Supplementary

A. QTRAN Training Algorithm

The training algorithms for QTRAN-base and QTRAN-alt are provided in Algorithm 1.

Algorithm 1 QTRAN-base and QTRAN-alt

Initialize replay memory D
Initialize $[Q_i]$, Q_{jt}, and V_{jt} with random parameters θ
Initialize target parameters $\theta^- = \theta$
for episode = 1 to M **do**
  Observe initial state $\boldsymbol{s}^0$ and observation $\boldsymbol{o}^0 = [O(\boldsymbol{s}^0, i)]_{i=1}^N$ for each agent i
  for $t = 1$ to T **do**
    With probability ϵ select a random action u_i^t
    Otherwise $u_i^t = \arg\max_{u_i^t} Q_i(\tau_i^t, u_i^t)$ for each agent i
    Take action $\boldsymbol{u}^t$, and retrieve next observation and reward $(\boldsymbol{o}^{t+1}, r^t)$
    Store transition $(\boldsymbol{\tau}^t, \boldsymbol{u}^t, r^t, \boldsymbol{\tau}^{t+1})$ in D
    Sample a random minibatch of transitions $(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau'})$ from D
    Set $y^{\text{dqn}}(r, \boldsymbol{\tau}'; \boldsymbol{\theta^-}) = r + \gamma Q_{\text{jt}}(\boldsymbol{\tau}', \bar{\boldsymbol{u}}'; \boldsymbol{\theta}^-)$, $\bar{\boldsymbol{u}}' = [\arg\max_{u_i} Q_i(\tau_i', u_i; \boldsymbol{\theta}^-)]_{i=1}^N$,
    If QTRAN-base, update θ by minimizing the loss:

$$
\begin{aligned}
L(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}'; \boldsymbol{\theta}) &= L_{\text{td}} + \lambda_{\text{opt}} L_{\text{opt}} + \lambda_{\text{nopt}} L_{\text{nopt}}, \\
L_{\text{td}}(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}'; \boldsymbol{\theta}) &= \left(Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - y^{\text{dqn}}(r, \boldsymbol{\tau}'; \boldsymbol{\theta}^-)\right)^2, \\
L_{\text{opt}}(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}'; \boldsymbol{\theta}) &= \left(Q'_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) - \hat{Q}_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) + V_{\text{jt}}(\boldsymbol{\tau})\right)^2, \\
L_{\text{nopt}}(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}'; \boldsymbol{\theta}) &= \left(\min\left[Q'_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - \hat{Q}_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) + V_{\text{jt}}(\boldsymbol{\tau}), 0\right]\right)^2.
\end{aligned}
$$

    If QTRAN-alt, update θ by minimizing the loss:

$$
\begin{aligned}
L(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}'; \boldsymbol{\theta}) &= L_{\text{td}} + \lambda_{\text{opt}} L_{\text{opt}} + \lambda_{\text{nopt-min}} L_{\text{nopt-min}}, \\
L_{\text{td}}(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}'; \boldsymbol{\theta}) &= \left(Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - y^{\text{dqn}}(r, \boldsymbol{\tau}'; \boldsymbol{\theta}^-)\right)^2, \\
L_{\text{opt}}(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}'; \boldsymbol{\theta}) &= \left(Q'_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) - \hat{Q}_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) + V_{\text{jt}}(\boldsymbol{\tau})\right)^2, \\
L_{\text{nopt-min}}(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}'; \boldsymbol{\theta}) &= \frac{1}{N}\sum_{i=1}^{N}\left(\min_{u_i \in \mathcal{U}}\left(Q'_{\text{jt}}(\boldsymbol{\tau}, u_i, \boldsymbol{u}_{-i}) - \hat{Q}_{\text{jt}}(\boldsymbol{\tau}, u_i, \boldsymbol{u}_{-i}) + V_{\text{jt}}(\boldsymbol{\tau})\right)\right)^2.
\end{aligned}
$$

    Update target network parameters $\theta^- = \theta$ with period I
  end for
end for

B. Proofs

In this section, we provide the proofs of theorems and propositions coming from theorems.

B.1. Proof of Theorem 1

Theorem 1. *A factorizable joint action-value function $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ is factorized by $[Q_i(\tau_i, u_i)]$, if*

$$
\sum_{i=1}^N Q_i(\tau_i, u_i) - Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) + V_{\text{jt}}(\boldsymbol{\tau}) = \begin{cases} 0 & \boldsymbol{u} = \bar{\boldsymbol{u}}, \quad (4a) \\ \geq 0 & \boldsymbol{u} \neq \bar{\boldsymbol{u}}, \quad (4b) \end{cases}
$$

where

$$V_{\text{jt}}(\boldsymbol{\tau}) = \max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - \sum_{i=1}^{N} Q_i(\tau_i, \bar{u}_i).$$

