Multi-Agent Chronological Planning with Model-Agnostic Meta Reinforcement Learning

Cong Hu, Kai Xu *, Zhengqiu Zhu, Long Qin and Quanjun Yin

Abstract: In this study, we propose an innovative approach to address a chronological planning problem involving the multiple agents required to complete tasks under precedence constraints. We model this problem as a stochastic game and solve it with multi-agent reinforcement learning algorithms. However, these algorithms necessitate relearning from scratch when confronted with changes in the chronological order of tasks, resulting in distinct stochastic games and consuming a substantial amount of time. To overcome this challenge, we present a novel framework that incorporates meta-learning into a multi-agent reinforcement learning algorithm. This approach enables the extraction of meta-parameters from past experiences, facilitating rapid adaptation to new tasks with altered chronological orders and circumventing the time-intensive nature of reinforcement learning. Then, the proposed framework is demonstrated through the implementation of a method named Reptile-MADDPG. The performance of the pre-trained model is evaluated using average rewards before and after fine-tuning. Our method, in two testing tasks, improves the average rewards from \(-44\) to \(-37\) through \(10,000\) steps of fine-tuning in two testing tasks, significantly surpassing the two baseline methods that only attained \(-51\) and \(-44\), respectively. The experimental results demonstrate the superior generalization capabilities of our method across various tasks, thus constituting a significant contribution towards the design of intelligent unmanned systems.

Keywords: meta reinforcement learning; MARL; chronological planning; cooperative navigation

1. Introduction

In recent years, Multi-Agent Reinforcement Learning (MARL) has attracted a great deal of interest in the AI community. As a cutting-edge research direction, MARL is closely related to decision theory, game theory, optimization methods, agent-based modeling, and so forth. In real-world scenarios, MARL has a broad range of promising applications, such as robotics control [1–3], power grid [4–6], real-time strategy games [7,8], etc. The use of MARL has achieved remarkable performance and exhibits potential economic benefits.

Multi-agent chronological planning problems, where multiple agents need to complete sequential tasks with precedence constraints, are important in both theoretical and real-world domains. However, these problems are yet to be well addressed. In this paper, we consider a chronological scenario where several agents have to cooperatively occupy some landmarks in sequence. This scenario is applicable in many multi-agent systems, such as real-time strategy games and pickup-and-delivery problems.

Many problems related to multi-agent systems can be described with Markov Decision Processes (MDPs) and addressed within the framework of MARL. However, when it comes to chronological planning, the process of training policies for multiple agents could be extremely time-consuming to obtain optimal policies since there might be too many task completion orders. Meanwhile, learned policies can be unstable in performance and easily deteriorate when faced with tiny disturbances or other noise. The existing bottleneck in...
this domain is that the learned policies may suffer from unsatisfactory performance in a new environment (a new task completion order) even though the drift of an underlying Markov decision process, e.g., system dynamics or the reward mechanism, is quite slight. These challenges render the direct deployment of a previously learned policy in a new task order impractical. However, learning from scratch is not only time-consuming but also fails to leverage prior experience effectively.

To circumvent the above-mentioned concerns about MARL, researchers have proposed a novel paradigm—meta-learning—to improve the generalizability of machine learning algorithms with regard to adapting to new environments. Meta-learning is a universal paradigm that can be combined with supervised learning and reinforcement learning. Recent advances in meta-learning enable reinforcement learning to learn from a distribution of different yet related tasks and solve new tasks within a few trials. In the realm of reinforcement learning, meta-learning can be used to learn good initialization parameters [9,10], improve exploration efficiency [11,12], and learn proper hyperparameters [13]. The use of meta-learning techniques enables us to leverage learned skills in the past from MARL, and simultaneously improve the robustness to the disturbance of MDPs.

Although meta-learning has proved to be effective in adapting policies to new environments, applying meta-learning to multi-agent chronological planning remains less investigated. To close this knowledge gap, we first design a reward setting for the chronological scenario that can effectively describe scenarios of different chronological tasks. Then, we use a classical MARL algorithm—Multi-Agent Deep Deterministic Policy Gradients (MADDPG)—to train the policies of several cooperative agents in the environment. To improve the generalization across different tasks of the learned policies, we encapsulate the MARL part into a meta-learning framework—Reptile—to produce more general policies that can adapt to a new chronological planning scenario with a few interactions.

We respectively examine methods on MADDPG with and without a meta-learning framework. We pre-train these two methods on training tasks and then fine-tune the learned policies on two testing tasks to assess the generalization capability. We also consider a random policy as a baseline when fine-tuning on testing stage. The empirical results show that our method—Reptile-MADDPG—adapts to the testing tasks fastest, and achieves the highest rewards. We prove that the meta-learning framework can help MARL to quickly adapt to different but related tasks.

In summary, this work aims to contribute to these aspects:

• First, we propose a framework for addressing chronological planning problems by bridging the knowledge gap through the application of meta-learning to multi-agent reinforcement learning techniques. Based on the framework, we instantiate a method called Reptile-MADDPG to improve the generalizability of chronological planning across different tasks.

• Second, we design a reward setting that can effectively describe scenarios of different chronological tasks.

