Locality Matters: A Scalable Value Decomposition Approach for Cooperative Multi-Agent Reinforcement Learning

Roy Zohar, [1] **Shie Mannor,** [2] **Guy Tennenholtz** [2]
[1] Hebrew University of Jerusalem
[2] Nvidia Research, Technion Institute of Technology
roy.zohar@mail.huji.ac.il, shie@ee.technion.ac.il, guytenn@gmail.com

Abstract

Cooperative multi-agent reinforcement learning (MARL) faces significant scalability issues due to state and action spaces that are exponentially large in the number of agents. As environments grow in size, effective credit assignment becomes increasingly harder and often results in infeasible learning times. Still, in many real-world settings, there exist simplified underlying dynamics that can be leveraged for more scalable solutions. In this work, we exploit such locality structures effectively whilst maintaining global cooperation. We propose a novel, value-based multi-agent algorithm called LOMAQ, which incorporates local rewards in the Centralized Training Decentralized Execution paradigm. Additionally, we provide a direct reward decomposition method for finding these local rewards when only a global signal is provided. We test our method empirically, showing it scales well compared to other methods, significantly improving performance and convergence speed.

1 Introduction

The field of Reinforcement Learning (RL) is concerned with an agent taking actions in an environment in order to maximize a cumulative reward. Recent work has witnessed major success in various tasks, including Atari games (Mnih et al. 2015), and Go (Silver et al. 2016). A popular extension of RL is cooperative multi-agent RL (cooperative MARL), in which a group of agents attempts to interact with an environment together. Research on MARL has gained much attention in recent years, with examples in the Star-Craft multiagent challenge (Vinyals et al. 2019) and traffic control (Chu et al. 2019).

A common paradigm used in cooperative MARL is Centralized Training Decentralised Execution (CTDE, Kraemer and Banerjee (2016)). In this approach, agents are trained simultaneously by a centralized controller. Decentralized policies are then derived from the training process and used for execution. Centralized training can be highly beneficial, granting access to additional global information, which helps agents coordinate their actions. Nevertheless, utilizing such information effectively is a challenging problem for cooperative MARL, due to exponential state and action spaces. As the environment scales, coordination becomes increasingly difficult, rendering centralized training impractical. Still, in many real-world settings, there exist simplified underlying dynamics that can help tackle this problem.

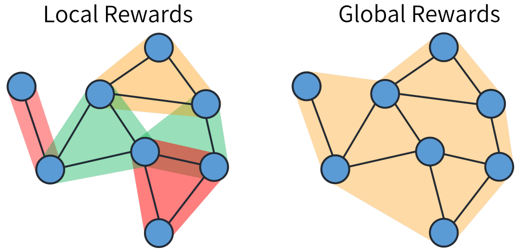


Figure 1: A visualization of training MARL for a graph of agents with local rewards vs. with global rewards. The colored regions represent the feedback that the agents exhibit during training

In this paper, we utilize *local rewards*, a principal component of our work. While local rewards are often used in competitive settings (i.e., where every agent attempts to maximize its own local reward), in most cooperative approaches, cooperation is weakly enforced through a shared global reward that all agents aim to maximize (Rashid et al. 2018; Lowe et al. 2017). A visualization of this paradigm is depicted in Figure 1

Local rewards are critical for effective learning in scalable settings. As an example, consider the problem of coaching a large soccer team. If a certain player loses the ball to the other team, punishing that player directly (and possibly neighboring players) with targeted feedback, may be far more effective than punishing the entire team with general feedback. The latter will often leave players confused, believing they should have acted differently.

Despite the effectiveness of local rewards, naively training with local rewards may result in greedy agents that fail to cooperate. Concurrent approaches that aim to exploit local reward structures for our setting often pay a price in terms of cooperation and usually resort to training with global rewards (Lowe et al. 2017). This is particularly true for the value decomposition approach for cooperative MARL, which has become increasingly popular in recent years (Sunehag et al. 2018; Rashid et al. 2018; Son et al. 2019; Rashid et al. 2020; Wang et al. 2020). To the best of our knowledge, there are no value decomposition methods that utilize local rewards effectively. Rather, they rely on the global reward signal for decomposing the joint state-action value function into individual state-action value functions. As we show in our work, such an approach hurts overall performance and convergence speed in large environments.

In this work, we present a scalable value decomposition method for the cooperative CTDE setting. Our method leverages local agent rewards for improving credit assignment, whilst maintaining a cooperative objective. In addition, we provide a direct decomposition method for finding local rewards when only a global reward is provided. We empirically show that our method is scalable, improving upon state-of-the-art methods for this setting.

Our contributions are as follows. We define the Q-Summation Maximization (QSM) Condition (Section 3.1), showing its theoretical benefits in a linear bandit setting (Theorem 1). We show that a monotonic decomposition of utilities can be derived to establish the QSM condition (Section 3.3), and provide a value-based algorithm to enforce it (Section 4). Finally, we construct a reward decomposition method for learning local rewards when a global reward is given (Section 4.2).

2 Preliminaries

We define a multi-agent Markov decision process (MAMDP) as the tuple $\mathcal{M} = (\mathcal{G}, \mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an undirected graph of agents, where $\mathcal{V} = [n] = \{1, \ldots, n\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, $\mathcal{S} = \bigtimes_{i=1}^{n} \mathcal{S}_i$ is the global state space, $\mathcal{A} = \bigtimes_{i=1}^{n} \mathcal{A}_i$ is the global action space, $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ is the global transition function, $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the global reward, and $\gamma \in (0, 1)$ is the discount factor.

An agent $i \in \mathcal{V}$ is associated with the underlying graph $\mathcal{G}$, state s_i and action a_i. For a set $B \subseteq \mathcal{V}$ we define s_B, a_B as the subset of agent states and actions in B, i.e., $s_B = (s_i)_{i \in B}$ and $a_B = (a_i)_{i \in B}$, respectively. At time t, the environment is at state $s = (s_1, \ldots, s_n)$ and the agents take an action $a = (a_1, \ldots, a_n)$, after which the environment returns a reward r and transitions to state s' according to the factored dynamics $P(s'|s, a) = \prod_{i \in \mathcal{V}} P_i(s'_i | s_{N(i)}, a_i)$, where here we used $N(i)$ to denote the neighborhood of agent i, including i, i.e., $N(i) = \{j \in \mathcal{V} : (i, j) \in \mathcal{E}\} \cup \{i\}$.

We define a global Markovian policy π as a mapping $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ such that $\pi(a|s)$ is the probability to choose action $a = (a_0, \ldots, a_n)$ at state $s = (s_0, \ldots s_n)$. We define the value of policy π starting at a state $s \in \mathcal{S}$ and taking action $a \in \mathcal{A}$ as

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s(t), a(t)) \;\middle|\; s(0) = s, a(0) = a \right].$$

The value function is then defined by $v^\pi(s) = \mathbb{E}_{a \sim \pi(s)}[Q^\pi(s, a)]$. We define the optimal value and optimal policy by $v^*(s) = \max_\pi v^\pi(s)$ and $\pi^* \in \arg\max_\pi v^\pi(s)$, respectively.

Finally, we denote by $\mathcal{P}$ a partition of $\mathcal{V} = [n]$ (i.e., of agents), such that $\bigcup_{J \in \mathcal{P}} J = \mathcal{V}$ and $\bigcap_{J \in \mathcal{P}} J = \emptyset$. We say that $\mathcal{P}'$ is a refinement of $\mathcal{P}$ if for every $J' \in \mathcal{P}'$ there exists $J \in \mathcal{P}$ such that $J' \subseteq J$[1].

2.1 Reward Decomposition

A primary element of MARL is the decomposition of the reward function r over agent states and actions $\{s_i, a_i\}_{i \in \mathcal{V}}$. Given some decomposition of rewards $\{r_i : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}\}_{i \in \mathcal{V}}$, such that $r(s, a) = \sum_{i \in \mathcal{V}} r_i(s, a)$, we define the partial Q-function of π, denoted by $Q_i^\pi(s, a)$ as $Q_i^\pi(s, a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_i(s(t), a(t)) \mid s(0) = s, a(0) = a]$. It follows that $Q^\pi(s, a) = \sum_{i \in \mathcal{V}} Q_i^\pi(s, a)$. Note that such decomposition always exists, e.g., by choosing $r_1 = r, r_i = 0, i \geq 2$.

In this work, we consider a reward decomposition for which every agent is dependent only on its local state and action, as defined formally below. We refer the reader to Section 4.3 for a relaxation of this assumption.

Assumption 1 (Qu et al. (2020)). *We assume that the reward function r is additively decomposable. That is, there exist $\{r_i : \mathcal{S}_i \times \mathcal{A}_i \mapsto \mathbb{R}\}_{i \in \mathcal{V}}$ such that $r(s, a) = \sum_{i=1}^{n} r_i(s_i, a_i)$ for all $s = (s_1, \ldots, s_n)$, $a = (a_1, \ldots, a_n)$.*

3 Value Partitions for MARL

In this section, we focus on leveraging value-based partitions for credit assignment in MARL. We consider decoupling the problem into smaller problems, each of which can be viewed as a separate, easier estimation problem. Particularly, we generalize ideas from Rashid et al. (2018), and define a partition-based Q-maximization condition. We motivate this condition in a contextual bandit setting, proving it can exponentially improve regret. Then, for the general RL setting, we propose a monotonic decomposition of agent utilities for which our proposed condition holds. We show examples of the latter and prove that monotonic decomposition of utilities is indeed sufficient for partition-based maximization. Our decomposition will prove beneficial in Section 4, as we leverage value partitions to construct a scalable value-based algorithm for MARL.

3.1 Q-Summation Maximization (QSM)

We begin by defining the Q-Summation Maximization condition on which we build upon the rest of this section. The QSM condition states that the Q-function can be maximized using a partition of partial maximizers, as defined formally below.

Definition 1 (QSM Condition). *Let $\mathcal{P}$ be a partition of $\mathcal{V}$. We say that a MAMDP satisfies the Q-Summation Maximisation (QSM) Condition with $\mathcal{P}$, if for every $s \in \mathcal{S}$ and policy π*

$$\max_a \left\{ \sum_{i=1}^{n} Q_i^\pi(s, a) \right\} = \sum_{J \in \mathcal{P}} \left(\max_a \left\{ \sum_{i \in J} Q_i^\pi(s, a) \right\} \right)$$

[1] A refinement partition can be useful when multiple groups of agents concurrently attempt to solve relatively separable tasks.

Algorithm 1 Multi-OFUL

```
input: $\alpha, \lambda, \delta > 0$, $\mathcal{P}$ partition of $\mathcal{V}$
init: $V_{J,a_J} = \lambda I$, $J \in \mathcal{P}, a_J \in \bigtimes_{i \in J} \mathcal{A}_i$.
      $Y_J = 0$, $J \in \mathcal{P}$.
for $t = 1, 2, \dots$ do
    Receive context $x(t)$
    for $J \in \mathcal{P}, a_J \in \bigtimes_{i \in J} \mathcal{A}_i$ do
        $\hat{y}_{a_J}(t) = \left\langle x(t), V_{J,a_J}^{-1} Y_J \right\rangle$
        $\text{UCB}_{a_J}(t) = \sqrt{\beta_J(t,\delta)} \|x(t)\|_{V_{J,a_J}^{-1}}$
    end for
    $a(t) \in \bigtimes_{a_J} \arg\max_{J \in \mathcal{P}} \hat{y}_{a_J}(t) + \alpha \text{UCB}_{a_J}(t)$
    Play $a(t)$ and observe $\{r_J(t)\}$
    $V_{J,a_J(t)} = V_{J,a_J(t)} + x(t)x(t)^T$, $J \in \mathcal{P}$
    $Y_J = Y_J + x(t) r_J(t)$, $J \in \mathcal{P}$
end for
```

We note the two extremes of the QSM Condition. First, every MAMDP satisfies the condition trivially with $\mathcal{P} = \{\mathcal{V}\}$. Second, if $\mathcal{P}$ partitions $\mathcal{V}$ into singletons (i.e., $\mathcal{P} = \{\{1\}, \{2\}, \dots, \{n\}\}$), then for every $s \in \mathcal{S}$,

$$\max_a \left\{ \sum_{i=1}^n Q_i^\pi(s,a) \right\} = \sum_{i=1}^n \Big(\max_a \{ Q_i^\pi(s,a) \} \Big).$$

The QSM condition can greatly improve learning efficiency in settings in which the partial Q-functions are easier to approximate, effectively decoupling the problem to $|\mathcal{P}|$ simpler problems. We prove this for an instance of the linear bandits problem in the following subsection. Then, in Section 3.3 we discuss a sufficient assumption for which the QSM condition holds.

