RMIX: RISK-SENSITIVE MULTI-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Centralized training with decentralized execution (CTDE) has become an important paradigm in multi-agent reinforcement learning (MARL). Current CTDE-based methods rely on restrictive decompositions of the centralized value function across agents, which decomposes the global Q-value into individual Q values to guide individuals’ behaviours. However, such expected, i.e., risk-neutral, Q value decomposition is not sufficient even with CTDE due to the randomness of rewards and the uncertainty in environments, which causes the failure of these methods to train coordinating agents in complex environments. To address these issues, we propose RMIX, a novel cooperative MARL method with the Conditional Value at Risk (CVaR) measure over the learned distributions of individuals’ Q values. Our main contributions are in three folds: (i) We first learn the return distributions of individuals to analytically calculate CVaR for decentralized execution; (ii) We then propose a dynamic risk level predictor for CVaR calculation to handle the temporal nature of the stochastic outcomes during executions; (iii) We finally propose risk-sensitive Bellman equation along with Individual-Global-MAX (IGM) for MARL training. Empirically, we show that our method significantly outperforms state-of-the-art methods on many challenging StarCraft II tasks, demonstrating significantly enhanced coordination and high sample efficiency. Demonstrative videos and results are available in this anonymous link: https://sites.google.com/view/rmix

1 INTRODUCTION

Reinforcement learning (RL) has made remarkable advances in many domains, including arcade video games (Mnih et al., 2015), complex continuous robot control (Lillicrap et al., 2016) and the game of Go (Silver et al., 2017). Recently, many researchers put their efforts to extend the RL methods into multi-agent systems (MASs), such as urban systems (Singh et al., 2020), coordination of robot swarms (Hüttenrauch et al., 2017) and real-time strategy (RTS) video games (Vinyals et al., 2019). Centralized training with decentralized execution (CTDE) (Oliehoek et al., 2008; Kraemer & Banerjee, 2016) has drawn enormous attention via training policies of each agent with access to global trajectories in a centralized way and executing actions given only the local observations of each agent in a decentralized way. Empowered by CTDE, several MARL methods, including value-based and policy gradient-based, are proposed (Foerster et al., 2017a; Sunehag et al., 2017; Rashid et al., 2018; Son et al., 2019). These MARL methods propose decomposition techniques to factorize the global Q value either by structural constraints or by estimating state-values or inter-agent weights to conduct the global Q value estimation. Among these methods, VDN (Sunehag et al., 2017) and QMIX (Rashid et al., 2018) are representative methods that use additivity and monotonicity structure constraints, respectively. With relaxed structural constraints, QTRAN (Son et al., 2019) guarantees a more general factorization than VDN and QMIX. Some other methods include incorporating an estimation of advantage values (Wang et al., 2020a) and proposing a multi-head attention method to represent the global values (Yang et al., 2020).

Despite the merits, most of these works focus on decomposing the global Q value into individual Q values with different constraints and network architectures, but ignore the fact that the expected, i.e., risk-neutral, value decomposition is not sufficient even with CTDE due to the randomness of rewards and the uncertainty in environments, which causes the failure of these methods to train coordinating agents in complex environments. Specifically, these methods only learn the expected values over
returns (Rashid et al. 2018) and do not handle the high variance caused by events with extremely high/low rewards to agents but small probabilities, which cause the inaccurate/insufficient estimations of the future returns. Therefore, instead of expected values, learning distributions of future returns, i.e., Q values, is more useful for agents to make decisions. Recently, QR-MIX (Hu et al. 2020) decomposes the estimated joint return distribution (Bellemare et al. 2017; Dabney et al. 2018a) into individual Q values. However, the policies in QR-MIX are still based expected individual Q values. Even further, given that the environment is nonstationary from the perspective of each agent, each agent needs a more dynamic way to choose actions based on the return distributions, rather than simply taking the expected values. However, current MARL methods do not extensively investigate these aspects.