• Finally, we conduct extensive testing and comparison with existing methods. Our method shows faster adaption to testing tasks and obtains higher rewards, proving the efficiency of the meta-learning framework in quickly adapting to different but related tasks.

The rest of this paper is organized as follows: we first introduce related work on MARL and meta-reinforcement learning in Section 2 and then propose our method, referred to as Reptile-MADDPG, in Section 3. In Sections 4 and 5, we describe the design of our experiments and present the results to show the effectiveness of our method. Section 6 contains the concluding remarks about the whole work and directions for future research.

2. Related Work

2.1. Reinforcement Learning

The field of reinforcement learning (RL) holds significant importance in the realm of machine learning and has achieved noteworthy advancements across various domains. In
recent years, RL has greatly benefited from the progress made in deep learning. Combined with deep learning, RL can solve some tough decision-making tasks intractable before. As a landmark paper in the field of reinforcement learning, DeepMind introduced DQN in 2015 [14]. DQN uses a deep neural network to more effectively represent Q-values and achieves a remarkable performance in Atari games. Lin et al. introduced an innovative method termed Reinforcement Q-learning-based Deep Neural Network (RQDNN), which synergistically integrates the Deep Principal Component Analysis Network (DPCANet) and Q-learning for playing strategy video games [15]. Srinivasu et al. presented a robust reinforcement learning-based algorithm utilizing Probabilistic Roadmap and Inverse Kinematics for accurate path recognition and approximation in real-time surgical procedures, offering an optimal solution for performing precise surgeries on soft tissues [16]. To improve the applicability of DQN in real-world environments, Kim et al. strategically incorporated prior knowledge through a Bayesian-based loss function, notably improving the learning convergence performance [17].

2.2. Multi-Agent Reinforcement Learning

MARL extends RL to a Multi-Agent System (MAS). MARL is much more complex because the agents need to make decisions based on observations and interactions with dynamic environments and their partners [18]. The most natural approach to finding policies for MARL is that each agent learns its policy independently and treats the rest of the agents as part of the environment. This idea is implemented in the Independent Q-Learning (IQL) method [19]. In 2015, Tampuu et al. [20] proposed Independent Deep Q-Network (IDQN), which extends IQL with DQN. These methods suffer from the non-stationarity of the environment. That means the policies of other agents keep changing during the training process and result in unstable training for the learning algorithm. Non-stationarity also prevents the straightforward use of experience replay in methods like DQN. Meanwhile, policy gradient RL methods usually exhibit very high variance when the coordination of multiple agents is required [21].

Many approaches in MARL adopt the framework of centralized training and decentralized evaluation (CTDE). In this framework, the policies of a group of agents are trained in a centralized way and granted access to other agents’ information and the global states during the centralized training process. While in the decentralized execution phase, each agent makes its own decision based on its local action-observation information. Among these works, DIAL [22], CommNet [23], ATOC [24], and SchedNet [25] aim to enable agents to learn how to communicate with each other in multi-agent systems. Some other approaches try to use a fully observable critic to let agents learn to achieve cooperation directly from their own local observations. MADDPG is the first general-purpose MARL algorithm for stabilizing training. It uses an actor-critic learning framework and can be applied to mixed competitive and cooperative environments. With the fully observable critics, MADDPG eliminates the non-stationary in MARL by explicitly conditioning on the actions of other agents. As the critics in MADDPG concatenate all the local observations, it faces the curse of dimensionality when agents increase. To alleviate this problem, Iqbal et al. [26] proposed Multiple-Actor-Attention-Critic (MAAC) that involves an attention mechanism to select relevant information for each agent during training.

In cooperative settings, a group of agents has to coordinate to maximize a shared team reward. Therefore, the global Q function $Q^\pi(o, a)$ is conditioned on the joint states and actions of all the agents. However, in a multi-agent system, some agents may get lazy and not learn to cooperate as they are supposed to [27], which may lead to the breakdown of the whole system. To solve this issue, some works in MARL focus on the factorization of this global reward. Sunehag et al. [28] proposed Value-Decomposition Networks (VDN) that learn an optimal linear decomposition for Q values: $Q^\pi(o, a) := \sum_{i=1}^{N} Q^i(o^i, a^i)$. The implicit value function learned by each agent depends only on local observations. As the structure of Q values considered by VDN is too simple, Rashid et al. [7] proposed QMIX, which makes an improvement over VDN by adding the constraints that the joint-action
value is monotonic in the per-agent values. Wang et al. [29] also proposed QPLEX, which takes a duplex dueling network architecture to factorize the joint value function, making value function learning more efficient.

Even though the above MARL methods perform well in some scenarios, when the task varies, one has to retrain the policies from scratch, which takes plenty of time.

2.3. Meta Reinforcement Learning

Early works on the idea of meta-learning (also known as learning to learn) date back to the 1990s [30,31]. Recently, meta-learning has received much attention and has been applied in few-shot recognition [32], network routing [33], traffic control [34], and in biomedical [35] and other domains [36–41]. Meta-learning is regarded as the key to achieving human-level intelligence because it raises the learning level from data to tasks [42].

