Supplementary Material for Cooperative Multi-Agent Reinforcement Learning with Sequential Credit Assignment

Anonymous Author(s)
Affiliation
Address
email

A Proof of Claim 1

Claim 1. *The proposed sequential credit assignment achieves additive advantage-decomposition.*

Proof. We discuss a multi-agent system with n agents identified by $a_i (i \in \{1, ..., n\})$ under one specific sequence $<a_1, a_2, ..., a_n>$. Claim 1 can also be proved from the rest $(n! - 1)$ orders in the same way. Here we denote $\mathbf{u}_{a_1}^{a_i} = [u^{a_1}, u^{a_2}, ..., u^{a_i}]$ $(i = 1, 2, 3, ..., n)$.

For better understanding, we analyze in the reverse order (from agent a_n to agent a_1). As for the last agent a_n in the sequence, we evaluate its action based on all the preceding agents' fixed behaviors, so there is no need to calculate the expectations on the others' actions, and the advantage function for agent a_n is the same as in COMA:

$$A^{a_n}(s, \mathbf{u}) = Q(s, \mathbf{u}) - \sum_{u'^{a_n}} \pi^{a_n}\left(u'^{a_n} | \tau^{a_n}\right) \cdot Q\left(s, \left(\mathbf{u}^{-a_n}, u'^{a_n}\right)\right). \tag{A1}$$

When evaluating the second-to-last agent a_{n-1}, the actions of agent a_1 to a_{n-2} are fixed. We only consider the expectation on agent a_n's action for the first term in Equ.(6) and the expectation on the actions of agent a_{n-1} and a_n for the second term in Equ.(6) in our main paper:

$$\begin{aligned} A^{a_{n-1}}(s, \mathbf{u}) = & \sum_{u'^{a_n}} \pi^{a_n}\left(u'^{a_n} | \tau^{a_n}\right) \cdot Q\left(s, \left(\mathbf{u}^{-a_n}, u'^{a_n}\right)\right) \\ & - \sum_{u'^{a_{n-1}}} \sum_{u'^{a_n}} \pi^{a_{n-1}}\left(u'^{a_{n-1}} | \tau^{a_{n-1}}\right) \cdot \pi^{a_n}\left(u'^{a_n} | \tau^{a_n}\right) \cdot Q\left(s, \left(\mathbf{u}_{a_1}^{a_{n-2}}, u'^{a_{n-1}}, u'^{a_n}\right)\right). \end{aligned} \tag{A2}$$

We can conclude the advantage function of each agent a_i $(i \in \{2, ..., n\})$ as:

$$\begin{aligned} A^{a_i}(s, \mathbf{u}) = & \sum_{u'^{a_{i+1}}} \cdots \sum_{u'^{a_n}} \pi^{a_{i+1}}\left(u'^{a_{i+1}} | \tau^{a_{i+1}}\right) \cdots \pi^{a_n}\left(u'^{a_n} | \tau^{a_n}\right) \cdot Q\left(s, \left(\mathbf{u}_{a_1}^{a_i}, u'^{a_{i+1}}, \cdots, u'^{a_n}\right)\right) \\ & - \sum_{u'^{a_i}} \cdots \sum_{u'^{a_n}} \pi^{a_i}\left(u'^{a_i} | \tau^{a_i}\right) \cdots \pi^{a_n}\left(u'^{a_n} | \tau^{a_n}\right) \cdot Q\left(s, \left(\mathbf{u}_{a_1}^{a_{i-1}}, u'^{a_i}, \cdots, u'^{a_n}\right)\right), \end{aligned} \tag{A3}$$

and the advantage of the first agent in the sequence a_1 is:

$$\begin{aligned} A^{a_1}(s, \mathbf{u}) = & \sum_{u'^{a_2}} \cdots \sum_{u'^{a_n}} \pi^{a_2}\left(u'^{a_2} | \tau^{a_2}\right) \cdots \pi^{a_n}\left(u'^{a_n} | \tau^{a_n}\right) \cdot Q\left(s, \left(u^{a_1}, u'^{a_2}, \cdots, u'^{a_n}\right)\right) \\ & - \sum_{u'^{a_1}} \cdots \sum_{u'^{a_n}} \pi^{a_1}\left(u'^{a_1} | \tau^{a_1}\right) \cdots \pi^{a_n}\left(u'^{a_n} | \tau^{a_n}\right) \cdot Q\left(s, \left(u'^{a_1}, u'^{a_2}, \cdots, u'^{a_n}\right)\right) \end{aligned} \tag{A4}$$

