.

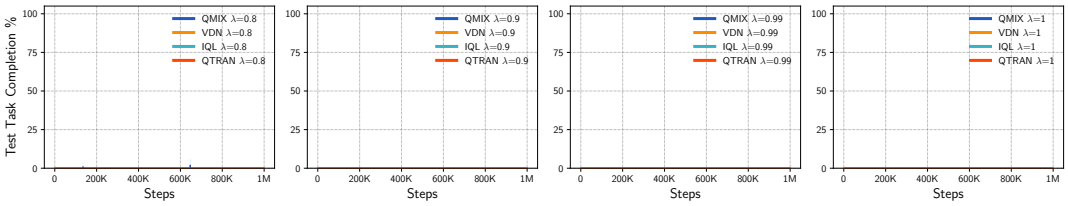

Figure 18: Results of TD(λ) Afforestation Game.

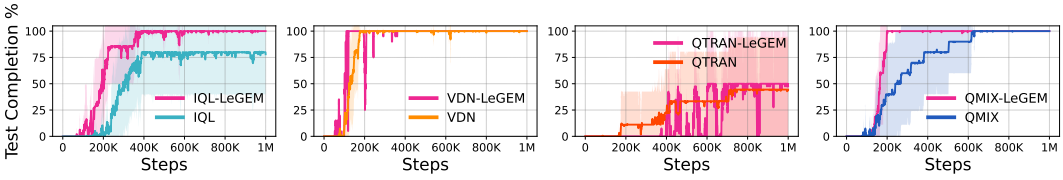

Figure 19: Results on Afforestation Game.

F.2 RUDDER

We present results of QMIX-RUDDER, VDN-RUDDER, IQL-RUDDER and QTRAN-RUDDER on Stag-Hunter Game, Quarry Game, Afforestation Game and SMAC via incorporating RUDDER [2] into QMIX, VDN, IQL and QTRAN. As rewards in off-beat Dec-POMDP are not episodic rewards (c.f. Remark 3), we convert rewards into episodic rewards via using total rewards of the episode as the reward of the last time step of the episode (rewards at other timesteps in the episode are zero) in MARL-RUDDER methods. We use 4 random seeds for training each MARL method with RUDDER.

On Stag-Hunter Game, Quarry Game, Afforestation Game and SMAC scenario 2m_vs_1z, QMIX-RUDDER, VDN-RUDDER, IQL-RUDDER and QTRAN-RUDDER all perform poorly as shown in Fig. 20, 21, 22 and 23. The performances are all zero. There are three reasons caused the poor performance: (i) RUDDER cannot capture the association between off-beat actions and off-beat rewards, making it challenging to detect the pivot timesteps and redistribute the episodic reward to pivot timesteps; (ii) RUDDER conducts the contribution analysis by estimating the reward of each timestep via regression. Due to the sparsity of off-beat rewards and the estimation error of RUDDER, RUDDER redistributes the reward to timesteps around the pivot timestep, rendering the failure of TD learning; (iii) Our setting is a partial observable multi-agent setting, simply redistributing rewards without considering the multi-agent setting can redistribute the reward to the wrong time steps. In TD learning, the estimation of the TD target is essential to the learning of the policy (or the Q value). However, with off-beat actions, n-step return and TD(λ) fail in Off-Beat MARL as shown in Table 2 in the main text and Sec. F.1 in Appendix. It is not surprising to see that RUDDER also fails in Off-Beat MARL.

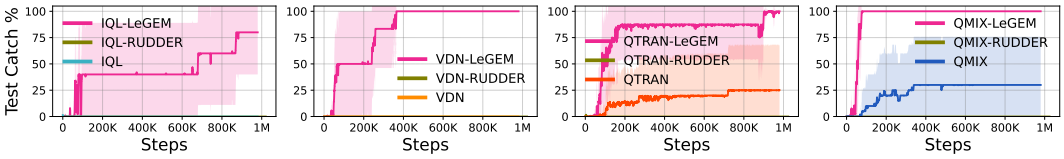

Figure 20: Results of MARL-RUDDER on Stag-Hunter Game.

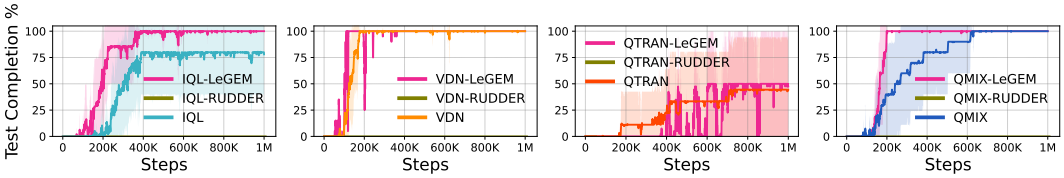

Figure 21: Results of MARL-RUDDER on Afforestation Game.

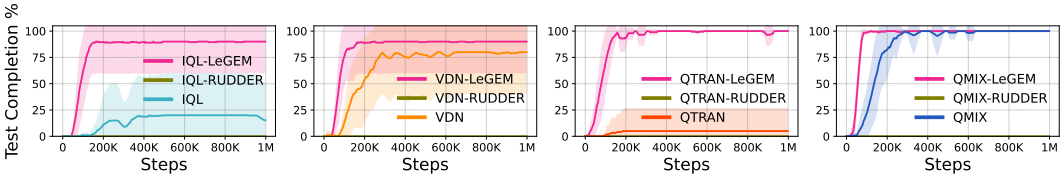

Figure 22: Results of MARL-RUDDER on Quarry Game.

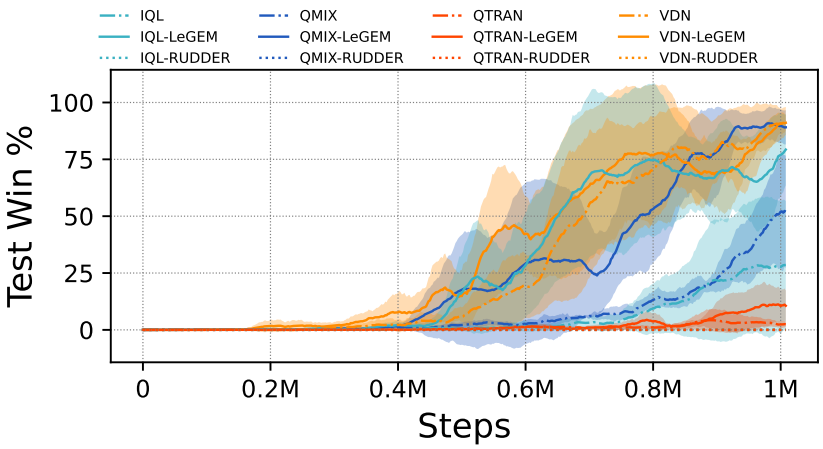

Figure 23: Results of MARL-RUDDER on 2m_vs_1z.