Here, we focus on meta-learning in the reinforcement learning community, meta reinforcement learning (meta-RL). The goal of meta-RL is to adapt to a new test task quickly using only a small amount of experience in the test setting without learning from scratch. In meta-RL, the train and test tasks are usually different but drawn from the same family of problems [43]. In this paper, we divide the recent methods of meta-reinforcement learning into recurrent neural network-based and optimization-based methods.

RNN-based meta-RL is very similar to the standard RL algorithms. The current state $s_t$, the last reward $r_{t-1}$, and the last action $a_{t-1}$ are also fed into the policy network of model-based meta-RL. Duan et al. [44] proposed a meta learning method that trains a gated recurrent unit (GRU) to remember the construct of the current MDP, so that it can adapt to an unseen but familiar MDP task quickly by fine-tuning the parameters of the GRU. During the training procedure, the hidden states of the GRU are not cleared between episodes. Similar to [44], Wang et al. [45] used an LSTM as the memory module. Frans et al. [46] present a hierarchical meta reinforcement learning method that learns hierarchically structured policies and uses shared primitives to improve sample efficiency on new tasks.

Optimization-based meta-RL aims to update the model parameters to achieve good generalization across new tasks. Finn et al. [9] proposed a general method called model-agnostic meta learning (MAML). MAML aims to optimize initialized parameters on some training tasks so that the parameters can quickly adapt to similar unseen tasks. MAML is model-agnostic so that it can combine with any learning models trained with gradient-descent. To reduce the expensive computation from the use of second-derivatives in MAML, Nichol et al. [10] proposed a first-order meta learning method, Reptile, that performs SGD that updates the initialized parameters towards the average of updated task-specific weights. Evolved Policy Gradient aims to build a differentiable loss function that is parameterized via temporal convolutions over the agent’s historical information [47]. When tackling new tasks, one can optimize the policy by minimizing the loss function. Meta Q-Learning (MQL) [48] is an off-policy meta learning method and draws upon ideas in propensity estimation to amplify the amount of available data for adaptation.

Recently, some works tried to apply meta learning to multi-agent systems. Al-Shedivat et al. [49] studied the continuous adaptation problem in a multi-agent environment, RoboSumo, and enabled a learning agent to adapt to different enemies in a short timeframe. However, they did not actually research multi-agent learning, but focused on a single agent in a two-agent competitive environment. Jia et al. [50] combined meta learning with MADDPG and successfully enabled MADDPG to adapt to new tasks, with different friction coefficients or numbers of agents. Li et al. [18] trained a meta-actor and a meta-critic to distill the meta-knowledge of a team that can help a new agent to better integrate into the group. Both [18,50] tested their meta learning methods in multi-agent systems. Existing works primarily focus on variations in kinetic parameters or the number of agents between training and testing tasks, neglecting the interrelationships among agents. In contrast, our study addresses the complexity of chronological constraints between agents. This leads to a more challenging scenario where the reward function in the stochastic game may vary across different tasks.
3. Methods

In this section, we first formulate the chronological planning problem in a mathematical form. Then, we elaborate on how to address the problem together with a reward function we designed. Furthermore, we show a meta-learning framework combined with multi-agent reinforcement learning to acquire learned policies for chronological planning with varying orders.

3.1. Chronological Planning Problem Formulation

We consider a multi-agent chronological planning problem in a navigation task with precedence constraints, where agents autonomously plan their routes toward their targets in a specific chronological order. There are $N$ agents and $N$ landmarks in our scenario. The task is to navigate the $N$ agents to occupy all the $N$ landmarks in an order and simultaneously avoid collisions with each other. At the start of each episode, the positions of both agents and landmarks are randomly reset. Moreover, we have excluded generated positions where any two landmarks are too close, preventing unavoidable collisions when the agents are approaching their destination.

A simple example is shown in Figure 1. In addition, the precedence constraint requires that agents must occupy the $N$ landmarks in a certain order. An agent can access the position and velocity of itself, as well as the positions of other agents and all the landmarks. Each agent can also observe whether each landmark is occupied.

As the agents in our scenario must consider the interaction with other agents when planning the routes (i.e., making decisions), we use stochastic game (SG), also known as Markov game, to describe the multi-agent decision-making process in our problem. A stochastic game can be regarded as a multi-player extension to an MDP, which allows agents to move simultaneously [51].

