Projection-Optimal Monotonic Value Function Factorization in Multi-Agent Reinforcement Learning

Yongsheng Mei*
The George Washington University
Washington, DC, USA
ysmei@gwu.edu

Hanhan Zhou*
The George Washington University
Washington, DC, USA
hanhan@gwu.edu

Tian Lan
The George Washington University
Washington, DC, USA
tlan@gwu.edu

ABSTRACT

Value function factorization has emerged as the prevalent method for cooperative multi-agent reinforcement learning under the centralized training and decentralized execution paradigm. Many of these algorithms ensure the coherence between joint and local action selections for decentralized decision-making by factorizing the optimal joint action-value function using a monotonic mixing function of agent utilities. Despite this, utilizing monotonic mixing functions also induces representational limitations, and finding the optimal projection of an unconstrained mixing function onto monotonic function classes remains an open problem. In this paper, we propose QPro, which casts this optimal projection problem for value function factorization as regret minimization over projection weights of different transitions. This optimization problem can be relaxed and solved using the Lagrangian multiplier method to obtain the optimal projection weights in a closed form, where we narrow the gap between optimal and restricted monotonic mixing functions by minimizing the policy regret of expected returns, thereby enhancing the monotonic value function factorization. Our experiments on Predator-Prey and StarCraft environments demonstrate the effectiveness of our method, indicating improved performance in environments with non-monotonic value functions.

KEYWORDS

Multi-agent Reinforcement Learning, Value Function Factorization, Optimization

ACM Reference Format:
Yongsheng Mei*, Hanhan Zhou*, and Tian Lan. 2024. Projection-Optimal Monotonic Value Function Factorization in Multi-Agent Reinforcement Learning. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 16 pages.

1 INTRODUCTION

Reinforcement learning has demonstrated its capability to solve challenging real-world problems, ranging from autonomous driving [3, 10] to robotics and planning [11, 17, 21]. In some scenarios, these tasks involve multiple agents operating in the same environment, requiring multi-agent reinforcement learning (MARL)[1, 13, 36, 38] to coordinate the agents and learn the desired behaviors from their experiences. However, due to practical communication constraints and the need to handle vast joint action space, MARL algorithms often rely on fully decentralized policies while learning them in a centralized fashion with access to additional information during training. Value function factorization methods such as QMIX[26], QPLEX [37], Qatten [40], FOP [41], and DOP [39] have been the dominant approach for centralized training and decentralized execution (CTDE) MARL [16]. These algorithms factorize the optimal joint action value function using a monotonic mixing function of per-agent utilities to ensure consistency between joint and local action selections for decentralized decision-making. Many MARL tasks, such as the StarCraft Multi-Agent Challenge (SMAC) [27], have reported superior performance with these value function factorization methods.

Value function factorization can be considered as an operator that first computes the optimal joint action value functions as targets and then projects them onto the space that can be represented by monotonic function classes [5]. The projected monotonic mixing functions enable efficient maximization while allowing decentralized decision-making. However, this approach also poses limitations regarding representation. For example, QMIX employs a universal approximator for non-linear monotonic mixing functions, which prevents it from efficiently representing joint action value functions where the agents' orderings of their action choices depend on each other [20]. To address this limitation, the authors proposed an improved projection method using Weighted QMIX (WQMIX) [25] that assigns higher weights to the values of optimal joint actions than the suboptimal ones, resulting in a better projection that more accurately represents these optimal values. However, WQMIX still relies on a heuristic design, such as Centrally-Weighted (CW) and Optimistically-Weighted (OW) QMIXs, where the weight terms are constant. Therefore, finding an optimal projection onto the monotonic function class remains an open problem.

To this end, we propose QPro, which formulates the optimal projection problem for value function factorization as regret minimization over the projection weights of different state-action values. Our method involves constructing an optimal policy based on the optimal joint action-value function and a restricted policy using its projection onto monotonic mixing functions. We then define policy regret as the difference between the expected discounted reward of the optimal policy and that of the restricted policy. By minimizing this policy regret through an upper bound, we can minimize the gap between the optimal and restricted policies, leading to an optimal monotonic factorization with minimum regret. It is worth noting that while regret has been used in various optimization problems in reinforcement learning, such as prioritized experience replay [18] and loss function design [14], our goal is to optimize the value function factorization in MARL through regret minimization.

*Equal contribution.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand.

Our proposed regret minimization problem can be solved using the Lagrangian method considering an upper bound. We derive the optimal projection weights in closed form by examining a weighted Bellman equation involving monotonic mixing functions and per-agent critics and leveraging the implicit function theorem (IFT) and Karush-Kuhn-Tucker (KKT) conditions. Our results shed light on the key principles that contribute to optimal monotonic value function factorization. The optimal projection weights consist of four components: Bellman error, value underestimation, the gradient of the monotonic mixing function, and the on-policiness of available transitions. We note that the first two components are consistent with the weighting heuristics proposed in WQMIX and provide a quantitative justification for this method. Furthermore, our analysis shows that an optimal value function factorization should also consider the gradient of the monotonic mixing function and the positive impact of more current transitions.

Following the theoretical results, we provide a tractable approximation of the optimal projection weights and propose QPro learning optimal monotonic value function factorization. We evaluate the performance of QPro in Predator-Prey [2] and SMAC. Compared with state-of-the-art factorization-based MARL algorithms (e.g., WQMIX, QPlex, ResQ, DOP), QPro is shown to better cope with environments with non-monotonic value functions, resulting in improved convergence and superior empirical performance.

The main contributions of our work are as follows:

- We propose a novel method, QPro, formulating the optimal value function factorization as regret minimization and obtaining the weights of the optimal projection in closed form.
- The theoretical contributions and tractable weight approximations of QPro enable cooperative MARL algorithms with improved value function factorization.
- Experiments on Predator-Prey and SMAC environments show that QPro outperforms state-of-the-art factorization-based methods regarding convergence and empirical performance. We further perform ablation studies to demonstrate the contribution of each component in our design.

2 BACKGROUND

2.1 Partially Observable Markov Decision Process

We describe a fully cooperative multi-agent sequential decision-making task as a decentralized partially observable Markov decision process (Dec-POMDP) [24]. The task consists of a tuple $G = \langle S, U, P, R, Z, O, n, \gamma \rangle$, where $s \in S$ describes the global state of the environment. Every time, each agent $a \in A \equiv \{1, \dots, n\}$ selects an action $u_a \in U$, and all selected actions are combined to form a joint action $\boldsymbol{u} \in \mathbf{U}$, which causes a transition in the environment based on the state transition function $P(s'|s, \boldsymbol{u}) : S \times \mathbf{U} \times S \to [0, 1]$. All agents share the same reward function $r(s, \boldsymbol{u}) : S \times \mathbf{U} \to \mathbb{R}$ with a discount factor $\gamma \in [0, 1)$. In the partially observable environment, the agents' individual observations $z \in Z$ are generated by the observation function $O(s, u) : S \times A \to Z$. Each agent has an action-observation history $\tau_a \in T \equiv (Z \times U)^*$, and the policy $\pi^a(u_a|\tau_a) : T \times U \to [0, 1]$ is conditioned on the history. The joint policy $\boldsymbol{\pi}$ has a joint action-value function: $Q^{\boldsymbol{\pi}}(s_t, \boldsymbol{u}_t) = \mathbb{E}_{s_{t+1:\infty}, \boldsymbol{u}_{t+1:\infty}}[R_t|s_t, \boldsymbol{u}_t]$, where t is the timestep and $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ is the discounted return. In this paper, we adopt the CTDE paradigm, where the algorithm has access to all local action-observation histories $\boldsymbol{\tau}$ and global state s during training, but every agent can only access its individual history during execution. Although we compute individual policy based on histories in practice, following the existing work [32] where the state solely includes state and history, we use $\boldsymbol{\pi}(\boldsymbol{u}|s)$ and $\pi^a(u_a|s)$ in analysis and proofs for simplicity.

2.2 Value Function Decomposition

In MARL, value function decomposition methods [26, 30, 33, 39] learn a joint action value functions $Q_{tot}(s, \boldsymbol{u})$ as a function of combined individual action value functions, conditioning individual local observation history, then these local action values are combined with a learnable mixing neural network to produce joint action values, given by:

$$Q_{tot}(s, \boldsymbol{u}) = \text{Mixer}\left(s, Q^a(\tau_a, u_a)\right) = f_s\left(Q^a(\tau_a, u_a)\right).$$

Under the principle of guaranteed consistency between global optimal joint actions and local optimal actions, a global argmax performed on Q_{tot} yields the same result as a set of individual arg max operations performed on each local value, also known as Individual Global Maximum (IGM):

$$\arg\max_{\boldsymbol{u}} Q_{tot} = \left(\arg\max_{u_1} Q^1, \cdots, \arg\max_{u_n} Q^n\right).$$

VDN [33] takes the joint action value function as a summation of local action value:

$$Q_{tot}(s, \boldsymbol{u}) = \sum_{i=1}^{n} Q^i(\tau_i, u_i),$$

while QMIX [26] proposed a more general case of VDN by approximating a broader class of monotonic functions to represent joint action value functions rather than summation of the local action values, such that:

$$\frac{\partial Q_{tot}(s, \boldsymbol{u})}{\partial Q^a(\tau_a, u_a)} > 0, \forall a \in A \equiv 1, \dots, n, \tag{1}$$

which restricts the joint action value function to be a monotonic mixing of agents' utilities, preventing it from projecting non-monotonic joint action representation. Later proposed WQMIX [25] solved the limitation by introducing the weights into the projection to retrieve the optimal policy. The WQMIX algorithms – OW and CW QMIXs – can place more importance on the better Q_{tot} in minimizing the loss: $\sum_{i=1}^{b} w(s, \mathbf{u})(Q_{tot}(s, \mathbf{u}; \theta) - \bar{y}_i)^2$, where $\bar{y}_i = r + \gamma \hat{Q}^*(s', \arg\max_{\mathbf{u}'} Q_{tot}(s', \mathbf{u}'; \theta^-))$ is the fixed target, $\hat{Q}^*$ is the unrestricted joint action-value function, b is the batch size and w is the weighting function. For example, in OW, the weight w is given by:

$$w(s, \mathbf{u}) = \begin{cases} 1 & Q_{tot}(s, \mathbf{u}) < \bar{y}_i \\ \alpha & \text{otherwise.} \end{cases} \tag{2}$$

When a transition is overestimated in the OW paradigm, it will be assigned with a constant weight $\alpha \in (0, 1]$. Compared to OW, CW has a similar mechanism but assigns weights to a transition whose joint action $\mathbf{u}$ is not the best. We note that while insightful, these methods are based on heuristic designs of projection weights.

