Appendix
ResQ: A Residual Q Function-based Approach for Multi-Agent Reinforcement Learning Value Factorization

Siqi Shen[†], **Mengwei Qiu**[†], **Jun Liu**[†], **Weiquan Liu**[†], **Yongquan Fu**[‡,*], **Xinwang Liu**[‡], **Cheng Wang**[†]
[†]Fujian Key Lab of Sensing and Computing for Smart Cities, School of Informatics, Xiamen University, China
[‡]School of Computer, National University of Defense Technology, China
{siqishen,wqliu,cwang}@xmu.edu.cn, {yongquanf,xinwangliu}@nudt.edu.cn
{mengweiqiu,junliu}@stu.xmu.edu.cn

A Appendix

In Section A.1, we provide the ResQ theorems and proofs. Section A.2 describes more experimental results. We describe the exeprimental setup in Section A.2.1 in detail, the factorization results in Section A.2.2 and shows another payoff matrix that cannot represent well by state-of-the-art algorithms in Section A.2.3. We show more results about the SMAC benchmark in Section A.2.4.

In Section A.3, we study the differences among ResQ and others. We compare ResQ with DMIX in Section A.3.1, with QTRAN in Section A.3.2, with QPlex in Section A.3.3, with weighted QMIX in Section A.3.4, and study the impact of the implementation of the inequality condition in Section A.3.5, and discuss the limitation of ResQ in Section A.3.6.

A.1 Theorems

In this section, we show the proof of all the Theorems of this work.

Theorem 1. *A joint state-action function*

$$Q_{jt}(\tau, u) = Q_{tot}(\tau, u) + w_r(\tau, u) Q_r(\tau, u) \tag{1}$$

is factorized by $[Q_i(\tau_i, u_i)]_{i=1}^N$*, if* $Q_r(\tau, u) \leq 0$*,* $Q_{tot}(\tau, u)$ *and* $[Q_i(\tau_i, u_i)]_{i=1}^N$ *satisfy the monotonicity conditions, and*

$$w_r(\tau, u) = \begin{cases} 0 & u = \bar{u}, \quad (2a) \\ 1 & u \neq \bar{u}, \quad (2b) \end{cases}$$

Proof. Theorem 1 shows that if condition (2) holds and $Q_r(\tau, u) \leq 0$, then $[Q_i(\tau_i, u_i)]_{i=1}^N$ satisfy the IGM principle for Q_{jt}. We will show that $\arg\max_u Q_{\text{jt}}(\tau, u) = \bar{u}$, where $\bar{u}_i = \arg\max_{u_i} Q_i(\tau_i, u_i)$ and $\bar{u} = [\bar{u}_i]_{i=1}^N$.

$$\begin{aligned} Q_{jt}(\tau, \bar{u}) &= Q_{tot}(\tau, \bar{u}) \quad \text{(From (1)(2a))} && (3) \\ &\geq Q_{tot}(\tau, u) \quad \forall u \neq \bar{u} \quad \bar{u} \text{ maximize } Q_{tot} && (4) \\ &\geq Q_{tot}(\tau, u) + w_r(\tau, u) Q_r(\tau, u) \quad \forall u \neq \bar{u} \quad \text{(Monotonicity)} && (5) \\ &= Q_{\text{jt}}(\tau, u) \quad \forall u \neq \bar{u} \quad \text{(From (1))} && (6) \end{aligned}$$

(5) comes from $w_r(\tau, u) = 1 \; \forall u \neq \bar{u}$ and $Q_r(\tau, u) \leq 0$. (3) to (6) mean that $\bar{u} = [\bar{u}_i]_{i=1}^N$ maximizes $Q_{jt}(\tau, u)$. Thus $[Q_i(\tau_i, u_i)]_{i=1}^N$ satisfies IGM for $Q_{jt}(\tau, u)$. □

*Corresponding author

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Lemma 1. *For any joint state-action function $Q(\tau, u)$, we can find a Q_{tot} that satisfy the monotonicity conditions, and they share the same optimal policy.*

Proof. Theorem 1 of [1] shows that Weighted QMIX can always find a monotonically increasing function Q_{tot} that shares the same optimal policy as the true value function of $Q(s, u)$. This indicates that for any $Q(s, u)$, there exists a monotonically increasing function $Q_{tot}(s, u)$ that shares the same optimal policy as $Q(s, u)$. □

Theorem 2. *For any joint state-action function $Q(\tau, u)$, we can find $Q_{jt}(\tau, u) = Q_{tot}(\tau, u) + w_r(\tau, u)Q_r(\tau, u)$ that*

$$\bar{u} = \arg\max_u Q(\tau, u) = \arg\max_u Q_{jt}(\tau, u) \tag{7}$$

$$Q(\tau, u) = Q_{jt}(\tau, u) \quad \forall u \neq \bar{u} \tag{8}$$

$Q_{tot}(\tau, u)$ is a monotonic increasing function with respect to $[Q_i(\tau, u)]_{i=1}^N$, $w_r(\tau, u)$ satisfies the condition (2), and $Q_r(\tau, u) \leq 0$.