The stochastic game of our problem is defined by the number of agents $N$, the state set $S$, the action sets $\{A^1, A^2, \ldots, A^N\}$, and the observation sets $\{O^1, \ldots, O^N\}$. For each agent $i$, $o^i \in O^i$ is an observation of the global state $s \in S$. In our scenario, $o^i$ contains the absolute position and velocity of the $i$-th agent itself and the relative locations of all the other agents and landmarks. For the sake of universality and reasonability, we define the global state $s \in S$ as the combination of observations from the $N$ agents, $(o^1, \ldots, o^N)$, where $o^i \in O^i$. To describe the environmental dynamics, state transition $P(s'|s,a)$ gives the distribution of the next state $s'$ when taking a joint action $a = (a^1, \ldots, a^N)$ under the current state $s$, and per-agent reward function $R(s,a,s')$ returns a scalar value for agent $i$ for a transition from $(s,a)$ to $s'$. Our reward function will be explained in detail in Section 3.2. The policy of agent $i$ is represented by $\pi^i = P(a^i|s)$, which gives the action probabilities with regard to a given state and we denote the joint policy of all the $N$ agents by $\pi = (\pi^1, \ldots, \pi^N)$.
In a stochastic game, the goal of each agent is to find a behavioral policy \( \pi^i \) that can take sequential actions at every step \( t \) such that a discounted cumulative reward in Equation (1) is maximized, where \( \gamma \) is the decay factor. Here, we use the superscript of \((i, \cdot^{-i})\) to distinguish between agent \( i \) and all the other \( N-1 \) teammates. In \((\pi^i, \pi^{-i})\), the former item means the policy of agent \( i \) while the latter one means the joint policy of all the agents except agent \( i \).

\[
V^{\pi^i, \pi^{-i}}(s) = \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t), a_{t+1} \sim \pi^{-i}(\cdot|s_{t+1})} \left[ \sum_{t \geq 0} \gamma^t R^i_t(s_t, a_t, s_{t+1}) \mid a^i_t \sim \pi^i(\cdot|s_t), s_0 \right]
\]  

As shown in Equation (1), the optimal strategy of each agent is not only determined by its own policy \( \pi^i \), but is also affected by the joint policy of other agents in the environment \( \pi^{-i} \). This brings non-stationarity to the training process, the fundamental difference in the solution concept between single-agent RL and multi-agent RL. Furthermore, we can clearly see in Equation (1) that the key to optimizing strategies is to design a reward function \( R^i \) that can correctly guide agents to complete the task in chronological order. In the next section, we explain our design of the reward function in detail.

3.2. Reward Function Design

In reinforcement learning, a reward function gives an assessment to the actions of agents and can guide the agents to maximize their objectives. The design of the reward function can largely affect the convergence of RL algorithms [52,53]. Based on our chronological scenario, we design a reward function that can well reflect the tasks in chronological order.

The reward function consists of three parts. First, the agents need to reach their landmarks as soon as possible, to achieve efficient task completion. Therefore, we give a negative reward at each step according to the distance \( d^i \) between agent \( i \) and its corresponding landmark:

\[
r^i_{\text{dis}} = -d^i.
\]

This part of the reward motivates the agents to approach their landmarks as soon as possible to reduce the penalty (negative reward).

Second, the agents should not collide with each other. Therefore, we give a penalty part if the distance between any two agents is smaller than the sum of their radii.

\[
r^i_{\text{collide}} = \begin{cases} 
0 & d^{ij} \geq (\text{Rad}^i + \text{Rad}^j), \forall j \neq i \\
-1 & d^{ij} < (\text{Rad}^i + \text{Rad}^j), \exists j \neq i
\end{cases}
\]

\( d^{ij} \) means the distance between agent \( i \) and agent \( j \) and \( \text{Rad}^i \) and \( \text{Rad}^j \) represents their radii.

The last part is a negative reward that penalizes the agents who break the precedence order. In our setting, the \( N \) agents are required to occupy their landmarks one by one. We represent a precedence constraint as \((v^1, v^2, \ldots, v^N)\). This means that, for any \( i > 1 \), agent \( v^i \) must occupy its landmark after agent \( v^{i-1} \), which is called the precedent agent of agent \( v^i \). In our setting, we describe this chronological constraint using the distances between agents and their targeted landmarks. We give a penalty, i.e., a negative reward, to the agent \( i \) (whose precedent agent is agent \( j \)) if the distance from agent \( i \) to its landmark is less than that of agent \( j \). The penalty is described in Equation (4).

\[
r^i_{\text{order}} = -\max(0, d^i - d^j).
\]

Considering all three penalties together, the reward function in our scenario is:

\[
R^i = r^i_{\text{dis}} + r^i_{\text{collide}} + \beta r^i_{\text{order}}.
\]
where \( \beta \) is the coefficient of the precedence penalty. For simplicity, the coefficients of \( r_{\text{dis}}^i \) and \( r_{\text{collide}}^i \) are set to 1 as default as in previous research. Specifically, the third part of the reward varies in tasks with different \( r_{\text{order}} \).

3.3. Framework

Our proposed solution to the aforementioned stochastic game involves the application of a multi-agent reinforcement learning algorithm to train cooperative policies for the \( N \) agents. To address the generalizability problem, where tasks may present in various chronological orders, we incorporate a meta-learning method. This leads to a two-part framework comprised of the MARL and meta components, as illustrated in Figure 2.

This dual-stage framework operates in the following manner. The MARL component is responsible for handling agent interactions and coordination to solve a specific stochastic game (depicted as the inner loop in Figure 2). The meta component, on the other hand, concentrates on generalization across different tasks (represented as the outer loop in Figure 2). This element of the framework distills meta-knowledge from the training outcomes of the inner loop. This meta-knowledge is instrumental in swiftly adapting to new tasks.