The second term of the above equation, i.e., the expected Q value of all possible joint actions, is recognized as the value of the state $V(s)$. We then rewrite Equ.(A4) as:

$$A^{a_1}\left(s, \mathbf{u}\right) = \sum_{u'^{a_2}} \cdots \sum_{u'^{a_n}} \pi^{a_2}\left(u'^{a_2}|\tau^{a_2}\right) \cdots \pi^{a_n}\left(u'^{a_n}|\tau^{a_n}\right) \cdot Q\left(s, \left(u^{a_1}, u'^{a_2}, \cdots, u'^{a_n}\right)\right) - V(s) \tag{A5}$$

It can be seen from Equ.(A3) that the second term of $A^{a_i}\left(s, \mathbf{u}\right)$ is the same as $A^{a_{i-1}}\left(s, \mathbf{u}\right)$'s first term $(i \in \{2, 3, ..., n\})$. We can eliminate most of the terms by summing from $A^{a_1}\left(s, \mathbf{u}\right)$ to $A^{a_n}\left(s, \mathbf{u}\right)$, acquiring the following equation that only retains the first term of A^{a_n} and the second term of A^{a_1}. Therefore, we derive the following equation from Equ.(A1) and Equ.(A5):

$$\sum_{i=1}^{n} A^{a_i}\left(s, \mathbf{u}\right) = Q\left(s, \mathbf{u}\right) - V(s) = A\left(s, \mathbf{u}\right). \tag{A6}$$

From Equ.(A6), we can see that our explicit credit assignment through the proposed sequential advantage function decomposes the total advantage function $A\left(s, \mathbf{u}\right)$ in an additive form. □

B Pseudocode

We provide SeCA's pseudocode below for a better understanding of its optimization procedure.

Algorithm 1 Sequential Credit Assignment Optimization Procedure

```
Randomly initialize the policy network parameter θ and the critic network parameter ϕ.
Initialize target critic network parameter ϕ⁻ ← ϕ.
while not terminated do
    Sample b episodes {τ^i} (i ∈ {1, 2, ..., b}) where τ^i = {s_0^i, z_0^i, u_0^i, r_0^i, ..., s_T^i, z_T^i, u_T^i, r_T^i}.
    for each episode i = 1 to b do
        for timestep t = T to 0 do
            Compute target y_t^i = r_t^i + γ (λ y_{t+1}^i + (1 − λ) f_{ϕ⁻}(s_{t+1}^i, u_{t+1}^i)).
            Compute critic loss L_t(ϕ)^i = (y_t^i − f_ϕ(s_t^i, u_t^i))^2.
        end for
        for all agents a ∈ {1, 2, ..., n} do
            Compute a's contribution to τ^i: c_a^i = Σ_{x_j ∈ π^a} PathIG_j^{τ^i}(π^a) according to Equ.(1).
        end for
        Decide the sequence based on c_a^i. Arrange agents with higher contributions in the front.
        for all agents in the sequence do
            Compute each agent a's sequential advantage A_a^i(s, u) according to Equ.(7) and (8).
        end for
    end for
    // Critic Learning:
    Update the critic parameter ϕ by descending the gradient ∇_ϕ (1/(bT)) Σ_i Σ_t L_t(ϕ)^i.
    // Policy Learning:
    Update the policy parameter θ by by maximizing the following objective:
```

$$\frac{1}{n}\sum\nolimits_a \left[\log \pi_a(u_a|\tau_a)\left(\frac{1}{b}\sum\nolimits_i \left(\frac{1}{T}\sum\nolimits_t A_a^i(s_t, \mathbf{u}_t)\right)\right) + \mathcal{H}\left(\pi^a(\cdot|\tau^a)\right)\right]$$

```
    if at target update interval then
        Update the target critic parameter ϕ⁻ ← ϕ.
    end if
end while
```

C Experiment Details on Multi-Agent Particle Environments

We evaluate our proposed sequential advantage and COMA's advantage in two multi-agent particle environments [3] in Section 3.3. The code for these experiment environments is available in the Codes folder in our Supplementary Material. Here we introduce these two environments and our settings in detail.