Finding optimal projection weights for monotonic value function factorization is still an open problem. In this paper, we reformulate the problem as a policy regret minimization and solve the optimal projection weights in closed form by relaxing the objective and the Lagrangian method.

2.3 Regret of Expected Returns

The object of MARL is to find a joint policy $\boldsymbol{\pi}$ that can maximize the expected return: $\eta(\boldsymbol{\pi}) = \mathbb{E}_{\boldsymbol{\pi}}[\sum_{i=0}^{\infty} \gamma^i r_{t+i}]$. For a fixed policy, the Markov decision process becomes a Markov reward process, where the discounted state distribution is defined as $d^{\pi}(s)$. Similarly, the discounted state-action distribution is defined as $d^{\boldsymbol{\pi}}(s, \boldsymbol{u}) = d^{\boldsymbol{\pi}}(s)\boldsymbol{\pi}(\boldsymbol{u}|s)$. Thus, we will have the expected return rewritten as $\eta(\boldsymbol{\pi}) = \frac{1}{1-\gamma}\mathbb{E}_{d^{\boldsymbol{\pi}}(s,\boldsymbol{u})}[r(s, \boldsymbol{u})]$. We assume there exists an optimal joint policy $\boldsymbol{\pi}^*$ such that $\boldsymbol{\pi}^* = \arg\max_{\boldsymbol{\pi}} \eta(\boldsymbol{\pi})$. The regret of the joint policy $\boldsymbol{\pi}$ is defined as $\text{regret}(\boldsymbol{\pi}) = \eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi})$. The policy regret measures the expected loss when following the current policy $\boldsymbol{\pi}$ instead of optimal policy $\boldsymbol{\pi}^*$. Since $\eta(\boldsymbol{\pi}^*)$ is a constant, minimizing the regret is consistent with maximizing of expected return $\eta(\boldsymbol{\pi})$. In this paper, we use regret as an alternative optimization objective for finding the optimal projection in MARL, along with multiple constraints, e.g., the Bellman equation and the sum of projection weights. By minimizing the regret, the current policy $\boldsymbol{\pi}_k$ following a monotonic value factorization will approach the optimum $\boldsymbol{\pi}^*$ following an unrestricted value function.

3 RELATED WORK

3.1 Multi-Agent Reinforcement Learning

MARL algorithms have developed into neural-network-based methods that can cope with high-dimensional state and action spaces. Early methods practice finding policies for a multi-agent system by directly learning decentralized value functions or policies. For example, independent Q-learning [35] trains independent action-value functions for each agent via Q-learning. [34] extends this technique to DQN [23]. Recently, approaches for CTDE have come up as centralized learning of joint actions that can conveniently solve coordination problems without introducing non-stationary. COMA [6] uses a centralized critic to train decentralized actors to estimate a counterfactual advantage function for every agent. Similar works [8, 19] are also proposed based on such analysis. Under the CTDE manner, value decomposition methods, such as QMIX [26], perform well in solving cooperative problems. Besides, other mechanisms can also solve competitive problems or mixed problems. For instance, MADDPG [19] utilizes the ensemble of policies for each agent that leads to more robust multi-agent policies, showing strength in cooperative and competitive scenarios, and the extension [12] of MADDPG has been proposed to realize further optimization towards the original algorithm. This paper focuses on the cooperative setting in MARL and aims to find the optimal weighting scheme to retrieve the best projection onto the monotonic function class.

3.2 Value Decomposition Approaches

Value decomposition approaches [4, 7] are widely used in value-based MARL. Such methods integrate each agent's local action-value functions through a learnable mixing function to generate global action values. For instance, VDN [33] and QMIX estimate the optimal joint action-value function Q^* as Q_{tot} with different formations. VDN aims to learn a joint action-value function Q_{tot} of the sum of individual utilities for each agent. QMIX calculates Q_{tot} by combining mentioned utilities via a continuous state-dependent monotonic function, generated by a feed-forward mixing network with non-negative weights. QTRAN [30] and QPLEX further extend the class of value functions that can be represented. Besides value-based factorization algorithms, some works extend the value decomposition method to policy-based actor-critic algorithms. In VDAC [31], a factorized actor-critic framework compatible with A2C can obtain a reasonable trade-off between training efficiency and algorithm performance. Recently proposed FOP [41] provides a new way to factorize the optimal joint policy induced by maximum-entropy MARL into individual policies. DOP [39] addresses the issue of centralized-decentralized mismatch and credit assignment in both discrete and continuous action spaces in the multi-agent actor-critic framework. In this paper, we recast the problem of projecting an unrestricted value function onto monotonic function classes as a policy regret minimization, whose solution allows us to find the optimal projection weights to obtain an improved value function factorization.

4 OPTIMAL PROJECTION ONTO MONOTONIC VALUE FUNCTIONS

4.1 Problem Formulation

Let Q^* be the unrestricted joint action value function and $Q_{tot} = f_s(Q^1(\tau_1, u_1), \ldots, Q^n(\tau_n, u_n))$ be its estimation obtained through a monotonic mixing function $f_s(\cdot)$ of per-agent utilities $Q^a(\tau_i, u_i)$ for $a = 1, \ldots, n$. To update Q_{tot}, we use a weighted Bellman equation $\mathbb{E}_{\mu}[w_k(f_s(Q_k^1, \ldots, Q_k^n) - \mathcal{B}^* Q_{k-1}^*)^2]$ where μ is the uniform distribution that we sample data from the replay buffer, and $\mathcal{B}^* Q_{k-1}^*$ is the Bellman target. We update $f_s(Q_k^1, \ldots, Q_k^n)$ in tandem with a set of non-negative projection weights w_k for different transitions that need optimizing.

The projection maps the unrestricted action value to monotonically mixed action value regarding local utilities and weights. To formulate the regret with respect to this projection, we consider a Boltzmann policy $\boldsymbol{\pi}_k$ following the agent's individual utilities Q_k^a at step k obtained from such monotonic value factorization, as well as a similar policy $\boldsymbol{\pi}^*$ following the unrestricted value function Q^* defined over joint actions. Our objective is to minimize the regret $\eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi})$ over non-negative projection weights under relevant

constraints, i.e.,

$$\begin{aligned}
\min_{w_k} \quad & \eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi}_k) \\
\text{s.t.} \quad & (Q_k^1, \ldots, Q_k^n) = \\
& \quad \underset{(Q^1,\ldots,Q^n)\in\mathcal{Q}}{\arg\min} \quad \mathbb{E}_\mu[w_k(s,\boldsymbol{u})(f_s(Q^1,\ldots,Q^n) - \mathcal{B}^* Q_{k-1}^*)^2], \\
& \boldsymbol{\pi}_k = \{\pi_k^a\}_{a=1}^n, \\
& \pi_k^a = \frac{\exp(Q_k^a(\tau_a, u_a))}{\sum_{\tau_a, u_a'} \exp(Q_k^a(\tau_a, u_a'))}, \\
& \mathbb{E}_\mu[w_k(s,\boldsymbol{u})] = 1, \quad w_k(s,\boldsymbol{u}) \geq 0.
\end{aligned} \tag{3}$$

This is a challenging MARL projection problem since learning the optimal Q_{tot} requires optimizing the mixing networks and policies of local utilities Q^a, on which the projection weight $w(s,\boldsymbol{u})$ is jointly defined. Although the weight hyperparameters in the monotonic mixing function $f_s(\cdot)$ can be updated, we solely analyze the KKT conditions regarding Q^a, which is sufficient to obtain the optimal projection weights. According to the first constraint in equation (3), we also need to consider an implicit function where each Q^a is implicitly defined through a projection minimization of weight $w(s,\boldsymbol{u})$. We will provide our solution to this optimization problem in the coming sections. A summary of common notations is given in Appendix A.

4.2 Solving Optimal Projection Weights

The optimization problem can be solved by analyzing the KKT conditions using the Lagrangian method. To do so, we need to find the first-order derivative of the monotonic mixing network $f_s(\cdot)$. The mixing network is a universal approximator that consists of a two-layer network with non-negative weights [5]. The following lemma gives the derivative of the mixing network.

Lemma 1. *Considering a two-layer mixing network of the weight matrix W_1, W_2, bias b_1, b_2 and activation function $h(\cdot)$, the derivative of Q_{tot} over one of the local utilities Q^a is:*

$$f'_{s,Q^a} = \frac{\partial Q_{tot}}{\partial Q^a} = h'_{Q^a}([Q^1,\ldots,Q^n]^{\mathrm{T}} W_1 + b_1) \sum_{j=1}^m w_{aj}^1 w_j^2,$$

where W_1, W_2 are the $n \times m$ and $1 \times m$ matrix correspondingly, with the respective elements w_{ij}^1 and w_j^2 in each matrix. n is the agent number, and m is the width of the mixing network.

Proof. See Appendix B. □

To simplify the notations, we denote $Q_k = f_s(Q_k^1, \ldots, Q_k^n)$ as the estimated joint action-value function at step k. Since the original optimization objective is non-tractable, we adopt an upper bound of the regret via a relaxation, given by the lemma below.

Lemma 2. *The regret of expected return between current and optimal joint policy can be bounded by a tractable upper bound, which is:*

$$\begin{aligned}
&\eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi}_k) \leq \\
&\mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}(Q^*(s,\boldsymbol{u}^*) - Q_k(s,\boldsymbol{u}^*)) + \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}(Q_k(s,\boldsymbol{u}) - Q^*(s,\boldsymbol{u})).
\end{aligned}$$

Proof. See Appendix C □

We leverage the relaxed optimization problem and formulate its Lagrangian by introducing Lagrangian multipliers with respect to the constraints. To handle this problem, we need to compute the partial derivative of constructed Lagrangian, requiring the gradient of Q_k regarding projection weights. However, the local utilities Q_k^a are defined implicitly by the minimization constraint of weights in equation (3). Thus, we use the implicit function theorem (IFT) to obtain this gradient, given by the following lemma where $p_k = w_k \cdot \mu$ absorbs the data distribution μ into weights w_k.