Motivated by the previous reasons, we intend to extend the risk-sensitive RL (Chow & Ohavamzadeh 2014; Keramati et al. 2020; Zhang et al. 2020) to MARL settings, where risk-sensitive RL optimizes policies with a risk measure, such as variance, power formula measure value at risk (VaR) and conditional value at risk (CVaR). Among these risk measures, CVaR has been gaining popularity due to both theoretical and computational advantages (Rockafellar & Uryasev 2002; Ruszczyński 2010). However, there are two main obstacles: i) most of the previous works focus on risk-neutral or static risk level in single-agent settings, ignoring the randomness of reward and the temporal structure of agents’ trajectories (Dabney et al. 2018a; Tang et al. 2019; Ma et al. 2020; Keramati et al. 2020); ii) many methods use risk measures over Q values for policy execution without getting the risk measure values used in policy optimization in temporal difference (TD) learning, which causes the global value factorization on expected individual values to sub-optimal behaviours in MARL. We provide a detailed review of related works in Appendix A due to the limited space.

In this paper, we propose RMIX, a novel cooperative MARL method with CVaR over the learned distributions of individuals’ Q values. Specifically, our contributions are in three folds: (i) We first learn the return distributions of individuals by using Dirac Delta functions in order to analytically calculate CVaR for decentralized execution. The resulting CVaR values at each time step are used as policies for each agent via \( \arg \max \) operation; (ii) We then propose a dynamic risk level predictor for CVaR calculation to handle the temporal nature of stochastic outcomes during executions. The dynamic risk level predictor measures the discrepancy between the embedding of current individual return distributions and the embedding of historical return distributions. The dynamic risk levels are agent-specific and observation-wise; (iii) We finally propose risk-sensitive Bellman equation along with IGM for centralized training. The risk sensitive Bellman equation enables CVaR value update in a recursive form and can be trained with TD learning via a neural network. These also allow our method to achieve temporally extended exploration and enhanced temporal coordination, which are key to solving complex multi-agent tasks. Empirically, we show that RMIX significantly outperforms state-of-the-art methods on many challenging StarCraft II tasks, demonstrating enhanced coordination in many symmetric & asymmetric and homogeneous & heterogeneous scenarios and revealing high sample efficiency. To the best of our knowledge, our work is the first attempt to investigate cooperative MARL with risk-sensitive policies under the Dec-POMDP framework.

2 Preliminaries

In this section, we provide the notation and the basic notions we will use in the following. We consider the probability space \((\Omega, \mathcal{F}, \Pr)\), where \(\Omega\) is the set of outcomes (sample space), \(\mathcal{F}\) is a \(\sigma\)-algebra over \(\Omega\) representing the set of events, and \(\Pr\) is the set of probability distributions. Given a set \(\mathcal{X}\), we denote with \(\mathcal{P}(\mathcal{X})\) the set of all probability measures over \(\mathcal{X}\).

**DEC-POMDP** A fully MARL problem can be described as a *decentralised partially observable Markov decision process* (Dec-POMDP) (Oliehoek et al. 2016), which can be formulated as a tuple \(\mathcal{M} = (\mathcal{S}, \mathcal{U}, \mathcal{P}, R, \mathcal{Y}, O, N, \gamma)\), where \(s \in \mathcal{S}\) denotes the true state of the environment. Each agent \(i \in N := \{1, \ldots, N\}\) chooses an action \(u_i \in \mathcal{U}\) at each time step, giving rise to a joint action vector, \(u := [u_i]_{i=1}^N \in \mathcal{U}^N\). \(\mathcal{P}(s'|s, u) : \mathcal{S} \times \mathcal{U}^N \times \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})\) is a Markovian transition function and governs all state transition dynamics. Every agent shares the same joint reward function \(R(s, u) : \mathcal{S} \times \mathcal{U}^N \rightarrow \mathcal{R}\), and \(\gamma \in (0, 1)\) is the discount factor. Due to partial observability, each agent has individual partial observation \(v \in \mathcal{Y}\), according to some observation function \(O(s, i) : \mathcal{S} \times \mathcal{N} \rightarrow \mathcal{Y}\).

---

1 “Risk” refers to the uncertainty of future outcomes (Dabney et al. 2018a).
2 StarCraft II is a trademark of Blizzard Entertainment, Inc.
Each agent also has an action-observation history $\tau_i \in T := (\mathcal{Y} \times \mathcal{U})^*$, on which it conditions its stochastic policy $\pi_i(\tau_i) : T \times \mathcal{U} \mapsto [0, 1]$.