Proof. Theorem 1 shows that if condition (4) holds, then Q_i satisfies **IGM**. Thus, for some given Q_i that satisfies (4), we will show that $\arg\max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) = \bar{\boldsymbol{u}}$. Recall that $\bar{u}_i = \arg\max_{u_i} Q_i(\tau_i, u_i)$ and $\bar{\boldsymbol{u}} = [\bar{u}_i]_{i=1}^{N}$,.

$$\begin{aligned} Q_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) &= \sum_{i=1}^{N} Q_i(\tau_i, \bar{u}_i) + V_{\text{jt}}(\boldsymbol{\tau}) \quad \text{(From (4a))} \\ &\geq \sum_{i=1}^{N} Q_i(\tau_i, u_i) + V_{\text{jt}}(\boldsymbol{\tau}) \\ &\geq Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) \quad \text{(From (4b))}. \end{aligned}$$

It means that the set of optimal local actions $\bar{\boldsymbol{u}}$ maximizes Q_{jt}, showing that Q_i satisfies **IGM**. This completes the proof. □

B.2. Necessity in Theorem 1 Under Affine-transformation

As mentioned in Section 3.1, the conditions (4) in Theroem 1 are necessary under an affine transformation. The necessary condition shows that for some given factorizable Q_{jt}, there exists Q_i that satisfies (4), which guides us to design the QTRAN neural network. Note that the affine transformation ϕ is $\phi(\boldsymbol{Q}) = A \cdot \boldsymbol{Q} + B$, where $A = [a_{ii}] \in \mathbb{R}_{+}^{N \times N}$ is a symmetric diagonal matrix with $a_{ii} > 0, \forall i$ and $B = [b_i] \in \mathbb{R}^N$. To abuse notation, let $\phi(Q_i(\tau_i, u_i)) = a_{ii} Q_i(\tau_i, u_i) + b_i$.

Proposition 1. *If $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ is factorized by $[Q_i(\tau_i, u_i)]$, then there exists an affine transformation $\phi(\boldsymbol{Q})$ such that $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ is factorized by $[\phi(Q_i(\tau_i, u_i))]$ and the condition (4) holds by replacing $[Q_i(\tau_i, u_i)]$ with $[\phi(Q_i(\tau_i, u_i))]$.*

Proof. To prove, we will show that, for the factors $[Q_i]$ of Q_{jt}, there exists an affine transformation of Q_i that also satisfies conditions (4).

By definition, if $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ is factorized by $[Q_i(\tau_i, u_i)]$, then the followings hold: (i) $Q_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) - \max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) = 0$, (ii) $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - Q_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) < 0$, and (iii) $\sum_{i=1}^{N}(Q_i(\tau_i, u_i) - Q_i(\tau_i, \bar{u}_i)) < 0$ if $\boldsymbol{u} \neq \bar{\boldsymbol{u}}$. Now, we consider an affine transformation, in which $a_{ii} = \alpha$ and $b_i = 0\ \forall i$, where $\alpha > 0$, and $\phi(Q_i) = \alpha Q_i$ with this transformation. Since this is a linearly scaled transformation, it satisfies the **IGM** condition, and thus (4a) holds. We also prove that $\phi(Q_i)$ satisfies condition (4a) by showing that there exists a constant α small enough such that

$$\sum_{i=1}^{N} \alpha Q_i(\tau_i, u_i) - Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) + V_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) = \sum_{i=1}^{N} \alpha(Q_i(\tau_i, u_i) - Q_i(\tau_i, \bar{u}_i)) - (Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - Q_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}})) \geq 0,$$

where $V_{\text{jt}}(\boldsymbol{\tau})$ is redefined for linearly scaled αQ_i as $\max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - \sum_{i=1}^{N} \alpha Q_i(\tau_i, \bar{u}_i)$. This completes the proof. □

B.3. Special Case: Theorem 1 in Fully Observable Environments

If the task is a fully observable case (observation function is bijective for all i), the state-value network is not required and all $V_{\text{jt}}(\tau)$ values can be set to zero. We show that Theorem 1 holds equally for the case where $V_{\text{jt}}(\boldsymbol{\tau}) = 0$ for a fully observable case. This fully observable case is applied to our example of the simple matrix game. The similar necessity under an affine-transformation holds in this case.