Lemma 3. *The gradient of Q_k regarding projection weights can be derived via IFT given the first constraint in equation (3), which is:*

$$\frac{\partial Q_k}{\partial p_k} = -\frac{\operatorname{diag}(Q_k - \mathcal{B}^* Q_{k-1}^*)}{\operatorname{diag}(p_k)}.$$

Proof. See Appendix D. □

Considering all mentioned lemmas, we can solve the proposed regret minimization problem and obtain optimal projection weights in closed form (albeit with a normalization factor Z^*).

Theorem 1 (Optimal weighting scheme). *The optimal weight $w_k(s,\boldsymbol{u})$ to a relaxation of the regret minimization problem in equation (3) with discrete action space is given by:*

$$w_k(s,\boldsymbol{u}) = \frac{1}{Z^*}(E_k(s,\boldsymbol{u}) + \epsilon_k(s,\boldsymbol{u})), \tag{4}$$

where when $Q_k \leq \mathcal{B}^* Q_{k-1}^*$, we have:

$$E_k(s,\boldsymbol{u}) = \frac{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}{\mu(s,\boldsymbol{u})}(\mathcal{B}^* Q_{k-1}^* - Q_k)\exp(Q_{k-1}^* - Q_k)\left(\sum_{j=1}^n \frac{1-\pi^j}{f'_{s,Q^j}} - 1\right),$$

and otherwise (i.e., when $Q_k > \mathcal{B}^* Q_{k-1}^*$), we have:

$$E_k(s,\boldsymbol{u}) = 0,$$

where Z^* is the normalization factor, and $\epsilon_k(s,\boldsymbol{u})$ is negligible when the probability of reversing back to the visited state is small, or the number of steps agents take to revisit a previous state is large.

Proof (sketch). The complete proof is provided in Appendix E. The derivation of optimal weights consists of four major steps: (i) use a relaxation and Jensen's inequality to obtain a tractable upper bound of the regret objective for minimization; (ii) formulate the Lagrangian for the new optimization problem and analyze its KKT conditions; (iii) compute terms in the KKT conditions and particularly analyze the gradient of Q_k regarding weights; (iv) derive the optimal projection weights in closed form by setting the Lagrangian gradient to zero and applying KKT and its slackness conditions.

Step 1: Relaxing the objective and adopting Jensen's inequality. Since the original optimization is non-tractable, we replace the original optimization objective regret with a relaxed upper bound, as provided in Lemma 2. After that, we adopt Jensen's inequality to continue relaxing the intermediate objective function based on a convex function $g(x) = \exp(-x)$. The new optimization objective is:

$$\begin{aligned}
\min_{w_k} \quad & -\log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}[\exp(Q_k - Q_{k-1}^*)(s,\boldsymbol{u}^*)] - \\
& \log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}[\exp(Q_{k-1}^* - Q_k)(s,\boldsymbol{u})],
\end{aligned} \tag{5}$$

where the constraints in equation (3) still hold for the new objective.

Step 2: Formulating the Lagrangian. We leverage the Lagrangian multiplier method to solve the new optimization problem in equation (5). For simplicity, we use p_k that absorbs the distribution μ into w_k. The constructed Lagrangian with multipliers λ, ν is:

$$\begin{aligned}\mathcal{L}(p_k;\lambda,\nu) = &-\log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}[\exp(Q_k - Q^*_{k-1})(s,\boldsymbol{u}^*)]- \\ &\log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}[\exp(Q^*_{k-1} - Q_k)(s,\boldsymbol{u})]+ \\ &\lambda(\sum_{s,\boldsymbol{u}} p_k - 1) - \nu^{\mathrm{T}} p_k.\end{aligned}$$

Step 3: Computing the gradients required in the Lagrangian. The gradient $\frac{\partial Q_k}{\partial p_k}$ can be computed via IFT following Lemma 3. We also derive the gradient $\frac{\partial d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}{\partial p_k}$ for solving the Lagrangian, and its derivation is given in the appendix.

Step 4: Deriving the Optimal Weight. After obtaining the equation for gradients and an expression of the Lagrangian, we can compute the optimal p_k by applying the KKT conditions, requiring setting the partial derivative of the Lagrangian equaling to zero, as:

$$\frac{\partial \mathcal{L}(p_k;\lambda,\nu)}{\partial p_k} = 0,$$

where the optimal weight w_k can be acquired from the p_k. □

The theoretical results shed light on the key factors determining an optimal projection onto monotonic mixing functions. Specifically, the optimal projection weights consist of four components relating to Bellman error, value underestimation, the gradient of the monotonic mixing function, and the on-policiness of available transitions. We will interpret these four components next and develop a deep MARL algorithm through approximations of the optimal projection weights.

Bellman error $\mathcal{B}^*Q^*_{k-1} - Q_k$: Q_k is the current estimate of the action-value function after the Bellman update. This term quantifies the deviation between the estimate and the Bellman target, with a larger difference indicating a higher hindsight Bellman error. Our analysis shows that, according to the KKT slackness condition, the optimal projection weight is zero when $Q_k > \mathcal{B}^*Q^*_{k-1}$ is an overestimate of the target value, and contrarily, a higher weight should be assigned when Q_k is more underestimated.

Value underestimation $\exp(Q^*_{k-1} - Q_k)$: If Q_{tot} after the Bellman update at current step k is smaller than optimal Q^*_{k-1}, it results in an underestimate. In this case, we will assign a higher weight (always larger than 1) to this transition, which is proportional to the exponential of this underestimation gap. In contrast, when overestimating (with a negative gap), the assigned weight becomes lower and always smaller than 1. This is important because an underestimate of function approximation may lead to a sub-optimal Q_k estimation and thus non-optimal action selections.

Gradient of the mixing network $\sum_{j=1}^{n} \frac{1-\pi^j}{f'_{s,Q^j}} - 1$: The optimal projection weights also depend on the inverse of the gradient of the monotonic mixing function $f_s(\cdot)$, which is a new discovery. Intuitively, the optimal projection weights would become higher when the monotonic mixing function is insensitive to underlying per-agent utility values (i.e., having a small, positive gradient). We view this result as a form of normalization with respect to different shapes of monotonic mixing function. In practice, a two-layer mixing network with non-negative weights is often used to approximate the monotonic function to produce Q_k. The parameters of the mixing network are updated every step, and the gradient value can be readily computed from these parameters. We have provided an instance regarding calculating the gradient of a two-layer mixing network in Lemma 1. It is worth noting that similar gradients can also be obtained for other value function factorization methods.

Measurement of on-policy transitions $\frac{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}{\mu(s,\boldsymbol{u})}$: To update the joint action value function efficiently, we can prioritize transitions that are more likely to be encountered by the current policy, i.e., those with a higher $d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})$. Adding this term can accelerate the search for the optimal Q_k that is closer to Q^*_{k-1}.

4.3 Proposed Algorithm

Our analytical results in Theorem 1 identify four key factors determining the optimal projection weights. Interestingly, the first two terms, relating to Bellman error and value underestimation, recover the heuristic designs in WQMIX. Specifically, when the Bellman error of a particular transition is high, which indicates a wide gap between Q_k and Q^*_{k-1}, we may consider assigning a larger weight to this transition. Similarly, value underestimation works as a correction term for incoming transitions: based on the difference of current Q_k and ideal Q^*_{k-1}, it will compensate the underestimated Q_k with larger importance while penalizing overestimated Q_k with a smaller weighting modifier, consistent with OW scheme in equation (2).

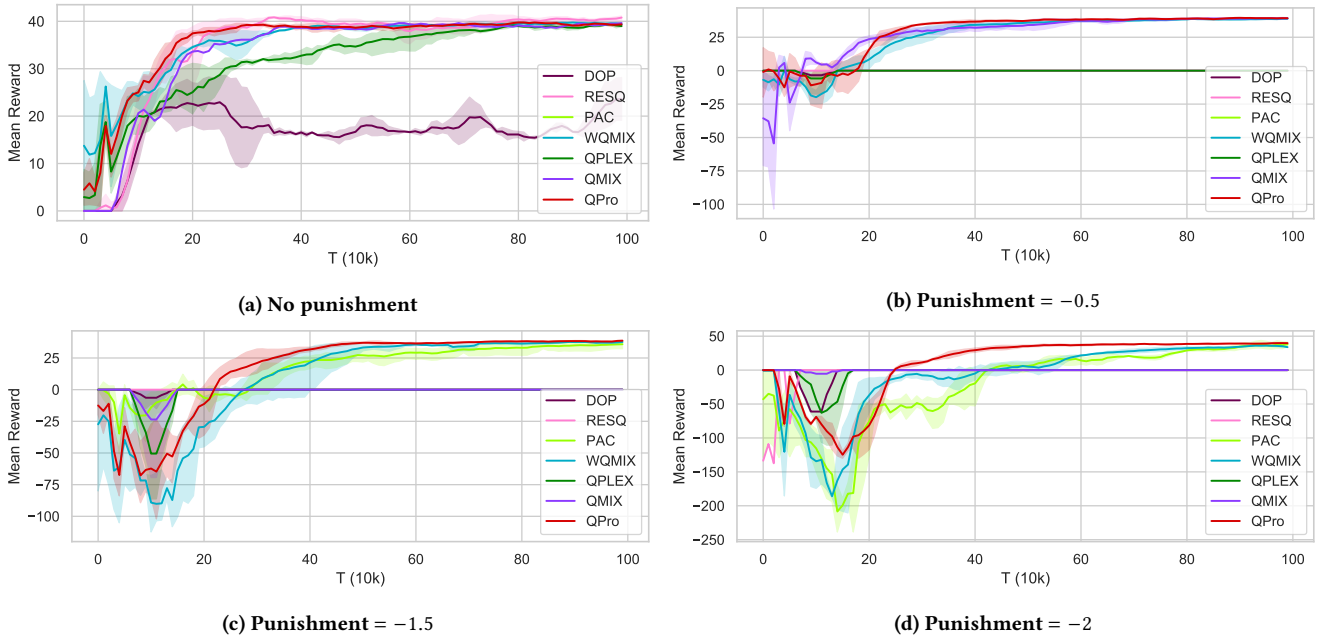
Additionally, our analysis identifies two new terms: the gradient of the monotonic mixing function and measurement of on-policy transitions, which are crucial in obtaining an optimal projection onto monotonic value function factorization. As discussed, we interpret the gradient term in optimal weights as a form of normalization – by increasing the weights for transitions, where the monotonic mixing function is less sensitive to the underlying per-agent utility, and decreasing the weights otherwise. The measurement of on-policy transitions in the weighting expression emphasizes the useful information carried by more current, on-policy transitions.