Proof. Lemma (1) shows that for any joint state-action function $Q(\tau, u)$, we can find a Q_{tot} that satisfy the monotonicity conditions and $\bar{u} = \arg\max Q_{tot}(\tau, u) = \arg\max Q(\tau, u)$. Let

$$\Delta = Q(\tau, \bar{u}) - Q_{tot}(\tau, \bar{u}) \tag{9}$$

$$Q_{tot}^{'}(\tau, u) = Q_{tot}(\tau, u) + \Delta \quad \forall u \tag{10}$$

Without loss of generality, let's assume that $\Delta \geq 0$. Obviously, the new function $Q_{tot}^{'}(\tau, u)$ satisfy the monotonic conditions, and $\arg\max_u Q(\tau, u) = \arg\max_u Q_{tot}^{'}(\tau, u)$ and $Q(\tau, \bar{u}) = Q_{tot}^{'}(\tau, \bar{u})$. Define the residual Q function as

$$Q_r(\tau, u) = Q(\tau, u) - Q_{tot}^{'}(\tau, u) \tag{11}$$

Then

$$Q_r(\tau, \bar{u}) = Q(\tau, \bar{u}) - Q_{tot}^{'}(\tau, \bar{u}) \tag{12}$$

$$= Q(\tau, \bar{u}) - Q_{tot}(\tau, \bar{u}) - \Delta \tag{13}$$

$$= 0 \quad \text{(From(9))} \tag{14}$$

We can write $Q(\tau, u)$ as follows.

$$Q(\tau, u) = Q_{tot}^{'}(\tau, u) + Q_r(\tau, u) \tag{15}$$

$$= Q_{tot}^{'}(\tau, u) + w_r(\tau, u)Q_r(\tau, u) \tag{16}$$

where

$$w_r(\tau, u) = \begin{cases} 0 & u = \bar{u}, \quad (17a) \\ 1 & u \neq \bar{u}, \quad (17b) \end{cases}$$

Let

$$Q_r^{'}(\tau, u) = \begin{cases} 0 & u = \bar{u}, \quad (18a) \\ Q_r(\tau, u) - \max_u Q_r(\tau, u) & u \neq \bar{u}, \quad (18b) \end{cases}$$

Through definition, $Q_r^{'}(\tau, u) \leq 0$. Let

$$Q_{tot}^{''}(\tau, u) = Q_{tot}^{'}(\tau, u) + \max_u Q_r(\tau, u) \tag{19}$$

Obviously, $Q_{tot}^{''}(\tau, u)$ satisfy the monotonic conditions with $[Q_i]_{i=1}^N$. $\arg\max_u Q(\tau, u) = \arg\max_u Q_{tot}^{'}(\tau, u) = \arg\max_u Q_{tot}^{''}(\tau, u)$. Combining (18) and (19), we can find a function

$$Q_{jt}(\tau, u) = Q_{tot}^{''}(\tau, u) + w_r(\tau, u)Q_r^{'}(\tau, u) \tag{20}$$

$\arg\max_u Q_{jt}(\tau, u) = \arg\max_u Q(\tau, u)$. Thus, we show that for any joint state action function $Q_(\tau, u)$, it can find a function Q_{jt} that can be expressed in the form of ResQ (1), and share the same optimal actions with Q.

Further, we analysis the property of $Q_{jt}(\tau, u) \quad \forall u \neq \bar{u}$.

$$Q_{jt}(\tau, u) = Q_{tot}^{''}(\tau, u) + w_r(\tau, u)Q_r^{'}(\tau, u) \quad \forall u \neq \bar{u} \tag{21}$$
$$= Q_{tot}^{'}(\tau, u) + w_r(\tau, u)Q_r(\tau, u) \quad \forall u \neq \bar{u} \tag{22}$$
$$= Q(\tau, u) \quad \forall u \neq \bar{u} \tag{23}$$

Then, we analysis the property of $Q_{jt}(\tau, \bar{u})$.

$$Q_{jt}(\tau, \bar{u}) = Q_{tot}^{''}(\tau, \bar{u}) + w_r(\tau, \bar{u})Q_r^{'}(\tau, \bar{u}) \tag{24}$$
$$= Q_{tot}^{''}(\tau, \bar{u}) \tag{25}$$
$$= Q_{tot}^{'}(\tau, \bar{u}) + \max_u Q_r(\tau, u) \tag{26}$$
$$= Q(\tau, \bar{u}) + \max_u Q_r(\tau, u) \tag{27}$$

Combining (23) and (27), we reach the conclusion that the two functions $Q(\tau, u)$ and $Q_{jt}(\tau, u)$ differ only in the maximal values. And $Q_{jt}(\tau, u)$ can be expressed as the form of the addition of one monotonicity function $Q_{tot}(\tau, u)$ and a residual state-action function $Q_r(\tau, u)$ □

Theorem 3. *A stochastic joint state-action function*

$$Z_{jt}(\tau, u) = Z_{dmix}(\tau, u) + w_r(\tau, u)Z_r(\tau, u) \tag{28}$$

is factorized by $[Z_i(\tau_i, u_i)]_{i=1}^N$*, if* $Z_r(\tau, u) \leq 0$ *and* $w_r(\tau, u) = 0$ *when* $u = \bar{u}$*, otherwise* 1. $Z_{dmix}(\tau, u)$ *is the factorization method proposed in [2].* $Z_{dmix}(\tau, u) = Z_{mean}(\tau, u) + Z_{shape}(\tau, u)$, $\mathbb{E}[Z_{shape}(\tau, u)] = 0$, $Q_i = \mathbb{E}[Z_i(\tau_i, u_i)]$. $Z_{mean}(\tau, u)$ *is a monotonically increasing function with respect to* Q_i *or the sum of* $[Q_i]_1^N$.