Theorem 1a. (Fully observable case) *A factorizable joint action-value function $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ is factorized by $[Q_i(\tau_i, u_i)]$, if*

$$\sum_{i=1}^{N} Q_i(\tau_i, u_i) - Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) = \begin{cases} 0 & \boldsymbol{u} = \bar{\boldsymbol{u}}, \quad \text{(9a)} \\ \geq 0 & \boldsymbol{u} \neq \bar{\boldsymbol{u}}, \quad \text{(9b)} \end{cases}$$

Proof. We will show $\arg\max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) = \bar{\boldsymbol{u}}$ (*i.e.*, **IGM**), if the (9) holds.

$$Q_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) = \sum_{i=1}^{N} Q_i(\tau_i, \bar{u}_i) \geq \sum_{i=1}^{N} Q_i(\tau_i, u_i) \geq Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}).$$

The first equality comes from (9a), and the last inequality comes from (9b). We note that $\arg\max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) = [\arg\max_{u_i} Q_i(\tau_i, u_i)]$, so this completes the proof. □

Proposition 1a. *If $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ is factorized by $[Q_i(\tau_i, u_i)]$, then there exists an affine transformation $\phi(\boldsymbol{Q})$, such that $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ is factorized by $[\phi(Q_i(\tau_i, u_i))]$ and condition (9) holds by replacing $[Q_i(\tau_i, u_i)]$ with $[\phi(Q_i(\tau_i, u_i))]$.*

Proof. From Theorem 1a, if $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ is factorizable, then there exist $[Q_i]$ satisfying both **IGM** and (9). Now, we define an additive transformation $\phi'_i(Q_i(\tau_i, u_i)) = Q_i(\tau_i, u_i) + \frac{1}{N}\max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - Q_i(\tau_i, \bar{u}_i)$ for a given τ_i, which is uniquely defined for fully observable cases. $[\phi'_i(Q_i(\tau_i, u_i))]$ also satisfy **IGM**, and the left-hand side of (9) can be rewritten as:

$$\sum_{i=1}^{N} Q_i(\tau_i, u_i) - Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - \sum_{i=1}^{N} Q_i(\tau_i, \bar{u}_i) + \max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) = \sum_{i=1}^{N} \phi'_i(Q_i(\tau_i, u_i)) - Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$$

So there exist individual action-value functions $[\phi'_i(Q'_i(\tau_i, u_i))]$ that satisfy both **IGM** and (9), where $V_{\text{jt}}(\boldsymbol{\tau})$ is redefined as 0. This completes the proof of the necessity.

□

B.4. Proof of Theorem 2

Theorem 2. The statement presented in Theorem 1 and the necessary condition of Theorem 1 holds by replacing (4b) with the following (7): if $\boldsymbol{u} \neq \bar{\boldsymbol{u}}$,

$$\min_{u_i \in \mathcal{U}} \left[Q'_{\text{jt}}(\boldsymbol{\tau}, u_i, \boldsymbol{u}_{-i}) - Q_{\text{jt}}(\boldsymbol{\tau}, u_i, \boldsymbol{u}_{-i}) + V_{\text{jt}}(\boldsymbol{\tau}) \right] = 0, \quad \forall i = 1, \dots, N, \tag{7}$$

where $\boldsymbol{u}_{-i} = (u_1, \dots, u_{i-1}, u_{i+1}, \dots, u_N)$, *i.e.*, the action vector except for i's action.

Proof. ($\Rightarrow$) Recall that condition (7) is stronger than (4b), which is itself sufficient for Theorem 1. Therefore, by transitivity, condition (7) is sufficient for Theorem 2. Following paragraphs focus on the other direction, *i.e.*, how condition (7) is *necessary* for Theorem 2.

($\Leftarrow$) We prove that, if there exist individual action-value functions satisfying condition (4), then there exists an individual action-value function Q'_i that satisfies (7). In order to show the existence of such Q'_i, we propose a way to construct Q'_i.

We first consider the case with $N = 2$ and then generalize the result for any N. The condition (7) for $N = 2$ is denoted as:

$$\min_{u_i \in \mathcal{U}} \left[Q_1(\tau_1, u_1) + Q_2(\tau_2, u_2) - Q_{\text{jt}}(\boldsymbol{\tau}, u_1, u_2) + V_{\text{jt}}(\boldsymbol{\tau}) \right] = 0.$$

Since this way of constructing Q'_i is symmetric for all i, we present its construction only for u_1 without loss of generality. For Q_1 and Q_2 satisfying (4), if $\beta := \min_{u_1 \in \mathcal{U}} \left[Q_1(\tau_1, u_1) + Q_2(\tau_2, u_2) - Q_{\text{jt}}(\boldsymbol{\tau}, u_1, u_2) + V_{\text{jt}}(\boldsymbol{\tau}) \right] > 0$ for given $\boldsymbol{\tau}$ and u_2, then $u_2 \neq \bar{u}_2$. This is because $Q_1(\tau_1, \bar{u}_1) + Q_2(\tau_2, \bar{u}_2) - Q_{\text{jt}}(\boldsymbol{\tau}, \bar{u}_1, \bar{u}_2) + V_{\text{jt}}(\boldsymbol{\tau}) = 0$ by condition (4a). Now, we replace $Q_2(\tau_2, u_2)$ with $Q'_2(\tau_2, u_2) = Q_2(\tau_2, u_2) - \beta$. Since $Q_2(\tau_2, \bar{u}_2) > Q_2(\tau_2, u_2) > Q_2(\tau_2, u_2) - \beta$, it does not change the optimal action and other conditions. Then, (7) is satisfied for given $\boldsymbol{\tau}$ and u_2. By repeating this replacement process, we can construct Q'_i that satisfies condition (7).