We provide a tractable approximation of the optimal projection weights and propose a MARL algorithm, QPro, with regret-minimizing projections onto the monotonic value function factorization. The procedure of QPro can be found in Algorithm 1.We consider a new loss function with respect to the optimal projection weights w_k applied to the Bellman equation of Q_k, i.e.,

$$L_{\text{QPro}} = \sum_{i=1}^{b} \left[w_i(s,\boldsymbol{u})(Q_k - y_i)^2(s,\boldsymbol{u}) \right], \tag{6}$$

where b is the batch size, and $y_i = \mathcal{B}^*Q^*_{k-1}$ is a fixed target using an unrestricted joint action-value function that can be approximated using a separate network similar to WQMIX.

To compute the projection weights for Bellman error and value underestimation terms, we again leverage the unrestricted joint action-value function Q^* to compute them quantitatively. We note that the Bellman error term also works as the condition in Theorem 1 for deciding whether the weight should be zero. The gradient of the monotonic mixing network can be directly computed using Lemma 1. Since the distribution $d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})$ in the numerator of the measurement of on-policy transitions term is not readily available,



(a) No punishment

(b) Punishment $= -0.5$

(c) Punishment $= -1.5$

(d) Punishment $= -2$

Figure 1: Average reward per episode on the Predator-Prey tasks for QPro and other baseline algorithms of 4 settings.

we approximated this term via an additional replay buffer storing the recently generated transitions and sample from it to obtain trajectories that are more possibly to be visited by current policy, which has been empirically illustrated in [29]. To account for the unknown normalization factor Z^* and improve the stability of the training process, we model Z^* as a hyperparameter and map the projection weights to several ranges, and the sensitivity results are provided in Appendix G.

5 EXPERIMENTS

In this section, we present our experimental results on Predator-Prey and SMAC and demonstrate the effectiveness of QPro by comparing the results with several state-of-the-art MARL baselines. Besides, we visualize the optimal weight pattern in heat maps to show the step-wise weight assignment for each transition and conduct the ablation experiments by disabling each term in Theorem 1. More details are provided in Appendix F.

5.1 Predator-Prey

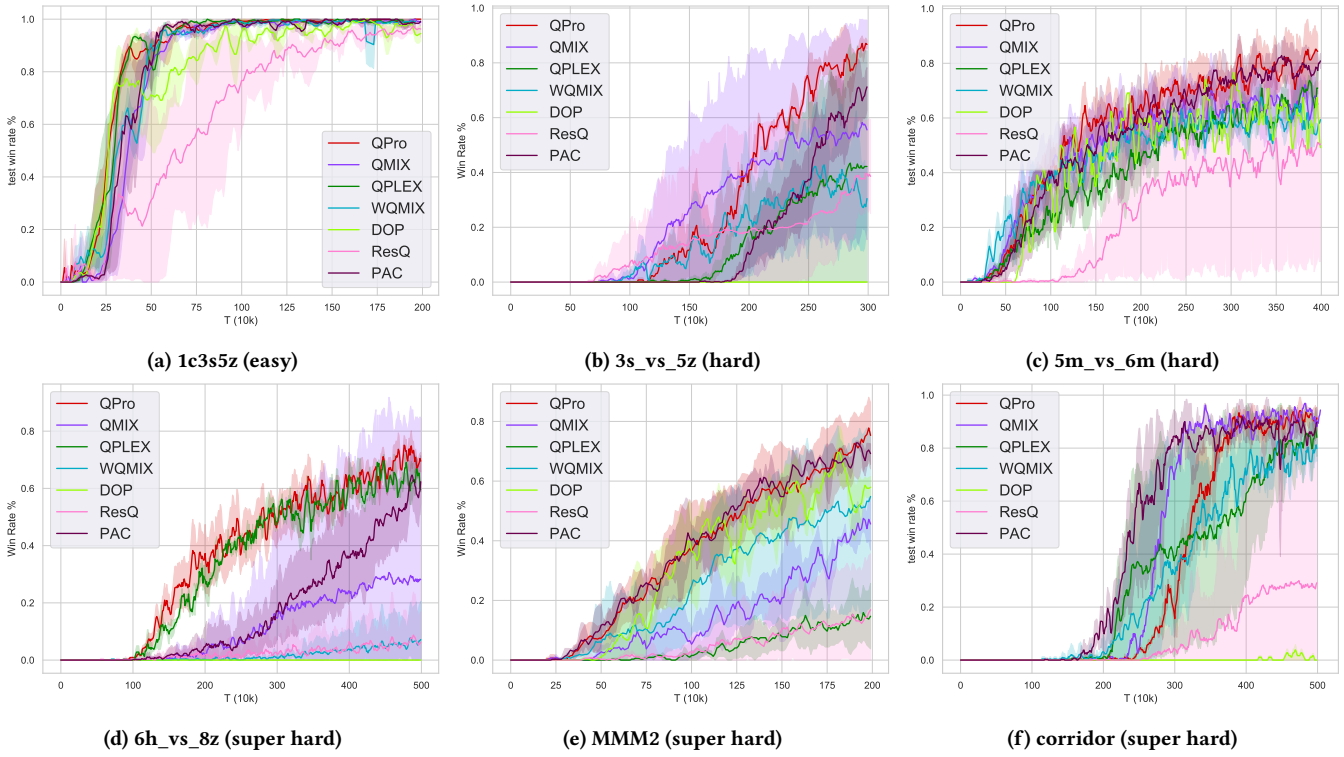
To start with, we consider a complex partially-observable multi-agent cooperative environment, Predator-Prey, that involves 8 agents in cooperation as predators to catch 8 prey on a 10×10 grid. In this task, a successful capture with the positive reward of 1 must include two or more predator agents surrounding and catching the same prey simultaneously, requiring a high level of cooperation. The greater punishment determines the degree of monotonicity. A failed coordination between agents to capture the prey, which happens when only one predator catches the prey, will receive a negative punishment reward. The greater punishment determines the degree of monotonicity. Algorithms that suffer from relative overgeneralization issues or make poor trade-offs in joint action-value function projection will fail to solve this task.

We select multiple state-of-the-art MARL approaches as baseline algorithms for comparison, which includes value-based factorization algorithms (i.e., QMIX [26], WQMIX[25], ResQ [28] and QPLEX[37]), and decomposed actor-critic approaches (i.e., PAC [42] and DOP [39]). All mentioned baseline algorithms have shown strength in handling MARL tasks in existing works.

Figure 1 shows the performance of seven algorithms with different punishments, where all results demonstrate the superiority of QPro over others. Besides, regarding efficiency, we can spot that QPro has the fastest convergence speed in seeking the best policy. In Figure 1c and 1d, QPro significantly outperforms other state-of-the-art algorithms in a hard setting requiring a higher level of coordination among agents as learning the best policy with improved joint action representation is required in this setting. Most algorithms, such as QMIX, ResQ, and DOP, end up learning a sub-optimal policy where agents learn to work together with limited coordination. Although QPro and WQMIX acquired good results eventually, compared to the latter, QPro achieves better performance and converges to the optimal policy profoundly faster than WQMIX, demonstrating that our optimal weighting approach can generate a better joint action-value projection.

5.2 SMAC

Next, we evaluate QPro on the SMAC benchmark. We report the experiments on six maps consisting of one easy map, two hard maps, and three super-hard maps. The selected state-of-the-art baseline algorithms for this experiment are consistent with those in the Predator-Prey environment. The empirical results are provided in Figure 2, demonstrating that QPro can effectively generate



(a) 1c3s5z (easy) **(b) 3s_vs_5z (hard)** **(c) 5m_vs_6m (hard)**

(d) 6h_vs_8z (super hard) **(e) MMM2 (super hard)** **(f) corridor (super hard)**

Figure 2: Results of 6 maps (from easy to super hard) on the SMAC benchmark.

optimal weight projection for joint actions on SMAC for achieving a higher win rate, especially when the environment becomes substantially complicated and harder, such as *MMM2*. We can see that several state-of-the-art policy-based factorization algorithms are brittle when significant exploration is undergone since joint action representations generated by them are sub-optimal.

Specifically, QPro performs well on an easy map *1c3s5z* in Figure 2a, albeit holding the comparable performance among algorithms. Specifically, on hard map *3s_vs_5z*, the best policy found by our optimal weighting approach significantly outperforms the remaining baseline algorithms regarding winning rate. For super-hard map *6h_vs_8z* and *MMM2*, QPro can learn a better policy than ResQ, DOP, and WQMIX. We achieve the highest winning rate by adopting our algorithm on *6h_vs_8z* and *MMM2*. Compared to our method, QMIX and WQMIX suffer from this map as their joint action representations are oblivious to some latent factors, such as the shape of the monotonic mixing network, and therefore fail to generate an accurate joint action representation. On *corridor*, QPro manages to learn the model with better performance than WQMIX, QPLEX, and other policy-based algorithms, though standard QMIX has the fastest convergence rate among all baseline algorithms.

5.3 Optimal Weight Pattern

In this test, we generate the heat maps of the projecting weight probability distributions of QPro and WQMIX as the training proceeds to better visualize and compare the weight evolution pattern of transitions sampled as in a minibatch, shown in Figure 3. Adopted weights are generated from the Predator-Prey task with a punishment of -2. We re-scale the absolute value of the transition number to logarithmic probability for scale normalization. As shown in the figure, the probability value of a certain weight is represented by colors, decreasing from 0 in light yellow to -10 in black. The vertical axis represents the training steps, and the horizontal axis represents the normalized weight value, where ours ranges from 0.1 to 1 and WQMIX is either 0.1 or 1.