Proof. We will show that $\arg\max_u \mathbb{E}[Z_{jt}(\tau, u)] = \bar{u}$, where $\bar{u}_i = \arg\max_{u_i} \mathbb{E}[Z_i(\tau_i, u_i)]$ and $\bar{u} = [\bar{u}_i]_{i=1}^N$.

$$\mathbb{E}[Z_{jt}(\tau, \bar{u})] = \mathbb{E}[Z_{dmix}(\tau, \bar{u})] \quad \text{(From (28) } w(\tau, \bar{u}) = 0) \tag{29}$$
$$= \mathbb{E}[Z_{mean}(\tau, \bar{u}) + Z_{shape}(\tau, \bar{u})] \tag{30}$$
$$= \mathbb{E}[Z_{mean}(\tau, \bar{u})] \quad \text{(From } \mathbb{E}[Z_{shape}(\tau, u)] = 0) \tag{31}$$
$$= \mathbb{E}[Q_{mix}(\tau, \bar{u})] \quad \text{(Definition)} \tag{32}$$
$$\geq \mathbb{E}[Q_{mix}(\tau, u)] \quad \forall u \neq \bar{u} \quad \text{(Mononicity relation)} \tag{33}$$
$$= \mathbb{E}[Z_{mean}(\tau, u)] \quad \forall u \neq \bar{u} \quad \text{(Definition)} \tag{34}$$
$$= \mathbb{E}[Z_{mean}(\tau, u) + Z_{shape}(\tau, u)] \quad \forall u \neq \bar{u} \quad \text{(From } \mathbb{E}[Z_{shape}(\tau, u)] = 0) \tag{35}$$
$$= \mathbb{E}[Z_{dmix}(\tau, u)] \quad \forall u \neq \bar{u} \quad \text{(Definition)} \tag{36}$$
$$\geq \mathbb{E}[Z_{dmix}(\tau, u)] + w_r(\tau, u)\mathbb{E}[Z_r(\tau, u)] \quad \forall u \neq \bar{u} \tag{37}$$
$$= \mathbb{E}[Z_{dmix}(\tau, u) + w_r(\tau, u)Z_r(\tau, u)] \quad \forall u \neq \bar{u} \tag{38}$$
$$= \mathbb{E}[Z_{jt}(\tau, u)] \quad \forall u \neq \bar{u} \quad \text{(From (28))} \tag{39}$$

(37) comes from $w_r(\tau, u) = 1 \ \forall u \neq \bar{u}$ and $Z_r(\tau, u) \leq 0$. (29) to (39) mean that $\bar{u} = [\bar{u}_i]_{i=1}^N$ maximizes $\mathbb{E}[Z_{jt}(\tau, u)]$. Thus $[Z_i(\tau_i, u_i)]_{i=1}^N$ satisfies DIGM for $Z_{jt}(\tau, u)$. □

Theorem 4. *A stochastic joint state-action function*

$$Z_{jt}(\tau, u) = Z_{tot}(\tau, u) + w_r(\tau, u)Z_r(\tau, u) \tag{40}$$

is factorized by $[Z_i(\tau_i, u_i)]_{i=1}^N$*, if* $Z_r(\tau, u) \leq 0$*,* $Z_{tot}(\tau, u) = \sum_{i=1}^N k_i Z_i(\tau_i, u_i) \ k_i \geq 0$ *and* $w_r(\tau, u) = 0$ *when* $u = \bar{u}$*, otherwise* 1.

Proof. Theorem 4 shows that if $Z_r(\tau, u) \leq 0$, $Z_{tot}(\tau, u) = \sum_{i=1}^N k_i Z_i(\tau_i, u_i) \ k_i \geq 0$ and $w_r(\tau, u) = 0$ when $u = \bar{u}$, otherwise 1, then $[Z_i(\tau_i, u_i)]_{i=1}^N$ satisfy the DIGM principle for $Z_{jt}(\tau, u)$.

We will show that $\arg\max_u \mathbb{E}[Z_{jt}(\tau, u)] = \bar{u}$, where $\bar{u}_i = \arg\max_{u_i} \mathbb{E}[Z_i(\tau_i, u_i)]$ and $\bar{u} = [\bar{u}_i]_{i=1}^N$.

$$\mathbb{E}[Z_{jt}(\tau, \bar{u})] = \mathbb{E}[Z_{tot}(\tau, \bar{u})] \quad \text{(From (40) } w(\tau, \bar{u}) = 0) \tag{41}$$

$$\geq \mathbb{E}[Z_{tot}(\tau, u)] \quad \forall u \neq \bar{u} \tag{42}$$

$$\geq \mathbb{E}[Z_{tot}(\tau, u)] + w_r(\tau, u)\mathbb{E}[Z_r(\tau, u)] \quad \forall u \neq \bar{u} \tag{43}$$

$$= \mathbb{E}[Z_{tot}(\tau, u) + w_r(\tau, u)Z_r(\tau, u)] \quad \forall u \neq \bar{u} \tag{44}$$

$$= \mathbb{E}[Z_{jt}(\tau, u)] \quad \forall u \neq \bar{u} \quad \text{(From (40))} \tag{45}$$

Because $Z_{tot}(\tau, u) = \sum_{i=1}^N k_i Z_i(\tau_i, u_i)\, k_i \geq 0$, the inequality formula (42) is established. (43) comes from $w_r(\tau, u) = 1\ \forall u \neq \bar{u}$ and $Q_r(\tau, u) \leq 0$. (44) comes from $w_r(\tau, u)Z_r(\tau, u) = \mathbb{E}[w_r(\tau, u)Z_r(\tau, u)]$. (41) to (45) mean that $\bar{u} = [\bar{u}_i]_{i=1}^N$ maximizes $\mathbb{E}[Z_{jt}(\tau, u)]$. Thus $[Z_i(\tau_i, u_i)]_{i=1}^N$ satisfies DIGM for $Z_{jt}(\tau, u)$. □

In Theorem 4, the main function Z_{tot} is a positive weighted sum of Z_i instead of a monotonic mixing of Z_i, because not all monotonic mixing functions satisfy the DIGM principle. We will show an example in the proof of Theorem 5.

Theorem 5. *The factorization method CW/OW QMIX may fail to satisfy the DIGM theorem.*

Proof. We prove this theorem through an example.

