DARL1N: Distributed multi-Agent Reinforcement Learning with One-hop Neighbors

Baoqian Wang[1] Junfei Xie[2] Nikolay Atanasov[3]

***Abstract*— Multi-agent reinforcement learning (MARL) methods face a curse of dimensionality in the policy and value function representations as the number of agents increases. The development of distributed or parallel training techniques is also hindered by the global coupling among the agent dynamics, requiring simultaneous state transitions. This paper introduces Distributed multi-Agent Reinforcement Learning with One-hop Neighbors (DARL1N). DARL1N is an off-policy actor-critic MARL method that breaks the curse of dimensionality and achieves distributed training by restricting the agent interactions to one-hop neighborhoods. Each agent optimizes its value and policy functions over a one-hop neighborhood, reducing the representation complexity, yet maintaining expressiveness by training with varying numbers and states of neighbors. This structure enables the key contribution of DARL1N: a distributed training procedure in which each compute node simulates the state transitions of only a small subset of the agents, greatly accelerating the training of large-scale MARL policies. Comparisons with state-of-the-art MARL methods show that DARL1N significantly reduces training time without sacrificing policy quality as the number of agents increases.**

SUPPLEMENTARY MATERIAL

Software and demos supplementing this paper: `https://github.com/BaoqianWang/IROS22_DARL1N`

I. INTRODUCTION

Recent years have witnessed tremendous success of reinforcement learning (RL) in challenging decision making problems, such as robot control and video games. Research efforts are currently focused on multi-agent settings, such as cooperative robot navigation, multi-player games, and traffic management. A direct application of RL techniques in a multi-agent setting by simultaneously running a single-agent algorithm at each agent exhibits poor performance [1]. This is because, without considering interactions among the agents, the environment becomes non-stationary from the perspective of a single agent.

Multi-agent reinforcement learning (MARL) [2] addresses the aforementioned challenge by considering all agent dynamics collectively when learning the policy of an individual agent. This is achieved by learning a centralized value or action-value (Q) function that involves the states and actions of all agents. Most effective MARL algorithms, such as multi-agent deep deterministic policy gradient (MADDPG) [1], counterfactual multi-agent (COMA) [3], and multi actor attention critic (MAAC) [4], adopt this strategy. However, learning a joint Q function is challenging due to the exponentially growing size of the joint state and action space with the increasing number of agents [5]. A policy obtained by parametrizing the joint Q function directly has poor performance in large-scale settings as shown in [6], [7].

Recently, MARL algorithms that reduce the representation complexity of the Q function have been shown to significantly improve the quality of the learned policies for large-scale multi-agent settings. Successful methods include value factorization algorithms, such as mean-field MARL [6], evolutionary population curriculum (EPC) [7] and scalable actor critic (SAC) [5]. While these methods achieve excellent performance, their training time can be exceedingly slow as the number of agents increases because they require simultaneous state transitions for all agents. This requirement prevents fully distributed training over a computing cluster because the compute nodes need to receive the state transitions of all agents simultaneously.

Our **contribution** is a distributed MARL training method called Distributed multi-Agent Reinforcement Learning with One-hop Neighbors (DARL1N). Its main advantage over state-of-the-art methods is a fully distributed training procedure, in which each compute node only simulates a very small subset of the agents locally. This is made possible by representing the agent topology as a proximity graph and approximating the Q function over one-hop neighborhoods. When agent interactions are restricted to one-hop neighbors, training the Q function of an agent requires simulation only of the agent itself and its potential two-hop neighbors. This enables fully distributed training and greatly accelerates the training of large-scale MARL policies.

II. RELATED WORK

State-of-the-art MARL algorithms like MADDPG [1], COMA [3], MAAC [4] and MAPPO [8] learn joint Q/value functions over all agent states and actions. As the number of agents increases, the exponential growth of the joint state and action spaces increases the representation complexity required to model the Q functions drastically. Moreover, the need to perform simultaneous state transitions for all agents prevents distributed or parallel training.

Two directions have been explored in the literature to scale MARL up. The first direction aims to reduce the complexity

We gratefully acknowledge support from ARL DCIST CRA W911NF-17-2-0181, NSF CAREER-2048266 and CRI-1953048.

[1] Baoqian Wang is with the Department of Electrical and Computer Engineering, University of California San Diego and San Diego State University, La Jolla, CA, 92093 (e-mail: `bawang@ucsd.edu`).

[2] Junfei Xie is with the Department of Electrical and Computer Engineering, San Diego State University, San Diego, CA, 92182 (e-mail: `jxie4@sdsu.edu`).

[3] Nikolay Atanasov is with the Department of Electrical and Computer Engineering, University of California San Diego, La Jolla, CA, 92093 (e-mail: `natanasov@ucsd.edu`).

of the Q function representation by factorizing it using local value functions that only depend on the states and actions of some agents. The second direction considers distributed or parallel computing architectures to speed MARL training up.

We first review factorization techniques for the Q and policy functions in MARL. VDN [9] proposes a decomposition of the joint Q function into a sum of local value functions. QMIX [10] improves VDN by combining the local value functions monotonically using a mixing neural network, which provides a more expressive Q function. QTRAN [11] further extends VDN and QMIX by factorizing an alternative Q function having equivalent optimal actions with the original Q function without requiring additivity and monotonicity assumptions. The Q function can also be factorized according to a coordination graph specifying the agent interactions. For instance, [12]–[15] decomposed the global Q function into a set of local value functions with dependencies specified by agents that are connected in a static undirected graph. Dynamic coordination graphs for cooperative MARL were considered in [16]. To approximate the local value functions, Böhmer et al. [17] used deep neural networks. Mean-Field MARL techniques [6], including Mean-Field Actor Critic (MFAC) and Mean-Field Q (MF-Q), factorize the Q function into a weighted sum of local Q functions, each depending only on one agent's action and a mean value of the actions of its neighboring agents. SAC [5] approximates the Q function of each agent using the states and actions of its κ-hop neighbors. The Q and policy factorization of SAC with $\kappa = 1$ is adopted in DARL1N. The main difference with respect to SAC is that DARL1N also uses the graph structure to devise a distributed training approach. While training SAC requires simultaneous state transitions, we show that the Q function of an agent can be trained off-policy using state transitions only for the agent itself and its potential two-hop neighbors. The relationship between SAC and DARL1N will be discussed in more detail in Sec.VI-A after introducing DARL1N.