More generally, when $N \neq 2$, if $\min_{u_i \in \mathcal{U}} \left[Q'_{\text{jt}}(\boldsymbol{\tau}, u_i, \boldsymbol{u}_{-i}) - Q_{\text{jt}}(\boldsymbol{\tau}, u_i, \boldsymbol{u}_{-i}) + V_{\text{jt}}(\boldsymbol{\tau}) \right] = \beta > 0$ for given $\boldsymbol{\tau}$ and $\boldsymbol{u}_{-i}$, then there exists some $j \neq i$ that satisfies $u_j \neq \bar{u}_j$. Therefore, by repeating the same process as when $N = 2$ through j, we can construct Q'_i for all i, and this confirms that individual action-value functions satisfying condition (7) exist. This completes the proof.

□

B.4.1. Process of constructing Q'_i in the matrix game using Theorem 2

We now present the process of how we have demonstrated through the example in Section 3.5. In the original matrix shown in Tables 2a and 2d, the second row does not satisfy the condition (7), and $\beta = 0.23$ for $u_1 = B$. Then, we replace $Q_1(B)$ as shown in Table 2e. Table 2b shows that its third row does not satisfy the condition (7). Finally, we replace $Q_1(C)$ as shown in Table 2f. Then, the resulting Table 2c satisfies the condition (7).

u_1 \ u_2	A	B	C
A	0.00	18.14	18.14
B	**14.11**	**0.23**	**0.23**
C	13.93	0.05	0.05

(a) $Q'_{\text{jt}} - Q_{\text{jt}}$

u_1 \ u_2	A	B	C
A	0.00	18.14	18.14
B	13.88	0.00	0.00
C	**13.93**	**0.05**	**0.05**

(b) $Q'_{\text{jt}} - Q_{\text{jt}}$ after one replacement

u_1 \ u_2	A	B	C
A	0.00	18.14	18.14
B	13.88	0.00	0.00
C	13.88	0.00	0.00

(c) $Q'_{\text{jt}} - Q_{\text{jt}}$ after two replacements

Q_1 \ Q_2	4.16(A)	2.29(B)	2.29(C)
3.84(A)	8.00	6.13	6.12
-2.06(B)	2.10	0.23	0.23
-2.25(C)	1.92	0.04	0.04

(d) Q_1, Q_2, Q'_{jt}

Q_1 \ Q_2	4.16(A)	2.29(B)	2.29(C)
3.84(A)	8.00	6.13	6.12
-2.29(B)	1.87	0.00	0.00
-2.25(C)	1.92	0.04	0.04

(e) Q_1, Q_2, Q'_{jt} after one replacement

Q_1 \ Q_2	4.16(A)	2.29(B)	2.29(C)
3.84(A)	8.00	6.13	6.12
-2.29(B)	1.87	0.00	0.00
-2.30(C)	1.87	-0.01	-0.01

(f) Q_1, Q_2, Q'_{jt} after two replacements

Table 2. The process of replacing $[Q_i]$ satisfying the condition (9b) with $[Q_i]$ satisfying the condition (7)

C. Details of environments and implementation

C.1. Environment

Matrix game In order to see the impact of QTRAN-alt, we train the agents in a single state matrix game where two agents each have 21 actions. Each agent i takes action u_i, ranging over $\in \{0, ..., 20\}$. The reward value R for a joint action is given as follows:

$$f_1(u_1, u_2) = 5 - \left(\frac{15 - u_1}{3}\right)^2 - \left(\frac{5 - u_2}{3}\right)^2$$
$$f_2(u_1, u_2) = 10 - \left(\frac{5 - u_1}{1}\right)^2 - \left(\frac{15 - u_2}{1}\right)^2$$
$$R(u_1, u_2) = \max(f_1(u_1, u_2), f_2(u_1, u_2))$$