In expectation case, CW/OW QMIX learn $Q_{tot}(\tau, u)$ to approximate the optimal policy of the state-action value function $Q(\tau, u)$, where Q_{tot} is a monotonic increasing function with respect to Q_i. For the stochastic case, CW/OW QMIX learn $Z_{tot}(\tau, u)$ to approximate the true value function, where Z_{tot} is a monotonic increasing function with respect to Z_i. That is $\partial Z_{tot}(\tau, u)/\partial Z_i(\tau_i, u_i) \geq 0$

Let's consider a simple one-step matrix game where there are only two agents each with actions a and b. Let's assume that the state is full observational. The learned function is $Z_{tot}(\tau, u) = Z_{tot}(s, u) = Z_1^2(s, u_1) + Z_2^2(s, u_2)$, where $Z_1 = Z_2$. Z_{tot} is the summation of the square of the stochastic utilities. Z_{tot} could lead to incorrect estimation of the optimal actions. Let's assume that, for action a, $Z_1(s, a) = 2$ for 100% of the time; for action b $Z_1(s, b) = 3$ for 50% of the time and $Z_1(s, b) = 0$ for 50% of the time. Clearly $\mathbb{E}Z_1(s, a) = 2$ and $\mathbb{E}Z_1(s, b) = 1.5$. If Z_{tot} and Z_i satisfy the DIGM theorem, then the optimal action for Z_{tot} should be $(a, a) = (\arg\max \mathbb{E}Z_1(\tau, u), \arg\max \mathbb{E}Z_2(\tau, u))$. However, $\mathbb{E}Z_1^2(\tau, a) = 4$ and $\mathbb{E}Z_1^2(\tau, b) = 4.5$. $\arg\max \mathbb{E}Z_{tot} = (b, b)$ rather than (a, a)

□

A.2 Experimental Results

A.2.1 Experimental Setup

We adopt the PyMARL [3] implementation of QMIX,VDN, QAtten, QTran, OW QMIX, CW QMIX, QPlex, and DMIX from their open-source repositories[2][3][4]. Their code are released under the Apache License V2.0. Their hyper-paramters are the same as that in PyMARL. ResQ is designed based on PyMARL as well. All the algorithms is trained with 1 rollout process for 2 million steps (the default setup for Weighted QMIX). At each 10,000 training step, the algorithm is tested on test episodes to evaluate their performance. For ResQ, the RMSProp optimizer ($l_r = 1e^{-3}$) is used. The batch size and buffer size are 32 and 5,000, respectively. In the matrix game and SMAC, QMIX is chosen as the main function Q_{tot}. Q_{tot} has 32 hidden dimensions, equiped with 2 hypernets, each with 64 dimensions. In predator-predy, VDN is chosen as Q_{tot}. The ϵ used in $\epsilon-$greedy annealed from 1 to 0.05 within 100K steps. TD lambda is used, and $\lambda = 0.6$. All the experiments are ran within a local cluster. Each experiment is ran within a NVIDIA 3090 GPU.

A.2.2 A one-step Matrix Game

Table 1 show the detailed factorization results of the one-step pay-off matrix (depicted in Figure 1).

Table 2 shows another difficult pay-off matrix. In this matrix, only ResQ and QTran can recover the optimal action, the other algorithms cannot.

[2]https://github.com/oxwhirl/pymarl

[3]https://github.com/oxwhirl/wqmix

[4]https://github.com/wjh720/QPLEX

Table 1: The Q_{jt}, Q_{tot}, V, and Adv of different methods

Q_1 \ Q_2	**0.108 (A)**	-0.300 (B)	0.106 (C)
0.108(A)	8.03	-12.00	-11.99
-0.300(B)	-12.00	0.00	0.00
0.106(C)	-12.00	0.00	7.87

(a) ResQ: Q_1, Q_2, Q_{jt}

u_1 \ u_2	**A**	B	C
A	8.03	7.50	8.02
B	7.50	6.97	7.50
C	8.02	7.50	8.02

(b) ResQ: Q_{tot}.

u_1 \ u_2	**A**	B	C
A	0	-19.51	-20.04
B	-19.48	-6.96	-7.50
C	-20.00	-7.50	-0.16

(c) ResQ: $w_r Q_r$.

u_1 \ u_2	**A**	B	C
A	8.00	4.67	7.98
B	4.88	1.55	4.86
C	7.99	4.65	7.97

(d) QTran: $Q_{jt} = \sum_{i=1}^{n} Q_i + V$

u_1 \ u_2	**A**	B	C
A	6.74	3.62	6.73
B	3.41	0.30	3.40
C	6.71	3.60	6.71

(e) QTran: $Q_{tot} = \sum_{i=1}^{n} Q_i$

u_1 \ u_2	**A**	B	C
A	8.00	-5.04	-5.04
B	-5.04	-5.04	-5.04
C	-5.04	-5.04	-5.04

(f) CW QMIX: Q_{tot}

u_1 \ u_2	**A**	B	C
A	6.07	-0.87	6.86
B	-0.86	-0.87	-0.16
C	5.49	-0.87	6.29

(g) OW QMIX: Q_{tot}

u_1 \ u_2	**A**	B	C
A	11.82	-26197	13.80
B	-26918	-53128	-26916
C	13.85	-26195	15.83

(h) QPlex: V

u_1 \ u_2	**A**	B	C
A	3.94	26194	-13.46
B	26915	53140	26928
C	-15.06	26208	0

(i) QPlex: Adv

Table 2: Payoff matrix of a one-step matrix game and reconstructed Q_{jt}. Boldface means greedy actions. Red color indicates *wrongly* estimated optimal actions, whereas blue color represents the opposite.

u_1 \ u_2	**A**	B	C
A	2.5	0	-100
B	0	2	0
C	-100	-100	**3**

(a) Game Payoff matrix.