3.2 QSM in Linear Bandits

To motivate the QSM condition, we generalize the linear bandit model of Abbasi-Yadkori, Pál, and Szepesvári (2011). Specifically, at each round t, the environment generates a context $x(t) \in \mathcal{X} \subseteq \mathbb{R}^d$ (from a possibly adaptive adversary), where $\|x(t)\|_2 \leq S_x$. The learner must then choose an action $a(t) \in \mathcal{A} = \bigtimes_{i=1}^n \mathcal{A}_i$, where $\mathcal{A}_i = [K]$. Given a partition $\mathcal{P}$ of $\mathcal{V}$, the learner then receives $|\mathcal{P}|$ noisy observations $\Big\{ r_J(t) = \sum_{i \in J} \Big\langle x(t), \theta^*_{i,a_i(t)} \Big\rangle + \eta_J(t) \Big\}_{J \in \mathcal{P}}$, where $\big\{ \theta^*_{i,j} \in \mathbb{R}^d : i \in [n], j \in [K] \big\}$ are unknown vectors, $\big\| \theta^*_{i,j} \big\|_2 \leq S_\theta$, and $\{\eta_J(t)\}_{J \in \mathcal{P}}$ are independent random variables (for every t). We assume $\eta_J(t)$ is conditionally R_J-subgaussian random noise, such that

$$\mathbb{E}\left[e^{\lambda \eta_J(t)} \,\middle|\, a_J(1), \dots, a_J(t), \eta_J(1), \eta_J(t-1) \right] \leq e^{\lambda^2 R_J^2 / 2}.$$

We define the regret at time T by

$$\text{Regret}(T) = \sum_{t=0}^T \sum_{i=1}^n \left[\left\langle x(t), \theta^*_{i,a_i^*(t)} \right\rangle - \left\langle x(t), \theta^*_{i,a_i(t)} \right\rangle \right],$$

where $a^*(t) \in \arg\max_{a \in \mathcal{A}} \sum_{i=1}^n \big\langle x(t), \theta^*_{i,a_i} \big\rangle$.

Algorithm 1 uses the structured partition under which the QSM condition holds. At every iteration of the algorithm, a least square problem is solved for every $J \in \mathcal{P}$, after which an action is chosen according to an upper confidence defined by $\sqrt{\beta_J(t,\delta)} = \lambda^{1/2} |J| S_\theta + R_{\text{max}} \sqrt{d \log\Big(\frac{|\mathcal{P}| K^{|J|} (1 + t S_x)/\lambda}{\delta} \Big)}$.

Denote $K_{\mathcal{P}} = \sum_{J \in \mathcal{P}} K^{|J|}$ and $R_{\text{max}} = \max_{J \in \mathcal{P}} R_J$. Then, we have the following result.

Theorem 1. *Assume $\mathbb{E}[r_J] \in [-1,1]$ for all $J \in \mathcal{P}$. For all $T \geq 0$, with probability at least $1 - \delta$, the regret of Algorithm 1 is bounded by*

$$\textit{Regret}(T) \leq 2\sqrt{T} \sqrt{d \log\left(\lambda + \frac{T S_x^2}{Kd} \right) K_{\mathcal{P}}} \times$$
$$\left(\lambda^{1/2} n S_\theta + R_{\textit{max}} \sqrt{d \log\left(\frac{|\mathcal{P}| K^n (1 + t S_x)/\lambda}{\delta} \right)} \right).$$

This leads to, Regret$(T) \leq \widetilde{\mathcal{O}}\big(d R_{\textit{max}} \sqrt{T K_{\mathcal{P}}} \big)$.

The above result achieves regret that is dependent on the maximum subgassuian constant R_{max} and $\sqrt{K_{\mathcal{P}}}$. This upper bound is significantly lower than the regret of a naive application of the OFUL algorithm in Abbasi-Yadkori, Pál, and Szepesvári (2011). As the latter doesn't assume the QSM condition, it achieves an exponentially larger regret, $\text{Regret}(T) \leq \widetilde{\mathcal{O}}\Big(d R_{\text{tot}} \sqrt{T K^n} \Big)$, where $R_{\text{tot}} = \sum_{J \in \mathcal{P}} R_J$. Indeed, whenever $\max_{J \in \mathcal{P}} |J| \ll n$ Algorithm 1 achieves regret which is exponentially smaller in K. Particularly, when $\mathcal{P} = \{\{1\}, \{2\}, \dots \{n\}\}$, we get that $\text{Regret}(T) \leq \widetilde{\mathcal{O}}\Big(d R_{\text{max}} \sqrt{T n K} \Big)$.

3.3 Monotonic Decomposition of Utilities

In Section 3.1 we defined the QSM condition and showed it can significantly improve performance in a linear bandit setting, suggesting its benefits for cooperative MARL. Still, a question arises, when does the QSM condition hold? In this section, we show a sufficient monotonicity assumption under which the QSM condition holds. We formalize this assumption below.

Assumption 2 (Monotonic Decomposition)**.** *We assume there exists a partition $\mathcal{P}$ of $\mathcal{V}$, utility functions $\{U_i^\pi : S_i \times \mathcal{A}_i \mapsto \mathbb{R}\}_{i \in \mathcal{V}}$, and partition functions $\{F_J^\pi : \mathbb{R}^n \mapsto \mathbb{R}\}_{J \in \mathcal{P}}$ such that for all $J \in \mathcal{P}$,*

$$F_J^\pi(\mathbf{U}(s,a)) = \sum_{i \in J} Q_i^\pi(s,a), \textit{ and}$$
$$\nabla_{\mathbf{U}} F_J^\pi \geq \mathbf{0},$$

where $\mathbf{U}(s,a) := (U_1^\pi(s_1,a_1), \dots, U_n^\pi(s_n,a_n))^T$.

Remark 1. *The monotonic decomposition assumption generalizes to trajectory-dependent utilities $U_i^\pi : \mathcal{T} \mapsto \mathbb{R}$, such that $F_J^\pi(\mathbf{U}(\tau)) = \sum_{i \in J} Q_i^\pi(s,a)$, where s, a are the final state and action in the trajectory τ.*

A basic setting for which Assumption 2 holds is decoupled MAMDPs. Indeed, for any $\mathcal{M} = (\mathcal{G}, \mathcal{S}, \mathcal{A}, P, r, \gamma)$ such that $\mathcal{E} = \emptyset$ and $\mathcal{M}$ is additively decomposable (see Assumption 1), we have that Assumption 2 holds for any partition $\mathcal{P}$. We refer the reader to the appendix for a proof as well as examples of Assumption 2.

3.4 Monotonic Utilities are Sufficient for QSM

Next, we show that monotonic utilities (Assumption 2) are sufficient for the QSM condition. Additionally, we show that, under Assumption 2, local utilities are enough for global Q-maximization. This result is closely related to maximization results in previous work (Son et al. 2019; Rashid et al. 2018). See Appendix for proof.

Theorem 2. *Suppose Assumption 2 holds for some partition $\mathcal{P}$. Then the QSM condition (Definition 1) is satisfied with $\mathcal{P}$. Moreover, for any state $s = (s_1, \dots, s_n) \in \mathcal{S}$, $\arg\max_{a \in \mathcal{A}} Q^\pi(s, a) = \times_{i=1}^{n} \arg\max_{a_i \in \mathcal{A}_i} U_i^\pi(s_i, a_i)$.*

Assumption 2 is a generalization of the monotonicity assumption of Q-mix (Rashid et al. 2018), which holds when $\mathcal{P} = \{\mathcal{V}\}$. In contrast, when $\mathcal{P} = \{\{1\}, \{2\}, \dots, \{n\}\}$, each Q_i can be expressed as a function of $\mathbf{U}(\mathbf{s}, \mathbf{a})$, and is monotonic w.r.t to its inputs. The following proposition shows that, if Assumption 2 holds for $\mathcal{P}'$, a refinement of $\mathcal{P}$, then it holds for $\mathcal{P}$ as well. Particularly, this means that if Assumption 2 holds for any $\mathcal{P}$, then the assumption holds for $\mathcal{P} = \{\mathcal{V}\}$.

Proposition 1. *Let $\mathcal{M} = (\mathcal{G}, \mathcal{S}, \mathcal{A}, P, r, \gamma)$, and let $\mathcal{P}, \mathcal{P}'$ be partitions such that $\mathcal{P}'$ is a refinement of $\mathcal{P}$. If $Assumption$ 2 holds for $\mathcal{P}'$, then it also holds for $\mathcal{P}$.*

The above proposition suggests a certain trade-off between the refinement of $\mathcal{P}$ and the number of MAMDP's that satisfy Assumption 2. Assumption 2 can thus be viewed as a trade-off between expressibility and speed, as controlled by the refinement of $\mathcal{P}$.

In the next section, we build upon Assumption 2 to construct a scalable value-based MARL algorithm that efficiently leverages local value-partitions and local rewards.

4 Local Multi-Agent Q-Learning

In this section, we describe a value-based approach that leverages the QSM condition using an application of Assumption 2. Algorithm 2 provides pseudo-code of our method, which we call LOcal Multi-Agent Q-learning (LOMAQ). We assume local agent rewards are observable during learning (this assumption will be lifted in Section 4.2). Instead of approximating the global Q-function, LOMAQ builds upon Assumption 2 to approximate the partition functions $\{F_J\}_{J \in \mathcal{P}}$.

Algorithm 2 receives as input a partition $\mathcal{P}$ and enforces the monotonicity assumption of $Assumption$ 2. At every iteration of the algorithm, a greedy action is taken w.r.t. each utility. After an action has been selected, $\{F_J\}_{J \in \mathcal{P}}$ are updated using a bellman update for every $J \in \mathcal{P}$. Finally, in line 10, monotonicity is enforced to ensure Assumption 2 holds. After training is complete, we use the learned utilities U_i for decentralized execution. We note that, due to Theorem 2, choosing the greedy action in line 6 w.r.t. the local utilities is equivalent to acting greedily w.r.t. the global Q-function. We refer the reader to the appendix for a discussion regarding the convergence of Algorithm 2.

Algorithm 2 LOMAQ with local rewards

```
 1: Input: Partition $\mathcal{P}$ of $\mathcal{V}$, exploration parameter $\epsilon$
 2: Init: $F_J(\{\mathbf{U}(s', a')) = 0$, for all $J \in \mathcal{P}$
 3: for $t = 1, 2 \dots$ do
 4:     Take action $a$
 5:     Observe $s'$ and local rewards $\{r_J\}_{J \in \mathcal{P}}$
 6:     $a'_{\text{greedy}} \in \left(\arg\max_{a'_i} U_i(s'_i, a'_i)\right)_{i \in \mathcal{V}}$
 7:     $a' \leftarrow \begin{cases} \text{random action} & \text{, w.p. } \epsilon \\ a'_{\text{greedy}} & \text{, w.p. } 1 - \epsilon \end{cases}$
 8:     for $J \in \mathcal{P}$ do
 9:         $F_J(\mathbf{U}(s, a)) \overset{\alpha_t}{\leftarrow} r_J(s, a) + \gamma F_J(\mathbf{U}(s', a'))$
10:         Project $F_J$ to the set $\{f : \mathbb{R}^n \mapsto \mathbb{R} \text{ s.t. } \nabla f \geq 0\}$
11:     end for
12: end for
```

4.1 Practical Implementation of LOMAQ

We implement LOMAQ in a deep Q-learning framework (Rashid et al. 2018). Specifically, we approximate F_J^π for every $J \in \mathcal{P}$, and U_i^π for every $i \in \mathcal{V}$ using neural networks with parameters θ. We denote these approximations by F_J^θ and U_i^θ, respectively. The outputs of U_i^θ are forwarded as inputs into F_J^θ, i.e., $F_J^\theta(\{U_i^\theta\}_{i=1}^n)$.