While value factorization techniques reduce the representation complexity of the MARL policy and value functions, simultaneous simulation and parameter updates for all agents are required in a single compute node. To speed up training for MARL, distributed/parallel computing is a promising technique, which has been rarely studied for MARL. Only a few distributed or parallel approaches have been proposed to accelerate MARL training. Multi-agent A3C was explored in [18], where each parallelly running compute node performs independent simulation and training for all agents, and asynchronously updates and fetches the parameters stored in the central controller. EPC [7] applies curriculum learning and adopts population invariant policy and Q functions to support varying numbers of agents in different learning stages. EPC was implemented in a parallel computing architecture that consists of multiple compute nodes. Each compute node simulates all agents in multiple independent environments, and each environment and agent runs in an independent and parallel process. Also of interest is the distributed architecture presented in [19], which was designed to reduce the communication overhead between compute nodes and the central controller. Although these methods can improve training efficiency by leveraging distributed or parallel computing, their training costs are still high when the number of agents is large, as they require each compute node to simulate all agents.

III. Background

We consider the MARL problem in which M agents learn to optimize their behaviors by interacting with the environment. Denote the state and action of agent $i \in [M] := \{1, \dots, M\}$ by $s_i \in \mathcal{S}_i$ and $a_i \in \mathcal{A}_i$, respectively, where $\mathcal{S}_i$ and $\mathcal{A}_i$ are the corresponding state and action spaces. Let $\mathbf{s} := (s_1, \dots, s_M) \in \mathcal{S} := \prod_{i\in[M]} \mathcal{S}_i$ and $\mathbf{a} := (a_1, \dots, a_M) \in \mathcal{A} := \prod_{i\in[M]} \mathcal{A}_i$ denote the joint state and action of all agents. At time t, a joint action $\mathbf{a}(t)$ applied at state $\mathbf{s}(t)$ triggers a transition to a new state $\mathbf{s}(t+1) \in \mathcal{S}$ according to a conditional probability density function (pdf) $p(\mathbf{s}(t+1)|\mathbf{s}(t), \mathbf{a}(t))$. After each transition, each agent i receives a reward $r_i(\mathbf{s}(t), \mathbf{a}(t))$, determined by the joint state and action according to the function $r_i : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. The objective of each agent i is to choose a deterministic policy $\mu_i : \mathcal{S} \to \mathcal{A}_i$ to maximize the expected cumulative discounted reward:

$$V_i^{\boldsymbol{\mu}}(\mathbf{s}) := \mathbb{E}_{\substack{\mathbf{a}(t)=\boldsymbol{\mu}(\mathbf{s}(t)) \\ \mathbf{s}(t)\sim p}} \left[\sum_{t=0}^{\infty} \gamma^t r_i(\mathbf{s}(t), \mathbf{a}(t)) \;\middle|\; \mathbf{s}(0) = \mathbf{s} \right],$$

where $\boldsymbol{\mu} := (\mu_1, \dots, \mu_M)$ denotes the joint policy of all agents and $\gamma \in (0, 1)$ is a discount factor. The function $V_i^{\boldsymbol{\mu}}(\mathbf{s})$ is known as the *value function* of agent i associated with joint policy $\boldsymbol{\mu}$.

An optimal policy μ_i^* for agent i can also be obtained by maximizing the *action-value (Q) function*:

$$Q_i^{\boldsymbol{\mu}}(\mathbf{s}, \mathbf{a}) := \mathbb{E}_{\substack{\mathbf{a}(t)=\boldsymbol{\mu}(\mathbf{s}(t)) \\ \mathbf{s}(t)\sim p}} \left[\sum_{t=0}^{\infty} \gamma^t r_i(\mathbf{s}(t), \mathbf{a}(t)) \;\middle|\; \mathbf{s}(0) = \mathbf{s}, \mathbf{a}(0) = \mathbf{a} \right]$$

and setting $\mu_i^*(\mathbf{s}) \in \arg\max_{a_i} \max_{\mathbf{a}_{-i}} Q_i^*(\mathbf{s}, \mathbf{a})$, where $Q_i^*(\mathbf{s}, \mathbf{a}) := \max_{\boldsymbol{\mu}} Q_i^{\boldsymbol{\mu}}(\mathbf{s}, \mathbf{a})$ and $\mathbf{a}_{-i}$ denotes the actions of all agents except i. In the rest of the paper, we omit the time notation t for simplicity, when there is no risk of confusion.

IV. Problem Statement

To develop a distributed MARL algorithm, we impose additional structure on the MARL problem. Assume that all agents share a common state space, i.e., $\mathcal{S}_i = \mathcal{S}_j, \forall i, j \in [M]$. Let $dist : \mathcal{S}_i \times \mathcal{S}_i \to \mathbb{R}$ be a distance metric on the homogeneous state space. We introduce a proximity graph [20] to model the topology of the agent team. A d-disk proximity graph is defined as a mapping that associates the joint state $\mathbf{s} \in \mathcal{S}$ with an undirected graph $(\mathcal{V}, \mathcal{E})$ such that $\mathcal{V} = \{s_1, s_2, \dots, s_M\}$ and $\mathcal{E} = \{(s_i, s_j) | dist(s_i, s_j) \leq d, i \neq j\}$. Define the set of *one-hop neighbors* of agent i as $\mathcal{N}_i := \{j | (s_i, s_j) \in \mathcal{E}\} \cup \{i\}$. We make the following regularity assumption about the agents' motion.

Assumption 1. *The distance between two consecutive states, $s_i(t)$ and $s_i(t+1)$, of agent i is bounded, i.e., $dist(s_i(t), s_i(t+1)) \leq \epsilon$, for some $\epsilon > 0$.*

This assumption is justified in many problems of interest where, e.g., due to physical limitations, the agent states can only change by a bounded amount in a single time step. Following this assumption, we define the set of *potential neighbors* of agent i at time t as $\mathcal{P}_i(t) := \{j | dist(s_j(t), s_i(t)) \leq 2\epsilon + d\}$, which captures the set of agents that may become one-hop neighbors of agent i at time $t+1$.