C.1 Environments Introduction

Predator-Prey. Three slower predators cooperate to chase a faster prey that acts randomly in an area containing two obstacles impeding the way. The action space of the predators is [`move_up`, `move_down`, `move_left`, `move_right`, `stay`]. Each predator can observe its current position and velocity and the displacement to the prey, other predators, and the obstacles. The predators' goal is to capture the prey in as few steps as possible, and the shared reward is the negative minimal distance between any predator and the prey. When any predator captures the prey, an additional positive reward is given to the team, and the game terminates.

Cooperative Navigation. This environment initializes three agents and three landmarks with random locations in an area. The agents aim to acquire a bigger shared reward and cooperate to cover all the landmarks with action space [`move_up`, `move_down`, `move_left`, `move_right`, `stay`]. Each agent observes its own position, velocity, and the displacement to the other agents and the targets. The global reward is the negative sum of the distance between each target and the nearest agent to it. If the agents collide, the team will receive a penalty, so agents must avoid collisions.

C.2 Training Settings

We train agents for 5000 episodes for these two environments. Each episode has a maximum of 200 steps. We adapt the original open-source environment implementation[1] and the training framework[2] provided by [7]. We follow the pre-set environment hyperparameters, including the shared reward for capturing the prey in Predator-Prey, the collision penalty in Cooperative Navigation, and the spawn regions size of the agents and obstacles. The prey acts randomly in Predator-Prey is a random agent with a uniform action probability distribution.

We utilize the default setting of the training framework. The policy networks and critic network implement one hidden layer MLPs, with 128 units and 32 units for Predator-Prey and Cooperative Navigation respectively. Agents share parameters in these environments. The number of transitions for each update is 128 and 32, and the discount factor is 0.99 and 0.9. We leverage target networks that update every 200 training iteration. Our models are trained by Adam Optimizer with a learning rate of 0.0002 for critic learning in Predator-Prey and 0.0001 in Cooperative Navigation, and 0.0001 for policy network in both environments.

Note, we only compare our sequential advantage with COMA's counterfactual advantage in Figure 2 in the main paper, so we do not implement SeCA's whole architecture. The three agents' advantages for policy learning here are directly calculated according to Equ.(5) or Equ.(7) in our main paper. The credit assignment sequence is fixed and initialized as <Agent 1, Agent 2, Agent 3>.

D Details of StarCraft Multi-Agent Challenge Experiments

We mainly evaluate methods on StarCraft II micromanagement in Section 4 and follow the default setup of the StarCraft Multi-Agent Challenge (SMAC) [5].[3] We utilize the open-source implementations of the baseline algorithms, including COMA, QMIX, QTRAN[4], and LICA[5]. All these methods are all based on the PyMARL framework [5]. The code for SeCA in SMAC is available in the Codes folder in Supplementary Material.

D.1 Detailed Information about SMAC and Scenarios

A group of units controlled by decentralized agents cooperates to defeat the enemy agent system controlled by handcrafted heuristics in each SMAC micromanagement problem. Each agent's partial observation comprises of the attributes (such as `health`, `location`, `unit_type`) of all units shown up in its view range. The global state information includes all agents' positions and `health`, and allied units' last actions and `cooldown`, which is only available to agents during centralized