Q_1 \ Q_2	-0.04 (A)	-0.84 (B)	**0.08 (C)**
-0.04(A)	2.34	0.15	-99.4
-0.84(B)	-0.17	2.11	0.03
0.08(C)	-99.4	-99.3	**2.98**

(b) ResQ: Q_1, Q_2, Q_{jt}

Q_1 \ Q_2	1.79(A)	1.66(B)	**1.81**(C)
0.89(A)	2.89	2.76	2.91
0.78(B)	2.78	2.65	2.80
1.01(C)	3.01	2.87	**3.02**

(c) QTran: Q_1, Q_2, Q_{jt}

Q_1 \ Q_2	1.98(A)	1.86(B)	**1.99(C)**
2.82(A)	1.80	-1.51	-5.78
2.83(B)	1.70	1.79	**1.80**
2.81(C)	-11.32	-11.32	1.79

(d) QPlex: Q_1, Q_2, Q_{jt}

Q_1 \ Q_2	**-0.02(A)**	-0.03(B)	-3.35(C)
-0.02(A)	**2.73**	1.88	-57.38
-0.03(B)	1.82	0.97	-57.38
-4.42(C)	-57.38	-57.38	-57.38

(e) CW QMIX: Q_1, Q_2, Q_{jt}

Q_1 \ Q_2	**-0.17(A)**	-2.27(B)	-20.31(C)
10.40(A)	1.71	-0.51	-19.49
19.04(B)	**19.28**	17.07	-2.02
-20.71(C)	-19.52	-19.52	-19.52

(f) OW QMIX: Q_1, Q_2, Q_{jt}

A.2.3 A distributional one-step Matrix Game

We study the representation power of multipe methods for a distributional matrix game. The matrix game is modified from the matrix depicted in Figure 1. We add a nomral distribution (mean = 0, std=1) value into all action-value of the matrix. The payoff matrix of the game is depicted in Table 3 (a). All the cells show the mean and the variance of a normal distributed reward. For example, (8,1) indicates that the mean and the variance of the reward distribution are 8 and 1, respectively.

In this matrix game, all the algorithm are ran through a full exploration $\epsilon = 1$ for ϵ-greedy conducted over 20,000 steps. This setting guarantees the exploration of all possible actions.

As it is observed in Table 3, ResQ/ResZ, and QTRAN can obtain the optimal policy, DMIX, DDN, CW QMIX, and QPlex learn the second-best policy, and OW QMIX learns a wrong policy.

These result demonstrates the ability of ResQ to factorize difficult state-action value functions with low approximation error and high sample efficiency. Moreover, we find that ResZ has lower approximation error than ResQ for this stochastic pay-off matrix.

A.2.4 StarCraft II Multi-Agent Challenge (SMAC)

We study the performance of ResQ and other algorithms on a well known MARL benchmark, the StarCraft II Multi-Agent Challenge (SMAC). In each environment of SMAC, two teams of agents fight against each other. One team is controlled by the built-in game artificial intelligence, which is handcrafted carefully. We choose the highest difficult built-in game script. Another team of agents are controlled by the decentralized policies learned by the MARL algorithms. Each agent can receive

Table 3: Payoff matrix of a didactic **distributional** matrix game and reconstructed Q_{jt}. Boldface means greedy actions. Red color indicates wrongly estimated optimal actions, whereas blue color represents the opposite.

u_1 \ u_2	**A**	B	C
A	**(8, 1)**	(-12,1)	(-12,1)
B	(-12,1)	(0,1)	(0,1)
C	(-12,1)	(0,1)	(7.9,1)

(a) Game Payoff matrix.

Q_1 \ Q_2	**0.003 (A)**	-0.007 (B)	0.0001 (C)
0.003(A)	**8.57**	-11.46	-11.43
-0.007(B)	-11.42	0.54	0.54
0.0001(C)	-11.42	0.55	8.53

(b) ResQ Q_1, Q_2, Q_{jt}

Z_1 \ Z_2	**0.68(A)**	-0.27(B)	0.66(C)
0.68(A)	**8.52**	-11.59	-11.71
-0.27(B)	-11.39	0.53	0.44
0.66(C)	-11.35	0.53	8.43

(c) ResZ: $\mathbb{E}[Z_{tot}], \mathbb{E}[Z_1], \mathbb{E}[Z_2]$

Q_1 \ Q_2	-6.33(A)	-0.08(B)	**0.04(C)**
-6.72(A)	-9.48	-9.03	-8.99
-0.09(B)	-9.04	0.24	0.67
0.04(C)	-9.01	0.66	**8.44**

(d) DMIX: Q_1, Q_2, Q_{jt}

Q_1 \ Q_2	-6.31(A)	-0.00(B)	**1.64(C)**
-6.34(A)	-12.65	-6.34	-4.71
0.07(B)	-6.24	0.06	1.70
1.56(C)	-4.74	1.56	**3.20**

(e) DDN: Q_1, Q_2, Q_{jt}

Q_1 \ Q_2	**4.49(A)**	0.20(B)	4.47
2.12(A)	**8.62**	4.33	8.60
0.12(B)	8.62	2.33	6.60
2.11(C)	8.61	4.32	8.59

(f) QTran: $Q_1, Q_2, Q_{\mathrm{jt}}$

Q_1 \ Q_2	0.08(A)	-134(B)	**0.11(C)**
0.07(A)	10.19	-8.96	-6.90
-134(B)	-7.65	5.55	5.43
0.01(C)	-6.02	5.43	**10.20**

(g) QPlex: Q_1, Q_2, Q_{jt}

Q_1 \ Q_2	-0.57(A)	-760.2(B)	**0.24(C)**
-0.58(A)	4.33	-4.46	4.39
-760.3(B)	-4.47	-4.48	-4.47
0.24(C)	4.36	0.48	**8.32**

(h) CW QMIX: Q_1, Q_2, Q_{jt}

Q_1 \ Q_2	**4.58(A)**	-17.83(B)	3.32(C)
6.73(A)	6.54	-0.30	6.15
-132(B)	-0.33	-0.33	-0.33
7.29(C)	**6.98**	0.10	6.59

(i) OW QMIX: Q_1, Q_2, Q_{jt}

circular observation around the agent, and attack enemies close-by. In each episode, if the agents eliminate all the enemies within a limited time, the game episode is counted as won. Otherwise, the game is counted as a failure. For the SMAC tasks, the agents must learn the policies which maximize their expected total rewards. The reward of SMAC is based on the damage dealt, enemy agents killed, and the game won. We adopt the default SMAC reward schema.