Given a mini-batch of tuples (s, a, r, s') sampled from a replay memory, we train the neural networks end-to-end by minimizing the loss

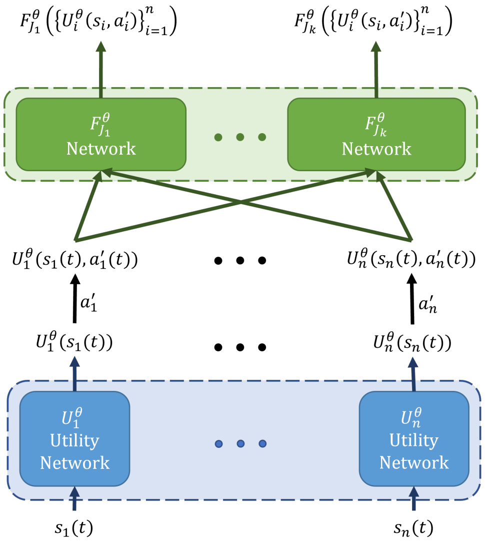
$$L_F(\theta) = \mathbb{E}_{s,a,s'}\left[\sum_{J \in \mathcal{P}} \left(y_J - F_J^\theta(\{U_i^\theta(s_i, a_i)\}_{i=1}^n)\right)^2\right], \tag{1}$$

where, $y_J = \sum_{j \in J} r_j + \gamma \max_{a'}\{F_J^\theta(\{U_i^\theta(s'_i, a'_i)\}_{i=1}^n)\}$.

Figure 2 depicts the feed-forward architecture for LOMAQ. The local agents states s_i are fed into U_i^θ, which outputs a vector of size $\mathcal{A}_i$, representing the utility of every state-action pair (s_i, a_i). The utilities of the chosen actions a'_i are then forwarded as inputs into $F_J^\theta(\{U_i^\theta(s_i, a'_i)\}_{i=1}^n)$. Finally, the outputs of F_J^θ are trained according to $L_F(\theta)$ in Equation (1).

In practice, every agent $i \in \mathcal{V}$ views a trajectory of local states, represented by τ_i. We use recurrent networks for estimating U_i^θ, and fully-connected networks for F_J^θ. We utilize the graph structure for approximating F_J, by redirecting U_i into F_J only if there exists a $j \in J$ such that $i \in N(j)$. We refer the reader to the appendix for an exhaustive overview of specific implementation details.

Monotonic Regularization To enforce the monotonicity criterion of Assumption 2, we implement line 10 of Algorithm 2 by regularizing the loss in Equation (1). We propose



$F^{\theta}_{J_1}\left(\{U^{\theta}_i(s_i,a'_i)\}^n_{i=1}\right)$
$F^{\theta}_{J_k}\left(\{U^{\theta}_i(s_i,a'_i)\}^n_{i=1}\right)$
$F^{\theta}_{J_1}$ Network
$F^{\theta}_{J_k}$ Network
$U^{\theta}_1(s_1(t),a'_1(t))$
$U^{\theta}_n(s_n(t),a'_n(t))$
a'_1
a'_n
$U^{\theta}_1(s_1(t))$
$U^{\theta}_n(s_n(t))$
U^{θ}_1 Utility Network
U^{θ}_n Utility Network
$s_1(t)$
$s_n(t)$

Figure 2: The architecture for the LOMAQ network. The agent's states s_i are fed into the utility networks U_i^{θ}, which are then forwarded as inputs into F_J^{θ}.

two such regularizations; namely, using hard and soft projection regularizers.

For hard regularization, we project all parameters θ to be positive through a Relu activation function, i.e., $\theta \leftarrow \text{Relu}(\theta)$ for all θ corresponding to F_J^{θ}. Alternatively, to allow for softer regularization, we penalize Equation (1) by the negative derivatives of F_J^{θ} w.r.t. U_i^{θ} for every $J \in \mathcal{P}$. That is, $L(\theta) = L_F(\theta) + \lambda \mathcal{R}_{reg}(\theta)$, where $\lambda > 0$, and

$$\mathcal{R}_{reg}(\theta) = \sum_{J \in \mathcal{P}} \text{Relu}(-\nabla_{\mathbf{U}} F_J^{\theta}).$$

Here, the regularization parameter λ reflects a trade-off between efficiency (due to QSM) and accuracy (whenever Assumption 2 does not hold exactly).

4.2 Global Reward

While LOMAQ relies on observable local rewards for estimating F_J^{π}, they may not always be provided. In this section, we propose a new method for decomposing the global reward function into local reward functions, whenever these are not available.

We assume the global reward signal can be approximately additively decomposed (see Assumption 1). We approximate each local reward $r_i(s_i, a_i)$ using a deep neural network with parameters ϕ. Our prediction for the global reward is then given by $r^{\phi}_{\text{pred}}(s,a) = \sum_{i=1}^{n} r_i^{\phi}(s_i, a_i)$, which is trained to match the global reward signal r_{global}, by minimizing the loss

$$L_r(\phi) = \mathbb{E}_{s,a}\left[\left(r^{\phi}_{\text{pred}}(s,a) - r_{\text{global}}(s,a)\right)^2\right]. \tag{2}$$

Training $r_i^{\phi}(s_i, a_i)$ is done in parallel to LOMAQ, where $(s, a, r_{\text{global}})$ are sampled from a replay memory. We refer the reader to the appendix for an exhaustive overview and further implementation details.

4.3 Beyond Additive Decomposition

In certain settings, Assumption 1 may be too restrictive, e.g. when interactions between agents are exhibited in the global reward signal. To overcome this, we consider an alternative decomposition of the reward, where every learned reward function can be dependent on a *group* of agents.

Formally, for any $i \in \mathcal{V}$ we denote by $\mathcal{I}(i)$ the power set of agents in $\{i\} \cup N(i)$. That is,

$$\mathcal{I}(i) = \{I : I \text{ is in the power set of } \{i\} \cup N(i)\}.$$

Next, for every set $I \in \mathcal{I}(i)$ we define a reward function relating to the agents in I, $r_I : \mathcal{S}_I \times \mathcal{A}_I \mapsto \mathbb{R}$. Finally, we define the reward of agent $i \in \mathcal{V}$ by

$$r_i(s,a) = \sum_{I \in \mathcal{I}(i)} \frac{1}{|I|} r_I(s_I, a_I), \tag{3}$$

where here, every reward r_I in the summand is normalized according to the cardinality of I. Notice that this decomposition is a generalization of Assumption 1. Indeed, Equation (3) coincides with Assumption 1 whenever $\mathcal{E} = \emptyset$.

The reward decomposition in Equation (3) creates a hierarchy for every agent i, as every local reward r_i is comprised of multiple learned reward functions $\{r_I(s_I, a_I)\}_{I \in \mathcal{I}(i)}$ which have less effect on agent i as $|I|$ increases.

In most cases, the number of local reward functions is exponential in $N(i)$, rendering large decompositions infeasible. Moreover, as $|I|$ increases, the learned rewards become dependent on more agents, reducing their effectiveness (due to normalization in $|I|$). We therefore focus on learning reward functions of small cardinality in $|I|$. We enforce this in practice using a regularization term that is dependent on $|I|$. Specifically, we regularize the loss in Equation (2) by

$$\mathcal{R}_{\text{reg}}(\phi) = \sum_{I \in \mathcal{I}} w(|I|) \times |r_I^{\phi}(s_I, a_I)|,$$

where $w(|I|)$ are weights that grow proportionally to $|I|$, penalizing $r_I^{\phi}(s_I, a_I)$ as $|I|$ increases. This regularization reflects a trade-off between the overall accuracy of the learned rewards and the complexity of the reward decomposition.

5 Experiments

In this section we test the performance of LOMAQ and compare it to previous MARL approaches on two large-scale multi agent tasks.

5.1 Environments

We tested our algorithm on two environments, Coupled-Multi-Cart-Pole and Bounded-Cooperative-Navigation. Both environments include minor modifications of the well-known Cart-Pole (Brockman et al. 2016) and Cooperative-Navigation (Lowe et al. 2017) environments.

The Coupled-Multi-Cart-Pole consists of n cartpoles, residing on the 1d axis. Each cart is viewed as an agent, controlled by applying a force of ±1. Every pair of neighboring

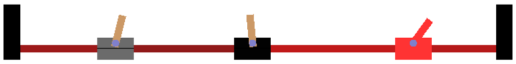


Figure 3: The Coupled-Multi-Cart-Pole environment with 3 cartpoles. The right-most cartpole has fallen (marked in red). The global reward for this timestep is +2.

carts is connected by a spring. Every cart receives a local reward of $+1$ for every timestep that the pole is upright. The global reward for the environment is the total number of cartpoles that are currently upright. The dependency graph for this environment can be modeled as a line graph, where every cartpole has two neighbors excluding the edges which only have one. Figure 3 depicts this environment for three cartpoles.

The Bounded-Cooperative-Navigation consists of n agents (particles) and n landmarks. Agents must strive to cooperatively cover as many landmarks as possible. In this environment, particles aren't able to move freely in 2d space. Every particle is bound to a fixed distance from its starting position and is thereby restricted to a certain region. This restriction resembles a simplified food-delivery service, where the landmarks represent customers and the agents represent delivery people. Consequently, not all particles interact with each other directly, since direct interactions only occur when two particles are in the same location. This induces a dependency graph for which every two particles are neighbors in the graph if and only if their regions overlap. The environment rewards $+1$ for every landmark that is covered by a particle at a certain timestep. Figure 4 shows a conceptual visualization of the task.

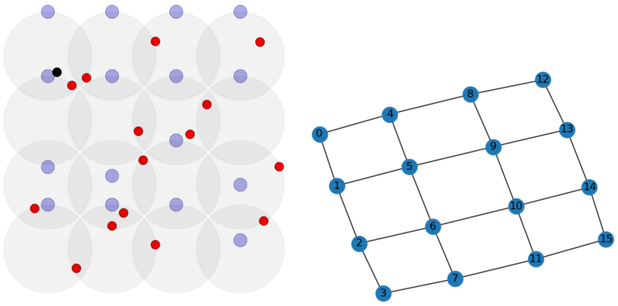
5.2 Comparative Experiments

Scalability We tested LOMAQ on both cooperative environments with $n = 15$ agents in two setups; namely, with and without access to a local reward signal. We denote these by LOMAQ and LOMAQ+RD, respectively. In both cases, we used the refined partition $\mathcal{P} = \{\{1\}, \{2\}, \ldots, \{n\}\}$.

We compared LOMAQ to a wide range of contemporary cooperative methods. In addition, we compared LOMAQ to two versions of IQL (Tan 1993), trained with environment local rewards and global rewards, which we denote by IQL-local and IQL, respectively.

Figure 5 depicts the results of the Coupled-Multi-Cart-Pole and Bounded-Cooperative-Navigation environments. It is evident that both versions of LOMAQ significantly outperform all of the compared methods in both performance and convergence speed. We note that LOMAQ+RD converges to LOMAQ's policy, with a slight delay due to the time taken to learn the reward decomposition.

In both environments various cooperative methods exhibit slow learning compared to LOMAQ, due to the use of global rewards. Additionally, while IQL-local does learn quickly (primarily due to the use of local rewards), it converged to a sub-optimal solution. This occurs as IQL acts greedily w.r.t its local rewards. In contrast, LOMAQ incentivizes cooperation, enabling both fast convergence as well as improved performance.



(a) Particles in purple, landmarks in red, regions in gray

(b) Interaction graph based on region overlap

Figure 4: The Bounded-Cooperative-Navigation environment with 16 agents and circular regions.

Reward Decomposition We visualize multiple reward decompositions for Bounded-Cooperative-Navigation. We run our decomposition method with a global reward signal, for $n = 2$ agents and a single landmark. If both agents are on the landmark at the same time, the global reward remains 1. We plot the learned reward functions as a function of Agent 1 and Agent 2's distance from the landmark, which we denote by Δx. These results are depicted in Figure 6.

The first row in Figure 6 assumes a decomposition according to Assumption 1. Assumption 1 does not hold for this setup, since the reward function is dependent on both agents when they share a landmark. The approximated reward is overly optimistic and wrongly rewards $+2$ when the landmark is shared. The second row approximates a decomposition that allows $|I| \leq 2$ with no regularization $\lambda = 0$. In this case, the global reward is approximated correctly, and the local reward functions $r^{\phi}_{\{i\}} = 0$. The third row visualizes a decomposition with regularization $w(|I| = 2) = 1, \lambda = 0.0001$. In this case, the local reward functions $r^{\phi}_{\{i\}}$ convey information for each agent i, and $r^{\phi}_{\{1,2\}}$ conveys information regarding their joint dynamic.

6 Related Work

Graph Based MARL. The underlying structure of the team of agents in the environment can often be modeled using a graph topology. Jiang et al. (2019) propose DGN - a MARL algorithm based on the graph convolutional network (GCN) architecture which assumes centralized execution and homogeneous agents. Naderializadeh et al. (2020) propose GraphMIX for CTDE, that uses global rewards for learning. Qu et al. (2020) propose Scalable Actor-Critic - An Actor-Critic approach for the discrete space case which utilizes a dependency graph with theoretical guarantees.