![Figure 2. The framework of the proposed method that combines meta-learning and multi-agent reinforcement learning.](image)

We use MADDPG [21] and Reptile [10] to instantiate a method called Reptile-MADDPG based on the proposed framework. In the rest of this section, we will explain our method in detail.

3.3.1. Multi-Agent Deep Deterministic Policy Gradients

MADDPG is an actor-critic method and adopts the paradigm of centralized training and distributed execution. It extends the Deep Deterministic Policy Gradient (DDPG) algorithm to the multi-agent context. The extension facilitates decentralized execution where each agent takes actions based only on its local observations while still permitting centralized training with access to the observations and actions of all agents.

In the MADDPG framework shown in Figure 3, each agent employs two primary components: a critic \( V^i \) and an actor \( \pi^i \). The actor takes the current observation \( o \) as input and outputs an action \( a \), dictating the policy the agent should follow. The critic, meanwhile, predicts the Q-value of a given state–action pair, estimating the expected return for taking an action in a particular state following the actor’s policy.

Unlike traditional DDPG, where the critic only evaluates the Q-value of a state-action pair from a single agent’s perspective, in MADDPG, the critic of each agent takes into account the actions and observations of all other agents. Having global information, MADDPG can eliminate the non-stationary in MARL by explicitly conditioning on the actions of other agents. As we know the actions of other agents, and the environment is stationary even as the policies change, since \( P(s'|s,a^1,\ldots,a^N,\pi^1,\ldots,\pi^N) = P(s'|s,a^1,\ldots,a^N) = P(s'|s,a^1,\ldots,a^N,\tilde{\pi}^1,\ldots,\tilde{\pi}^N) \) for any \( \pi^i \neq \tilde{\pi}^i \). This design enables agents to learn policies that are aware of the actions of other agents and promote cooperative behavior.
Overall, MADDPG represents an efficient and effective method for training multi-agent systems, leveraging the power of deep learning and reinforcement learning within a framework specifically designed to handle the complexities of multi-agent environments.

3.3.2. Meta Learning for MARL

To quickly adapt to unseen but related tasks, we incorporate MADDPG into a meta-learning method. Researchers have proposed many meta-learning methods and some of them aim to find a proper initialization of the neural network’s parameters that can quickly be fine-tuned for new tasks. Their objective is training to maximize the expectation of rewards across a task set:

\[
\max_{\theta} E_{T_i \sim p(T)} [R(\phi)], \quad \text{where } \phi = f_{\theta}(T_i),
\]

where \( p(T) \) is the distribution of all possible tasks and \( T_i \) means a specific task sampled from \( p(T) \). \( \theta \) is the meta parameters to be trained and \( \phi \) is the parameters adapted from \( \theta \) for a specific task \( T_i \). \( f_\theta \) indicates the base learner parameterized by \( \theta \).

In our method, MADDPG is the base learner that learns parameters \( \phi \) starting from \( \theta \) to reduce the loss on a specific task \( T_i \).

Here, we take Reptile [10] as the meta-learning part in our method. Reptile is a simple but effective meta-learning method, involving only first-order gradient information, bringing a faster training speed with negligible accuracy loss.

Similar to other meta-learning methods, Reptile consists of two stages, as illustrated in Figure 4. During the inner update stage, Reptile uses the base learner (a policy gradient reinforcement learning algorithm) to train on a certain task, updating the network parameters according to the gradient \( \nabla_i \) at each step. After several steps, the network parameters \( \phi_i \) specific to that task are obtained. Subsequently, the meta update stage takes place. In this phase, the meta-parameters \( \theta \) are updated based on the results of the training performed on the individual task, as Equation (7) shows. Through gradient descent of the outer update, Reptile learns easily adaptable model parameters for various tasks. This two-step iterative process allows Reptile to generalize across a variety of tasks and adapt swiftly to new ones. Reptile is suitable for any gradient-based learning algorithms and does not introduce any extra parameters to be learned.

\[
\theta' = \theta + \epsilon (\phi_i - \theta).
\]
3.4. Procedure

Our method consists of two stages, meta-training and meta-testing. In the meta-training stage, we optimize the model parameters of each agent to minimize their losses on a set of training tasks. In the meta-testing stage, we use the learned meta-parameters to adapt to unseen but related tasks. We collect experiences from a new task and use them to fine-tune the meta parameters to reduce the loss on a specific task.

As we have shown in the framework of our proposed method (Figure 2), the overall method is a two-level optimization, where the inner loop corresponds to the base learner that only considers a single task, while the outer loop trains the meta parameters \( \theta \) that determine the base learner in the inner loop. In Figure 2, different tasks are implicitly expressed in their corresponding environments. The pseudo-code of meta training is shown in Algorithm 1.