**CVaR** CVaR is a coherent risk measure and enjoys computational properties (Rockafellar & Uryasev 2002) that are derived for loss distributions in discreet decision-making in finance. It gains popularity in various engineering and finance applications. CVaR (as illustrated in Figure 1) is the expectation of values that are less equal than the $\alpha$-percentile value of the distribution over returns. Formally, let $X \in \mathcal{X}$ be a bounded random variable with cumulative distribution function $F(x) = \mathcal{P}[X \leq x]$ and the inverse CDF is $F^{-1}(u) = \inf\{x : F(x) \geq u\}$. The conditional value at risk (CVaR) at level $\alpha \in (0, 1]$ of a random variable $X$ is then defined as $\text{CVaR}_\alpha(X) := \sup_{\nu} \left\{ \nu - \frac{1}{\alpha} \mathbb{E}_{\nu}[(\nu - X)_+] \right\}$ (Rockafellar et al., 2000) when $X$ is a discrete random variable. Correspondingly, $\text{CVaR}_\alpha(X) = \mathbb{E}_{X \sim F}[X | X \leq F^{-1}(\alpha)]$ (Acerbi & Tasche 2002) when $X$ has a continuous distribution. The $\alpha$-percentile value is value at risk (VaR). For ease of notation, we write $\text{CVaR}$ as a function of the CDF $F$, $\text{CVaR}_\alpha(F)$.

**Risk-sensitive RL** Risk-sensitive RL uses risk criteria over policy/value, which is a sub-field of the Safety RL (García et al., 2015). Von Neumann & Morgenstern (1947) proposed the expected utility theory where a decision policy behaves as though it is maximizing the expected value of some utility functions. The theory is satisfied when the decision policy is a consistent and has a particular set of four axioms. This is the most pervasive notion of risk-sensitivity. A policy maximizing a linear utility function is called risk-neutral, whereas concave or convex utility functions give rise to risk-averse or risk-seeking policies, respectively. Many measures are used in RL such as CVaR (Chow et al., 2015; Dabney et al., 2018a) and power formula (Dabney et al., 2018a). However, few works have been done in MARL and it cannot be easily extended. Our work fills this gap.

**CTDE** CTDE has recently attracted attention from deep MARL to deal with nonstationarity while learning decentralized policies. One of the promising ways to exploit the CTDE paradigm is value function decomposition (Sunehag et al., 2017; Rashid et al., 2018; Son et al., 2019) which learns a decentralized utility function for each agent and uses a mixing network to combine these local Q values into a global action-value. It follows the IGM principle where the optimal joint actions across agents are equivalent to the collection of individual optimal actions of each agent (Son et al., 2019). To achieve learning scalability, existing CTDE methods typically learn a shared local value or policy network for agents.

### 3 Methodology

In this section, we present our framework RMIX, as displayed in Figure 2, where the agent network learns the return distribution of each agent, a risk operator network determines the risk level of each agent and the mixing network mixes the outputs of risk operators of agents to produce the global value. In the rest of this section, we first introduce the CVaR operator to analytically calculate the CVaR value with the modeled individual distribution of each agent in Section 3.1 and propose the dynamic risk level predictor to alleviate time-consistency issue in Section 3.2. Then, we introduce the risk-sensitive Bellman equation for both centralized training and decentralized execution in Section 3.3. Finally, we provide the details of centralized training of RMIX in Section 3.4. All proofs are provided in Appendix B.

#### 3.1 CVaR of Return Distribution

In this section, we describe how we estimate the CVaR value. The value of CVaR can be either estimated through sampling or computed from the parametrized return distribution (Rockafellar & Uryasev 2002). However, the sampling method is usually computationally expensive (Tang et al., 2019). Therefore, we let each agent learn a return distribution parameterized by a mixture
of Dirac Delta ($\delta$) functions, which is demonstrated to be highly expressive and computationally efficient\footnote{\cite{bellemare2017risk}}. For convenience, we provide the definition of the Generalized Return Probability Density Function (PDF).

**Definition 1.** (Generalized Return PDF). For a discrete random variable $R \in [-R_{\text{max}}, R_{\text{max}}]$ and probability mass function (PMF) $\mathcal{P}(R = r_k)$, where $r_k \in [-R_{\text{max}}, R_{\text{max}}]$, we define the generalized return PDF as: $f_R(r) = \sum_{r_k \in R} \mathcal{P}(r_k) \delta(r - r_k)$. Note that for any $r_k \in R$, the probability of $R = r_k$ is given by the coefficient of the corresponding $\delta$ function, $\delta(r - r_k)$.