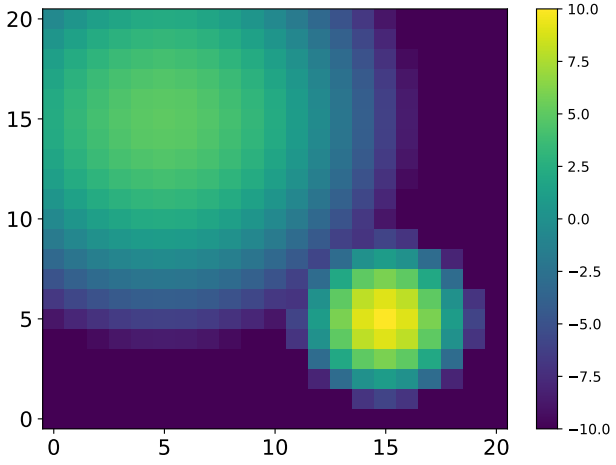
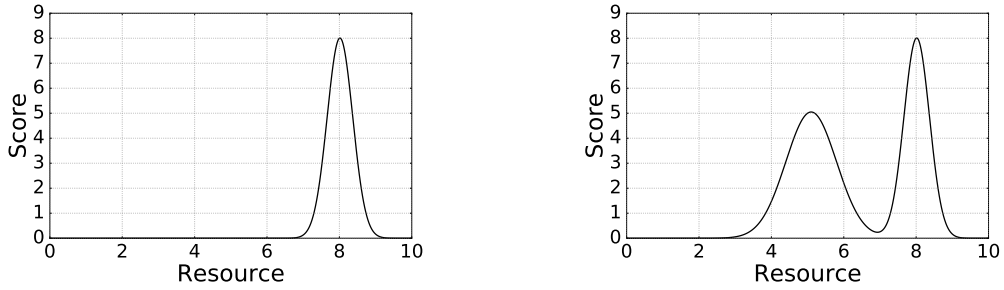


Figure 5. x-axis and y-axis: agents 1 and 2's actions, respectively. Colored values represent the reward for selected actions.

Figure 5 shows reward for selected action. Colored values represent the values. In the simple matrix game, the reward function has a global maximum point at $(u_1, u_2) = (5, 15)$ and a local maximum point at $(u_1, u_2) = (15, 5)$.

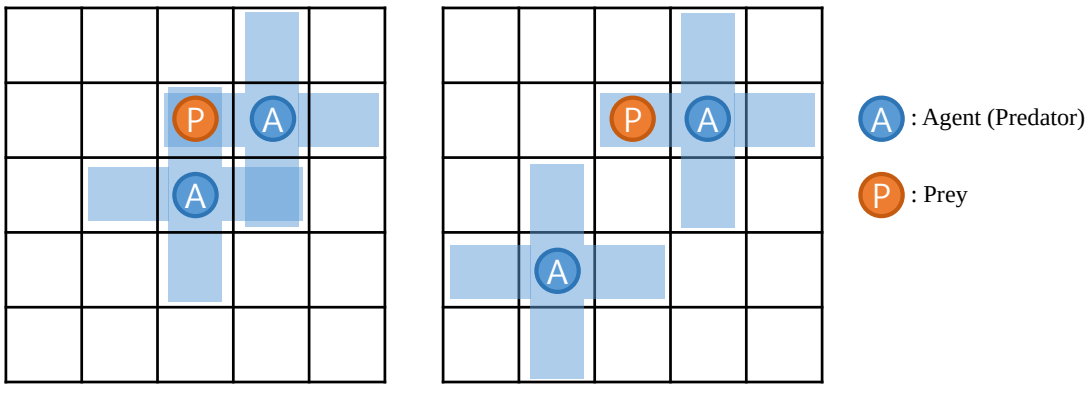
Multi-domain Gaussian Squeeze We adopt and modify Gaussian Squeeze, where agent numbers ($N = 10$) and action spaces ($u_i \in \{0, 1, ..., 9\}$) are much larger than a simple matrix game. In MGS, each of the ten agents has its own amount of unit-level resource $s_i \in [0, 0.2]$ given by the environment *a priori*. This task covers a fully observable case where all agents can see the entire state. We assume that there exist K domains, where the above resource allocation takes place. The joint action $\boldsymbol{u}$ determines the overall resource usage $x(\boldsymbol{u}) = \sum_i s_i \times u_i$. Reward is given as a function of resource usage as follows: $G(\boldsymbol{u}) = \sum_{k=1}^{K} x e^{-(x-\mu_k)^2/{\sigma_k}^2}$. We test with two different settings: (i) $K = 1, \mu_1 = 8, \sigma_1 = 1$ as shown in Figure 6a, and (ii) $K = 2, \mu_1 = 8, \sigma_1 = 0.5, \mu_2 = 5, \sigma_2 = 1$ as shown in Figure 6b. In the second setting, there are two local maxima for resource x. The maximum on the left is relatively easy to find through exploration – as manifested in the greater variance of the Gaussian distribution, but the maximum reward — as represented by the lower peak — is relatively low. On the other hand, the maximum on the right is more difficult to find through exploration, but it offers higher reward.