**Algorithm 1: Reptile-MADDPG**

1. Randomize \( \theta \)
2. for each epoch do
   3. Sample tasks set \( T \) from the task distribution.
   4. for \( T_i \in T \) do
      5. \( \phi_i \leftarrow \theta \)
      6. for \( k = 1, 2, \ldots, K \) do
         7. Sample \( D_i = \{(s, a, r, s')\} \) using \( f_{\phi_i} \) in task \( T_i \)
         8. Calculate \( \mathcal{L}_{T_i}(f_{\phi_i}) \)
         9. Inner Update: \( \phi_i \leftarrow \phi_i - \alpha \nabla_{\phi_i} \mathcal{L}_{T_i}(f_{\phi_i}) \)
      10. end
   11. end
   12. Meta Update: \( \theta \leftarrow \theta + \epsilon \sum_{i=1}^{\left| T \right|} (\phi_i - \theta) \)
13. end
14. Return \( \theta \)

\[ \phi_i = \theta - \alpha(g_1 + g_2 + g_3 + g_4) \]

**Figure 4.** Overview of Reptile algorithm.

For every cycle of the outer loop, one or more tasks are sampled from a set of tasks that share some common structure. The current meta parameters \( \theta \) and the sampled task(s) are given to the inner loop, while in the inner loop, agents initialize their network parameters \( \phi \) with \( \theta \) and then interact with the sampled training tasks and optimize their parameters.
for the maximal average reward. In detail, \( N \) agents collect experiences from each training task and store them in the corresponding replay buffers. For every training step, we randomly sample a batch of transitions \( \{(s_k, a_k, r_k, s'_{k})\}^B \) from the replay buffer, where \( B \) means the batch size. Then, each agent uses the batch to update its actor and critic based on Equations (8) and (9).

\[
\nabla_{\phi} J(\mu) = \frac{1}{B} \sum_k [\nabla_{\phi} \mu(o_k) Q_i(x, a_1^k, \ldots, a_N^k) | a_i^k = \mu(o_i^k)] \quad (8)
\]

\[
L(\phi) = \frac{1}{B} \sum_k [(Q_i^t(x_k, a_1^k, \ldots, a_N^k) - y_k)^2], \quad (9)
\]

\[
y_k = r_k^t + \gamma Q_i^t(x_k', a_1^k, \ldots, a_N^k) | a'_j^k = \mu'(o_j^k).
\]

Every time the policy networks and \( Q \) networks are updated, the corresponding target networks will be soft-updated with hyperparameters \( \tau \) like Equation (10).

\[
\phi' \leftarrow (1 - \tau) \phi' + \tau \phi. \quad (10)
\]

When the inner loop finishes training, the outer loop concludes its results on the batch of tasks to update the meta parameters. To be specific, the outer loop collects the learned policy parameters \( \{\phi_i\}_{i=1}^{\mid T\}} \) and updates the meta parameters \( \theta \) as Equation (11).

\[
\theta \leftarrow \theta + \epsilon \sum_{i=1}^{\mid T\}} (\phi_i - \theta). \quad (11)
\]

After training, the meta parameters \( \theta \) that can quickly adapt to a specific task in the stage of meta-testing are acquired. The process of meta-testing is the same as the inner loop, representing a typical MARL process. With the initialization of \( \theta \), meta-testing could potentially learn good parameters quickly.

4. Experiments

In this section, we introduce the experimental settings where we evaluate the effectiveness of our method to solve the chronological planning problem, as well as the details of the state and action features, network architectures, and training and testing regimes. As explained in the Introduction, we mainly focus on whether the trained policies have acceptable generalizability and quick adaption across various tasks in the experiments.

4.1. Experimental Settings

To evaluate our method, we use a popular test-bed in MARL community, Multi-Agent Particle Environment (MPE) (https://github.com/openai/multiagent-particle-envs (accessed on 20 April 2022)), to instantiate our chronological planning problem. MPE is a simulation environment for multi-agent systems and provides several built-in scenarios that represent some typical cases of collaboration and competition. MPE is widely used in the validation of MARL algorithms due to its usability and extensibility.

As MPE is not intentionally designed for meta-learning, we design our own scenario that supports multiple tasks. In our scenario, there are \( N \) cooperative agents in a continuous environment with a size of \( 2 \times 2 \) and the radius of an agent is 0.15. They can choose a continuous action and move omnidirectionally by setting the forces in two orthogonal directions within a range of \([-1, 1]\), and aim to cover \( N \) landmarks without collisions. Each agent can obtain the \( x - y \) locations of all the agents and landmarks, and make decisions by itself without any communication with others. The positions of agents and landmarks are randomly reset at the beginning of each episode. To evaluate the ability to adapt to unseen tasks, we obtain six tasks by changing the chronological orders of the three agents in our
scenario. We randomly select four of them as training tasks and the other two as testing tasks in the experiments.

4.2. Experimental Conditions

The proposed Reptile-MADDPG approach is referred to as META in our experiments. To evaluate the generalizability and the ability of quick adaption, we select two baseline conditions.

As MADDPG is not specifically defined for multiple tasks, we introduce Domain Randomization [54,55] and provide MADDPG diverse training data from different tasks to improve its generalization (referred to as DR).

Furthermore, during the testing stage, we also consider the policy consisting of random parameters without pre-training (referred to as RANDOM).

4.3. Implementation Details

We train an actor and a critic for each agent. For all experiments, we use MLPs with three layers (ReLU) for both the actor and critic networks and the number of neurons for the hidden layers is 64.