Denote the joint state and action of the one-hop neighbors of agent i by $\mathbf{s}_{\mathcal{N}_i} = (s_{j_1}, \ldots, s_{j_{|\mathcal{N}_i|}})$ and $\mathbf{a}_{\mathcal{N}_i} = (a_{j_1}, \ldots, a_{j_{|\mathcal{N}_i|}})$, respectively, where $j_1, \ldots, j_{|\mathcal{N}_i|} \in \mathcal{N}_i$. Our key idea is to let agent i's policy, $a_i = \mu_i(\mathbf{s}_{\mathcal{N}_i})$, only depend on the one-hop neighbor states $\mathbf{s}_{\mathcal{N}_i}$. The intuition is that agents that are far away from agent i at time t have little impact on its current action $a_i(t)$. To support this policy model, we make two additional assumptions on the problem structure. To emphasize that the output of a function $f : \prod_{i\in[M]} \mathcal{S}_i \mapsto \mathbb{R}$ is affected only by a subset $\mathcal{N} \subseteq [M]$ of the input dimensions, we use the notation $f(\mathbf{s}) = f(\mathbf{s}_{\mathcal{N}})$ for $\mathbf{s} \in \mathcal{S}$ and $\mathbf{s}_{\mathcal{N}} \in \prod_{i\in\mathcal{N}} \mathcal{S}_i$.

Assumption 2. *The reward of agent i can be fully specified using its one-hop neighbor states $\mathbf{s}_{\mathcal{N}_i}$ and actions $\mathbf{a}_{\mathcal{N}_i}$, i.e., $r_i(\mathbf{s}, \mathbf{a}) = r_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$ and its absolute value is upper bounded by $|r_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})| \leq \bar{r}$, for some $\bar{r} > 0$.*

Assumption 2 is satisfied in many multi-agent problems where the reward of one agent is determined only by the states and actions of nearby agents. Examples are provided in Sec. VI. Similar assumptions are adopted in [5], [21], [22].

Assumption 3. *The transition model of agent i depends only on its action a_i and one-hop neighbor states $\mathbf{s}_{\mathcal{N}_i}$, i.e., $p_i\left(s_i(t+1) \mid \mathbf{s}(t), a_i(t)\right) = p_i\left(s_i(t+1) \mid \mathbf{s}_{\mathcal{N}_i}(t), a_i(t)\right)$.*

Assumption 3 is common for multi-agent networked systems as in [5], [22]. As a result, the joint state transition pdf decomposes as:

$$p(\mathbf{s}(t+1) \mid \mathbf{s}(t), \mathbf{a}(t)) = \prod_{i=1}^{M} p_i\left(s_i(t+1) \mid \mathbf{s}_{\mathcal{N}_i}(t), a_i(t)\right).$$

The objective of each agent i is to obtain an optimal policy μ_i^* by solving the following problem:

$$\mu_i^*(\mathbf{s}_{\mathcal{N}_i}) = \arg\max_{a_i} \max_{\mathbf{a}_{-i}} Q_i^*(\mathbf{s}, \mathbf{a}),$$

where $Q_i^*(\mathbf{s}, \mathbf{a}) := \max_{\boldsymbol{\mu}} Q_i^{\boldsymbol{\mu}}(\mathbf{s}, \mathbf{a})$ is the optimal action-value (Q) function introduced in the previous section.

V. Distributed MARL with One-hop Neighbors

In this section, we develop the DARL1N algorithm to solve the MARL problem with proximity-graph structure. DARL1N limits the interactions among agents within one-hop neighbors to significantly reduce the learning complexity. To further speed training up, DARL1N adopts a distributed training framework that exploits local interactions to decompose and distribute the computation load.

A. One-hop Neighborhood Value and Policy Factorization

The Q function of each agent associated with the MARL problem over the proximity graph can be written as

$$Q_i^{\boldsymbol{\mu}}(\mathbf{s}, \mathbf{a}) = Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-})$$

where $\mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}$ denote joint states and actions of agents except one-hop neighbors of agent i. Inspired by the SAC algorithm [5], we approximate the Q function as $\tilde{Q}_i^{\boldsymbol{\mu}}$ that depends only on one-hop neighbor states and actions:

$$\tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}) = \sum_{\mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}} w_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-}) Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-})$$

where the weights $w_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-}) > 0$ satisfy $\sum_{\mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}} w_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-}) = 1$. The approximation error is given in the following lemma with proof provided in Appendix A.

Lemma 1. *Under Assumptions 2 and 3, the approximation error between $\tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$ and $Q_i^{\boldsymbol{\mu}}(\mathbf{s}, \mathbf{a})$ is bounded by:*

$$|\tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}) - Q_i^{\boldsymbol{\mu}}(\mathbf{s}, \mathbf{a})| \leq \frac{2\bar{r}\gamma}{1-\gamma}.$$

We then parameterize the approximated Q function $\tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$ and the policy $\mu_i(\mathbf{s}_{\mathcal{N}_i})$ by θ_i and ϕ_i, respectively. To handle the varying sizes of $\mathbf{s}_{\mathcal{N}_i}$ and $\mathbf{a}_{\mathcal{N}_i}$, in the implementation, we let the input dimension of $\tilde{Q}_i^{\boldsymbol{\mu}}$ to be the largest possible dimension of $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$, and apply zero-padding for agents that are not in the one-hop neighborhood of agent i. The same procedure is applied to represent $\mu_i(\mathbf{s}_{\mathcal{N}_i})$. More implementation details are provided in Appendix C.b.

To learn the approximated Q function $\tilde{Q}_i^{\boldsymbol{\mu}}$, instead of incremental on-policy updates to the Q function as in SAC, we adopt off-policy temporal-difference learning with a buffer similar to MADDPG. The parameters θ_i of the approximated Q function are updated by minimizing:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}_j'}\}_{\forall j \in \mathcal{N}_i'}) \sim \mathcal{D}_i} \left[\left(\tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}) - y \right)^2 \right]$$
$$y = r_i + \gamma \hat{Q}_i^{\hat{\boldsymbol{\mu}}}(\mathbf{s}_{\mathcal{N}_i'}, \mathbf{a}_{\mathcal{N}_i'}) \quad (1)$$

where $\mathcal{D}_i$ is the replay buffer for agent i that contains information only from $\mathcal{N}_i, \mathcal{N}_i'$, the one-hop neighbors of agent i at the current and next time step, and the one-hop neighbors $\mathcal{N}_j'$ for $j \in \mathcal{N}_i'$. To stabilize the training, a target Q function $\hat{Q}_i^{\hat{\boldsymbol{\mu}}}$ with parameters $\hat{\theta}_i$ and a target policy function $\hat{\mu}_i$ with parameters $\hat{\phi}_i$ are used. The parameters $\hat{\theta}_i$ and $\hat{\phi}_i$ are updated using Polyak averaging, $\hat{\theta}_i = \tau\hat{\theta}_i + (1-\tau)\theta_i$ and $\hat{\phi}_i = \tau\hat{\phi}_i + (1-\tau)\phi_i$, where τ is a hyperparameter. In contrast to MADDPG, the replay buffer $\mathcal{D}_i$ for agent i only needs to store its local interactions $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}_j'}\}_{\forall j \in \mathcal{N}_i'})$ with nearby agents. Note that $\{\mathbf{s}_{\mathcal{N}_j'}\}_{\forall j \in \mathcal{N}_i'}$ is used to calculate $\mathbf{a}_{\mathcal{N}_i'}$. Also, in contrast to SAC, each agent i only needs to collect its own training data by simulating local two-hop interactions.