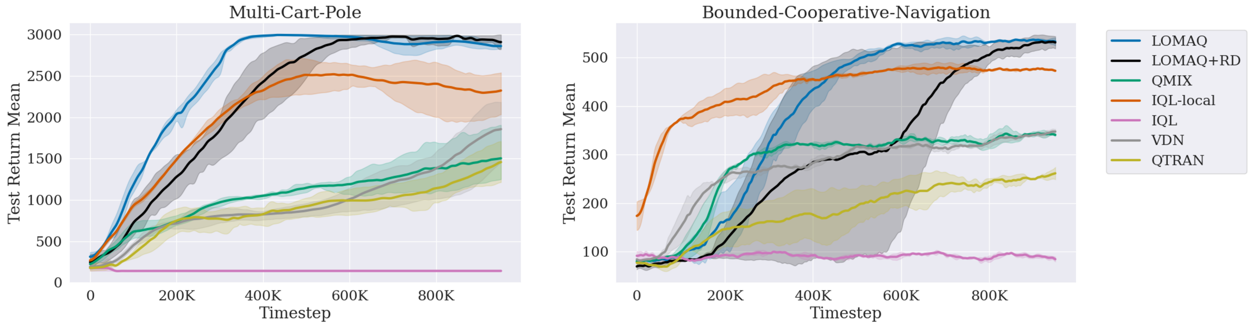


Figure 5: Test returns for the Coupled-Multi-Cart-Pole environment and for the Bounded-Cooperative-Navigation environment.

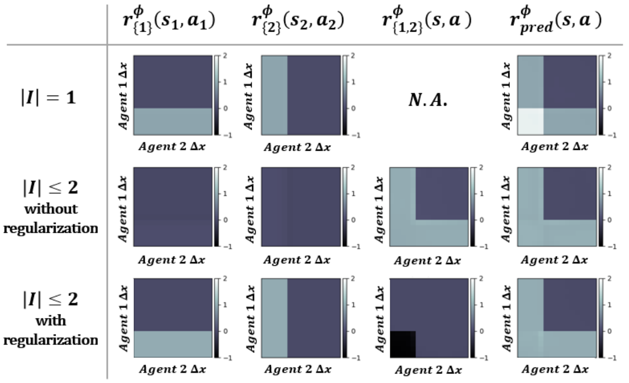


Figure 6: Visualization of learned reward functions r^{ϕ}_{I} for different decompositions in Bounded-Cooperative-Navigation for $n = 2$ agents and a single landmark. We plot the learned reward functions as a function of Agent 1 and Agent 2's distance from the landmark, denoted by Δx. The first row assumes $|I| \leq 1$, the second row assumes $|I| \leq 2$ with no regularization, and the third row adds regularization.

Cooperative MARL. Our approach enhances the popular value decomposition family (Son et al. 2019; Wang et al. 2020; Rashid et al. 2020), which consider a cooperative multi-agent problem in which each agent observes its own state and action history. Sunehag et al. (2018) propose VDN for decomposing the value function into a sum of utility functions. Rashid et al. (2018) offer Q-mix which generalizes this concept, by decomposing the value function into a monotonic function of individual utility functions. All of these approaches implicitly measure the impact of every agent on the observed global reward, whereas we propose to combine this line of work with an explicit approach for credit assignment using local rewards.

Credit Assignment. Various approaches have attempted to tackle the credit assignment problem. A common approach for credit assignment is by estimating the individual Q functions Q_i directly, which are often substantially simpler and significantly easier to learn than Q (Qu et al. 2020; Kok and Vlassis 2004; Russell and Zimdars 2003; van Seijen et al. 2017; Juozapaitis et al. 2019). Our work extends this line of work for value-based CTDE, and focuses on reward decompositions that expedite learning alongside global cooperation.

Reward Decomposition. Multiple works recognize the benefits of local rewards and attempt to learn them in settings where only a global reward signal is provided. RD^2 (Lin et al. 2020b) learns a reward decomposition with minimally-dependent features for factored-state MDP setting. Our method can be seen as an extension of RD^2 for MARL, where the action is also factored.

Large Action Spaces. Finally, our work is related to work on large and combinatorial action spaces. From action elimination (Zahavy et al. 2018), to action embeddings (Tennenholtz and Mannor 2019; Chandak et al. 2019), through action redundancy (Baram, Tennenholtz, and Mannor 2021), our work can be viewed as an additional method for reducing the effective dimensionality of the problem.

7 Conclusion and Future Work

In this work we tackled the credit assignment problem of cooperative MARL through local, partition based value functions. We used the QSM condition and a monotonic decomposition of utilities to construct a value-based approach, effectively reducing the problem to simpler ones. We showed that local rewards are highly beneficial, both when provided as well as learned implicitly from a global reward. These greatly improved overall performance and convergence speed, suggesting that local structures can be efficiently used to improve MARL algorithms.

In this work we have assumed that an underlying, static dependency graph $\mathcal{G}$ is provided during training. In many cases, these assumptions are limiting. We look to further generalize our method by learning such dynamic dependencies between agents through interaction with the environment. In addition, our work has assumed that $Assumption\ 2$ holds for some partition $\mathcal{P}$ and local reward decomposition $\{r_i\}$. We look to generalize our algorithm to automatically identify effective decompositions.

References

Abbasi-Yadkori, Y.; Pál, D.; and Szepesvári, C. 2011. Improved algorithms for linear stochastic bandits. *Advances in neural information processing systems* 24: 2312–2320.

Baram, N.; Tennenholtz, G.; and Mannor, S. 2021. Action Redundancy in Reinforcement Learning. *arXiv preprint arXiv:2102.11329* .

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym.

Chandak, Y.; Theocharous, G.; Kostas, J.; Jordan, S.; and Thomas, P. 2019. Learning action representations for reinforcement learning. In *International Conference on Machine Learning*, 941–950. PMLR.

Chu, T.; Wang, J.; Codecà, L.; and Li, Z. 2019. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems* 21(3): 1086–1095.

Jiang, J.; Dun, C.; Huang, T.; and Lu, Z. 2019. Graph Convolutional Reinforcement Learning. In *International Conference on Learning Representations*.

Juozapaitis, Z.; Koul, A.; Fern, A.; Erwig, M.; and Doshi-Velez, F. 2019. Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI Workshop on Explainable Artificial Intelligence*.

Kok, J. R.; and Vlassis, N. 2004. Sparse cooperative Q-learning. In *Proceedings of the twenty-first international conference on Machine learning*, 61.

Kraemer, L.; and Banerjee, B. 2016. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing* 190: 82–94.

Lin, Y.; Qu, G.; Huang, L.; and Wierman, A. 2020a. Multi-agent reinforcement learning in time-varying networked systems. *arXiv preprint arXiv:2006.06555* .

Lin, Z.; Yang, D.; Zhao, L.; Qin, T.; Yang, G.; and Liu, T.-Y. 2020b. RD2: Reward Decomposition with Representation Decomposition. *Advances in Neural Information Processing Systems* 33.

Lowe, R.; WU, Y.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; and Mordatch, I. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *Advances in Neural Information Processing Systems* 30: 6379–6390.

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

Naderializadeh, N.; Hung, F. H.; Soleyman, S.; and Khosla, D. 2020. Graph Convolutional Value Decomposition in Multi-Agent Reinforcement Learning. *CoRR* abs/2010.04740. URL https://arxiv.org/abs/2010.04740.

Qu, G.; Lin, Y.; Wierman, A.; and Li, N. 2020. Scalable Multi-Agent Reinforcement Learning for Networked Systems with Average Reward. In *Thirty-fourth Conference on Neural Information Processing Systems*.

Rashid, T.; Farquhar, G.; Peng, B.; and Whiteson, S. 2020. Weighted QMIX: Expanding Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. *Advances in Neural Information Processing Systems* 33.

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

Russell, S. J.; and Zimdars, A. 2003. Q-decomposition for reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 656–663.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529(7587): 484–489.

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

Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W. M.; Zambaldi, V. F.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J. Z.; Tuyls, K.; et al. 2018. Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward. In *AAMAS*.

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

Tennenholtz, G.; and Mannor, S. 2019. The natural language of actions. In *International Conference on Machine Learning*, 6196–6205. PMLR.

van Seijen, H.; Fatemi, M.; Romoff, J.; Laroche, R.; Barnes, T.; and Tsang, J. 2017. Hybrid reward architecture for reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 5398–5408.

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

Wang, J.; Ren, Z.; Liu, T.; Yu, Y.; and Zhang, C. 2020. QPLEX: Duplex Dueling Multi-Agent Q-Learning. In *International Conference on Learning Representations*.

Zahavy, T.; Haroush, M.; Merlis, N.; Mankowitz, D. J.; and Mannor, S. 2018. Learn What Not to Learn: Action Elimination with Deep Reinforcement Learning. *Advances in Neural Information Processing Systems* 31: 3562–3573.

Appendix A: Monotonic Decomposition

This section further analyzes the properties of the monotonic decomposition that we introduced in Section 3.3. We introduce a few additional results that (proofs can be found in Appendix D), discuss the limitations of Assumption 2 with a bandit example, and formally justify the approximation of truncated F_J mentioned in Section 4.

Additional Results

First, the following proposition claims that under Assumption 2, the global Q function is also monotonic w.r.t the utility functions. This is a useful and important result that we use in multiple proofs.

Proposition 2. *[Monotonicity of the global Q function] Suppose Assumption 2 holds for some partition $\mathcal{P}$. Then the global Q function $Q^\pi(s, a)$ can be written as a function of utilities $\mathbf{U}(\mathbf{s}, \mathbf{a})$ and is monotonic w.r.t every utility*

In Section 3.3 we claimed that Assumption 2 holds for decoupled MAMDPs. The following proposition formalizes this claim.

Proposition 3. *Let $\mathcal{M} = (\mathcal{G}, \mathcal{S}, \mathcal{A}, P, r, \gamma)$ such that $\mathcal{E} = \emptyset$ and $\mathcal{M}$ is additively decomposable by Assumption 1. Then Assumption 2 holds for any partition $\mathcal{P}$*

In this work, we have considered reward decompositions that are additively decomposable, such that $r(s, a) = \sum_{i=1}^{n} r_i(s_i, a_i)$. This assumption is limiting in many cases, and can be generalized in many ways, for instance with β_2 as done in (Lin et al. 2020a).

In practice, even if local rewards depend on a small group of agents, they still expedite LOMAQ substantially. The following proposition, shows that the refinement of $\mathcal{P}$ can be deepened by modifying the reward decomposition. Note that the mentioned reward decompositions are dependent on the *global* state-action $r_i(s, a)$.

Proposition 4. *Let $\mathcal{M} = (\mathcal{G}, \mathcal{S}, \mathcal{A}, P, r, \gamma)$ such that $Assumption$ 2 holds from some partition $\mathcal{P}$ and reward decomposition $\{r_i(s, a)\}_{i=1}^{n}$. Then, for any refinement $\mathcal{P}'$ of $\mathcal{P}$ there exists a reward decomposition $\{r'_i(s, a)\}_{i=1}^{n}$ such that $\sum_{i=1}^{n} r'_i(s, a) = \sum_{i=1}^{n} r_i(s, a)$ and $Assumption$ 2 holds with $\mathcal{P}'$.*

In the extreme case where $\mathcal{P} = \{\mathcal{V}\}$ and $\mathcal{P}' = \{\{1\}, \{2\}, \ldots, \{n\}\}$, the proposed reward decomposition assumes the global reward function for every agent: $r'_i(s, a) = \frac{1}{n} \sum_{j \in \mathcal{V}} r_j(s, a)$.

Limitations of Monotonic Utilities

In this section, we formalize an example that outlines some limitations of Assumption 2. Figure 7 depicts a bandit setting example (with $\gamma = 0$) for which Assumption 2 only holds for certain partitions. Note that the reward decomposition in Figure 7 does not satisfy Assumption 1 purposely, since bandit settings that satisfy Assumption 1, satisfy Assumption 2 as an immediate corollary of Proposition 3.

For the payoff matrices in Figure 7, Assumption 2 holds for $\mathcal{P} = \{\{1, 2\}\}$ by setting $U_i^\pi(s_i, a_i) = a_i$, and $Q^\pi(U_1^\pi, U_2^\pi) = F^\pi_{\{\{1,2\}\}} = 2 + \max\{0, U_1^\pi + U_2^\pi - 1\}$ which is monotonic.