We define the return distribution of each agent $i$ at time step $t$ as:

$$Z_i^t(\tau_i, u_i^{t-1}) = \sum_{j=1}^m \mathcal{P}_j(\tau_i, u_i^{t-1}) \delta_j(\tau_i, u_i^{t-1})$$

(1)

where $m$ is the number of Dirac Delta functions. $\delta_j(\tau_i, u_i^{t-1})$ is the $j$-th Dirac Delta function and indicates the estimated value which can be parameterized by neural networks in practice. $\mathcal{P}_j(\tau_i, u_i^{t-1})$ is the corresponding probability of the estimated value given local observations and actions. $\tau_i$ and $u_i^{t-1}$ are trajectories (up to that timestep) and actions of agent $i$, respectively. With the individual return distribution $Z_i^t(\tau_i, u_i^{t-1}) \in \mathbb{Z}$ and cumulative distribution function (CDF) $F_{Z_i^t(\tau_i, u_i^{t-1})}$, we define the CVaR operator $\Pi_{\alpha_i}$, at a risk level $\alpha_i$ ($\alpha_i \in (0, 1]$ and $i \in \mathcal{A}$) over return as:\footnote{\cite{kolla2019generalized}}

$$C_i^t(\tau_i, u_i^{t-1}, \alpha_i) = \Pi_{\alpha_i} Z_i^t(\tau_i, u_i^{t-1}) = \text{CVaR}_{\alpha_i}(F_{Z_i^t(\tau_i, u_i^{t-1})}),$$

(2)

where $C \in \mathbb{C}$. As we use CVaR on return distributions, it corresponds to risk-neutrality (expectation, $\alpha_i = 1$) and indicates the improving degree of risk-aversion ($\alpha_i \rightarrow 0$). CVaR can be estimated in a nonparametric way given ordering of Dirac Delta functions $\{\delta_j\}^m_{j=1}$ by leveraging the individual distribution:

$$\text{CVaR}_{\alpha_i} = \sum_{j=1}^m \mathcal{P}_j \delta_j 1 \{ \delta_j \leq \hat{v}_{m, \alpha_i} \},$$

(3)

where $1\{\cdot\}$ is the indicator function and $\hat{v}_{m, \alpha_i}$ is estimated value at risk from $\hat{v}_{m, \alpha_i} = [\delta_m(1-\alpha_i)]$ with $[\cdot]$ being floor function. This is a closed-form formulation and can be easily implemented in practice. The optimal action of agent $i$ can be calculated via $\arg \max_{u_i} C_i(\tau_i, u_i^{t-1}, \alpha_i)$. We will introduce the decentralized execution in detail in Section 4.3.
3.2 Dynamic Risk Level Prediction

The values of risk levels, i.e., $\alpha_i$, $i \in \mathcal{A}$, are important for the agents to make decisions. Most of the previous works take a fixed value of risk level and do not take into account any temporal structure of agents’ trajectories, which can impede centralized training in the evolving multi-agent environments. Therefore, we propose the dynamic risk level prediction, which determines the risk levels of agents by explicitly taking into account the temporal nature of the stochastic outcomes, to alleviate time-consistency issue (Ruszczyński, 2010; Iancu et al., 2015) and stabilize the centralized training. Specifically, we represent the risk operator $\Pi_\alpha$ by a deep neural network, which calculates the CVaR value with predicted dynamic risk level $\alpha$ over the return distribution.

We note that the predictor differs from the attention network used in previous works (Iqbal & Sha, 2019; Yang et al., 2020) because the agent’s current return distribution and its return distribution during training. Specifically, we represent the risk operator $\Pi_\alpha$ by a deep neural network, which calculates the CVaR value with predicted dynamic risk level $\alpha$ over the return distribution.

Then we get the $\alpha_i^k$ output from $\psi_i$ and the probability of $\alpha_i^k$ is defined as:

$$
\mathcal{P}(\alpha_i^k) = \frac{\exp \left( \left( f_{\text{emb}}(Z_i^k), \phi_i^k \right) \right)}{\sum_{k=0}^{K-1} \exp \left( \left( f_{\text{emb}}(Z_i^k), \phi_i^k \right) \right)},
$$

(4)

Then we get the $\alpha_i = k/K$. The prediction risk level $\alpha_i$ is a scalar value and it is converted into a $K$-dimensional mask vector where the first $|\alpha_i \times K|$ items are one and the rest items are zero. This mask vector is used to calculate the CVaR value (Eqn. 2 and 3) of each action-return distribution that contains $K$ Dirac functions. Finally, we obtain $C_i$ and the policy $\pi_i$ as illustrated in Figure 3. During training, $f_{\text{emb}}$ updates its weights and the gradients of $f_{\text{emb}}$ are blocked (the dotted arrow in Figure 3) in order to prevent changing the weights of the network of agent $i$.