This allows an efficient and distributed training framework as we explain in the next subsection.

Agent i's policy parameters ϕ_i are updated using gradients from the policy gradient theorem [23]:

$$G(\phi_i) = \mathbb{E}_{\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i} \sim \mathcal{D}_i} \left[\nabla_{\phi_i} \mu_i \left(\mathbf{s}_{\mathcal{N}_i} \right) \nabla_{a_i} \tilde{Q}_i^{\boldsymbol{\mu}} \left(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i} \right) \right], \tag{2}$$

where again data $\mathcal{D}_i$ only from local interactions is needed.

B. Distributed Training with Local Interactions

To implement the parameter updates proposed above, agent i needs training data $\mathcal{D}_i = (\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}_j'}\}_{j \in \mathcal{N}_i'})$ from its one-hop neighbors at the current and next time steps, whose dynamics obey the following proposition (see proof in Appendix B.).

Proposition 1. *Under Assumption 1, if an agent j is not a potential neighbor of agent i at time t, i.e., $j \notin \mathcal{P}_i(t)$, it will not be a one-hop neighbor of agent i at time $t+1$, i.e., $j \notin \mathcal{N}_i(t+1)$.*

Proposition 1 allows us to decouple the global interactions among agents and limit message exchanges or observations to be among one-hop neighbors. It also allows parallel training on a distributed computing architecture, where each compute node only needs to simulate a small subset of the agents. This leads to significant training efficiency gains as demonstrated in Sec. VI.

To collect training data, at each time step, agent i first interacts with its one-hop neighbors to obtain their states $\mathbf{s}_{\mathcal{N}_i}$ and actions $\mathbf{a}_{\mathcal{N}_i}$ and compute its reward $r_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$. To obtain $\mathbf{s}_{\mathcal{N}_j'}$ for all $j \in \mathcal{N}_i'$, we first determine agent i's one-hop neighbors at the next time step, $\mathcal{N}_i'$. Using Proposition 1, we let each potential neighbor $k \in \mathcal{P}_i$ perform a transition to a new state $s_k' \sim p_k(\cdot|\mathbf{s}_{\mathcal{N}_k}, a_k)$, which is sufficient to determine $\mathcal{N}_i'$. Then, we let the potential neighbors $\mathcal{P}_j$ of each new neighbor $j \in \mathcal{N}_i'$ perform transitions to determine $\mathcal{N}_j'$ and obtain $\mathbf{s}_{\mathcal{N}_j'}$. Fig. 1(a) illustrates the data collection process. At time t, agent i obtains $\mathbf{s}_{\mathcal{N}_i}$, $\mathbf{a}_{\mathcal{N}_i}$, and $r_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$ for $\mathcal{N}_i = \{i, 1\}$. Then, the potential neighbors of agent i, $\mathcal{P}_i = \{1, 2, i\}$, proceed to their next states at time $t+1$. This is sufficient to determine that $\mathcal{N}_i' = \{i, 2\}$ and obtain $\mathbf{s}_{\mathcal{N}_i'}$. Finally, we let agent 3, which belongs to set $\mathcal{P}_2 = \{i, 1, 2, 3\}$, perform a transition to determine that $\mathcal{N}_2' = \{i, 2, 3\}$ and obtain $\mathbf{s}_{\mathcal{N}_2'}$.

We now describe a distributed learning framework that exploits the local interactions among the agents to optimize the policy and Q function parameters. Our training framework consists of a central controller and M learners, each training a different agent. The central controller stores a copy of all policy and target policy parameters, ϕ_m, $\hat{\phi}_m$, $\forall m \in [M]$. In each training iteration, the central controller broadcasts the parameters to all learners. Each learner i updates its own policy parameters ϕ_i, $\hat{\phi}_i$ and returns the updated values to the central controller.

Each learner i maintains the parameters θ_i and $\hat{\theta}_i$ of agent i's approximated Q and target Q function. In each training iteration, learner i uses policies with parameters $\{\phi_m, \hat{\phi}_m\}_{m \in [M]}$ received from the central controller. Transitions are simulated only for agent i, its potential neighbors $\mathcal{P}_i$, and the potential neighbors $\mathcal{P}_j$ of each new neighbor $j \in \mathcal{N}_i'$ as described above. DARL1N achieves significant computation savings because (i) the Q function parameters θ_i and $\hat{\theta}_i$ are stored locally at each learner i and do not need to be communicated, and (ii) the agent transition simulation occurs only over small groups of agents, distributed among the M learners instead of centralized over all agents at the central controller. The interaction data $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}_j'}\}_{j \in \mathcal{N}_i'})$ are stored in the replay buffer $\mathcal{D}_i$. Finally, learner i updates θ_i and ϕ_i using (1) and (2), respectively, and the target network parameters $\hat{\theta}_i$ and $\hat{\phi}_i$ via Polyak averaging. Fig. 1(b) illustrates the distributed training procedure. The pseudocode of DARL1N is provided in Alg. 1.

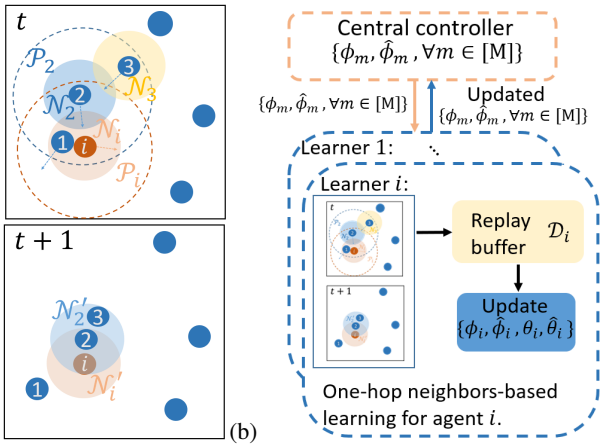

Fig. 1: (a) One-hop neighbor transitions from one time step to the next in a d-disk proximity graph; (b) Distributed MARL with local interactions.

VI. Experiments

In this section, we conduct experiments to evaluate the performance of DARL1N.