The heat map effectively shows the general trend of the weight evolution pattern at different steps. For WQMIX on the Figure 3 right, with the training of the algorithm, the transitions with the smaller weight (0.1) will become more, and those with the larger weight (1) will become fewer. Evolution like this happens since the transitions will approach optimal as the training goes on, while the algorithm will still take all transitions as potential overestimations and assign smaller weights to them as adjustments. A similar evolution pattern can be found in our weight pattern. On the left of Figure 3, during the training, the transitions with higher weights become less, and most transitions will migrate to the bottom right with lower weights, which empirically recovers the heuristic in WQMIX.

Moreover, as an optimal weight projection is used in QPro, we will assign different weights to transitions based on evaluating every one of them. We notice that some transitions are assigned with medium weight during the training, given by the light yellow spots on the left of Figure 3. Such a phenomenon demonstrates that the binary-weighted projections in WQMIX are not always

Algorithm 1 QPro

Initialize step, the parameters of mixing network, agent networks, and hyper-network.
Set the learning rate δ, replay buffer $\mathcal{D}$, and fast replay buffer $\mathcal{D}_+$ (smaller than $\mathcal{D}$)
let $\theta^- = \theta$
for step = 1 : step_{max} **do**
  $k = 0, s_0 =$ initial state
  while $s_k \neq$ terminal and $k <$ episode limit **do**
    for each agent a **do**
      $\tau_k^a = \tau_{k-1}^a \cup (o_k, u_{k-1})$
      $u_k^a = \begin{cases} \arg\max_{u_k^a} Q(\tau_k^a, u_k^a) & \text{with probability } 1-\epsilon \\ \text{randint}(1, |U|) & \text{with probability } \epsilon \end{cases}$
    end for
    Obtain the reward r_k and next state s_{k+1}
    Store the current trajectory into the replay buffer $\mathcal{D} = \mathcal{D} \cup (s_k, \boldsymbol{u}_k, r_k, s_{k+1})$
    Store the current trajectory into the fast replay buffer $\mathcal{D}_+ = \mathcal{D}_+ \cup (s_k, \boldsymbol{u}_k, r_k, s_{k+1})$
    $k = k + 1$, step = step + 1
  end while
  Collect b_1 samples from the replay buffer $\mathcal{D}$ following uniform distribution μ
  Collect b_2 samples from the fast replay buffer $\mathcal{D}_+$ following uniform distribution μ
  Obtain the batch $b = b_1 \cup b_2$
  for each timestep k in each episode in batch b **do**
    Evaluate Q_k, Q^* and target values
    Obtain the utilities Q_a from agents' local networks, and compute the individual policy π_k^a
    Compute the weight:
    (i) when $Q_k \leq \mathcal{B}^* Q_{k-1}^*$:
    $w_k \propto (\mathcal{B}^* Q_{k-1}^* - Q_k) \exp(Q_{k-1}^* - Q_k) \left(\sum_{j=1}^n \frac{1-\pi^j}{f'_{s,Q^j}} - 1 \right)$
    (ii) when $Q_k > \mathcal{B}^* Q_{k-1}^*$:
    $w_k \approx \epsilon$
  end for
  Minimize the Bellman error for Q_k weighted by w_k, update the network parameter θ:
  $\theta = \theta - \delta(\nabla_\theta \frac{1}{b} \sum_i^b w_k (Q_k - y_i)^2)$.
  if update-interval steps have passed **then**
    $\theta^- = \theta$
  end if
end for

accurate. Hence, QPro considers all transitions by applying optimal weights to their projections, leading to better results, which also illustrates the performance gap with other algorithms like WQMIX in previous experiments.

5.4 Ablation Experiment

For ablations, we conduct experiments by disabling one single term (in Theorem 1) each at a time to investigate their contribution to finding optimal projection weights, respectively. The ablation results are given in Figure 4 showing the results on *MMM2*. The terms

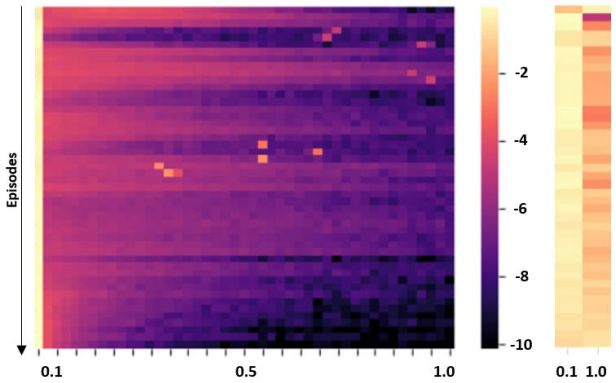


Figure 3: Heatmap pattern of generated optimal weights of QPro (left) and WQMIX weights (right) used in the Predator-Prey environment. The training episodes range from 0 to 1M.

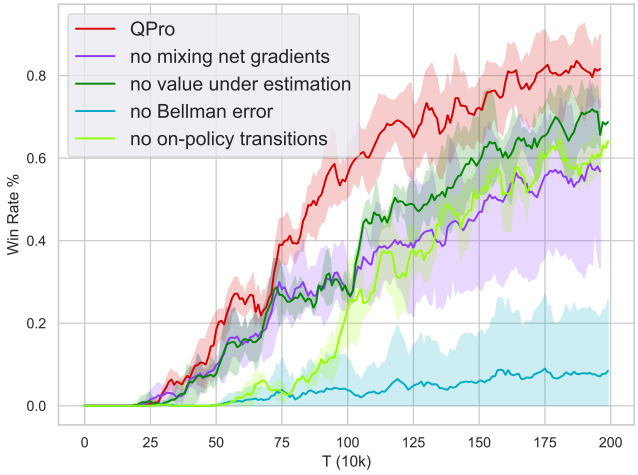


Figure 4: Ablation by disabling one term each for QPro on MMM2 (super hard), where the Bellman error and gradient of the mixing network terms have more significant impacts to optimal results than other terms.

considered in these experiments are the Bellman error, value underestimation, gradient of the mixing network, and measurement of on-policy transitions. Compared to the original result, missing any terms will harm the performance. The tests without Bellman error have the lowest final winning rate, which is less than 10%, and the test result without on-policy transitions shows a relatively slower convergence speed with less optimal performance. Furthermore, when we disable the gradient of the mixing network term, the result is only around 60%, demonstrating that providing a quantitative weight factorization for the value projection is the critical factor in value-factorization-based MARL tasks. The designing of an optimal weighting scheme without considering the mixing network's influence will be less capable of achieving the ideal final results.

6 CONCLUSION

In this paper, we formulate the optimal value function factorization as a policy regret minimization and solve the optimal projection weights for the cooperative multi-agent reinforcement learning problems in closed form. The theoretical results shed light on key factors for an optimal projection. Therefore, we propose QPro as a tractable weight approximation approach to enable MARL algorithms with improved value function factorization. Our experiment results in multiple MARL environments show the effectiveness of QPro by demonstrating superior convergence and empirical performance over state-of-the-art factorization-based methods. Beyond that, our theoretical results rely on several necessary assumptions and the relaxed optimization objective, where the gap between the original objective and the upper bound needs analyzing in the future work.

REFERENCES

[1] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. 2019. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528* (2019).

[2] Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. 2020. Deep coordination graphs. In *International Conference on Machine Learning*. PMLR, 980–991.

[3] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. 2012. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial informatics* 9, 1 (2012), 427–438.

[4] Jacopo Castellini, Frans A Oliehoek, Rahul Savani, and Shimon Whiteson. 2019. The representational capacity of action-value networks for multi-agent reinforcement learning. *arXiv preprint arXiv:1902.07497* (2019).

[5] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. 2009. Incorporating Functional Knowledge in Neural Networks. *Journal of Machine Learning Research* 10, 6 (2009).

[6] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2018. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[7] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. 2002. Coordinated reinforcement learning. In *ICML*, Vol. 2. Citeseer, 227–234.

[8] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. 2017. Cooperative multi-agent control using deep reinforcement learning. In *International conference on autonomous agents and multiagent systems*. Springer, 66–83.

[9] Jian Hu, Siyang Jiang, Seth Austin Harding, Haibin Wu, and Shih-wei Liao. 2021. RIIT: Rethinking the Importance of Implementation Tricks in Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2102.03479* (2021).

[10] Yeping Hu, Alireza Nakhaei, Masayoshi Tomizuka, and Kikuo Fujimura. 2019. Interaction-aware decision making with adaptive strategies under merging scenarios. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 151–158.

[11] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. 2017. Guided deep reinforcement learning for swarm systems. *arXiv preprint arXiv:1709.06011* (2017).

[12] Shariq Iqbal and Fei Sha. 2019. Actor-attention-critic for multi-agent reinforcement learning. In *International conference on machine learning*. PMLR, 2961–2970.

[13] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. 2019. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International conference on machine learning*. PMLR, 3040–3049.

[14] Peter Jin, Kurt Keutzer, and Sergey Levine. 2018. Regret minimization for partially observable deep reinforcement learning. In *International conference on machine learning*. PMLR, 2342–2351.

[15] Sham Kakade and John Langford. 2002. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer.

[16] Landon Kraemer and Bikramjit Banerjee. 2016. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing* 190 (2016), 82–94.

[17] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research* 17, 1 (2016), 1334–1373.

[18] Xu-Hui Liu, Zhenghai Xue, Jingcheng Pang, Shengyi Jiang, Feng Xu, and Yang Yu. 2021. Regret Minimization Experience Replay in Off-Policy Reinforcement Learning. *Advances in Neural Information Processing Systems* 34 (2021), 17604–17615.

[19] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems* 30 (2017).

[20] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. 2019. Maven: Multi-agent variational exploration. *arXiv preprint arXiv:1910.07483* (2019).

[21] Laëtitia Matignon, Laurent Jeanpierre, and Abdel-Illah Mouaddib. 2012. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In *Twenty-sixth AAAI conference on artificial intelligence*.

[22] Edward James McShane. 1937. Jensen's inequality. *Bull. Amer. Math. Soc.* 43, 8 (1937), 521–527.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[24] Frans A Oliehoek and Christopher Amato. 2016. *A concise introduction to decentralized POMDPs*. Springer.

[25] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. 2020. Weighted QMIX: Expanding Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. arXiv:2006.10800 [cs.LG]

[26] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR, 4295–4304.

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

[28] Siqi Shen, Mengwei Qiu, Jun Liu, Weiquan Liu, Yongquan Fu, Xinwang Liu, and Cheng Wang. 2022. ResQ: A Residual Q Function-based Approach for Multi-Agent Reinforcement Learning Value Factorization. *Advances in Neural Information Processing Systems* 35 (2022), 5471–5483.