Moreover, there exist $F^\pi_{\{1\}}$, $F^\pi_{\{2\}}$ that are monotonic w.r.t *different* utility functions, e.g.,

$$U_i^\pi(s_i, a_i) = \begin{cases} a_i & \text{for} F^\pi_{\{1\}} \\ 1 - a_i & \text{for} F^\pi_{\{2\}} \end{cases}$$

such that

$$F^\pi_{\{1\}} = U_1^\pi + U_2^\pi$$

$$F^\pi_{\{2\}} = 1 + \max\{0, U_1^\pi + U_2^\pi - 1\}.$$

Which are both monotonic. Still, it can be shown that Assumption 2 does not hold for $\mathcal{P} = \{\{1\}, \{2\}\}$. To see this, assume by contradiction that Assumption 2 holds for $\mathcal{P} = \{\{1\}, \{2\}\}$. Let $s \in S$ denote some global state, and $a_x, a_y \in \mathcal{A}$ denote global actions such that $a_x = (a_1, a_2) = (0, 0)$, and $a_y = (a_1, a_2) = (1, 0)$.

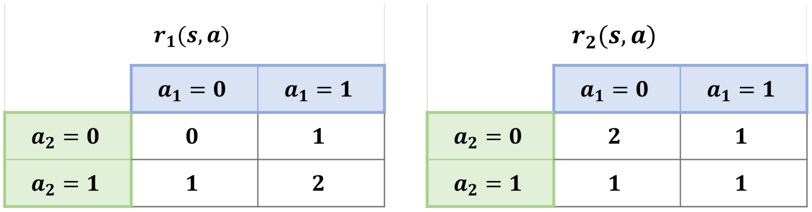
By definition $F^\pi_{\{1\}} = Q_1(s, a)$. Therefore, $F^\pi_{\{1\}}(s, a_x) = 0$, and $F^\pi_{\{1\}}(s, a_y) = 1$. Note that $U_2^\pi(s_2, (a_y)_2) = U_2^\pi(s_2, 0) = U_2^\pi(s_2, (a_x)_2)$, since U_2^π is not dependent on a_1. Therefore, we can write:

$$F^\pi_{\{1\}}(s, a_x) = F^\pi_{\{1\}}(U_1^\pi(s_1, 1), U_2^\pi(s_2, 0)) = 0$$

$$F^\pi_{\{1\}}(s, a_y) = F^\pi_{\{1\}}(U_1^\pi(s_1, 0), U_2^\pi(s_2, 0)) = 1$$

Fixing $U_2^\pi(s_2, 0)$, $F^\pi_{\{1\}}$ is monotonically non-decreasing w.r.t to U_1^π due to Assumption 2. Since $F^\pi_{\{1\}}(s, a_y) > F^\pi_{\{1\}}(s, a_x)$, then $U_1^\pi(s_1, 0) > U_1^\pi(s_1, 1)$.

Conversely, since $F^\pi_{\{2\}}(s, a_x) > F^\pi_{\{2\}}(s, a_y)$ and since $F^\pi_{\{2\}}$ is also monotonically non-decreasing w.r.t to U_1^π, we get that $U_1^\pi(s_1, 0) < U_1^\pi(s_1, 1)$. This of course in contradiction to $U_1^\pi(s_1, 0) > U_1^\pi(s_1, 1)$.



$r_1(s,a)$

	$a_1 = 0$	$a_1 = 1$
$a_2 = 0$	0	1
$a_2 = 1$	1	2

$r_2(s,a)$

	$a_1 = 0$	$a_1 = 1$
$a_2 = 0$	2	1
$a_2 = 1$	1	1

Figure 7: Payoff matrices with 2 local reward functions. Q_1 and Q_2 are monotonic w.r.t to different utility functions.

Truncated Value Decomposition

In this subsection, we expand upon our approximated decomposition scheme, which leverages the specific topological properties of the underlying agent graph $\mathcal{G}$. While Assumption 2 defines F_J as functions of *global* states and actions, by using a graph-dependent additive reward decomposition (see Section 2.1), this assumption can be simplified to *local* states and actions, characterized by neighboring agents.

First, we define the κ-hop neighborhood of agent $i \in \mathcal{V}$ $\mathcal{N}_i^\kappa$, as the set of all agents that are at most κ edges from i: $\mathcal{N}_i^\kappa = \{j \in \mathcal{V} | \text{dist}(i,j) \leq \kappa\}$. Note that for $\kappa = 1$ we get our original neighborhood definition $N(i)$ from Section 2.1.

Theoretically, every F_J is somewhat dependent on the entire global action space (s, a). However, agents that are distant from agent i in $\mathcal{G}$, have a negligible effect on agent i. This result is especially important in terms of credit assignment - by truncating distant agents, credit assignment becomes substantially easier and increases the effectiveness of each local reward.

(Qu et al. 2020) study this effect, and define truncated Q-functions $\tilde{Q}_i^\pi(s_{N_i^\kappa}, a_{N_i^\kappa})$ that are only dependent on a local neighborhood N_i^κ. We extend this result for our case be defining truncated F_J^π, such that $\tilde{F}_J^\pi = \sum_{i \in J} \tilde{Q}_i^\pi(s_{N_i^\kappa}, a_{N_i^\kappa})$. The following proposition describes the accuracy of this approximation.

Proposition 5. *[Truncated Decomposition] If the reward function is additively decomposable, and the exponential decay property (Qu et al. 2020) holds with (c, ρ), then for every $J \in \mathcal{P}$, and $(s, a) \in \mathcal{S} \times \mathcal{A}$,*

$$|F_J^\pi - \tilde{F}_J^\pi| \leq |J| c \rho^{\kappa+1}$$

Appendix B describes the full use of the truncated $\tilde{F}_J^\pi$ in LOMAQ.

Appendix B: Experiments

This section depicts further details regarding our experimentation. We give further explanations regarding LOMAQ and our reward decomposition method, depict the full algorithms and the used hyper-parameters for both methods, and provide a couple of further ablations for LOMAQ.

All experiments in the main paper were run with 3 randomly selected seeds. Every 10,000 time-steps, we tested all algorithms in a decentralized manner on 20 episodes with $\epsilon = 0$ and recorded the test return mean. This metric is suitable since some methods implement global mixing layers that defy decentralized execution. We averaged the test return mean for every seed, and plotted the result. We also added the minimal and maximal value for every test in a translucent color. All experiments were run on various Linux machines with varying architectures (including both CPU and GPU GeForce RTX 2080 Ti) and memory constraints. We refer the reader to the README.md file in the code that describes the full procedure for replicating our results. We did a coarse independent sweep for every hyper-parameter.

Additional Implementation Details

LOMAQ Similar to previous work (Mnih et al. 2015), we store two sets of parameters to stabilize the training of LOMAQ: θ for the current network parameters, and θ^-, as target parameters that are used for future estimates. The target parameters are updated at a slower time scale. Additionally, we implement parameter sharing between networks. If agents are homogeneous, we share parameters between U_i^θ.

Rashid et al. (2018) propose using *hyper-networks* that depend on the global state for generating non-negative weights for Q. LOMAQ also allows the use of hyper-networks for every F_J^θ, however since (1) the number of hyper-networks can be very large ($O(|J|)$) and (2) for large settings only a small portion of the global state is relevant, our implementation uses hard regularization that is independent of the global state. We leave testing LOMAQ with hyper-networks and global ”sub-states” as future work.

As shown in Proposition 5, it is often useful to approximate F_J^θ with a local neighborhood $\mathcal{N}_i^\kappa$. In practice, we redirect U_j^θ into F_J^θ, if and only if there exist i, j such that $i \in J$ and $j \in \mathcal{N}_i^\kappa$, where κ is a hyper-parameter that controls the accuracy of approximation F_J^θ. We have also implemented this redirection using a GCN (Graph Convolutional Network), where the number of convolution layers is κ, which yielded similar results. This concept is similar to the mixing layer in Graph-mix (Naderializadeh et al. 2020).

Reward Decomposition We chose a non-recurring architecture for approximating r_I^ϕ, since each decomposition should be independent of other time-steps. If the reward is independent of agent id, we assume parameter sharing between classes of r_I^ϕ with equal $|I|$.

We note that the approximated r_I^ϕ are only used when r_{pred}^ϕ achieves approximate convergence to r_{global}. That is, we only use r_I^ϕ if $\left|r_{\text{pred}}^\phi(s,a) - r_{\text{global}}\right| \leq \Delta$ where Δ is a tolerance hyper-parameter. If the inequality doesn't hold, we disregard r_I^ϕ.

We have also implemented a classification variant of our decomposition method that assumes that for every $I \in \mathcal{I}$, and $(s,a) \in \mathcal{S} \times \mathcal{A}$, $r_I(s_I, a_I) \in K_I$, where K_I is some small group of possible reward values. This method is exponential in n, but converged very well for small n's. Nevertheless, all experiments use the regression variant of our decomposition method, depicted in Section 4.2.

Full Algorithms

LOMAQ Algorithm 3 includes the full training scheme for LOMAQ, outlined in Algorithm 2. The full algorithm consists of 4 main parts (1) Storing trajectories in the replay buffer (2) Training the reward decomposition if we are running LOMAQ-RD (3) Updating our estimation for the Q-values and (4) Enforcing monotonicity. The algorithm references Algorithm 4 and Algorithm 5, that describe our reward decomposition method and are presented in the next section.

Reward Decomposition Algorithm 4 and Algorithm 5 depict our reward decomposition method in full. Algorithm 4 describes the training procedure of every r_I^ϕ using a global reward signal, and Algorithm 5 describes the prediction stage, where $r_I^\phi(s_I, a_I)$ are inferred and translated into local agent rewards as seen in Section 4.3.

Hyper-Parameters

Table 1 depicts the hyper-parameters used in LOMAQ and Table 2 depicts additional hyper-parameters used in our reward decomposition method for LOMAQ-RD.

Additional Experiments

Representability We evaluate the representability of LOMAQ on the matrix game described in Figure 7 for different partitions. We set $\gamma = 0$ for simplicity, such that $Q_i(s,a) = r_i(s,a)$, and $\epsilon = 1$ for testing representability of the entire state-action space. The results can be seen in Figure 8. As expected LOMAQ converged for $P = \{\{1,2\}\}$ but not for $P = \{\{1\},\{2\}\}$.

Algorithm 3 Full algorithm for training LOMAQ / LOMAQ+RD

```
Require: MAMDP $\mathcal{M}$, partition $\mathcal{P}$, hyperparameters $\kappa$, $\epsilon$, $\alpha$, $L$
Initialize replay memory D
Initialize $[F_J^\theta]$, $[U_i^\theta]$ with random parameters $\theta$
Redirect $[U_i^\theta]$ outputs into $F_J^\theta$ only if there exists a $j \in J$ such that $i \in \mathcal{N}_j^\kappa$ according to $\mathcal{G}$
Initialize target parameters $\theta^- = \theta$
for episode $= 1 \dots$ do
    // Sample and Store a trajectory
    Observe initial state $s(0)$
    for $t = 0, \dots$ do
        $a'_{\text{greedy}} \in \left(\arg\max_{a'_i} U_i(\tau'_i(t), a'_i)\right)_{i\in\mathcal{V}}$
        $a'(t) \leftarrow \begin{cases} \text{random action} & \text{, w.p. } \epsilon \\ a'_{\text{greedy}} & \text{, w.p. } 1-\epsilon \end{cases}$
        Take action $a'(t)$ and retrieve next observation $s(t+1)$ and reward $r(t)$
        if LOMA$Q$+RD and $r(t)$ is global then
            $r(t) \leftarrow$ Algorithm 5
        end if
        Store transition $(\tau(t), a(t), r(t), \tau(t+1))$ in D
    end for

    // Train Decomposer
    if LOMA$Q$+RD then Algorithm 4
    end if

    // Train Agents
    Sample a random mini-batch $B$ of transitions $(\tau, a, r, \tau')$ from D
    Update $\theta$ by minimizing the loss using learning rate $\alpha$ whilst enforcing monotonicity

    $$L_F(\theta) = \sum_{(\tau,a,r,\tau')\in B}\left(\sum_{J\in\mathcal{P}}\left(y_J - F_J^\theta(\{U_i^\theta(s_i,a_i)\}_{i=1}^n)\right)^2\right),$$

    $$y_J = \sum_{j\in J} r_j + \gamma\left(F_J^{\theta^-}\left(\left\{\max_{a'_i}\left\{U_i^{\theta^-}(\tau'_i,a'_i)\right\}\right\}_{i=1}^n\right)\right)$$

    // Enforce Monotonicity
    if Hard regularization then
        Enforce Monotonicity by $\theta \leftarrow$ Relu$(\theta)$
    else
        $L_F(\theta) \leftarrow L_F(\theta) + \lambda\mathcal{R}_{reg}(\theta)$, where $\mathcal{R}_{reg}(\theta) = \sum_{J\in\mathcal{P}}$ Relu$(-\nabla_{\mathbf{U}} F_J^\theta)$
    end if

    Anneal $\epsilon$
    Update target network parameters $\theta^- \leftarrow \theta$ with period $L$
end for
```

Nevertheless, we note that even when $P = \{\{1\},\{2\}\}$, although the values of Q_2^θ are wrong, the values of Q_1^θ are approximately correct, and the global Q^θ resembles the correct Q. We believe this is due to the fact that Q_1 induces larger TD-errors, that have a larger effect on the utilities.