A. Experiment Settings

a) Environments: We evaluate DARL1N in four environments, Ising Model [6], Food Collection, Grassland, and Adversarial Battle [7], which cover cooperative and mixed cooperative competitive games.

b) Benchmarks: We compare DARL1N with three state-of-the-art MARL algorithms: MADDPG [1], MFAC [6], and EPC [7]. All benchmark methods need states of all agents through observation or communication during training and execution. In contrast, DARL1N only needs states of one-hop neighbors during execution and two-hop potential neighbors during training. While the most closely related method to DARL1N is SAC [5], we do not compare to SAC because DARL1N is a distributed training version of SAC with the same Q function factorization. DARL1N utilizes off-policy training and allows each compute node to simulate state transitions only for one agent and its potential two-hop neighbors. In contrast, SAC is an on-policy approach running

Algorithm 1: DARL1N: Distributed multi-Agent Reinforcement Learning with One-hop Neighbors

// **Central controller:**

Initialize policy, target policy parameters $\boldsymbol{\phi} = \{\phi_m, \hat{\phi}_m\}_{m\in[M]}$.

Broadcast $\boldsymbol{\phi}$ to the learners.

do

  Listen to channel and collect updated $\boldsymbol{\phi}$ from learners.

while *updated* $\boldsymbol{\phi}$ *is not received*;

// **Learner** i**:**

Initialize parameters θ_i, $\hat{\theta}_i$ of $\tilde{Q}_i^{\boldsymbol{\mu}}$, $\hat{Q}_i^{\hat{\boldsymbol{\mu}}}$ and replay buffer $\mathcal{D}_i$.

for $iter = 1 : max\_iteration$ **do**

  Listen to channel.

  if $\boldsymbol{\phi}$ *received from the central controller* **then**

    // **Local interactions:**

    for $step = 1 : max\_transition\_number$ **do**

      Randomly initialize all agent states.

      Simulate one-step transitions for the potential neighbors $\mathcal{P}_i$ of agent i to determine $\mathcal{N}_i'$.

      Simulate one-step transitions for the potential neighbors $\mathcal{P}_j$ of each agent $j \in \mathcal{N}_i'$ to get $\mathbf{s}_{\mathcal{N}_j'}$.

      Store the obtained local interactions $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}_j'}\}_{j\in\mathcal{N}_i'})$ in the buffer $\mathcal{D}_i$.

    // **One-hop neighbor-based learning:**

    Sample a mini-batch from $\mathcal{D}_i$ and update ϕ_i, θ_i $\hat{\phi}_i, \hat{\theta}_i$.

  Send updated $\phi_i, \hat{\phi}_i$ to the central controller.

in a single compute node without parallel processing but with the same Q function factorization.

c) Evaluation Metrics: We evaluate all methods using two criteria: *training efficiency* and *policy quality*. To measure the training efficiency, we use two metrics: 1) *average training time* spent to run a specified number of training iterations and 2) *convergence time*. The convergence time is defined as the time when the variance of the average total training reward over 90 consecutive iterations does not exceed 2% of the absolute mean reward, where the average total training reward is the total reward of all agents averaged over 10 episodes in three training runs with different random seeds. To measure policy quality, we use *convergence reward*, which is the average total training reward at the convergence time.

d) Experiment Configurations: We run our experiments on Amazon EC2 computing clusters [24]. To understand the scalability of each method, for each environment, we consider four scenarios with increasing number of agents. The number of agents in the Ising Model and Food Collection environments are set to $M = 9, 16, 25, 64$ and $M = 3, 6, 12, 24$, respectively. In the Grassland and Adversarial Battle environments, the number of agents are set to $M = 6, 12, 24, 48$. In the experiments, the number of adversary agents, grass pellets and resource units are all set to $\frac{M}{2}$, and all adversary agents adopt policies trained by MADDPG. More experiment settings including training parameters, Q function and policy function representation, and neighborhood configurations are described in Appendix C.

To evaluate the training efficiency, we configure the computing resources used to train each method in a way so that DARL1N utilizes roughly the same or fewer resources measured by the money spent per hour on Amazon EC2 computing cluster. In particular, to train DARL1N, Amazon EC2 instance $c5n.large$ is used in all scenarios for all environments. To train MADDPG and MFAC, instance $z1d.3xlarge$ is used in the first scenario ($M = 9$) for Ising Model and in the first two scenarios for Food Collection, Grassland and Adversarial Battle. In the other scenarios, instance $z1d.6xlarge$ is used. To train EPC, we use instance $c5.12xlarge$ in all scenarios for Food Collection and in the first three scenarios for Ising Model, Grassland and Adversarial Battle. The other scenarios adopt instance $c5.18xlarge$. To configure the parallel computing architecture in EPC, we set the number of parallel computing instances and the number of independent environments to 3 and 25, respectively. The configurations of Amazon EC2 instances as the compute nodes are summarized in Tab. I.

TABLE I: Configurations of Amazon EC2 instances

Instances	CPU cores	CPU frequency	Memory	Network	Hourly price
$c5n.large$	2	3.4 GHz	5.3 GB	$\leq$ 25 Gb	$ 0.108
$z1d.3xlarge$	12	4 GHz	96 GB	$\leq$ 10 Gb	$ 1.116
$z1d.6xlarge$	24	4 GHz	192 GB	$\leq$ 10 Gb	$ 2.232
$c5.12xlarge$	48	3.6 GHz	96 GB	12 Gb	$ 2.04
$c5.18xlarge$	72	3.6 GHz	144 GB	25 Gb	$ 3.06

TABLE II: Convergence time and convergence reward of different methods in the Ising Model environment.

Method	Convergence Time (s)				Convergence Reward			
	$M = 9$	16	25	64	9	16	25	64
MADDPG	62	263	810	1996	460	819	**1280**	1831
MFAC	63	274	851	2003	**468**	814	1276	1751
EPC	101	**26**	**51**	**62**	**468**	**831**	1278	**3321**
EPC Scratch	101	412	993	2995	**468**	826	1275	2503
DARL1N	**38**	102	210	110	465	828	1279	2282

B. Experiment Results

a) Ising Model: Tab. II shows the convergence reward and convergence time of different methods. When the number of agents is small ($M = 9$), all methods achieve roughly the same reward. DARL1N takes the least amount of time to converge while EPC takes the longest time. When the number of agents increases, it can be observed that the EPC converges immediately and the convergence reward it achieves when $M = 64$ is much higher than the other methods. The reason is that, in the Ising Model, each agent only needs information of its four fixed neighbors, and hence in EPC the policy obtained from the previous stage can be applied to the current stage. The other methods train the agents from scratch without curriculum learning. For illustration, we also show the convergence reward and convergence time achieved by training EPC from scratch without curriculum learning (denoted as EPC Scratch in Tab. II). The results show that EPC Scratch converges much slower than EPC as the number of agents increases. Note that when the number of agents is 9, EPC and EPC Scratch are the same. Moreover, DARL1N achieves a reward comparable with that of EPC Scratch but converges much faster. Fig. 2(a) shows the average time taken to train each method for 10 iterations in different scenarios.