We pre-train our method and original MADDPG without meta-learning on the four training tasks and then evaluate them on the two testing tasks with fine-tuning. At the pre-training stage, we run the two methods for 10,000 epochs. In each epoch, we randomly sample three training tasks and train the models for an episode (25 steps) in each task. For every 25 epochs, we validate the current policies on the sampled tasks without fine-tuning, and record the average rewards.

In the following testing stage, we fine-tune the two pre-trained policies on each testing task for 400 episodes and record their rewards during adaptation. For every five episodes during fine-tuning, we evaluate the three policies and record the average rewards on the current testing task. We repeat the fine-tuning process for five runs and show the average results with 95% confidence intervals (95% CI). All the experiments are run on an AliCloud server of Intel(R) Xeon(R) Platinum 8163 CPU with 16 GB memory.

4.4. Parameter Study

We perform a parameter study on $\beta$ in the reward function of our setting to show the compatibility of our method. We first remove the chronological constraints by simply setting the coefficient of chronological penalty $\beta$ to zero. Therefore, the six tasks become indistinguishable. Then, we run the pre-training process of META and DR for 5000 epochs and fine-tune the learned policies for 10,000 steps, and we keep other details unchanged from the preceding experiments. We repeat the experiments for five runs with different random seeds and compare the achieved rewards of META and DR on the tasks without chronological constraints.

We also study the impact of different $\beta$. We separately set $\beta$ to each of $[1.0, 1.5, 2.0, 3.0]$, pre-train META and DR for 10,000 epochs, and then fine-tune the learned meta-policies for 10,000 steps and we keep the other details unchanged from the preceding experiments.

5. Results and Discussion

We present our results in terms of the meta-training stage, the meta-testing stage, and the parameter study.

5.1. Results of Meta Training

We show the learning curves during the pre-training process in Figure 5. We compare our method, meta-learning framework (Reptile-MADDPG, denoted as META), with the baseline condition MADDPG with Domain Randomization (denoted as DR). The learning curves are the average results of five runs with different random seeds; for each seed, it takes about 50 min for both META and DR in the pre-training process.
The figure indicates that the learning curves of both META and DR converge after around 2000 training epochs. META and DR perform similarly during pre-training, while META increases a bit more quickly and has slightly higher rewards; that is to say, in the pre-training stage, META can achieve better rewards in a comparable convergence time compared to DR.

![Figure 5. Average rewards on training tasks during pre-training and 95% CI.](image)

5.2. Results of Meta Testing

The learning curves of fine-tuning on testing tasks are demonstrated in Figure 6, and the statistical results are also shown in Tables 1 and 2, including the average rewards and standard deviations averaged over five runs. For each run, it takes about 90 min for the meta-testing stage.

The learning curves in Figure 6 show that, at the beginning of fine-tuning, both the rewards achieved by META and DR drop a little compared with those on training tasks because the chronological constraint varies. Then, in the next episodes, the rewards of the two pre-trained policies decrease because they have to adjust their policies to skip the local minima and find a set of parameters more suitable for the current testing task. After about 50 episodes, the performance of META starts to increase quickly and achieve very good rewards on the testing tasks. On the contrary, the performance of DR grows slowly and cannot even reach the level before fine-tuning within 400 episodes.

![Figure 6. Average rewards on testing tasks during fine-tuning and 95% CI.](image)

The quantitative results show the testing results on two testing tasks separately and further demonstrate the advantage of our method. We can see from Tables 1 and 2 that META always performs the best from 0 to 10,000 steps in comparison to DR and RANDOM baselines. META acquires about 15% higher rewards at step 10,000 compared with
that at 0 steps, while DR cannot adapt to the testing tasks within the given fine-tuning steps and only achieves rewards of about \(-50\) at 10,000 steps. Although the rewards of RANDOM grow most rapidly, META reaches \(-45\) before 6000 steps while RANDOM takes 10,000 steps, where META can achieve rewards higher than \(-40\). This means that META can achieve acceptable results in fewer fine-tuning steps during the process of fast adaptation.

These results prove that the meta-learning framework can effectively help MARL to adapt to unseen tasks quickly.

Table 1. Quantitative results of meta-testing on task 1.

<table>
<thead>
<tr>
<th>Policy</th>
<th>0 Steps</th>
<th>2000 Steps</th>
<th>4000 Steps</th>
<th>6000 Steps</th>
<th>10,000 Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>META</td>
<td>(-44.21 \pm 2.86)</td>
<td>(-49.26 \pm 3.66)</td>
<td>(-45.79 \pm 1.20)</td>
<td>(-43.13 \pm 2.51)</td>
<td>(-37.32 \pm 1.85)</td>
</tr>
<tr>
<td>DR</td>
<td>(-45.01 \pm 2.08)</td>
<td>(-53.17 \pm 2.76)</td>
<td>(-52.08 \pm 2.33)</td>
<td>(-52.15 \pm 3.10)</td>
<td>(-52.22 \pm 4.12)</td>
</tr>
<tr>
<td>RANDOM</td>
<td>(-121.32 \pm 3.54)</td>
<td>(-70.35 \pm 3.24)</td>
<td>(-58.12 \pm 3.24)</td>
<td>(-52.13 \pm 1.78)</td>
<td>(-43.32 \pm 2.66)</td>
</tr>
</tbody>
</table>

Table 2. Quantitative results of meta-testing on task 2.