[29] Samarth Sinha, Jiaming Song, Animesh Garg, and Stefano Ermon. 2022. Experience replay with likelihood-free importance weights. In *Learning for Dynamics and Control Conference*. PMLR, 110–123.

[30] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. 2019. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR, 5887–5896.

[31] Jianyu Su, Stephen Adams, and Peter Beling. 2021. Value-decomposition multi-agent actor-critics. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 11352–11360.

[32] Kefan Su and Zongqing Lu. 2022. Divergence-regularized multi-agent actor-critic. In *International Conference on Machine Learning*. PMLR, 20580–20603.

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

[34] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. 2017. Multiagent cooperation and competition with deep reinforcement learning. *PloS one* 12, 4 (2017), e0172395.

[35] Ming Tan. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*. 330–337.

[36] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354.

[37] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. 2020. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062* (2020).

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

[39] Yihan Wang, Beining Han, Tonghan Wang, Heng Dong, and Chongjie Zhang. 2020. Dop: Off-policy multi-agent decomposed policy gradients. In *International Conference on Learning Representations*.

[40] Yaodong Yang, Jianye Hao, Ben Liao, Kun Shao, Guangyong Chen, Wulong Liu, and Hongyao Tang. 2020. Qatten: A general framework for cooperative multiagent reinforcement learning. *arXiv preprint arXiv:2002.03939* (2020).

[41] Tianhao Zhang, Yueheng Li, Chen Wang, Guangming Xie, and Zongqing Lu. 2021. FOP: Factorizing Optimal Joint Policy of Maximum-Entropy Multi-Agent Reinforcement Learning. In *International Conference on Machine Learning*. PMLR, 12491–12500.

[42] Hanhan Zhou, Tian Lan, and Vaneet Aggarwal. 2022. PAC: Assisted Value Factorization with Counterfactual Predictions in Multi-Agent Reinforcement

Learning. *Advances in Neural Information Processing Systems* 35 (2022), 15757–15769.

A NOMENCLATURE

Table 1 summarizes the common notations in this paper.

Table 1: Definitions of notations.

Notation	Definition
s	State of the environment
a	Agent
$\boldsymbol{u}$	Agents' joint action
r	Reward
γ	Discount factor
π	Individual policy
$\boldsymbol{\pi}$	Joint policy
$\boldsymbol{\pi}^*$	Optimal joint policy
$\eta(\boldsymbol{\pi})$	Expected return under the joint policy $\boldsymbol{\pi}$
$d^{\boldsymbol{\pi}}(s)$	Discounted state distribution
$Q(\cdot)$	Action value function
$Q_{tot}(\cdot)$	Monotonic mixing of per-agent action value function
$Q^*(\cdot)$	Unrestricted joint action value function
$V(\cdot)$	Value function
$A(\cdot)$	Advantage function
$f_s(\cdot)$	Monotonic function with input state s
$\mathcal{B}^*$	Bellman operator: $\mathcal{B}^*Q(s,\boldsymbol{u}) \stackrel{\text{def}}{=} r(s,\boldsymbol{u}) + \gamma \arg\max_{\boldsymbol{u}'} \mathbb{E}_{s'} Q(s',\boldsymbol{u}')$
w	Projection weights of transitions
μ	Uniform data distribution
λ, ν	Lagrangian multipliers

B PROOF OF LEMMA 1

Proof. Considering a two-layer mixing network of the non-negative weight matrix W_1, W_2, bias b_1, b_2 and activation function $h(\cdot)$. Let $\vec{Q}$ denote the vector of all the agents' utilities. Assume there are n agents, $\vec{Q}$ is:

$$\vec{Q} = [Q^1, \ldots, Q^n]^{\mathrm{T}}$$

We assume the mixing network has the width of m, based on the input/output dimension, W_1 should be a $n \times m$ matrix as:

$$W_1 = \begin{bmatrix} w_{11}^1 & \cdots & w_{1m}^1 \\ \vdots & \ddots & \vdots \\ w_{n1}^1 & \cdots & w_{nm}^1 \end{bmatrix},$$

and W_2 is a m-dimension vector given by:

$$W_2 = [w_1^2, \ldots, w_m^2]^{\mathrm{T}}.$$

Therefore, Q_{tot} calculated from the utility vector $\vec{Q}$ becomes:

$$f_s(\vec{Q}) = h(\vec{Q}^{\mathrm{T}} W_1 + b_1) W_2^{\mathrm{T}} + b_2. \tag{7}$$

Considering one of the utilities Q^a, as long as the derivative of activation $h(\cdot)$ exists ($h(\cdot)$ is smooth and differentiable), based on equation (7), the result is:

$$f'_{s,Q^a} = \frac{\partial Q_{tot}}{\partial Q^a} = h'_{Q^a}(\vec{Q}^{\mathrm{T}} W_1 + b_1) \sum_{j=1}^{m} w_{aj}^1 w_j^2. \tag{8}$$

This concludes the proof. □

C PROOF OF LEMMA 2

Proof. Suppose $\boldsymbol{u}^*$ subject to $\boldsymbol{\pi}^*$. Let $\boldsymbol{\pi} = \boldsymbol{\pi}^*$ and $\tilde{\boldsymbol{\pi}} = \boldsymbol{\pi}_k$ in Lemma 6, the regret can be relaxed by:

$$\begin{aligned}
&\eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi}_k) \\
&= -\frac{1}{1-\gamma}\mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}[A^{\boldsymbol{\pi}^*}(s,\boldsymbol{u})] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}[V^*(s) - Q^*(s,\boldsymbol{u})] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}[V^*(s) - Q_k(s,\boldsymbol{u}^*) + Q_k(s,\boldsymbol{u}^*) - Q_k(s,\boldsymbol{u}) + Q_k(s,\boldsymbol{u}) - Q^*(s,\boldsymbol{u})] \\
&\overset{(a)}{\leq} \frac{1}{1-\gamma}\left[\mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}(Q^*(s,\boldsymbol{u}^*) - Q_k(s,\boldsymbol{u}^*)) + \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}(Q_k(s,\boldsymbol{u}) - Q^*(s,\boldsymbol{u})) + 1\right],
\end{aligned} \tag{9}$$

where (a) uses Lemma 7. Therefore, we have:

$$\eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi}_k) \leq \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}(Q^*(s,\boldsymbol{u}^*) - Q_k(s,\boldsymbol{u}^*)) + \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}(Q_k(s,\boldsymbol{u}) - Q^*(s,\boldsymbol{u})) \tag{10}$$

This concludes the proof. □

D PROOF OF LEMMA 3

Proof. To calculate the gradient $\frac{\partial Q_k}{\partial p_k}$, we leverage the implicit function theorem (IFT). Based on the first constraint in equation (3), we aim to find the minimum Q_k to satisfy the $\arg\min(\cdot)$, and therefore we need to ensure the derivative of the term inside $\arg\min(\cdot)$ (we use $g(p_k, Q_k)$ to denote this term) to be zero, which is:

$$g'_{Q_k} = 2\sum_{s,\boldsymbol{u}} p_k(Q_k - \mathcal{B}^*Q_{k-1}) = 0 \tag{11}$$

We can notice that $G(p_k, Q_k) : g'_{Q_k} = 0$ is an implicit function regarding Q_k and p_k. Hence, we apply the IFT on the $G(p_k, Q_k)$ considering the Hessian matrices of p_k and Q_k in $g(p_k, Q_k)$ as follows:

$$\frac{\partial Q_k}{\partial p_k} = -\frac{G'_{p_k}}{G'_{Q_k}} = -\frac{\mathrm{diag}(p_k)}{\mathrm{diag}(Q_k - \mathcal{B}^*Q^*_{k-1})}. \tag{12}$$

This concludes the proof. □

E PROOF OF THEOREM 1

Proof. We have provided the outline of the proof containing four key steps. In this section, we present detailed proof of the theorem. Following the existing work [32], we will use state-based joint policies for simplicity. Since we focus on the regret bound of the action value and define the Boltzmann policy in the regret via action value, i.e., Q_{tot} and Q^*, we aim to find the return gap between two action values characterized by Boltzmann policy function under centralized training.

To solve the optimization problem in equation (3), we first provide the definition of *universal approximator* [5].

Definition 1 (Universal approximator). *A class of function $\hat{\mathcal{F}}$ from $\mathbb{R}^n$ to $\mathbb{R}$ is a universal approximator for a class of functions $\mathcal{F}$ from $\mathbb{R}^n$ to $\mathbb{R}$ if for any $f \in \mathcal{F}$, any compact domain $D \subset \mathbb{R}^n$, and any positive ϵ, one can find a $\hat{f} \in \hat{\mathcal{F}}$ with $\sup_{x\in D} |f(x) - \hat{f}(x)| \leq \epsilon$.*

Furthermore, we introduce some assumptions in MARL environments:

Assumption 1. *The state space S, action space U, and observation space Z are compact metric spaces.*

Assumption 2. *The action-value and observation functions are continuous on $S \times U$ and Z, respectively.*

Assumption 3. *The joint policy $\boldsymbol{\pi}$ is the product of each agent's individual policy $\pi^a(u_a|s_a)$.*

Assumption 4. *The monotonic mixing function $f_s(\cdot)$ regarding per-agent action-value function Q^a for $\forall a \in A$ is smooth and differentiable.*

Let $d^{\pi^a}(s)$ denote the discounted state distribution of agent a, and $d_i^{\pi^a}(s)$ denote the distribution where the state is visited by the agent for the i-th time. Thus, we have $d^{\pi^a}(s) = \sum_{i=1}^{\infty} d_i^{\pi^a}(s)$, where each $d_i^{\pi^a}(s)$ is given by:

$$d_i^{\pi^a}(s) = (1-\gamma)\sum_{t_i=0}^{\infty} \gamma^{t_i} \Pr(s_{t_i} = s, s_{t_k} = s, \forall k = 1, ..., i-1), \tag{13}$$

where the $\Pr(s_{t_i} = s, s_{t_k} = s, \forall k = 1, ..., i-1)$ in this equation contains the probability of visiting state s for the i-th time at t_i and a sequence of times t_k, for $k = 1, ..., i$, such that state s is visited at each t_k. Thus, state s will be visited for i times at time t_i in total.