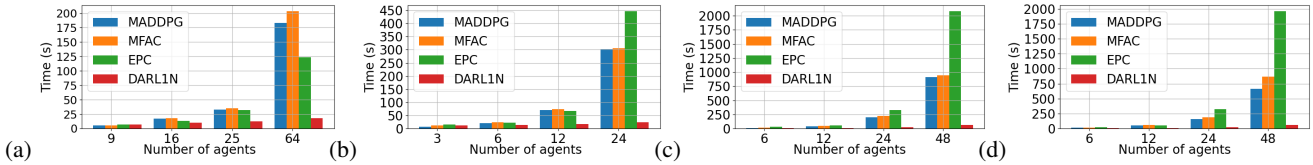

Fig. 2: Average training time of different methods to run (a) 10 iterations in the Ising Model, (b) 30 iterations in the Food Collection, (c) 30 iterations in the Grassland, and (d) 30 iterations in the Adversarial Battle environments.

TABLE III: Convergence time and convergence reward of different methods in the Food Collection environment.

Method	Convergence Time (s)				Convergence Reward			
	$M=3$	6	12	24	3	6	12	24
MADDPG	**501**	1102	4883	2005	24	24	-112	-364
MFAC	512	832	4924	2013	20	23	-115	-362
EPC	1314	723	2900	8104	**31**	**34**	-16	-87
DARL1N	502	**382**	**310**	**1830**	14	25	**13**	**-61**

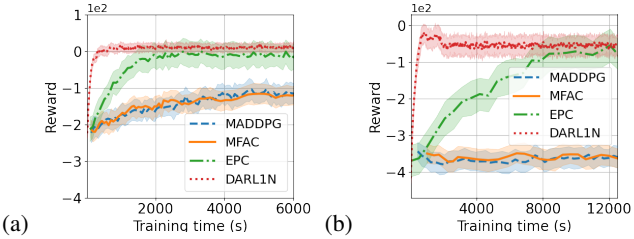

Fig. 3: Average total training reward of different methods in the Food Collection environment when there are (a) $M = 12$, (b) $M = 24$ agents.

DARL1N requires much less time to perform a training iteration than the benchmark methods.

b) Food Collection: The convergence rewards and convergence times in this environment are shown in Tab. III. The results show that, when the problem scale is small, DARL1N, MADDPG and MFAC achieve similar performance in terms of policy quality. As the problem scale increases, the performance of MADDPG and MFAC degrades significantly and becomes much worse than DARL1N or EPC when $M = 12$ and $M = 24$, which is also shown in Figs. 3(a)-3(b). The convergence reward achieved by DARL1N is comparable or sometimes higher than that achieved by EPC. Moreover, the convergence speed of DARL1N is the highest among all methods in all scenarios.

Fig. 2(b) shows the average training time for running 30 iterations. Similar as the results obtained in the Ising Model, DARL1N achieves the highest training efficiency and its training time grows linearly as the number of agents increases. When $M = 24$, EPC takes the longest training time. This is because of the complex policy and Q neural network architectures in EPC, the input dimensions of which grow linearly and quadratically, respectively, with more agents.

c) Grassland: Similar as the results in the Food Collection environment, the policy generated by DARL1N is equally good or even better than those generated by the benchmark methods, as shown in Tab. IV and Fig. 2(c), especially when the problem scale is large. DARL1N also has the fastest convergence speed and takes the shortest time to run a training iteration.

TABLE IV: Convergence time and convergence reward of different methods in the Grassland environment.

Method	Convergence Time (s)				Convergence Reward			
	$M=6$	12	24	48	6	12	24	48
MADDPG	423	6271	2827	1121	21	11	-302	-612
MFAC	431	7124	3156	**1025**	**23**	9	-311	-608
EPC	4883	2006	3324	15221	12	38	105	205
DARL1N	**103**	**402**	**1752**	5221	18	**46**	**113**	**210**

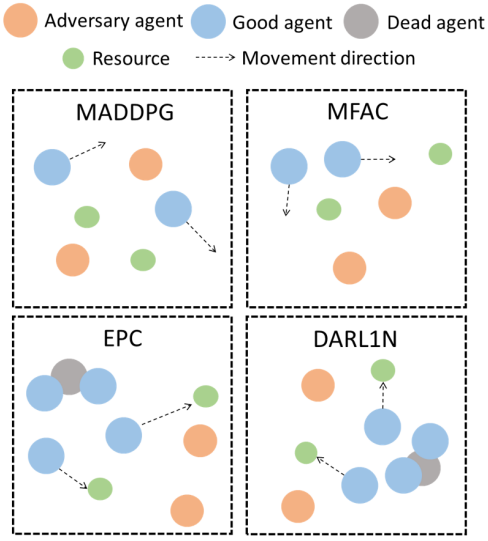

Fig. 4: States of a subset of agents during an episode in Adversarial Battle with agents trained by different methods when there are $M = 48$ agents.

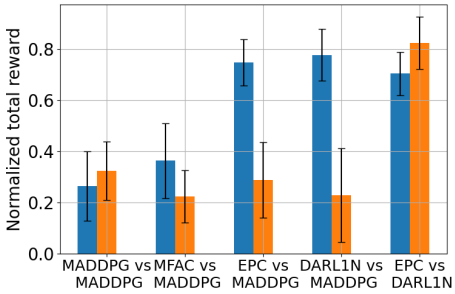

Fig. 5: Mean and standard deviation of normalized total reward of competing agents trained by different methods in the Adversarial Battle environment with $M = 48$.

d) Adversarial Battle: In this environment, DARL1N again achieves good performance in terms of policy quality and training efficiency compared to the baseline methods, as shown in Tab. V and Fig. 2(d). For illustration, the states of a subset of agents trained by different methods during an episode are shown in Fig. 4. It can be observed that both DARL1N and EPC agents can successfully collect resource

TABLE V: Convergence time and convergence reward of different methods in the Adversarial Battle environment.