We note that the predictor differs from the attention network used in previous works (Iqbal & Sha, 2019; Yang et al., 2020) because the agent’s current return distribution and its return distribution of previous time step are separate inputs of their embeddings and there is no key, query and value weight matrices. The dynamic risk level predictors allow agents to determine the risk level dynamically based on historical return distributions.

3.3 Risk-sensitive Bellman Equation

Motivated by the success of optimizing the CVaR value in single-agent RL (Chow & Ghavamzadeh, 2014), RMIX aims to maximize the CVaR value of the joint return distribution, rather than the expectation (Rashid et al., 2018; Son et al., 2019). As proved in Theorem 1, the maximizing operation of CVaR values satisfies the IGM principle, which implies that maximizing the CVaR value of joint return distribution is equivalent to maximizing the CVaR value of each agent.

**Theorem 1.** In decentralized execution, given $\alpha = \{\alpha_i\}_{i=0}^{n-1}$, we define the global $\arg \max$ performed on global CVaR $C^{\text{tot}}(\tau, u, \alpha)$ as:

$$
\arg \max_u C^{\text{tot}}(\tau, u, \alpha) = \left( \arg \max_{u_1} C_1(\tau_1, u_1, \alpha_1), \ldots, \arg \max_{u_n} C_n(\tau_n, u_n, \alpha_n) \right)
$$

(5)
where $\tau$ and $u$ are trajectories (up to that timestep) and actions of all agents, respectively. The individuals’ maximization operation over return distributions defined above satisfies IGM and allows each agent to participate in a decentralised execution solely by choosing greedy actions with respect to its $C_i(\tau_i, u_i, \alpha_i)$.

To maximize the CVaR value of each agent, we define the risk-sensitive Bellman operator $T$:

$$T C^\text{tot}(s, u, \alpha) := \mathbb{E} \left[ R(s, u) + \gamma \max_{u'} C^\text{tot}(s', u', \alpha') \right]$$

where $\alpha$ and $\alpha'$ are agents’ static risk levels or dynamic risk levels output from the dynamic risk level predictor $\psi$ at each time step. The risk-sensitive Bellman operator $T$ operates on the CVaR value of the agent and the reward, which can be proved to be a contracting operation, as showed in Proposition 1. Therefore, we can leverage the TD learning (Sutton & Barto, 2018) to compute the maximal CVaR value of each agent, thus leading to the maximal global CVaR value.

**Proposition 1.** $T : C \mapsto C$ is a $\gamma$-contraction.

### 3.4 Centralized Training

We introduce the centralized training of RMIX. We utilize the monotonic mixing network of QMIX, which is a value decomposition network via hypernetwork (Ha et al., 2017) to maintain the monotonicity and has shown success in cooperative MARL. Based on IGM (Son et al., 2019) principle, we define monotonic IGM between $C^\text{tot}$ and $C_i$ for RMIX:

$$\frac{\partial C^\text{tot}}{\partial C_i} \geq 0, \forall i \in \{1, 2, \ldots, N\},$$

where $C^\text{tot}$ is the total CVaR value and $C_i(\tau_i, u_i)$ is the individual CVaR value of agent $i$, which can be considered as a latent combination of agents’ implicit CVaR values to the global CVaR value. Following the CTDE paradigm, we define the TD loss of RMIX as:

$$\mathcal{L}_\Pi(\theta) := \mathbb{E}_{D^\text{tr}, D} \left[ (y^\text{tot}_t - C^\text{tot}(s_t, u_t, \alpha_t))^2 \right]$$