[1] `https://github.com/openai/multiagent-particle-envs`
[2] `https://github.com/hsvgbkhgbv/SQDDPG`
[3] `https://github.com/oxwhirl/smac`
[4] `https://github.com/oxwhirl/pymarl`
[5] `https://github.com/mzho7212/LICA`

training. The agents' discrete action space consists of `attack[enemy_id]`, `move[direction]`, `stop`, and `no-op` for the dead agents only. Particular unit Medivac has no action `attack[enemy_id]` but has `heal[enemy_id]`. Agents can only attack enemies within their shooting range. Proper micromanagement requires agents to maximize the damage to the enemies and take as little damage as possible in combat, so they need to cooperate with each other or even sacrifice themselves. We follow the default setup of SMAC in our experiments, and more settings, including rewards and detailed observation/state information, can be acquired from the original paper or implementation.

Based on baseline algorithms' performances, the scenarios in SMAC are broadly grouped into three categories: *Easy*, *Hard*, and *Super Hard*. The key point to win some *Hard* or *Super Hard* battles is mastering specific micro techniques, such as *focus fire*, *avoid overkill*, *kiting*, et cetera. The battles can be both symmetric or asymmetric, and the group of agents can be homogeneous or heterogeneous. We consider six scenarios with different difficulties and characteristics: `2s3z`, `1c3s5z` (*Easy*, heterogeneous, symmetric); `2c_vs_64zg`, `3s_vs_5z` (*Hard*, homogeneous, asymmetric); and `MMM2`, `3s5z_vs_3s6z` (*Super Hard*, heterogeneous, asymmetric). Here we provide some characteristics of each scenario to help gain insights into the good or poor performance of the methods:

- Both `2s3z` and `1c3s5z` are symmetric combats where two heterogeneous teams battle against each other. `s` represents Stalkers that can attack enemies at a distance. `z` represents Zealots, melee units with a short attack range. `c` for Colossus. It is a ranged and endurable unit that can harm an area instead of a single agent. These two *Easy* scenarios that do not need to cooperate too much can be solved easily by most of the recognized methods.
- `2c_vs_64zg` is a *Hard* asymmetric scenario where two Colossi battle against 64 Zerglings, which are melee units with low health and low attack damage. The number of the enemy units in this map is the largest in the SMAC benchmark, making the agents' action space much larger than other maps. It can be utilized as an example to test methods' performance on large action spaces.
- `3s_vs_5z` is also a *Hard* asymmetric battle between two different homogeneous teams. The allied Stalkers have to master the *kiting* technique and disperse in the area to kill the Zealots that chase them one after another. This map faces the delayed reward problem; however, it is not very strict about micro-cooperation between agents because of their scattering.
- `MMM2` is a representative *Super Hard* asymmetric battle between two heterogeneous teams with three kinds of units. One Medivac, two Marauders, and seven Marines have to battle against a team with one more Marine. Marauder has greater attack damage and health than Marine but with a longer attack `cooldown`. Medivac has no damage but can heal any other agent in the team. This map with three kinds of units and many agents requires more cooperation between agents, so we picked this map for our ablation studies.
- `3s5z_vs_3s6z` is another *Super Hard* map that requires breaking the bottleneck of exploration, where three Stalkers and five Zealots battle against three Stalkers and six Zealots.

D.2 Training Settings

We follow most of the training hyperparameters in the original PyMARL implementation. Figure 3 in the main paper illustrates SeCA's network structure. The policy network consists of two FC layers and a GRU layer between them. The critic network maps the state into two weight matrices and biases, which, in turn, maps the concatenated action-policy vector into the Q estimate. We follow existing methods [1, 9] where all methods align on the batch size, the number of batch updates, and the total number of environment steps. We train all the methods for 32 million steps in *Easy* scenarios and 64 million steps in *Hard* and *Super Hard* scenarios. We use 32 actors to generate the trajectories in parallel and use one NVIDIA Titan V GPU for training. More training details of our method utilized in SMAC are shown in Table A1 and can be found in our code.

D.3 Additional Results on SMAC

We do not compare QPD's learning curves with other methods in Figure 5 of our main paper, as QPD modifies the original SMAC's implementation, and it is unfair to compare QPD's learning speed with other methods that trained in the original environment. Here we follow the original paper of QPD, providing a test win percentage table of median and mean performance at the end of the

Table A1: Hyperparameters of SeCA in SMAC.