(a) Gaussian Squeeze (b) Multi-domain Gaussian Squeeze

Figure 6. Gaussian Squeeze and Multi-domain Gaussian Squeeze

Modified predator-prey Predator-prey involves a grid world, in which multiple predators attempt to capture randomly moving prey. Agents have a 5×5 view and select one of five actions $\in$ {Left, Right, Up, Down, Stop} at each time step. Prey move according to selecting a uniformly random action at each time step. We define the "catching" of a prey as when the prey is within the cardinal direction of at least one predator. Each agent's observation includes its own coordinates, agent ID, and the coordinates of the prey relative to itself, if observed. The agents can separate roles even if the parameters of the neural networks are shared by agent ID. We test with two different grid worlds: (i) a 5×5 grid world with two predators and one prey, and (ii) a 7×7 grid world with four predators and two prey. We modify the general predator-prey, such that a positive reward is given only if multiple predators catch a prey simultaneously, requiring a higher degree of cooperation. The predators get a team reward of 1 if two or more catch a prey at the same time, but they are given negative reward $-P$, if only one predator catches the prey as shown in Figure 7. We experimented with three varying P vales, where $P = 0.5, 1.0, 1.5$. The terminating condition of this task is when a prey is caught with more than one predator. The prey that has been caught is regenerated at random positions whenever the task terminates, and the game proceeds over fixed 100 steps.



Reward = +1 Reward = -P

Figure 7. Predator-prey environment

C.2. Experiment details

Matrix game Table 3 shows the values of the hyperparameters for the training in the matrix game environment. Individual action-value networks, which are common in all VDN, QMIX, and QTRAN, each consist of two hidden layers. In addition to the individual Q-networks, QMIX incorporates a monotone network with one hidden layer, and the weights and biases of this network are produced by separate hypernetworks. QTRAN has an additional joint action-value network with two hidden layers. All hidden layer widths are 32, and all activation functions are ReLU. All neural networks are trained using the Adam optimizer. We use ϵ-greedy action selection with $\epsilon = 1$ independently for exploration.

Hyperparameter	Value	Description
training step	20000	Maximum time steps until the end of training
learning rate	0.0005	Learning rate used by Adam optimizer
replay buffer size	20000	Maximum number of samples to store in memory
minibatch size	32	Number of samples to use for each update
λ_{opt}, λ_{nopt}	1,1	Weight constants for loss functions L_{opt}, L_{nopt} and $L_{nopt-min}$

Table 3. Hyperparameters for matrix game training

Multi-domain Gaussian Squeeze Table 4 shows the values of the hyperparameters for the training in the MGS environment. Each individual action-value network consists of three hidden layers. In addition to the individual Q-networks, QMIX incorporates a monotone network with one hidden layer, and the weights and biases of this network are produced by separate hypernetworks. QTRAN has an additional joint action-value network with two hidden layers. All hidden layer widths are 64, and all activation functions are ReLU. All neural networks are trained using the Adam optimizer. We use ϵ-greedy action selection with decreasing ϵ from 1 to 0.1 for exploration.

Hyperparameter	Value	Description
training step	1000000	Maximum time steps until the end of training
learning rate	0.0005	Learning rate used by Adam optimizer
replay buffer size	200000	Maximum number of samples to store in memory
final exploration step	500000	Number of steps over which ϵ is annealed to the final value
minibatch size	32	Number of samples to use for each update
λ_{opt}, λ_{nopt}	1,1	Weight constants for loss functions L_{opt}, L_{nopt} and $L_{nopt-min}$

Table 4. Hyperparameters for Multi-domain Gaussian Squeeze

Modified predator-prey Table 5 shows the values of the hyperparameters for the training in the modified predator-prey environment. Each individual action-value network consists of three hidden layers. In addition to the individual Q-networks, QMIX incorporates a monotone network with one hidden layer, and the weights and biases of this network are produced by separate hypernetworks. QTRAN has additional joint action-value network with two hidden layers. All hidden layer widths are 64, and all activation functions are ReLU. All neural networks are trained using the Adam optimizer. We use ϵ-greedy action selection with decreasing ϵ from 1 to 0.1 for exploration. Since history is not very important in this task, our experiments use feed-forward policies, but our method is also applicable with recurrent policies.

Hyperparameter	Value	Description
training step	10000000	Maximum time steps until the end of training
discount factor	0.99	Importance of future rewards
learning rate	0.0005	Learning rate used by Adam optimizer
target update period	10000	Target network update period to track learned network
replay buffer size	1000000	Maximum number of samples to store in memory
final exploration step	3000000	Number of steps over which ϵ is annealed to the final value
minibatch size	32	Number of samples to use for each update
λ_{opt}, λ_{nopt}	1,1	Weight constants for loss functions L_{opt}, L_{nopt} and $L_{nopt-min}$