Truncated Approximation κ We test our algorithm by changing κ, that control the accuracy of approximation of every F_J^θ on Multi-Cart-Pole. These results can be seen in Figure 9. As can be seen in the figure, convergence is slightly slower as κ grows, and the converged policy is slightly better. We believe that for environments that are coupled in a stronger manner, this effect will be more dominant in both convergence speed and performance.

Algorithm 4 Training Reward Decomposition

```
Require: MAMDP $\mathcal{M}$, Replay memory D, hyperparameters $\mathcal{I}$, $\alpha_2$, $\lambda$
if not initialized then
    Initialize $[r_I]_{I \in \mathcal{I}}$ with random parameters $\phi$
end if
Sample a random mini-batch $B$ of transitions $(s, a, r_{\text{global}})$ from D
$r^{\phi}_{\text{pred}}(s, a) \leftarrow \sum_{I \in \mathcal{I}} r^{\phi}_I(s_I, a_I)$
Update $\theta$ by minimizing the loss with learning rate $\alpha_2$

    $$L_r(\theta) = \sum_{(s,a,r_{\text{global}}) \in B} \left( r^{\phi}_{\text{pred}}(s, a) - r_{\text{global}(s,a)} \right)^2 + \lambda \mathcal{R}_{\text{reg}}(\phi)$$

    $$\mathcal{R}_{\text{reg}}(\phi) = \sum_{(s,a,r_{\text{global}}) \in B} \left( \sum_{I \in \mathcal{I}} w(|I|) \times |r^{\phi}_I(s_I, a_I)| \right)$$
```

Algorithm 5 Inferring Reward Decomposition

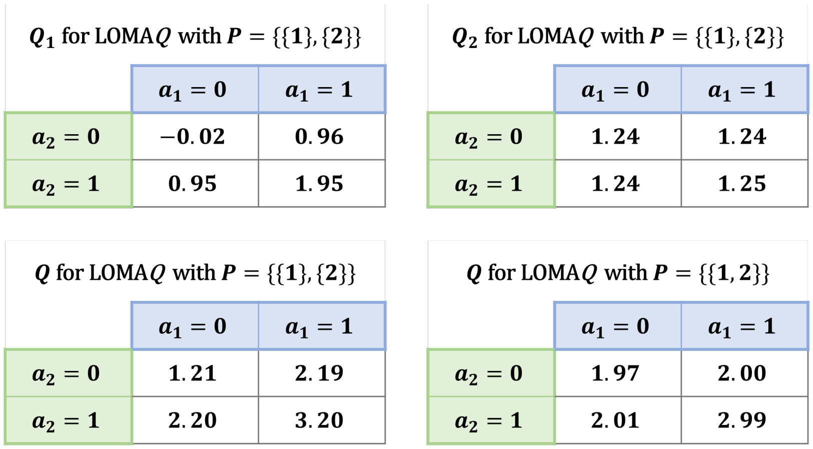
```
Require: $(s, a, r_{\text{global}})$, Trained Networks $\{r_I(\theta)\}_{I \in \mathcal{I}}$, hyper-parameter $\Delta$
$r_{\mathcal{I}} \leftarrow \{r_I(s_I, a_I; \theta)\}_{I \in \mathcal{I}}$
$r_{\text{pred}} \leftarrow \sum_{r_I \in r_{\mathcal{I}}} r_{\mathcal{I}}$
if $|r_{\text{pred}} - r_{\text{global}}| \geq \Delta$ then
    return Decomposition has failed
end if
$r_i \leftarrow \left[ \sum_{r_I \in r_{\mathcal{I}}} \frac{1}{|I|} r_I \right]_{i=1}^{n}$
return $r_i$
```

Hyper-parameter	**Values**	**Description**
$\mathcal{P}$	$\mathcal{P} = \{\{1\}, \{2\}, \dots, \{n\}\}$	Partition used in LOMAQ
ϵ	Linear anneal from 1.0 to 0.05 for 100K timesteps	Parameter used for ϵ-greedy action selection
γ	0.99	Used in Bellman updates
L Target update interval	50 episodes	Frequency of switching between θ and θ^{-}
batch size	50	Batch of trajectories sampled from replay memory
learning rate α	0.0005	Used with RMSProp optimizer. Also used $\alpha_{\text{RMS}} = 0.99$, $\epsilon_{\text{RMS}} = 0.00001$
κ	1	Radius of neighborhood taken for estimating F_J. Equal to $N(i)$ in this case.
Monotonicity Method	Hard	We used hard regularization, $\theta \leftarrow \text{Relu}(\theta)$
Layers in U^{θ}_i	Linear, Relu, GRU, Linear	Input shape is obs size, 64 for hidden dim in GRU, output shape is N$\underline{\text{o}}$ actions
Parameter Sharing in U^{θ}_i	all	Both environments include homogeneous agents
Layers in F^{θ}_J	Linear, elu, Linear, elu, Linear	Input shape for F^{θ}_J is $\lvert \bigcup_{j \in J} N(j) \rvert$, 32 for hidden dim, output shape is 1
Parameter Sharing in F^{θ}_J	Multi-Cart-Pole: $J \in \{\{2\}, \{3\}, \dots, \{13\}\}$, Bounded-Cooperative-Navigation: $J \in \{\{3\}, \{4\}, \dots, \{12\}\}$,	Since both environments exhibit symmetry, we share parameters between inner F^{θ}_J since mixing should be similar between them.

Table 1: Hyper-parameters used for LOMAQ in both environments

Hyper-parameter	Values	Description
$\mathcal{I}$	$\mathcal{I} = \{\{1\}, \{2\}, \dots, \{n\}\}$	We only decomposed the reward according to Assumption 1 in LOMAQ+RD. In Bounded-Cooperative-Navigation, we allowed every agent to observe how many agents are currently with it on the landmark, therefore making the reward additively decomposable by Assumption 1. This also effectively means $\lambda = 0$
Δ	n * 0.1 = 1.5	Parameter used for disregarding unsuccessful decompositions
batch size	5	Batch of trajectories sampled from replay memory for reward decomposition
learning rate α_2	0.01	Used with Adam optimizer with default settings.
Layers in r_I^ϕ	Linear, Relu, Linear, Leaky Relu, Linear, Tanh, Linear	Input shape is obs size * $\lvert I \rvert$, 64 for first hidden dim, 128 for second hidden dim, output shape is 1
Parameter Sharing in r_I^ϕ	all	Both environments include homogeneous agents, and $\lvert I \rvert \leq 1$

Table 2: Hyper-parameters used for reward decomposition



Q_1 for LOMAQ with $P = \{\{1\}, \{2\}\}$

	$a_1 = 0$	$a_1 = 1$
$a_2 = 0$	-0.02	0.96
$a_2 = 1$	0.95	1.95

Q_2 for LOMAQ with $P = \{\{1\}, \{2\}\}$

	$a_1 = 0$	$a_1 = 1$
$a_2 = 0$	1.24	1.24
$a_2 = 1$	1.24	1.25

Q for LOMAQ with $P = \{\{1\}, \{2\}\}$

	$a_1 = 0$	$a_1 = 1$
$a_2 = 0$	1.21	2.19
$a_2 = 1$	2.20	3.20

Q for LOMAQ with $P = \{\{1, 2\}\}$

	$a_1 = 0$	$a_1 = 1$
$a_2 = 0$	1.97	2.00
$a_2 = 1$	2.01	2.99

Figure 8: Learned Q values for $P_1 = \{\{1, 2\}\}$, but not for $P_2 = \{\{1\}, \{2\}\}$

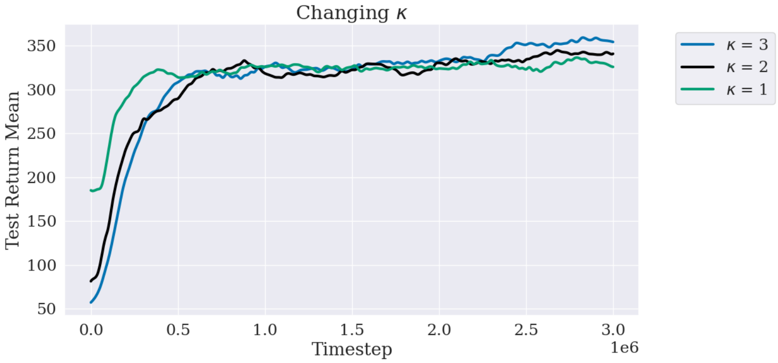


Figure 9: learned Q values changing κ

Appendix C: Discussion

This section presents a few further discussions regarding our work. We further compare our work to existing methods, present the differences between our methods for enforcing monotonicity in LOMAQ, and discuss the convergence of LOMAQ.

Relation to other works

Q-mix (Rashid et al. 2018) LOMAQ can be seen as a generalization of Q-mix that incorporates local rewards. The training process of LOMAQ can be described as training $|\mathcal{P}|$ instances of Q-mix, that are trained on shared utility functions. For $\mathcal{P} = \{\mathcal{V}\}$, LOMA$Q$'s architecture is equivalent to Q-mix, since $F_J = F_{\mathcal{V}} = Q$.

Our soft regularization monotonicity constraint can replace the hyper-network scheme proposed in (Rashid et al. 2018), and offers an alternative, soft approach for satisfying the IGM condition.

VDN (Sunehag et al. 2018) Secondly, our reward decomposition method resembles the VDN architecture for the supervised case. This choice is very suitable, since both the VDN decomposition and our reward decomposition are additive.

Comparison of monotonicity methods

In this section, we note 2 important differences between our hard and soft projection regularizers. For brevity, we denote these variations LOMAQ-h and LOMAQ-s.

First, LOMAQ-h ***must*** train and enforce monotonicity according to the ***same partition*** $\mathcal{P}$. Note however that LOMAQ-s permits us to approximate F_J where $J \in \mathcal{P}$, whilst enforcing Assumption 2 according to $\mathcal{P}'$, where $\mathcal{P}$ is only required to be a ***refinement*** of $\mathcal{P}'$, by enforcing monotonicity for sums of F_J where $J \in \mathcal{P}$.

This feature can be beneficial if one wishes to estimate Q-values according to a refined $\mathcal{P}$ (i.e: explainability (Juozapaitis et al. 2019)), for problems that only satisfy $Assumption$ 2 with a coarse $\mathcal{P}'$. Note that in this case, LOMAQ will probably not see a large increase in training speed, since $\mathcal{P}'$ is coarse.

Another important difference is that while optimizing LOMAQ-s, our optimizer can explore non-monotonic parameterizations θ since we enforce monotonicity softly, whereas LOMAQ-h enforces a hard monotonicity constraint that allows the only optimization of monotonic parameterizations.

Convergence of LOMAQ

In this section, we discuss details regarding the convergence of LOMAQ. (Juozapaitis et al. 2019) formalize dr$Q$, and offer convergence guarantees:

- dr$Q$ is guaranteed to converge to the global optimal $\sum_c Q_c \mapsto Q^*$.
- For every c, $Q_c \mapsto Q_c^*$

Note that in Algorithm 1 from (Juozapaitis et al. 2019), the action $a' = a(t+1)$ is chosen w.r.t to the global Q function: $a' = \arg\max_a \sum_{i=1}^{n} Q_{c_J}(s,a)$. This is a vital element for convergence to the global optimum. Similarly, the corresponding line in LOMAQ (Algorithm 2, Line 6), maximizes over the utility functions, and due to Theorem 2 this is equivalent to maximizing the global Q function directly.

Therefore, LOMAQ differs from dr$Q$ solely due to the monotonicity enforcement at every update. Due to these enforcements, the theoretical convergence results do not carry over directly and require a stronger argument. We leave the rigorous proof of convergence for LOMAQ as future work.

Appendix D: Missing Proofs

This section includes all of the missing proofs from the paper. This section is divided into 2 main parts, (1) proofs regarding monotonic decomposition and (2) proofs regarding linear bandits.