<table>
<thead>
<tr>
<th>Policy</th>
<th>0 Steps</th>
<th>2000 Steps</th>
<th>4000 Steps</th>
<th>6000 Steps</th>
<th>10,000 Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>META</td>
<td>(-43.78 \pm 2.81)</td>
<td>(-47.44 \pm 3.42)</td>
<td>(-47.10 \pm 2.73)</td>
<td>(-43.62 \pm 2.59)</td>
<td>(-38.18 \pm 2.05)</td>
</tr>
<tr>
<td>DR</td>
<td>(-46.04 \pm 4.04)</td>
<td>(-55.44 \pm 2.47)</td>
<td>(-53.09 \pm 2.32)</td>
<td>(-52.09 \pm 2.74)</td>
<td>(-50.28 \pm 2.88)</td>
</tr>
<tr>
<td>RANDOM</td>
<td>(-122.00 \pm 6.68)</td>
<td>(-75.98 \pm 2.98)</td>
<td>(-59.56 \pm 3.19)</td>
<td>(-52.83 \pm 2.03)</td>
<td>(-44.38 \pm 3.48)</td>
</tr>
</tbody>
</table>

5.3. Results of Parameter Study

Our parameter study first tests META and DR on a task without chronological constraints to show the compatibility of our method. We show the learning curves of pre-training and fine-tuning separately in Figures 7 and 8. Both of them are averaged from five runs with different random seeds. We can see that, without the precedence penalty (\(\beta = 0\)), META and DR perform very closely to one another. At the pre-training stage, both their rewards converge to about \(-10\) after 1000 epochs, while at the fine-tuning stage, the curves of META and DR continue to fluctuate and keep very close. This indicates that our method can also deal with a single task and the meta-learning framework does not harm the performance of the MARL algorithm in the inner loop.

In addition, the impact of varying the parameter \(\beta\) is illustrated in Figure 9. We expected that, with larger \(\beta\), the reinforcement learning would obey the chronological constraints more strictly. However, the reinforcement learning became very unstable when \(\beta\) increased. As the fourth sub-figure in Figure 9 shows, the training curve of META drops at 8000 epochs. We will try to find the reason for this in future work.

![Figure 7](image-url)
6. Conclusions

In this study, we addressed a complex multi-agent chronological planning problem by proposing an innovative method that combines multi-agent reinforcement learning with a meta-learning framework. Recognizing the critical role that a properly defined reward function plays, we designed one that encapsulates the chronological constraints inherent in
the problem. Additionally, the leverage of meta-learning allows our model to efficiently generate adaptive policies for agents, enabling them to swiftly adjust to new tasks with altered chronological sequences.

One of the primary contributions of this work lies in its successful enhancement of the generalization abilities of the learned policies. As a metric, we compared the average rewards before/after fine-tuning as other works have done. Compared to domain randomization, our method can achieve a slightly higher reward (from $-45.01/-46.04$ to $-44.21/-43.78$ on the two testing tasks) before fine-tuning, but this increases to $-37.32/-38.18$, which is much higher than the $-52.22/-50.28$ of DR. These findings validate the efficacy and practical utility of our model, making a strong case for its necessity and utility in the field of multi-agent chronological planning.

Although our method has improved the adaptability of learned policies, it might face limitations in its ability to generalize to all types of chronological planning tasks. The performance might depend on the specific nature of the tasks and the underlying Markov decision process. In addition, our method requires significant computational resources to extract the meta-parameters of past experiences, which could limit its practical application in large-scale or resource-constrained settings. Therefore, future work will focus on the application of our proposed method to more challenging scenarios. Specifically, we aim to scrutinize its adaptability in tasks featuring higher degrees of disturbance, pushing the boundaries of its potential utility. We will also consider optimizing the computational efficiency of the method. The work we present here paves the way for more sophisticated implementations of our methodology, promising advancements in the sphere of multi-agent systems.

Author Contributions: Conceptualization, C.H. and K.X.; methodology, C.H. and Z.Z.; validation, Q.Y. and K.X.; formal analysis, Z.Z.; investigation, K.X.; data curation, L.Q.; writing—original draft preparation, C.H.; writing—review and editing, C.H. and K.X.; supervision, K.X.; funding acquisition, L.Q. and Q.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Natural Science Foundation of China [grant number 62103420].

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data supporting the reported results can be found at https://github.com/openai/multiagent-particle-envs (accessed on 20 April 2022).

Acknowledgments: We thank Qi Wang and Sihang Qiu at the National University of Defense Technology for attending discussions and giving us important advice.

Conflicts of Interest: The authors declare no conflict of interest.

References

41. Liu, T.; Ma, X.; Li, S.; Li, X.; Zhang, C. A stock price prediction method based on meta-learning and variational mode decomposition. Knowl.-Based Syst. 2022, 252, 109324. [CrossRef]