where $y^\text{tot}_t = (r_t + \gamma \max_{u', \alpha'} C^\text{tot}_\theta(s_{t+1}, u', \alpha'))$, and $(y^\text{tot}_t - C^\text{tot}_\theta(s_t, u_t, \alpha_t))$ is our CVaR TD error for updating CVaR values. $\theta$ is the parameters of $C^\text{tot}$ which can be modeled by a deep neural network and $\theta$ indicates the parameters of the target network which is periodically copied from $\theta$ for stabilizing training (Mnih et al., 2015). While training, gradients from $Z_i$ are blocked to avoid changing the weights of the agents’ network from the dynamic risk level predictor. We train RMIX in an end-to-end manner. $\psi_i$ is trained together the agent network via the loss defined in Eq. 8. During training, $f_{\text{emb}}$ updates its weights while gradients of $f_{\text{emb}}$, are blocked in order to prevent changing the weights of the return distribution in agent $i$. The pseudo code of RMIX is shown in Algorithm 1 in Appendix C. We present our framework as shown in Figure 2. Our framework is flexible and can be easily used in many cooperative MARL methods.

### 4 Experiments

We empirically evaluate our methods on various StarCraft II scenarios. Especially, we are interested in the robust cooperation in complex asymmetric and homogeneous/heterogeneous scenarios. Additional introduction of baselines, scenarios and results are in Appendix C, E, F and G.

#### 4.1 Experiment Setup

**StarCraft II** We consider SMAC (Samvelyan et al., 2019) benchmark [screenshots of some scenarios are in Figure 5], a challenging set of cooperative StarCraft II maps for micromanagement, as our evaluation environments. We evaluate our methods for every 10,000 training steps during training by running 32 episodes in which agents trained with our methods battle with built-in game bots. We report the mean test won rate (test_battle_won_mean, percentage of episodes won of MARL agents) along with one standard deviation of won rate (shaded in figures). Due to limited...
page space, we present the results of our methods and baselines on 8 scenarios (we train our methods and baselines on 17 SMAC scenarios): corridor, 3s5z_vs_3s6z, 6h_vs_8z, 5m_vs_6m, 8m_vs_9m, 10m_vs_11m, 27m_vs_30m and MMM2. Table 1 shows detailed information on these scenarios.

**Baselines and training** The baselines are IQL (Tampuu et al., 2017), VDN (Sunehag et al., 2017), COMA (Foerster et al., 2017a), QMIX (Rashid et al., 2018), QTRAN (Son et al., 2019), MAVEN (Mahajan et al., 2019) and Qatten (Yang et al., 2020). We implement our methods on PyMARL and use 5 random seeds for training each method on 17 SMAC scenarios. We carry out experiments on NVIDIA Tesla V100 GPU 16G.

### 4.2 Experiment Results

RMIX demonstrates substantial superiority over baselines in asymmetric and homogeneous scenarios as depicted in Figure 7. RMIX outperforms baselines in asymmetric homogeneous scenarios: 5m_vs_6m, 8m_vs_9m, 10m_vs_11m and 27m_vs_30m (hard game). In 3s5z_vs_3s6z (asymmetric heterogeneous, very hard game) and MMM2 (symmetric heterogeneous, hard game), RMIX also shows leading performance over baselines. RMIX learns micro-trick (wall off) and micro-trick (focus fire) faster and better in very hard corridor and 6h_vs_8z. RMIX improves coordination in a sample efficient way via risk-sensitive policies. We summarize the performance of RMIX and QMIX on 17 SMAC scenarios in Figure 6. Readers can refer to Figure 19 and 20 for more results. We present results of RMIX in Figure 13 and 14 on 3s5z_vs_3s6z and 6h_vs_8z in 8 million training steps. Although training 27m_vs_30m is memory-consuming, we also present results of 2.5 million training steps, as depicted in Figure 15.

Interestingly, as illustrated in Figure 8, RMIX also demonstrates leading exploration performance over baselines on very hard corridor scenario.

![Figure 5: SMAC scenarios: 27m_vs_30m, 5m_vs_6m, 6h_vs_8z, corridor and MMM2.](https://github.com/oxwhirl/pymarl)
necting two separate rooms, and agents should learn to cooperatively combat the opponents to avoid being beaten by opponents with the divide-and-conquer strategy. RMIX outperforms MAVEN (Mahajan et al., 2019), which was originally proposed for tackling multi-agent exploration problems, both in sample efficiency and final results. After 4 million training steps, RMIX starts to converge while MAVEN starts to converge after over 7 million training steps.