7.1 Proofs for Monotonic Decomposition

Proposition 2. *[Monotonicity of the global Q function] Suppose Assumption 2 holds for some partition $\mathcal{P}$. Then the global Q function $Q^\pi(s,a)$ can be written as a function of utilities $\mathbf{U}(\mathbf{s},\mathbf{a})$ and is monotonic w.r.t every utility*

Proof. Since $Q^\pi(s,a) = \sum_{i=1}^n Q_i^\pi(s,a) = \sum_{J\in\mathcal{P}} F_J(\mathbf{U}(s,a))$, we can denote $Q^\pi(\mathbf{U}(s,a)) = Q^\pi(s,a)$.

Secondly, we have that Q^π is monotonic with respect to every $U_i^\pi \in \mathbf{U}$, as

$$\forall i \in \mathcal{V} : \frac{\partial Q^\pi}{\partial U_i^\pi} = \frac{\partial}{\partial U_i^\pi}\left(\sum_{J\in\mathcal{P}}\sum_{j\in\mathcal{J}} Q_j^\pi\right) = \frac{\partial}{\partial U_i^\pi}\left(\sum_{J\in\mathcal{P}} F_J^\pi\right) = \sum_{J\in\mathcal{P}}\left(\frac{\partial F_J^\pi}{\partial U_i^\pi}\right) \geq 0$$

□

Theorem 2. *Suppose Assumption 2 holds for some partition $\mathcal{P}$. Then the QSM condition (Definition 1) is satisfied with $\mathcal{P}$. Moreover, for any state $s = (s_1,\dots,s_n) \in \mathcal{S}$, $\arg\max_{a\in\mathcal{A}} Q^\pi(s,a) = \times_{i=1}^n \arg\max_{a_i\in\mathcal{A}_i} U_i^\pi(s_i,a_i)$.*

Proof. We will start by proving that the QSM condition holds. Let $s \in \mathcal{S}$ be some global state, and π be some global Markovian policy. For every $i \in \mathcal{V}$, let u_i^* denote the maximal value for every U_i^π i.e. $u_i^* = \max_{a_i}\{U_i^\pi(s_i,a_i)\}$.

Due to the monotonicity of $\{F_J^\pi\}_{J\in\mathcal{P}}$ and Q^π w.r.t to their inputs $\mathbf{U}(\mathbf{s},\mathbf{a})$ (Proposition 2), we can maximize each of these functions by maximizing $\mathbf{U}(\mathbf{s},\mathbf{a})$. That is:

$$\max_a \{Q^\pi(\mathbf{U}(s,a))\} = Q^\pi(\{\max_{a_i}\{U_i^\pi(s_i,a_i)\}\}_{i\in\mathcal{V}}), \tag{4}$$

and that:

$$\max_a \{F_J^\pi(\mathbf{U}(s,a))\} = F_J^\pi(\{\max_{a_i}\{U_i^\pi(s_i,a_i)\}\}_{i\in\mathcal{V}})$$

So overall we get that

$$\begin{aligned}
\max_a\left\{\sum_{i=1}^N Q_i^\pi(s,a)\right\} &= \max_a\{Q^\pi(s,a)\} \\
&= \max_a\{Q^\pi(\mathbf{U}(s,a))\} \\
&= Q^\pi(\{\max_{a_i}\{U_i^\pi(s_i,a_i)\}\}_{i\in\mathcal{V}}) \\
&= Q^\pi(\{u_i^*\}_{i\in\mathcal{V}}) \\
&= \sum_{J\in\mathcal{P}}\left(\sum_{j\in J} Q_j^\pi(\{u_i^*\}_{i\in\mathcal{V}})\right) \\
&= \sum_{J\in\mathcal{P}}\left(\max_a\left\{\sum_{j\in J} Q_j^\pi(s,a)\right\}\right).
\end{aligned}$$

Next, we show that $\arg\max_{a\in\mathcal{A}} Q^\pi(s,a) = \times_{i=1}^n \arg\max_{a_i\in\mathcal{A}_i} U_i^\pi(s_i,a_i)$. This condition closely resembles the IGM Condition from (Son et al. 2019; Rashid et al. 2018). This is straightforward due to Equation (4), completing the proof. □

Proposition 1. *Let $\mathcal{M} = (\mathcal{G},\mathcal{S},\mathcal{A},P,r,\gamma)$, and let $\mathcal{P},\mathcal{P}'$ be partitions such that $\mathcal{P}'$ is a refinement of $\mathcal{P}$. If $Assumption$ 2 holds for $\mathcal{P}'$, then it also holds for $\mathcal{P}$.*

Proof. Let $\mathcal{P}'$ be a refinement of $\mathcal{P}$. That is, for every $J' \in \mathcal{P}'$, there exists a $J \in \mathcal{P}$ such that $J' \subseteq J$. Let $\{F_{J'}^\pi : \mathbb{R}^n \mapsto \mathbb{R}\}_{J'\in\mathcal{P}'}$, $\{U_i^\pi : S_i \times \mathcal{A}_i \mapsto \mathbb{R}\}_{i\in\mathcal{V}}$ be functions that satisfy Assumption 2 for $\mathcal{P}'$.

We define new functions $\{F_J^\pi : \mathbb{R}^n \mapsto \mathbb{R}\}_{J\in\mathcal{P}}$ as follows:

$$\forall J \in \mathcal{P} : F^{\pi}_{J}(\mathbf{U}(s,a)) := \sum_{J' \in \mathcal{P}' | J' \in J} F^{\pi}_{J'}(\mathbf{U}(s,a))$$

Note that Assumption 2 is satisfied for $\mathcal{P}$, with the newly defined $\{F^{\pi}_{J}\}_{J \in \mathcal{P}}$ and the original utilities $\{U^{\pi}_{i}\}_{i \in \mathcal{V}}$, since

$$\forall J \in \mathcal{P} : \nabla_{\mathbf{U}} F^{\pi}_{J} = \nabla_{\mathbf{U}} \left(\sum_{J' \in \mathcal{P}' | J' \in J} F^{\pi}_{J'} \right) = \sum_{J' \in \mathcal{P}' | J' \in J} \nabla_{\mathbf{U}} F^{\pi}_{J'} \geq \mathbf{0}.$$

□

Proposition 3. *Let $\mathcal{M} = (\mathcal{G}, \mathcal{S}, \mathcal{A}, P, r, \gamma)$ such that $\mathcal{E} = \emptyset$ and $\mathcal{M}$ is additively decomposable by Assumption 1. Then Assumption 2 holds for any partition $\mathcal{P}$*

Proof. Due to Proposition 1, it suffices to show that Assumption 2 holds for $\mathcal{P} = \{\{1\}, \{2\}, \ldots, \{n\}\}$. Since $\mathcal{E} = \emptyset$, the global transition can be written as:

$$P(s'|s,a) = \prod_{i \in \mathcal{V}} P_i(s'_i | s_{N(i)}, a_i), = \prod_{i \in \mathcal{V}} P_i(s'_i | s_i, a_i),$$

This allows us to decouple the $\mathcal{M}$ into $|n|$ completely independent MDPs. For every $i \in \mathcal{V}$, M_i is defined with state space S_i, action space A_i, reward function $r_i(s_i, a_i)$ and transtion $P_i(s'_i | s_i, a_i)$.

Due to this decoupling, we can write:

$$\forall i \in \mathcal{V} : Q^{\pi}_{i}(s,a) = Q^{\pi}_{i}(s_i, a_i)$$

We can then define $\{F^{\pi}_{J}\}_{J \in \mathcal{P}}$, $\{U^{\pi}_{i}\}_{i \in \mathcal{V}}$ for the partition $\mathcal{P} = \{\{1\}, \{2\}, \ldots, \{n\}\}$:

$$\forall i \in \mathcal{V} : U^{\pi}_{i}(s_i, a_i) := Q^{\pi}_{i}(s_i, a_i)$$

$$\forall J \in \mathcal{P} : F^{\pi}_{J}(\mathbf{U}(s,a)) = F^{\pi}_{\{i\}}(\mathbf{U}(s,a)) = Q^{\pi}_{i}(s,a) = Q^{\pi}_{i}(s_i, a_i) = U^{\pi}_{i}(s_i, a_i)$$

And therefore for every $J \in \mathcal{P}$, $\nabla_{\mathbf{U}} F^{\pi}_{J} = \nabla_{\mathbf{U}} U^{\pi}_{i}(s_i, a_i) = \mathbf{e_i} \geq \mathbf{0}$

□

Proposition 4. *Let $\mathcal{M} = (\mathcal{G}, \mathcal{S}, \mathcal{A}, P, r, \gamma)$ such that Assumption 2 holds from some partition $\mathcal{P}$ and reward decomposition $\{r_i(s,a)\}_{i=1}^{n}$. Then, for any refinement $\mathcal{P}'$ of $\mathcal{P}$ there exists a reward decomposition $\{r'_i(s,a)\}_{i=1}^{n}$ such that $\sum_{i=1}^{n} r'_i(s,a) = \sum_{i=1}^{n} r_i(s,a)$ and Assumption 2 holds with $\mathcal{P}'$.*

Proof. Assume that Assumption 2 holds for partition $\mathcal{P}$ with $\{F^{\pi}_{J}\}_{J \in \mathcal{P}}$, $\{U^{\pi}_{i}\}_{i \in \mathcal{V}}$, and let $\mathcal{P}'$ be a refinement of $\mathcal{P}$. We denote $J'(i)$ as the parent set $J' \in \mathcal{P}'$ of $i \in \mathcal{V}$. Due to $\mathcal{P}'$ being a partition of $\mathcal{V}$, $J'(i)$ exists and is singular. Similarly, for every $J' \in \mathcal{P}'$ we denote $A(J')$ as the set in $\mathcal{P}$ that containts J'. Since $\mathcal{P}'$ is a refinement of $\mathcal{P}$ $A(J')$ exists and is singular.

We define a new reward decomposition $\{r'_i(s,a)\}_{i=1}^{n}$ for $\mathcal{P}'$ as follows:

$$\forall i \in \mathcal{V} : r'_i(s,a) = \frac{1}{|A(J'(i))|} \sum_{j \in A(J'(i))} r_j(s,a)$$

Note that for this new decomposition

$$\begin{aligned}
\sum_{i=1}^{n} r'_i(s,a) &= \sum_{i=1}^{n}\left(\frac{1}{|A(J'(i))|}\sum_{j\in A(J'(i))} r_j(s,a)\right)\\
&= \sum_{J'\in\mathcal{P}'}\left(\frac{|J'|}{|A(J')|}\sum_{j\in A(J')} r_j(s,a)\right)\\
&= \sum_{J\in\mathcal{P}}\left(\sum_{J'\in\mathcal{P}'|J'\subseteq J}\left(\frac{|J'|}{|J|}\sum_{j\in J} r_j(s,a)\right)\right)\\
&= \sum_{J\in\mathcal{P}}\left(\frac{1}{|J|}\left(\sum_{J'\in\mathcal{P}'|J'\subseteq J}|J'|\right)\left(\sum_{j\in J} r_j(s,a)\right)\right)\\
&= \sum_{J\in\mathcal{P}}\left(\sum_{j\in J} r_j(s,a)\right)\\
&= \sum_{i=1}^{n} r_i(s,a)
\end{aligned}$$

For this new decomposition, also note that

$$\begin{aligned}
{Q'}_i^{\pi}(s,a) = &= \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r'_i(s(t),a(t)) \,\middle|\, s(0)=s, a(0)=a\right]\\
&= \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t\left(\frac{1}{|A(J'(i))|}\sum_{j\in A(J'(i))} r_j(s,a)\right) \,\middle|\, s(0)=s, a(0)=a\right]\\
&= \frac{1}{|A(J'(i))|}\sum_{j\in A(J'(i))}\left(\mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_j(s(t),a(t)) \,\middle|\, s(0)=s, a(0)=a\right]\right)\\
&= \frac{1}{|A(J'(i))|}\sum_{j\in A(J'(i))} Q_j^{\pi}(s,a)
\end{aligned}$$

We can now define $\left\{{F'}_{J'}^{\pi}\right\}_{J'\in\mathcal{P}'}$, $\left\{{U'}_i^{\pi}\right\}_{i\in\mathcal{V}}$ for the new reward decomposition. We set the utilities to be identical to the utilities from before ${U'}_i^{\pi} = U_i^{\pi}$. Therefore:

$$\begin{aligned}
\forall J'\in\mathcal{P}' : {F'}_{J'}^{\pi}(\mathbf{U}(s,a)) &= \sum_{i\in J'} {Q'}_i^{\pi}(s,a)\\
&= \sum_{i\in J'}\frac{1}{|A(J'(i))|}\sum_{j\in A(J'(i))} Q_j^{\pi}(s,a)\\
&= \sum_{i\in J'}\frac{1}{|A(J')|}\sum_{j\in A(J')} Q_j^{\pi}(s,a)\\
&= \frac{|J'|}{|A(J')|}\sum_{j\in A(J')} Q_j^{\pi}(s,a)\\
&= \frac{|J'|}{|A(J')|}F_{A(J')}
\end{aligned}$$

Which therefore means:

$$\nabla_{\mathbf{U}}{F'}_{J'}^{\pi} = \nabla_{\mathbf{U}}\left(\frac{|J'|}{|A(J')|}F_{A(J')}\right) = \frac{|J'|}{|A(J')|}\nabla_{\mathbf{U}}F_{A(J')} \geq \mathbf{0}$$

Therefore satisfying Assumption 2 with partition $\mathcal{P}'$ and reward decomposition $\{r'_i(s,a)\}_{i=1}^n$.