Method	Convergence Time (s)				Convergence Reward			
	$M = 6$	12	24	48	6	12	24	48
MADDPG	452	1331	1521	7600	-72	-211	-725	-1321
MFAC	463	1721	1624	6234	-73	-221	-694	-1201
EPC	1512	1432	2041	9210	-75	-215	**-405**	**-642**
DARL1N	**121**	**756**	**1123**	**3110**	**-71**	**-212**	-410	-682

units and kill agents from the other team, but MADDPG and MFAC fail to do so. To further evaluate the performance, we reconsider the last scenario ($M = 48$) and train the good agents and adversary agents using two different methods. The trained good agents and adversarial agents are then compete with each other in the environment. We then apply the Min-Max normalization to measure the normalized total reward of agents at each side achieved in an episode. To reduce uncertainty, we generate 10 episodes and record the mean values and standard deviations. As shown in Fig. 5, DARL1N achieves the best performance, and both DARL1N and EPC significantly outperform MADDPG and MFAC.

VII. Conclusion

This paper introduces DARL1N, a scalable MARL algorithm. DARL1N features a novel training scheme that breaks the curse of dimensionality for action value function approximation by restricting the interactions among agents within one-hop neighborhoods. This reduces the learning complexity and enables fully distributed and parallel training, in which individual compute nodes only simulate interactions among a small subset of agents. To demonstrate the scalability and training efficiency of DARL1N, we conducted comprehensive evaluation in comparison with three state-of-the-art MARL algorithms, MADDPG, MFAC, and EPC. The results show that DARL1N generates equally good or even better policies in almost all scenarios with significantly higher training efficiency than benchmark methods, especially in large-scale problem settings.

References

[1] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, CA, USA, December 2017.

[2] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," *Innovations in Multi-agent Systems and Applications*, pp. 183–221, 2010.

[3] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *AAAI Conference on Artificial Intelligence*, Louisiana, USA, February 2018.

[4] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *International Conference on Machine Learning*, CA, USA, June 2019.

[5] G. Qu, A. Wierman, and N. Li, "Scalable reinforcement learning of localized policies for multi-agent networked systems," in *Learning for Dynamics and Control*, Online, June 2020.

[6] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," in *International Conference on Machine Learning*, June 2018.

[7] Q. Long, Z. Zhou, A. Gupta, F. Fang, Y. Wu, and X. Wang, "Evolutionary population curriculum for scaling multi-agent reinforcement learning," in *International Conference on Learning Representations (ICLR)*, Online, April 2020.

[8] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative, multi-agent games," *arXiv preprint arXiv:2103.01955*, 2021.

[9] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *International Conference on Autonomous Agents and Multi-Agent Systems*, Stockholm, Sweden, July 2018.

[10] 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*, Stockholm, Sweden, July 2018.

[11] 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 *Proceedings of 36th the International Conference on Machine Learning*, CA, USA, June 2019.

[12] L. Kuyer, S. Whiteson, B. Bakker, and N. Vlassis, "Multiagent reinforcement learning for urban traffic control using coordination graphs," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Antwerp, Belgium, September 2008.

[13] C. Guestrin, M. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *International Conference on Machine Learning*, Sydney, Australia, July 2002.

[14] J. R. Kok and N. Vlassis, "Collaborative multiagent reinforcement learning by payoff propagation," *Journal of Machine Learning Research*, vol. 7, pp. 1789–1828, 2006.

[15] C. Zhang and V. Lesser, "Coordinating multi-agent reinforcement learning with limited communication," in *International Conference on Autonomous Agents and Multi-Agent Systems*, MN, USA, May 2013.

[16] T. Wang, L. Zeng, W. Dong, Q. Yang, Y. Yu, and C. Zhang, "Context-aware sparse deep coordination graphs," *arXiv preprint arXiv:2106.02886*, 2021.

[17] W. Böhmer, V. Kurin, and S. Whiteson, "Deep coordination graphs," in *International Conference on Machine Learning*, Online, April 2020.

[18] D. Simões, N. Lau, and L. P. Reis, "Multi-agent actor centralized-critic with communication," *Neurocomputing*, 2020.

[19] T. Chen, K. Zhang, G. B. Giannakis, and T. Başar, "Communication-efficient policy gradient methods for distributed reinforcement learning," *IEEE Transactions on Control of Network Systems*, 2021.

[20] F. Bullo, J. Cortés, and S. Martinez, *Distributed control of robotic networks: a mathematical approach to motion coordination algorithms*. Princeton University Press, 2009.

[21] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1086–1095, 2019.

[22] Y. Lin, G. Qu, L. Huang, and A. Wierman, "Distributed reinforcement learning in multi-agent networked systems," *arXiv preprint arXiv:2006.06555*, 2020.

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

[24] AWS, "Amazon ec2," https://aws.amazon.com/ec2/, 2022, accessed: 2022-01-13.

Appendix

A. Proof of Lemma 1

We first prove the following inequality

$$|Q_i^{\boldsymbol{\mu}}(s_{\mathcal{N}_i}, s_{\mathcal{N}_i^-}, a_{\mathcal{N}_i}, a_{\mathcal{N}_i^-}) - Q_i^{\boldsymbol{\mu}}(s_{\mathcal{N}_i}, \hat{s}_{\mathcal{N}_i^-}, a_{\mathcal{N}_i}, \hat{a}_{\mathcal{N}_i^-})| \leq \frac{2\bar{r}\gamma}{1-\gamma} \tag{3}$$

where $\hat{\mathbf{s}}_{\mathcal{N}_i^-} \neq \mathbf{s}_{\mathcal{N}_i^-}$ and $\hat{\mathbf{a}}_{\mathcal{N}_i^-} \neq \mathbf{a}_{\mathcal{N}_i^-}$. Particularly, letting $(\mathbf{s}, \mathbf{a})$ and $(\hat{\mathbf{s}}, \hat{\mathbf{a}})$ denote $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-})$ and