The following lemmas are proposed in [18], where Lemma 4 support the derivation of the Lemma 5, and the latter demonstrates that $\left|\frac{\partial d^{\pi^a}(s)}{\partial \pi^a(s)}\right|$ is a small quantity. Two lemmas can be extended to suit the multi-agent scenario.

Lemma 4. *Let f be an Lebesgue integrable function. P and Q are two probability distributions, $f \leq C$, then:*

$$|\mathbb{E}_{P(x)} f(x) - \mathbb{E}_{Q(x)} f(x)| \leq C \cdot D(P, Q). \tag{14}$$

Lemma 5. *Let ρ be the probability of the agent a starting from (s, u^a) and coming back to s at time step t under policy π^a, i.e. $\Pr(s_0 = s, u_0^a = u^a, s_t = s, s_{1:t-1} \neq s; \pi^a)$, and $\epsilon = \sup_{s,u^a} \sum_{t=1}^{\infty} \gamma^t \rho^{\pi^a}(s, u^a, t)$. We have:*

$$\left|\frac{\partial d^{\pi^a}(s)}{\partial \pi^a(s)}\right| \leq \epsilon d_1^{\pi^a}(s), \tag{15}$$

where $d_1^{\pi^a}(s) = (1-\gamma) \sum_{t_1=0}^{\infty} \gamma^{t_1} \Pr(s_{t_1} = s)$ and $\epsilon \leq 1$.

The following lemma is introduced in [15]. It was originally proposed for the finite MDP, while it will also hold for the continuous scenario that is given by Assumption 1 and 2.

Lemma 6. *For any policy $\boldsymbol{\pi}$ and $\tilde{\boldsymbol{\pi}}$, given the advantage function $A^{\boldsymbol{\pi}}(s, \boldsymbol{u}) = Q^{\boldsymbol{\pi}}(s, \boldsymbol{u}) - V^{\boldsymbol{\pi}}(s)$, we have:*

$$\eta(\tilde{\boldsymbol{\pi}}) - \eta(\boldsymbol{\pi}) = \frac{1}{1-\gamma} \mathbb{E}_{d^{\tilde{\boldsymbol{\pi}}}(s,\boldsymbol{u})} [A^{\boldsymbol{\pi}}(s, \boldsymbol{u})]. \tag{16}$$

Since we are using the policy defined in Boltzmann manner, we have the following lemma measures the distance of expected action-value function under two different policies.

Lemma 7. *Given two policy $\boldsymbol{\pi}$ and $\bar{\boldsymbol{\pi}}$, the gap between two expected action-value function regarding each policy is:*

$$\mathbb{E}_{\boldsymbol{u}\sim\bar{\boldsymbol{\pi}}} [Q(s, \boldsymbol{u})] - \mathbb{E}_{\boldsymbol{u}\sim\boldsymbol{\pi}} [Q(s, \boldsymbol{u})] \leq 1. \tag{17}$$

Proof. Suppose there are two joint actions $\boldsymbol{u}$ and $\bar{\boldsymbol{u}}$. Let $Q(s, \boldsymbol{u}) < Q(s, \bar{\boldsymbol{u}})$. We have:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{u}\sim\bar{\boldsymbol{\pi}}} [Q(s, \boldsymbol{u})] - \mathbb{E}_{\boldsymbol{u}\sim\boldsymbol{\pi}} [Q(s, \boldsymbol{u})] &\leq Q(s, \bar{\boldsymbol{u}}) - \frac{Q(s, \boldsymbol{u}) \exp Q(s, \boldsymbol{u}) + Q(s, \bar{\boldsymbol{u}}) \exp Q(s, \bar{\boldsymbol{u}})}{\exp Q(s, \boldsymbol{u}) + \exp Q(s, \bar{\boldsymbol{u}})} \\
&= Q(s, \bar{\boldsymbol{u}}) - Q(s, \boldsymbol{u}) - \frac{(Q(s, \bar{\boldsymbol{u}}) - Q(s, \boldsymbol{u})) \exp (Q(s, \bar{\boldsymbol{u}}) - Q(s, \boldsymbol{u}))}{1 + \exp (Q(s, \bar{\boldsymbol{u}}) - Q(s, \boldsymbol{u}))} \\
&\stackrel{\text{def}}{=} F(Q(s, \bar{\boldsymbol{u}}) - Q(s, \boldsymbol{u}))
\end{aligned}$$

Let x denote $Q(s, \bar{\boldsymbol{u}}) - Q(s, \boldsymbol{u})$. To maximize the $F(x)$, we need to find a specific $x_0 = \arg\max_{x \in \mathcal{X}} F(x)$, i.e., the first-order derivative $F'(x) = 0$, from which we further have:

$$1 + \exp(x_0) = x_0 \exp(x_0) \tag{18}$$

where $x_0 \in (1, 2)$. Therefore, we can derive:

$$\mathbb{E}_{\boldsymbol{u}\sim\bar{\boldsymbol{\pi}}} [Q(s, \boldsymbol{u})] - \mathbb{E}_{\boldsymbol{u}\sim\boldsymbol{\pi}} [Q(s, \boldsymbol{u})] \leq F(Q(s, \bar{\boldsymbol{u}}) - Q(s, \boldsymbol{u})) = F(x) \leq x_0 - 1 \leq 1.$$

It is worth noting that the derived inequality can also be applied to the situation where we have joint action more than two or we consider the situation regarding per-agent action. □

Lemma 2 provides us with a upper bound to solve the original non-tractable optimization problem. Next, we introduce Jensen's inequality to further relax the objective, given as follows:

Lemma 8 (Jensen's inequality [22]). *The Jensen's inequality provide us with a tractable relaxation given by:*

$$\mathbb{E}[g(X)] \geq g(\mathbb{E}[X]), \tag{19}$$

when $g(x)$ is a convex function on real space $\mathbb{R}$.

We replace the original objective in equation (3) with the upper bound in Lemma 2 as both sides have the same minima at $Q_k = Q^*$, and further relax it with Jensen's inequality in Lemma 8. By additionally defining $p_k(s, \boldsymbol{u}) = w_k(s, \boldsymbol{u})\mu(s, \boldsymbol{u})$ to combine the weights and data distribution, the new relaxed optimization problem becomes:

$$\begin{aligned}
\min_{p_k} \quad & -\log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}[\exp(Q_k - Q^*_{k-1})(s, \boldsymbol{u}^*)] - \log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}[\exp(Q^*_{k-1} - Q_k)(s, \boldsymbol{u})] \\
\text{s.t.} \quad & (Q^1_k, \ldots, Q^n_k) = \underset{(Q^1,\ldots,Q^n)\in\boldsymbol{Q}}{\arg\min} \ \mathbb{E}_{p_k}[(f_s(Q^1, \ldots, Q^n) - \mathcal{B}^* Q^*_{k-1})^2], \\
& \boldsymbol{\pi}_k = \{\pi^a_k\}^n_{a=1}, \quad \pi^a_k = \frac{e^{Q^a_k(\tau_a, u_a)}}{\sum_{\tau_a, u'_a} e^{Q^a_k(\tau_a, u'_a)}}, \\
& \sum_{s,\boldsymbol{u}} p_k(s, \boldsymbol{u}) = 1, \quad p_k(s, \boldsymbol{u}) \geq 0.
\end{aligned} \tag{20}$$

In order to handle the optimization problem in equation (20), we follow the standard procedures of Lagrangian multiplier method, which is:

$$\begin{aligned}
\mathcal{L}(p_k; \lambda, \nu) = & -\log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}[\exp(Q_k - Q^*_{k-1})(s, \boldsymbol{u}^*)] - \log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}[\exp(Q^*_{k-1} - Q_k)(s, \boldsymbol{u})] \\
& + \lambda(\sum_{s,\boldsymbol{u}} p_k - 1) - \nu^{\mathrm{T}} p_k,
\end{aligned} \tag{21}$$

After constructing the Lagrangian, we obtain the gradient from Lemma 3 for solving the Lagrangian using IFT. The gradient of Q_k with respect to p_k is given by:

$$\frac{\partial Q_k}{\partial p_k} = -\frac{\operatorname{diag}(p_k)}{\operatorname{diag}(Q_k - \mathcal{B}^* Q^*_{k-1})}. \tag{22}$$

Next, we derive the expression for $\frac{\partial d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}{\partial p_k}$ in the following equation:

$$\begin{aligned}
\frac{\partial d^{\boldsymbol{\pi}_k}(s, \boldsymbol{u})}{\partial p_k} &= \frac{\partial d^{\boldsymbol{\pi}_k}(s, \boldsymbol{u})}{\partial \boldsymbol{\pi}_k} \frac{\partial \boldsymbol{\pi}_k}{\partial Q^a} \frac{\partial Q^a}{\partial Q_k} \frac{\partial Q_k}{\partial p_k} \\
&= \operatorname{diag}(d^{\boldsymbol{\pi}_k}(s) + \epsilon_0(s)) \frac{\partial \boldsymbol{\pi}_k}{\partial Q^a} \frac{\partial Q^a}{\partial Q_k} \frac{\partial Q_k}{\partial p_k} \\
&\overset{(b)}{=} \operatorname{diag}(d^{\boldsymbol{\pi}_k}(s) + \epsilon_0(s)) \operatorname{diag}(\boldsymbol{\pi}_k(1 - \boldsymbol{\pi}_k)) \frac{\partial Q^a}{\partial Q_k} \frac{\partial Q_k}{\partial p_k} \\
&\overset{(c)}{=} d^{\boldsymbol{\pi}_k}(s, \boldsymbol{u})(1 - \boldsymbol{\pi}_k) \frac{1}{f'_{s,Q_k}} \frac{\partial Q_k}{\partial p_k} + \epsilon_0(s) \boldsymbol{\pi}_k(1 - \boldsymbol{\pi}_k) \frac{1}{f'_{s,Q_k}} \frac{\partial Q_k}{\partial p_k},
\end{aligned} \tag{23}$$

where $\epsilon_0(s) = \frac{\partial d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}{\partial \boldsymbol{\pi}_k(s)}$ is a small quantity provided by Lemma 5. Besides, (b) is based on the the definition of the Boltzmann policy and Assumption 3, and (c) is based on Assumption 4 the gradient of the monotonic mixing function in Lemma 1.