Hyperparameters	**#**	**Description**
hidden units	64	Hidden units number for policy and critic network
batch size b	32	The number of transitions for each update
parallel runners	32	Number of environments to run in parallel
discount factor γ	0.99	The importance of future rewards
λ in TD(λ)	0.8	TD(λ) parameter for critic training
entropy regularization ξ	0.005	Weight or regularization for exploration
policy network's initial lr	0.0025	Initial learning rate for policy network
critic network's initial lr	0.0005	Initial learning rate for critic network
each IG step number	5	Summation of # intervals to approximate integrated gradients
target critic update frequency	200	Target network updates every # gradient steps
test interval	320000	Test the model every # steps
test episode number	32	Number of episodes to test for

training period. We use results from the SMAC paper [5] and the original QPD paper [8] because these reports show higher performance than the original works [6, 2, 4] and our implementation on QPD. Table A2 shows the evaluation results, where $\widetilde{m}$ is the median win percentage and $\overline{m}$ is the mean win percentage. In general, the median performance is more persuasive as it avoids the effect of any outliers [5]. We could see from Table A2 that our method SeCA achieves state-of-the-art performances on all these scenarios that mentioned in the original QPD paper, demonstrating our improvement on QPD that also utilizes integrated gradients for the credit assignment problem.

Table A2: Median and mean performance of the test win percentage.

Map	IQL		COMA		QMIX		QTRAN		QPD		SeCA (Ours)	
	$\widetilde{m}$	$\overline{m}$	$\widetilde{m}$	$\overline{m}$	$\widetilde{m}$	$\overline{m}$	$\widetilde{m}$	$\overline{m}$	$\widetilde{m}$	$\overline{m}$	$\widetilde{m}$	$\overline{m}$
3m	**100**	97	91	92	**100**	99	**100**	**100**	95	92	**100**	**100**
8m	91	90	95	94	**100**	96	**100**	97	94	93	**100**	**100**
2s3z	39	42	66	64	**100**	97	77	80	95	94	**100**	**99**
3s5z	0	3	0	0	16	25	0	4	85	81	**98**	**97**
1c3s5z	7	8	30	30	89	89	31	33	92	92	**100**	**100**

E Visualization

We visualize two `3s_vs_5z` battles and show the learned sequences to provide insights into our sequence adjustment result. We focus on this map because it contains only three agents, and its sequence analysis would be clear and illuminating. The allied Stalkers must master *kiting* technique and disperse in the map to kill the Zealots that chase them one after another. We number these three agents as Agent 1, Agent 2, and Agent 3 and print their positions at every step to follow their moves. The replays are available in the Visualization folder in our Supplementary Material.

Figure A1 shows the mini-maps of representative moments in one testing episode. Agent 3 only attracts one Zealot in this battle and kills it quickly, while Agent 1 and Agent 2 kite two enemies. After defeating the following Zealot, Agent 3 goes supporting alliances and observes Agent 1 in danger. Agent 3 blocks the enemies' way to shelter Agent 1, and they kill the following two Zealots together. Agent 2, who fights alone, defeats both of the following enemies and does not show in its alliances' field of view. The learned sequence of this episode is <Agent 3, Agent 2, Agent 1>.

We visualize another testing episode of `3s_vs_5z` in Figure A2. Agent 3 also kites only one Zealot, and both Agent 1 and Agent 2 attract two enemies. When going to rescue alliances, Agent 3 misses Agent 1 this time and observes Agent 2 afterward. Then Agent 3 support Agent 2, who already killed a Zealot but has low `health`. However, Agent 1, who fights alone, only defeats one enemy and is killed by another Zealot. This survived Zealot is defeated by Agent 3, who protects Agent 2 at death's door at last. The credit assignment sequence learned for this episode is <Agent 3, Agent 1, Agent 2>.