The Starcraft II version used in ResQ is 2.4.6.2.69232, which is the same version used in QMIX and Weighted QMIX. For pymarl2 [4], it used another version SC 2.4.10. As it is indicated in Weighted QMIX and confirmed by us, "*performance is not comparable across versions*". We have shown in Figure 5 (f), ResQ/ResZ perform better than the fine-tuned QMIX [4] for SC2.4.10 in the MMM2 scenario.

We have shown the results for ResQ, ResZ, QPLEX, QTran, QMIX, VDN, DMIX, OW QMIX, and CW QMIX. We have also evaluated the performance of DDN [2], and REFIL [5]. Their results are ploted in Figure 1. Due to the poor performance of REFIL, we only evaluate its performance on the MMM2, MMM, 3s_vs_5z, 1c3s5z, 8m_vs_9m scenarios. As it is shown in the picutre, ResQ performs better than REFIL and DDN.

Figure 2 depicts the results for three super-hard scenarios (Corridor, 3s5z_vs_3s6z, 6h_vs_8z) of the SMAC benchmark. As it is shown in the figure, all algorithms perform poorly for these super-hard scenarios. This indicates that better exploration techniques should be used together with the value factorization methods.

A.3 Comparison to DMIX, QTRAN, QPlex, and weighted QMIX

In this section, we compare ResQ with DMIX, QTRAN, QPlex, and weighted QMIX.

A.3.1 Larger DMIX

ResQ and ResZ use more parameters than DMIX with the residual functions. By modifying the dimension of the last layer of the hypernet inside DMIX, we have increased the number of parameters of DMIX from 85K to 350K, which is bigger than that of ResQ (320K) and that of ResZ(316K). Let's denote DMIX with more parameters as DMIX-larger, and test its performance in the MMM2, the MMM, and the 3s_vs_5z scenarios. The experimental results are depicted in Figure 3. DMIX-larger performs slightly better than DMIX in the MMM2 scenario but worse than DMIX in the MMM and the 3s_vs_5z scenarios. ResQ and ResZ performs better than DMIX and DMIX-larger in these scenarios. This means that the performance improvement of ResQ over DMIX does not come from the use of more parameters.

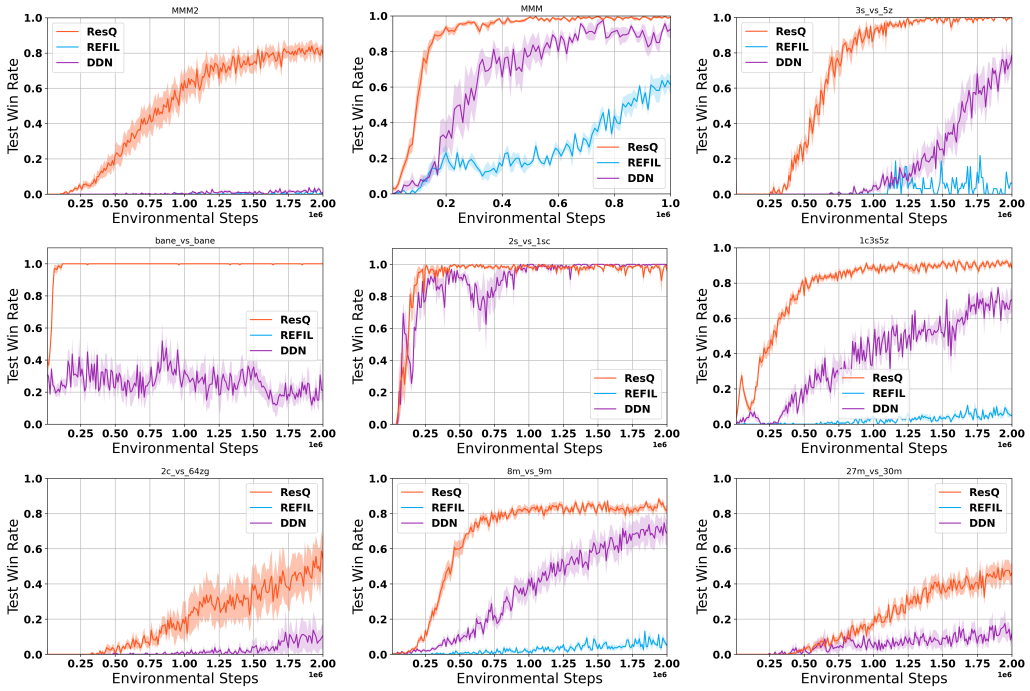

Figure 1: The test win rate of ResQ, DDN and REFIL for the SMAC benchmark.