Table 5. Hyperparameters for predator-prey training

D. Additional results for matrix games

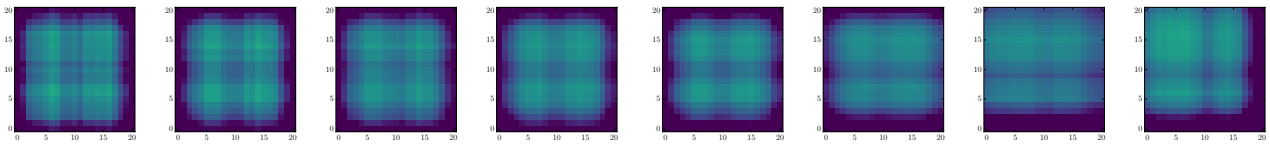
Table 6 shows the comparison between the final performance levels of VDN, QMIX, and QTRAN-base for 310 3×3 random matrix games, where each value of the payoff matrix is given as a random parameter from 0 to 1. Experimental settings are the same as in the previous matrix game. Since matrix games always satisfy **IGM** conditions, QTRAN-base always trains the optimal action for all 310 cases. On the other hand, VDN and QMIX were shown to be unable to learn an optimal policy for more than half of non-monotonic matrix games.

We briefly analyze the nature of the structural constraints assumed by VDN and QMIX, namely, additivity and monotonicity of the joint action-value functions. There have been only 19 cases in which the results of VDN and QMIX differ from each other. Interestingly, for a total of five cases, the performance of VDN is better than QMIX. QMIX was shown to outperform VDN in more cases (14) than the converse case (5). This supports the idea that the additivity assumption imposed by VDN on the joint action-value functions is indeed stronger than the monotonicity assumption imposed by QMIX.

QTRAN=VDN=QMIX	QTRAN>VDN=QMIX	VDN>QMIX	QMIX>VDN
114	177	5	14

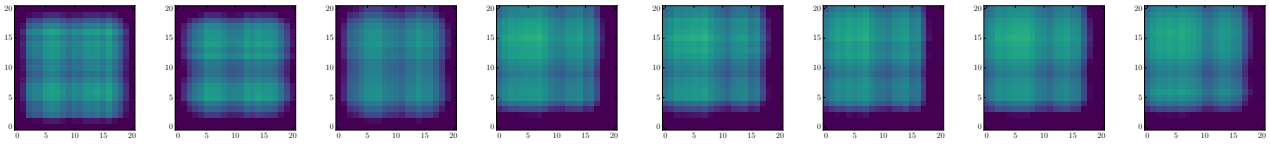
Table 6. Final performance comparison with 310 random matrices

Figures 8-9 show the joint action-value function of VDN and QMIX, and Figures 10-11 show the transformed joint action-value function of QTRAN-base and QTRAN-alt for a matrix game where two agents each have 20 actions. In the result, VDN and QMIX can not recover joint action-value, and these algorithms learn sub-optimal policy $u_1, u_2 = (15, 5)$. In the other hand, the result shows that QTRAN-base and QTRAN-alt successfully learn the optimal action, but QTRAN-alt has the ability to more accurately distinguish action from non-optimal action as shown in Figure 11.



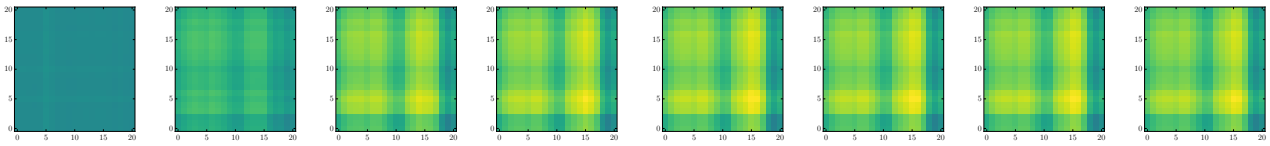
(a) 1000 step (b) 2000 step (c) 3000 step (d) 4000 step (e) 5000 step (f) 6000 step (g) 7000 step (h) 8000 step

Figure 8. x-axis and y-axis: agents 1 and 2's actions. Colored values represent the values of Q_{jt} for VDN



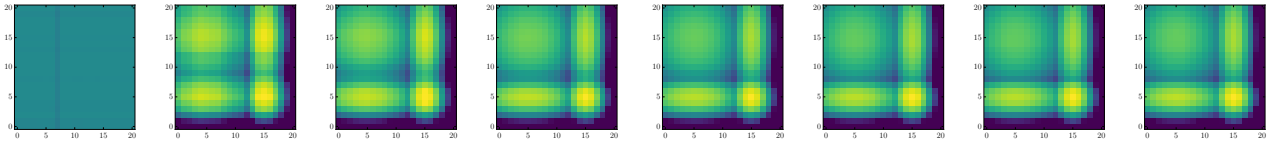
(a) 1000 step (b) 2000 step (c) 3000 step (d) 4000 step (e) 5000 step (f) 6000 step (g) 7000 step (h) 8000 step