Figure A1: Some critical moments in a `3s_vs_5z` battle. The big red squares represent Stalkers controlled by the agents, and each small blue square is an enemy Zealot. Agents win this battle by scattering and kiting the melee enemy units. Agent 3 kites only one Zealot and kills it quickly, and then it goes to the top of the map, protecting Agent 1 nearby. Agent 3 supports Agent 1 in defeating two following Zealots, while Agent 2 kites and kills two enemies by itself.

Although winning these two battles similarly, SeCA learns two different sequences in this map, illustrating that our dynamic adjustment algorithm (IG-episode) decides the sequence based on the battle's real-time circumstances. We find some similarities in these two learned sequences: Agent 3 that supports and protects alliances in both battles, is arranged at the front, while the agent who receives backup and flees to keep alive is always the last in the sequence. These similarities are consistent with our common understanding of team contribution. Thus, these two visualizations demonstrate the rationality of our methods in practice.

Figure A2: Another episode in `3s_vs_5z`. Similar to Figure A1, Agent 3 in this map still kites only one Zealot and kills it quickly. However, Agent 3 misses Agent 1 this time and supports Agent 2 in defeating the following Zealot. Agent 1 kills one enemy and does not survive from the other one, which is killed by Agent 3 later. Agent 2, with low `health`, choose to run away to keep alive.

F Additional Statement

We are very sorry for a typo in our main paper that may affect reviewers' understanding of our work and correct it here. We intended to show the objective to maximize when updating the policy network parameter θ in Equ.(11) in line 209 but gave a gradient form. Here we provide the correct version:

Policy Learning. We optimize each agent a's policy parameter θ_a by maximizing the following objective, which contains our proposed advantage function and an entropy regularization term $\mathcal{H}$:

$$J^a(\theta) = \mathbb{E}_{\tau\sim\pi}\left[\log \pi^a(u^a|\tau^a) A^a(s,\mathbf{u}) + \mathcal{H}\left(\pi^a(\cdot|\tau^a)\right)\right], \tag{A7}$$

where the derivative of the adaptive entropy regularization term $\mathcal{H}(\pi^a(\cdot|\tau^a))$ [9] with respect to the i-th action probability p_i^a is given by:

$$d\mathcal{H}_i := -\xi \cdot (\log p_i^a + 1)/H(\pi^a(\cdot|\tau^a)), \text{ and } H(\pi^a(\cdot|\tau^a)) = \mathbb{E}_{u^a\sim\pi^a}\left[-\log \pi^a(u^a|\tau^a)\right]. \tag{A8}$$

We share parameters among agents, and the gradient we use to train the actor shared by all agents is:

$$g = \mathbb{E}_{\tau\sim\pi}\left[\mathbb{E}_a\left[\nabla_{\theta_a}\left(\log \pi^a(u^a|\tau^a) A^a(s,\mathbf{u}) + \mathcal{H}\left(\pi^a(\cdot|\tau^a)\right)\right)\right]\right]. \tag{A9}$$

We sincerely apologize to the reviewers. Sorry for the inconvenience caused by this careless slip-up.

References

[1] Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4403–4414, 2019.

[2] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*, pages 2974–2982, 2018.

[3] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6382–6393, 2017.

[4] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation. In *Advances in Neural Information Processing Systems*, pages 10199–10210, 2020.

[5] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *CoRR*, abs/1902.04043, 2019.

[6] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *International Conference on Machine Learning*, pages 330–337, 1993.

[7] Jianhong Wang, Yuan Zhang, Tae-Kyun Kim, and Yunjie Gu. Shapley q-value: A local reward approach to solve global reward games. In *AAAI Conference on Artificial Intelligence*, pages 7285–7292, 2020.

[8] Yaodong Yang, Jianye Hao, Guangyong Chen, Hongyao Tang, Yingfeng Chen, Yujing Hu, Changjie Fan, and Zhongyu Wei. Q-value path decomposition for deep multiagent reinforcement learning. In *International Conference on Machine Learning*, pages 10706–10715, 2020.

[9] Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. Learning implicit credit assignment for cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.