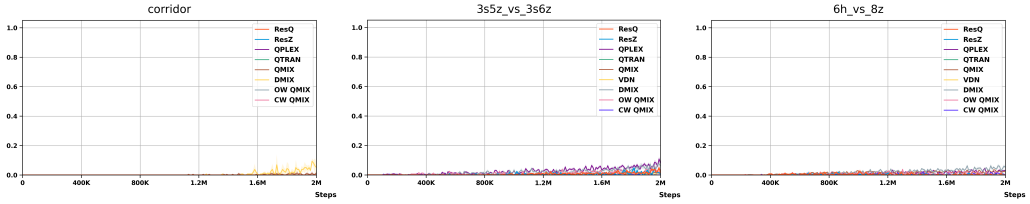

Figure 2: The test win rate of different algorithms for the SMAC benchmark: Corridor (Left), 3s5z_vs_3s6z (Middle), and 6h_vs_8z (Right).

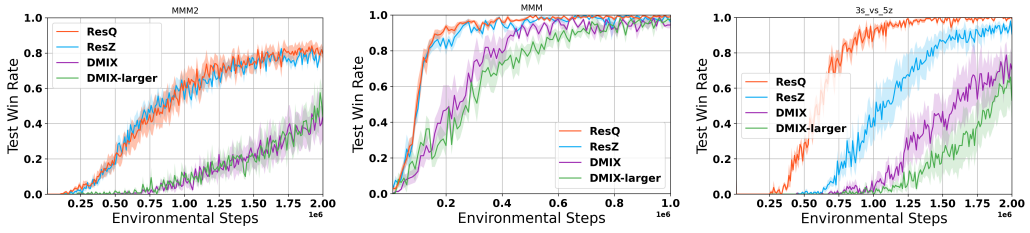

Figure 3: The test win rate of ResQ, ResZ, DMIX and DMIX-larger for the SMAC benchmark.

A.3.2 Comparison to QTRAN

We can reformulate QTRAN in the form of ResQ. QTRAN approximates the true value function $Q(\boldsymbol{\tau}, \boldsymbol{u})$ via $Q_{tran}(\boldsymbol{\tau}, \boldsymbol{u})$. $Q_{tran}(\boldsymbol{\tau}, \boldsymbol{u}) = Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) + w_r(\boldsymbol{\tau}, \boldsymbol{u})V(\boldsymbol{\tau})$, where $Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) = \sum_i Q_i(\tau_i, u_i)$ and $w_r(\boldsymbol{\tau}, \boldsymbol{u}) = 1$. And it must satisfy the inequality condition from 4b of Theorem 1 of QTRAN, which requires that $Q_{tran}(\boldsymbol{\tau}, \boldsymbol{u}) \geq Q(\boldsymbol{\tau}, \boldsymbol{u})$ for all non-optimal actions. Thus, QTRAN could over-estimate state-action value pairs. Further, QTRAN uses MSE loss to implement the inequality conditions, which could lead to violation of the IGM principle.

ResQ learns $Q_{jt}(\boldsymbol{\tau}, \boldsymbol{u})$ to approximate the state-action value function $Q(\boldsymbol{\tau}, \boldsymbol{u})$. Theorem 1 in ResQ shows that the state-action value function $Q_{jt}(\boldsymbol{\tau}, \boldsymbol{u})$ satisfy $Q_{jt}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) \geq Q_{jt}(\boldsymbol{\tau}, \boldsymbol{u})$, where $\bar{\boldsymbol{u}} = [\bar{u}_i]_{i=1}^n \quad \bar{u}_i = \text{argmax}_{u_i} Q_i(\tau_i, u_i)$. Further, as it is shown in Formula 8 of Theorem 2 in

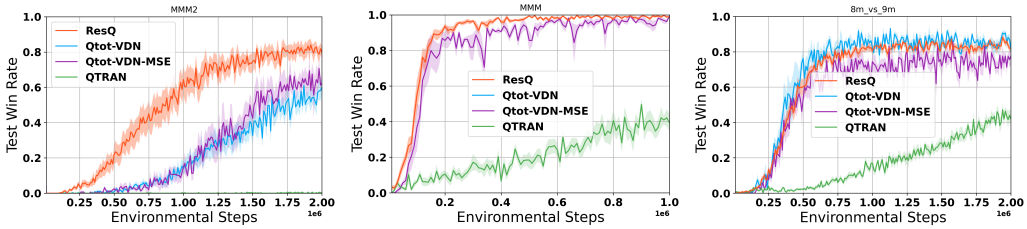

Figure 4: The test win rate of ResQ, Qtot-VDN, Qtot-VDN-MSE, and QTran for the SMAC benchmark.

ResQ, $Q_{jt}(\boldsymbol{\tau}, \boldsymbol{u}) = Q(\boldsymbol{\tau}, \boldsymbol{u}) \quad \forall u \neq \bar{\boldsymbol{u}}$, ResQ can find a $Q_{jt}(\boldsymbol{\tau}, \boldsymbol{u})$ that matches the sub-optimal state-actions of $Q(\boldsymbol{\tau}, \boldsymbol{u})$ closely. QTRAN cannot guarantee that the learned approximated function Q_{tran} satisfies this property. Further, ResQ uses $Q_r(\tau, u)$ to model the residual part; it has more input (i.e., u); it can consider the impact of actions. Thus, ResQ is more flexible than the residual part $V(\tau)$ of QTRAN.

To study the reason why ResQ performs better than QTRAN, We use two variants of ResQ: Qtot-VDN and Qtot-VDN-MSE. Qtot-VDN uses VDN as the main function same as QTRAN. The difference between Qtot-VDN between QTran is the residual function, the mask, and the inequality conditions. Qtot-VDN-MSE uses MSE loss to implement $Q_r \leq 0$, and it uses VDN as its main function. Their results are depicted in Figure 4. For MMM2, Qtot-VDN can obtain a win rate of 0.6. The win rate for QTRAN is 0. For the 8m_vs_9m scenario, Qtot-VDN can obtain a win rate of 0.8. And QTRAN can obtain a win rate of 0.4. The performance gap does not come from the implementation of the main function; it comes from the residual function, the mask function, and the inequality conditions. Further, Qtot-VDN-MSE performs similarly to Qtot-VDN, and it performs better than QTRAN. This suggests that the performance gap between ResQ and QTRAN does not come from the implementation of inequality conditions for ResQ.