Figure 9. x-axis and y-axis: agents 1 and 2's actions. Colored values represent the values of Q_{jt} for QMIX



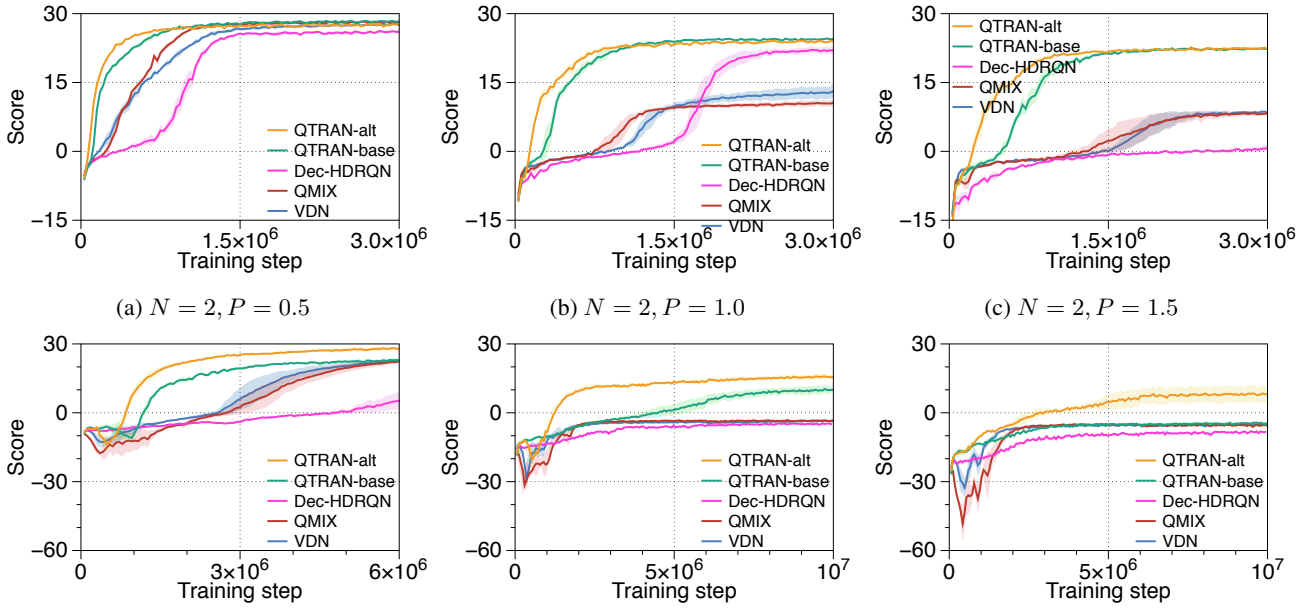
(a) 1000 step (b) 2000 step (c) 3000 step (d) 4000 step (e) 5000 step (f) 6000 step (g) 7000 step (h) 8000 step

Figure 10. x-axis and y-axis: agents 1 and 2's actions. Colored values represent the values of Q'_{jt} for QTRAN-base



(a) 1000 step (b) 2000 step (c) 3000 step (d) 4000 step (e) 5000 step (f) 6000 step (g) 7000 step (h) 8000 step

Figure 11. x-axis and y-axis: agents 1 and 2's actions. Colored values represent the values of Q'_{jt} for QTRAN-alt



(a) $N = 2, P = 0.5$ (b) $N = 2, P = 1.0$ (c) $N = 2, P = 1.5$

(d) $N = 4, P = 0.5$ (e) $N = 4, P = 1.0$ (f) $N = 4, P = 1.5$

Figure 12. Average reward per episode on the MPP tasks with 95% confidence intervals for VDN, QMIX, Dec-HDRQN and QTRAN

E. Comparison with other value-based methods for modified predator-prey

Additionally, we have conducted experiments with Dec-HDRQN (Omidshafiei et al., 2017). Dec-HDRQN can indeed solve problems similar to ours by changing the learning rate according to TD-error without factorization. However, Dec-HDRQN does not take advantage of centralized training. We implemented Dec-HDRQN ($\alpha = 0.001, \beta = 0.0002$) with modified predator-prey experiments and Figure 12 shows the performance of algorithms for six settings with different N and P values.

First, when the complexity of the task is relatively low, Dec-HDRQN shows better performance than VDN and QMIX as shown in the Figure 12b. However, QTRAN performs better than Dec-HDRQN in the case. When the penalty and the number of agents are larger, Dec-HDRQN underperforms VDN and QMIX. Figure 12c shows Dec-HDRQN scores an average of nearly 0 in the case of $N = 2, P = 1.5$. There is a limit of Dec-HDRQN since the method is heuristic and does not perform centralized training. Finally, Dec-HDRQN showed slower convergence speed overall.