$(\mathbf{s}_{\mathcal{N}_i}, \hat{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \hat{a}_{\mathcal{N}_i^-})$, respectively, we have:

$$
\begin{aligned}
&|Q_i^{\boldsymbol{\mu}}(\mathbf{s}, \mathbf{a}) - Q_i^{\boldsymbol{\mu}}(\hat{\mathbf{s}}, \hat{\mathbf{a}})| \\
&= |\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\mathbf{s}, \mathbf{a})] \\
&- \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\hat{\mathbf{s}}, \hat{\mathbf{a}})]| \\
&\leq \sum_{t=0}^{\infty} |\mathbb{E}\left[\gamma^t r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\mathbf{s}, \mathbf{a})\right] \\
&- \mathbb{E}\left[\gamma^t r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\hat{\mathbf{s}}, \hat{\mathbf{a}})\right]| \\
&\overset{(a)}{=} \sum_{t=1}^{\infty} |\mathbb{E}\left[\gamma^t r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\mathbf{s}, \mathbf{a})\right] \\
&- \mathbb{E}\left[\gamma^t r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\hat{\mathbf{s}}, \hat{\mathbf{a}})\right]| \\
&\leq \sum_{t=1}^{\infty} \gamma^t (|\mathbb{E}\left[r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\mathbf{s}, \mathbf{a})\right]| \\
&+ |\mathbb{E}\left[r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\hat{\mathbf{s}}, \hat{\mathbf{a}})\right]|) \\
&\leq \sum_{t=1}^{\infty} 2\gamma^t \bar{r} = \frac{2\bar{r}\gamma}{1-\gamma} \qquad (4)
\end{aligned}
$$

where (a) derives from the fact that $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$ are part of both $(\mathbf{s}, \mathbf{a})$ and $(\hat{\mathbf{s}}, \hat{\mathbf{a}})$. In the above equations, we have removed the subscription of the expectation function $\mathbb{E}()$ for simplicity, which should be $\mathbf{a}(t) = \boldsymbol{\mu}(\mathbf{s}(t)), \mathbf{s}(t) \sim p$. Then, we have

$$
\begin{aligned}
&\left|\tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}) - Q_i^{\boldsymbol{\mu}}(\mathbf{s}, \mathbf{a})\right| \\
&= |\sum_{\mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}} \omega_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}) Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}) \\
&\quad - Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \hat{\mathbf{s}}_{\mathcal{N}_i^-}, \hat{\mathbf{a}}_{\mathcal{N}_i^-})| \\
&\leq \sum_{\mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}} \omega_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}) |Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}) \\
&\quad - Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \hat{\mathbf{s}}_{\mathcal{N}_i^-}, \hat{\mathbf{a}}_{\mathcal{N}_i^-})| \leq \frac{2\bar{r}\gamma}{1-\gamma} \qquad \square
\end{aligned}
$$

B. Proof of Proposition 1

If agent $j \notin \mathcal{P}_i(t)$, then based on the definition of potential neighbors, we have $dist(\mathbf{s}_i(t), \mathbf{s}_j(t)) > d + 2\epsilon$. According to the triangle inequality, $dist(\mathbf{s}_i(t), \mathbf{s}_j(t+1)) + dist(\mathbf{s}_j(t+1), \mathbf{s}_j(t)) \geq dist(\mathbf{s}_i(t), \mathbf{s}_j(t))$, and according to Assumption 1, $dist(\mathbf{s}_j(t+1), \mathbf{s}_j(t)) \leq \epsilon$. Therefore, $dist(\mathbf{s}_i(t), \mathbf{s}_j(t+1)) > d + \epsilon$. Furthermore, using triangle inequality, we can obtain $dist(\mathbf{s}_i(t+1), \mathbf{s}_j(t+1)) + dist(\mathbf{s}_i(t+1), \mathbf{s}_i(t)) \geq dist(\mathbf{s}_i(t), \mathbf{s}_j(t+1))$. As $dist(\mathbf{s}_i(t+1), \mathbf{s}_i(t)) \leq \epsilon$, we have $dist(\mathbf{s}_i(t+1), \mathbf{s}_j(t+1)) > d$. Therefore, agent j will not be a one-hop neighbor of agent i at time $t+1$. $\square$

C. Experiment Settings

a) Training Parameters: All environments adopt the same training parameters. In particular, Adam optimizer is used to update the policy and Q function parameters with a learning rate of 0.01. The parameter τ in the Polyak averaging algorithm for updating target policy and target Q functions is set to $\tau = 0.01$. The discount factor γ is set to $\gamma = 0.95$. The sizes of the buffer are set to 10^4 and 10^6 for Food Collection and other environments, respectively. The parameters are updated after every 4 episodes. The $max\_transition\_number$ in Alg. 1 of DARL1N is set to 4 times of the length of one episode. In the Ising Model and Food Collection environments, the length of each episode is set to 25 in all scenarios. In the Grassland and Adversarial Battle environments, the length of an episode is set to $25, 30, 35$ and 40 for the scenarios of $M = 6, 12, 24$ and 48, respectively. The size of a mini batch is set to 32 in the Ising Model and 1024 in other environments.

b) Q Function and Policy Function Representation: In the implementations of DARL1N, MADDPG and MFAC, we use neural networks with fully connected layers to represent the approximated Q function and policy function. The neural networks have three hidden layers with each layer having 64 units and adopting ReLU as the activation function. To handle the varying sizes of $\mathbf{s}_{\mathcal{N}_i}$ and $\mathbf{a}_{\mathcal{N}_i}$ in approximated Q function in DARL1N, we let the input dimension of the approximated Q function to be the size of the joint state and action space of the maximum number of agents that can be in $\mathcal{N}_i$ and apply zero padding for agents that are not in the one-hop neighborhood of agent i. In particular, in the Ising Model, the maximum number of one-hop neighbors of an agent is 5, which is fixed. The input dimension of the approximated Q function for agent i is then $5 \times (|\mathcal{S}_i| + |\mathcal{A}_i|)$. For other environments, the maximum number of one-hop neighbors of an agent is the total number of agents. The EPC adopts a population-invariant neural network architecture with attention modules to support arbitrary number of agents in different stages for training the Q function and policy function.

c) Environment and Neighborhood Configurations: The one-hop neighbors of an agent are defined over the agent's state space using a distance metric, which is the prior knowledge of environment. In the Ising Model, the topology of the agents is fixed, and the one-hop neighbors of an agent are its vertically and horizontally adjacent agents and itself. In the other environments, the Euclidean distance between two agents in the 2-D space is used as the distance metric, and the neighbor distance d is set to 0.15, 0.2, 0.25, 0.3, 0.35 when $M = 3, 6, 12, 24, 48$, respectively. The bound ϵ is determined according to the maximum velocity and time interval between two consecutive time steps, and is set to 0.05, 0.10, 0.15, 0.20, 0.25 when $M = 3, 6, 12, 24, 48$, respectively. The size of agents' activity space is set to $[-1, 1] \times [-1, 1]$, $[-1.5, 1.5] \times [-1.5, 1.5]$, $[-2, 2] \times [-2, 2]$, $[-2.5, 2.5] \times [-2.5, 2.5]$, $[-3, 3] \times [-3, 3]$ when $M = 3, 6, 12, 24, 48$, respectively.