Since we have all the preparations ready, we now compute the Lagrangian by applying the KKT condition. We let the Lagrangian gradient to be zero, i.e.,

$$\frac{\partial \mathcal{L}(p_k; \lambda, \nu)}{\partial p_k} = 0 \tag{24}$$

Besides, the partial derivative of the Lagrangian can be computed as:

$$\begin{aligned}
\frac{\partial \mathcal{L}(p_k; \lambda, \nu)}{\partial p_k} &= -\frac{\partial \log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}[\exp(Q_k - Q^*_{k-1})(s, \boldsymbol{u}^*)]}{\partial p_k} - \frac{\partial \log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})}[\exp(Q^*_{k-1} - Q_k)(s, \boldsymbol{u})]}{\partial p_k} + \lambda - \nu_{s,\boldsymbol{u}} \\
&= -\frac{1}{Z} \exp(Q^*_{k-1} - Q_k) \left(\frac{\partial d^{\boldsymbol{\pi}_k}(s, \boldsymbol{u})}{\partial p_k} - d^{\boldsymbol{\pi}_k}(s, \boldsymbol{u}) \frac{\partial Q_k}{\partial p_k} \right) + \lambda - \nu_{s,\boldsymbol{u}},
\end{aligned} \tag{25}$$

where $Z = \mathbb{E}_{s',\boldsymbol{u}' \sim d^{\boldsymbol{\pi}_k}(s,\boldsymbol{u})} \exp(Q^* - Q_k)(s', \boldsymbol{u}')$.

Based on equation (24) and equation (25), and substituting the expression of $\frac{\partial Q_k}{\partial p_k}$ and $\frac{\partial d^{\boldsymbol{\pi}_k}(s,a)}{\partial p_k}$ with the derived results in equation (12) and equation (23), we obtain:

$$\begin{aligned}
p_k(s, \boldsymbol{u}) = & \frac{1}{Z(\nu^*_{s,\boldsymbol{u}} - \lambda^*)} \Bigg[d^{\boldsymbol{\pi}_k}(s, \boldsymbol{u})(Q_k - \mathcal{B}^* Q^*_{k-1}) \exp(Q^*_{k-1} - Q_k) \left(\sum_{j=1}^{n} \frac{1 - \pi^j}{f'_{s,Q^j}} - 1 \right) \\
& + \epsilon_0 \boldsymbol{\pi}_k (Q_k - \mathcal{B}^* Q^*_{k-1}) \exp(Q^*_{k-1} - Q_k) \sum_{j=1}^{n} \frac{1 - \pi^j}{f'_{s,Q^j}} \Bigg],
\end{aligned} \tag{26}$$

According to Lemma 5, the value of ϵ_0 is smaller than $d^{\boldsymbol{\pi}_k}(s)$ so the second term will not influence the sign of the equation. Equation (26) will always be larger or equal to zero. By KKT condition, when the $Q_k - \mathcal{B}^* Q^*_{k-1} < 0$, we have $\nu^*_{s,\boldsymbol{u}} = 0$. When equation (26) equal to zero, we let $\nu^*_{s,\boldsymbol{u}} = 0$ because the value of $\nu^*_{s,\boldsymbol{u}}$ will not affect p_k. In the contrast, when the $Q_k - \mathcal{B}^* Q^*_{k-1} > 0$, the p_k should equal to zero. Therefore, by introducing a normalization factor Z^*, equation (26) can be simplify as follows:

$$p_k(s, \boldsymbol{u}) = \frac{1}{Z^*}(D_k(s, \boldsymbol{u}) + \epsilon_k(s, \boldsymbol{u})), \tag{27}$$

where when $Q_k \leq \mathcal{B}^* Q^*_{k-1}$, we have

$$\begin{aligned} D_k(s, \boldsymbol{u}) &= d^{\boldsymbol{\pi}_k}(s, \boldsymbol{u})(\mathcal{B}^* Q^*_{k-1} - Q_k) \exp(Q^*_{k-1} - Q_k) \left(\sum_{j=1}^{n} \frac{1 - \pi^j}{f'_{s,Q^j}} - 1 \right) \\ \epsilon_k &= \epsilon_0 \boldsymbol{\pi}_k (Q_k - \mathcal{B}^* Q^*_{k-1}) \exp(Q^*_{k-1} - Q_k) \sum_{j=1}^{n} \frac{1 - \pi^j}{f'_{s,Q^j}} \end{aligned} \tag{28}$$

and when $Q_k > \mathcal{B}^* Q^*_{k-1}$, we have

$$D_k(s, \boldsymbol{u}) = 0, \quad \epsilon_k = 0 \tag{29}$$

This concludes the proof. □

F ENVIRONMENTAL DETAILS

We use more recent baselines (i.e., PAC and ResQ [28]) that are known to outperform QTRAN [30] and QPLEX [37] in the evaluation. In general, we tend to choose baselines that are more closely related to our work and most recent. This motivated the choice of QMIX (baseline for value-based factorization methods), WQMIX (close to our work that uses weighted projections so better joint actions can be emphasized), PAC [42], DOP [39] (SOTA actor-critic based methods). We acquired the results of QMIX, WQMIX based on their hyper-parameter tuned versions from pymarl2[9] and implemented our algorithm based on it.

F.1 Predator-Prey

A partially observable environment on a grid-world predator-prey task is used to model relative overgeneralization problem [2] where 8 agents have to catch 8 prey in a 10 × 10 grid. Each agent can either move in one of the 4 compass directions, remain still, or try to catch any adjacent prey. Impossible actions, i.e., moving into an occupied target position or catching when there is no adjacent prey, are treated as unavailable. If two adjacent agents execute the catch action, a prey is caught and both the prey and the catching agents are removed from the grid. An agent's observation is a 5 × 5 sub-grid centered around it, with one channel showing agents and another indicating prey. An episode ends if all agents have been removed or after 200 steps. Capturing a prey is rewarded with r = 10, but unsuccessful attempts by single agents are punished by a negative reward p. In this paper, we consider two sets of experiments with p = (0, -0.5, -1.5, -2). The task is similar to the matrix game proposed by [30] but significantly more complex, both in terms of the optimal policy and in the number of agents.

F.2 SMAC

For the experiments on StarCraft II micromanagement, we follow the setup of SMAC [27] with open-source implementation including QMIX [26], WQMIX [25], QPLEX [37], PAC [42], DOP [39] and ResQ [28]. We consider combat scenarios where the enemy units are controlled by the StarCraft II built-in AI and the friendly units are controlled by the algorithm-trained agent. The possible options for built-in AI difficulties are Very Easy, Easy, Medium, Hard, Very Hard, and Insane, ranging from 0 to 7. We carry out the experiments with ally units controlled by a learning agent while built-in AI controls the enemy units with difficulty = 7 (Insane). Depending on the specific scenarios(maps), the units of the enemy and friendly can be symmetric or asymmetric. At each time step each agent chooses one action from discrete action space, including noop, move[direction], attack[enemy_id], and stop. Dead units can only choose noop action. Killing an enemy unit will result in a reward of 10 while winning by eliminating all enemy units will result in a reward of 200. The global state information is only available in the centralized critic. Each baseline algorithm is trained with 4 random seeds and evaluated every 10k training steps with 32 testing episodes for main results, and with 3 random seeds for ablation results and additional results.

F.3 Implementation details and Hyperparameters

In this section, we introduce the implementation details and hyperparameters we used in the experiment. We carried out the experiments on NVIDIA 2080Ti with fixed hyperparameter settings. Recent work [9] demonstrated that MARL algorithms are significantly influenced by code-level optimization and other tricks, e.g. using TD-lambda, Adam optimizer, and grid-searched hyperparameters (where many state-of-the-art are already adopted), and proposed fine-tuned QMIX and WQMIX, which is demonstrated with significant improvements from their original implementation. We implemented our algorithm based on its open-sourced codebase and acquired the results of QMIX and WQMIX from it.

Table 2: Hyperparameter value settings.

Hyperparameter	Value
Batch size	128
Learning rate	0.001
Replay buffer size	10000
TD-lambda	0.6
Target network update interval	Every 200 episodes

We use one set of hyperparameters for each environment, i.e., no tuned hyperparameters for individual maps. We use epsilon greedy for action selection with annealing from $\epsilon = 0.995$ decreasing to $\epsilon = 0.05$ in 100000 training steps in a linear way. The performance for each algorithm is evaluated for 32 episodes every 1000 training steps. More hyperparameter values are given in Table 2.

G SENSITIVITY EXPERIMENT REGARDING NORMALIZATION

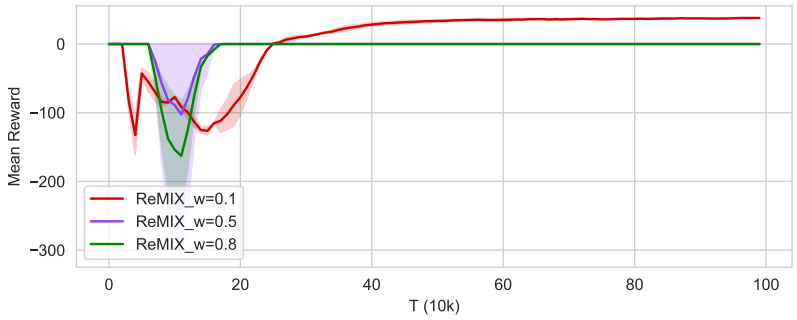


Figure 5: Sensitivity of normalizing the minimum weight to 0.1, 0.5, and 0.8.

We run the experiment in the Predator-Prey environment with a punishment of −1.5 to report the sensitivity with respect to the different normalization of weight ranges. We keep the maximum normalized weight as 1 but test the effects of using different minimums, which are 0.1, 0.5, and 0.8.

As shown in Figure 5, the experiment results are sensitive to the range of the normalized weight. When we map the weight to a minimum of 0.5, the agents in this task can only find a sub-optimal solution. It may be because there exist many overestimations in this task. The joint action representation generated at the is not accurate. Higher minimum weight normalization damages the capability of QPro to adjust the projection to retrieve a precise representation rapidly. Therefore, QPro performs well under 0.1 to 1 normalization of the weight in this scenario. Note in WQMIX, weights are used as $\alpha = 0.1$ for Predator-Prey and $\alpha = 0.5$ for SMAC according to their experiment settings.