□

Proposition 5. *[Truncated Decomposition] If the reward function is additively decomposable, and the exponential decay property (Qu et al. 2020) holds with (c,ρ), then for every $J \in \mathcal{P}$, and $(s,a) \in \mathcal{S} \times \mathcal{A}$,*

$$|F_J^\pi - \tilde{F}_J^\pi| \leq |J| c \rho^{\kappa+1}$$

Proof. Assume that the exponential decay property holds with (c,ρ), (Qu et al. 2020) showed that the truncated Q-functions prove to be a good approximation to the partial $Q_i^\pi(s,a)$:

$$\forall (s,a) \in S \times A : |Q_i^\pi(s,a) - \tilde{Q}_i^\pi(s_{N_i^\kappa}, a_{N_i^\kappa})| \leq c\rho^{\kappa+1}$$

This results immediately carries over for our F_J^π. We define the truncated F_J^π as $\tilde{F}_J^\pi = \sum_{j \in J} \tilde{Q}_i^\pi(s_{N_i^\kappa}, a_{N_i^\kappa})$. Therefore for every $(s,a) \in S \times A$:

$$\begin{aligned}
|F_J^\pi(s,a) - \tilde{F}_J^\pi| &= \left| \sum_{j \in J} Q_i^\pi(s,a) - \sum_{j \in J} \tilde{Q}_i^\pi(s_{N_i^\kappa}, a_{N_i^\kappa}) \right| \\
&= \left| \sum_{j \in J} \Big(Q_i^\pi(s,a) - \tilde{Q}_i^\pi(s_{N_i^\kappa}, a_{N_i^\kappa}) \Big) \right| \\
&\leq \sum_{j \in J} \Big| Q_i^\pi(s,a) - \tilde{Q}_i^\pi(s_{N_i^\kappa}, a_{N_i^\kappa}) \Big| \\
&\leq |J| c \rho^{\kappa+1}
\end{aligned}$$

□

Proofs for Linear Bandits

Theorem 1. *Assume $\mathbb{E}[r_J] \in [-1,1]$ for all $J \in \mathcal{P}$. For all $T \geq 0$, with probability at least $1-\delta$, the regret of Algorithm 1 is bounded by*

$$\textit{Regret}(T) \leq 2\sqrt{T} \sqrt{d \log\left(\lambda + \frac{TS_x^2}{Kd}\right) K_{\mathcal{P}}} \times \\ \left(\lambda^{1/2} n S_\theta + R_{\textit{max}} \sqrt{d \log\left(\frac{|\mathcal{P}| K^n (1 + tS_x)/\lambda}{\delta} \right)} \right).$$

This leads to, Regret$(T) \leq \widetilde{\mathcal{O}}\big(dR_{\textit{max}}\sqrt{TK_{\mathcal{P}}}\big)$.

Proof. Denote $\theta^*_{a_J} = \sum_{i \in J} \theta^*_{i,a_i}$ and let $\hat{\theta}_{a_J}(t) = V_{J,a_J}(t)^{-1} Y_J(t)$ be the least squares estimator at time step t, where $V_{J,a_J}(t) = \lambda I + \sum_{k=1}^{t-1} \mathbf{1}_{\{a_J(k) = a_J\}} x(k) x(k)^T$. Let $\sqrt{\beta_J(t,\delta)} = \lambda^{1/2} |J| S_\theta + R_{\max} \sqrt{d \log\Big(\frac{|\mathcal{P}| K^{|J|} (1 + tS_x)/\lambda}{\delta} \Big)}$ and

$$\mathcal{C}_{t,a_J} = \left\{ \theta \in \mathbb{R}^d : \left\| \theta_{a_J} - \hat{\theta}_{a_J}(t) \right\|_{V_{J,a_J}} \leq \sqrt{\beta_J(t)} \right\}.$$

Define the good events $\mathcal{G}_J = \big\{\theta^*_{a_J} \in \mathcal{C}_{t,a_J}, \forall t \geq 0, a_J \in \bigtimes_{i \in J} \mathcal{A}_i \big\}$. By Lemma 1, $P(G_J) \geq 1 - \frac{\delta}{|\mathcal{P}|}$. By the union bound, $P\big(\bigcup_{J \in \mathcal{P}} \mathcal{G}_J\big) \geq 1 - \delta$. Denote by $I_k(J, a_J) = \min\Big\{ t : \sum_{i=1}^t \mathbf{1}_{\{a_J(k) = a_J\}} = k \Big\}$ the k^{th} time action a_J was chosen in the sequence $x(1), a(2), \ldots, x(t), a(t)$ for set $J \in \mathcal{P}$, and by $N_t(J, a_J)$ the number of time action a_J was chosen at time t. Then, conditioned on $\bigcup_{J\in\mathcal{P}}\mathcal{G}_J$, for every $t \geq 0$

$$\begin{aligned}
\ell_t &= \sum_{i=1}^{n}\Big[\Big\langle x(t), \theta^*_{i,a_i^*(t)}\Big\rangle - \Big\langle x(t), \theta^*_{i,a_i(t)}\Big\rangle\Big] \\
&= \sum_{J\in\mathcal{P}}\sum_{i\in J}\Big[\Big\langle x(t), \theta^*_{i,a_i^*(t)}\Big\rangle - \Big\langle x(t), \theta^*_{i,a_i(t)}\Big\rangle\Big] \\
&= \sum_{J\in\mathcal{P}}\Big[\Big\langle x(t), \theta^*_{a_J^*(t)}\Big\rangle - \Big\langle x(t), \theta^*_{a_J(t)}\Big\rangle\Big] \\
&\leq \sum_{J\in\mathcal{P}} 2\sqrt{\beta_J(t)}\|x(t)\|_{V^{-1}_{J,a_J}}
\end{aligned}$$

Next, notice that $\ell_t \leq 2$ since $\sum_{i\in J}\big\langle x(t), \theta^*_{i,a}\big\rangle \in [-1,1]$. Therefore,

$$\ell_t \leq \sum_{J\in\mathcal{P}} 2\sqrt{\beta_J(t)}\min\Big\{\|x(t)\|_{V^{-1}_{J,a_J}}, 1\Big\}.$$

Combining the above we get that conditioned on $\bigcup_{J\in\mathcal{P}}\mathcal{G}_J$, for every $t \geq 0$

$$\begin{aligned}
R(T) &\leq \sqrt{T\sum_{t=0}^{T}\ell_t^2} \\
&\leq 2\sqrt{T\sum_{J\in\mathcal{P}}\beta_J(t)\sum_{t=0}^{T}\min\Big\{\|x(t)\|^2_{V^{-1}_{J,a_J}}, 1\Big\}} \\
&= 2\sqrt{T\sum_{J\in\mathcal{P}}\beta_J(t)\sum_{a_J}\sum_{k=1}^{N_T(J,a_J)}\min\Big\{\|x(I_k(J,a_J))\|^2_{V_{J,a_J}(I_k(J,a_J))^{-1}}, 1\Big\}} \\
&\leq 2\sqrt{T}\left(\lambda^{1/2}nS_\theta + R_{\max}\sqrt{d\log\left(\frac{|\mathcal{P}|K^n(1+tS_x)/\lambda}{\delta}\right)}\right)\sqrt{\sum_{J\in\mathcal{P}}\sum_{a_J}\sum_{k=1}^{N_T(J,a_J)}\min\Big\{\|x(I_k(J,a_J))\|^2_{V_{J,a_J}(I_k(J,a_J))^{-1}}, 1\Big\}} \\
&\overset{(1)}{\leq} 2\sqrt{T}\left(\lambda^{1/2}nS_\theta + R_{\max}\sqrt{d\log\left(\frac{|\mathcal{P}|K^n(1+tS_x)/\lambda}{\delta}\right)}\right)\sqrt{d\sum_{J\in\mathcal{P}}\sum_{a_J}\log\left(\lambda + \frac{N_T(J,a_J)S_x^2}{d}\right)} \\
&\overset{(2)}{\leq} 2\sqrt{T}\left(\lambda^{1/2}nS_\theta + R_{\max}\sqrt{d\log\left(\frac{|\mathcal{P}|K^n(1+tS_x)/\lambda}{\delta}\right)}\right)\sqrt{d\sum_{J\in\mathcal{P}}K^{|J|}\log\left(\lambda + \frac{TS_x^2}{K^{|J|}d}\right)} \\
&\leq 2\sqrt{T}\left(\lambda^{1/2}nS_\theta + R_{\max}\sqrt{d\log\left(\frac{|\mathcal{P}|K^n(1+tS_x)/\lambda}{\delta}\right)}\right)\sqrt{d\log\left(\lambda + \frac{TS_x^2}{Kd}\right)\sum_{J\in\mathcal{P}}K^{|J|}}
\end{aligned}$$

where in (1) we used Lemma 1 of Abbasi-Yadkori, Pál, and Szepesvári (2011), and in (2) Jensen's inequality and the fact that $\sum_{a_J} N_T(J,a_J) = T$, i.e.,

$$\begin{aligned}
\sum_{a_J}\log\left(\lambda + \frac{N_T(J,a_J)S_x^2}{d}\right) &= K^{|J|}\sum_{a_J}\frac{1}{K^{|J|}}\log\left(\lambda + \frac{N_T(J,a_J)S_x^2}{d}\right) \\
&\leq K^{|J|}\log\left(\lambda + \frac{\sum_{a_J}\frac{1}{K^{|J|}}N_T(J,a_J)S_x^2}{d}\right) \\
&= K^{|J|}\log\left(\lambda + \frac{TS_x^2}{K^{|J|}d}\right).
\end{aligned}$$

This completes the proof. □

Lemma 1. *Denote* $\theta^*_{a_J} = \sum_{i \in J} \theta^*_{i,a_i}$ *and let* $\hat{\theta}_{a_J}(t) = V_{J,a_J}(t)^{-1} Y_J(t)$ *be the least squares estimator at time step* t*, where* $V_{J,a_J}(t) = \lambda I + \sum_{k=1}^{t-1} \mathbf{1}_{\{a_J(k)=a_J\}} x(k) x(k)^T$*. Let* $\sqrt{\beta_J(t,\delta)} = \lambda^{1/2} |J| S_\theta + R_{max} \sqrt{d \log\left(\frac{|\mathcal{P}| K^{|J|}(1+tS_x)/\lambda}{\delta}\right)}$ *and*

$$\mathcal{C}_{t,a_J} = \left\{ \theta \in \mathbb{R}^d : \left\| \theta_{a_J} - \hat{\theta}_{a_J}(t) \right\|_{V_{J,a_J}} \leq \sqrt{\beta_J(t)} \right\}.$$

Then, for all $t \geq 0, a_J \in \times_{i \in J} \mathcal{A}_i$*, with probability at least* $1 - \frac{\delta}{|\mathcal{P}|}$*,* $\theta^*_{a_J} \in \mathcal{C}_{t,a_J}$*.*

Proof. Employing Theorem 2 of Abbasi-Yadkori, Pál, and Szepesvári (2011) with

$$\left\| \theta^*_{a_J} \right\|_2 = \left\| \sum_{i \in J} \theta^*_{i,a_i} \right\|_2 \leq |J| S_\theta$$

and taking the Union bound over all $a_J \in \times_{i \in J} \mathcal{A}_i$ yields that with probability at least $1 - \delta$,

$$\left\| \theta_{a_J} - \hat{\theta}_{a_J}(t) \right\|_{V_{J,a_J}} \leq \lambda^{1/2} |J| S_\theta + R_{\text{max}} \sqrt{d \log\left(\frac{K^{|J|}(1+tS_x)/\lambda}{\delta} \right)}.$$

Using $\tilde{\delta} = \frac{\delta}{|\mathcal{P}|}$ completes the proof. □