A.3.3 Comparison to QPlex

ResQ can be viewed as a generalization of QPlex. According to QPlex, QPlex learns a function $Q_{plex}(\tau, u)$ to approximate the state-action value function $Q(\tau, u)$. $Q_{plex}(\tau, u) = V(\tau) + Adv(\tau, u)$, where $V(\tau) = max_u Q(\tau, u)$.

According to Formula (11) of the QPlex paper [6], we can rewrite Q_{plex} in the form of ResQ as

$$Q_{plex}(\tau, u) = Q_{tot}(\tau, u) + w_r^{plex}(\tau, u) Q_r^{plex}(\tau, u) \tag{46}$$

where $w_r^{plex}(\tau, u) = 1$, $Q_{tot}(\tau, u) = \sum_i Q_i(\tau, u_i)$ and $Q_r^{plex}(\tau, u) = \sum_i (\lambda_i - 1) A_i(\tau, u_i)$. Q_{tot} is implemented as a summation of the utilities, which has limited representation abilities. To ensure the IGM principle, QPlex places restrictions $A_i(\tau, u_i) = Q_i(\tau, u_i) - max_{u_i} Q_i(\tau, u_i)$ among the main and the residual function. Different from QPlex, ResQ does not require any relationships among the main and the residual functions, it only requires that $Q_r \leq 0$.

In summary, ResQ places fewer restrictions on the relationships between the main and the residual function than QPlex. And ResQ can use a more expressive neural network to model the main function than QPlex. Thus, ResQ can model Q value function in a better and more flexible way than QPlex.

A.3.4 Comparison to weighted QMIX

CW/OW QMIX can be viewed as variants of ResQ (see Formula 4 in Section 4). CW/OW QMIX learns $Q_{wqmix} = w_{tot}(\tau, u) Q_{tot}(\tau, u) + (1 - w_{tot}(\tau, u)) Q_r(\tau, u)$ to approximate the true value function, where $w_{tot}(\tau, \bar{u}) = 1$ and $w_{tot}(\tau, u) = 0 \quad \forall u \neq \bar{u}$. They assign high learning priorities to Q_{tot}, which puts the learning of sub-optimal state-action pairs and Q_{wqmix} in trouble.

A.3.5 Inequality condition implementations

ResQ does not significantly affect by the implementation of the inequality conditions (6b of Theorem 1). We have evaluated the performance of a variant of ResQ, ResQ-MSE, which uses the MSE loss to implement $Q_r \leq 0$. For ResQ-MSE, using MSE loss could lead to potential violation of the

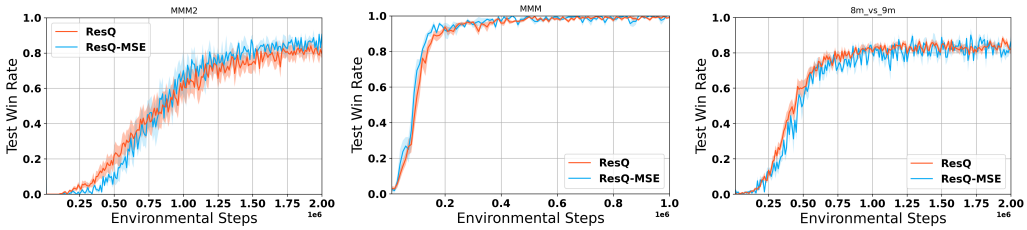

Figure 5: The test win rate of ResQ and ResQ-MSE for the SMAC benchmark.

inequality condition. We have studied the performance of ResQ-MSE in the MMM2, MMM, and 8m_vs_9m scenarios. The results are depicted in Figure 5. The performance of ResQ-MSE and ResQ are similar in these three scenarios. This indicates that using MSE loss does not significantly affect the performance for these two scenarios for ResQ.

A.3.6 Limitations

Knowing the optimal action over Q is computationally intractable, we make approximations to derive a practical algorithm. It is difficult for ResQ to find the optimal actions for a scenario requiring highly-coordinated agent exploration. The approximated mask function will fail if the approximated optimal action differs significantly from the optimal action. In ResQ, we assume that the argmax operator of Q_{tot} can lead to correct optimal actions. However, this assumption does not always hold. For a discrete-action environment with a tabular Q value, we think that all state-action values can be factorized using ResQ if the masking function can find the correct optimal action. As we use neural networks to represent states and actions, the approximation error of neural networks can interact with the error of the approximated argmax operator. This may make ResQ fail. We think that combining efficient-exploration approaches (e.g., MAVEN) with ResQ can make the mask function more robust than using ResQ alone.

References

[1] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted QMIX: expanding monotonic value function factorisation for deep multi-agent reinforcement learning. In *NeurIPS*, 2020.

[2] Wei-Fang Sun, Cheng-Kuang Lee, and Chun-Yi Lee. DFAC framework: Factorizing the value function via quantile mixture for multi-agent distributional q-learning. In *ICML*, 2021.

[3] Mikayel Samvelyan, Tabish Rashid, Christian Schröder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob N. Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *AAMAS*, pages 2186–2188, 2019.

[4] Jian Hu, Siyang Jiang, Seth Austin Harding, Haibin Wu, and Shih-wei Liao. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning, 2021.

[5] Shariq Iqbal, Christian A. Schröder de Witt, Bei Peng, Wendelin Boehmer, Shimon Whiteson, and Fei Sha. Randomized entity-wise factorization for multi-agent reinforcement learning. In *ICML*, 2021.

[6] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. QPLEX: duplex dueling multi-agent q-learning. In *ICLR*, 2021.