In addition, we compare RMIX with QR-MIX (Hu et al., 2020). We implement QR-MIX with PyMARL and train it on 5m_vs_6m (hard), corridor (very hard), 6h_vs_8z (very hard) and 3s5z_vs_3s6z (very hard) with 3 random seeds for each scenario. Hyper-parameters used during training are from QR-MIX paper. As shown in Figure 9, RMIX shows leading performance and superior sample efficiency over QR-MIX on 5m_vs_6m, 27m_vs_30m and 6h_vs_8z. With distributional RL, QR-MIX presents slightly better performance on 3s5z_vs_3s6z. We present more results of RMIX vs QR-MIX in Appendix G.3.

We conduct an ablation study by fixing the risk level with the value of 1, thus we get the risk-neutral method, which we name as RMIX (α = 1). We present results of RMIX, RMIX (α = 1) and QMIX.
on 4 scenarios in Figure 10. RMIX outperforms RMIX (α = 1) in many heterogeneous and asymmetric scenarios, demonstrating the benefits of learning risk-sensitive MARL policies in complex scenarios where the potential of loss should be taken into consideration in coordination. Intuitively, for asymmetric scenarios, agents can be easily defeated by the opponents. As a consequence, coordination between agents is cautious in order to win the game, and the cooperative strategies in these scenarios should avoid massive casualties in the starting stage of the game. Apparently, our risk-sensitive policy representation works better than vanilla expected Q values in evaluation. In heterogeneous scenarios, action space and observation space are different among different types of agents, and methods with vanilla expected action value are inferior to RMIX.

To show that our proposed method is flexible in other mixing network for the ablation study, we apply additivity of individual CVaR values to represent the global CVaR value as $C^{\text{tot}}(\tau, u, \alpha) = C_1(\tau_1, u_1, \alpha_1) + \cdots + C_n(\tau_n, u_n, \alpha_n)$. Following the training of RMIX, we name this method Risk Decomposition Network (RDN) for ablation study. We use experiment setup of VDN and train RDN on 5 SMAC scenarios. With CVaR values, RDN outperforms VDN and QMIX in 1c3s5z, 5m_vs_6m, 8m_vs_9m and MMM2 with convincing improvements, as depicted in Figure 11. In some scenarios, for example 1c3z5z and 8m_vs_9m, the converged performance is even equal to that of RMIX, which demonstrate that RMIX is flexible in additivity mixing networks. Overall, with the new policy representation and additivity decomposition network, we can gain convincing improvements of RDN over VDN.

We present how the risk level α of each agent changes during the episode and emergent cooperative strategies between agents in the results analysis of RMIX in Appendix G.2 due to limited space.

5 Conclusion

In this paper, we propose RMIX, a novel risk-sensitive MARL method with CVaR over the learned distributions of individuals’ Q values. Our main contributions are in three folds: (i) We first learn the return distributions of individuals to analytically calculate CVaR for decentralized execution; (ii) We then propose a dynamic risk level predictor for CVaR calculation to handle the temporal nature of the stochastic outcomes during executions; (iii) We finally propose risk-sensitive Bellman equation along with Individual-Global-MAX (IGM) for MARL training. Empirically, we show that RMIX significantly outperforms state-of-the-art methods on many challenging StarCraft II tasks, demonstrating enhanced coordination in many complex scenarios and revealing high sample efficiency. To the best of our knowledge, our work is the first attempt to investigate cooperative MARL with risk-sensitive policies under the Dec-POMDP framework.
REFERENCES


A RELATED WORKS

As deep reinforcement learning (DRL) becomes prevailing (Mnih et al., 2015; Schulman et al., 2017), recent years have witnessed a renaissance in cooperative MARL with deep learning. However, there are several inevitable issues, including the nonstationarity of the environment from the view of individual agents (Foerster et al., 2017b), the credit assignment in cooperative scenarios with shared global rewards (Sunehag et al., 2017; Rashid et al., 2018), the lack of coordination and communication in cooperative scenarios (Jiang & Lu, 2018; Wang et al., 2020a) and the failure to consider opponents’ strategies when learning agent policies (He et al., 2016). Aiming to address these issues, centralized training with decentralized execution (CTDE) (Oliehoek et al., 2008; Kraemer & Banerjee, 2016) has drawn enormous attention via training policies of each agent with access to global trajectories in a centralized way and executing actions given only the local observations of each agent in a decentralized way. Several MARL methods are proposed (Lowe et al., 2017; Foerster et al., 2017a; Sunehag et al., 2017; Rashid et al., 2018; Son et al., 2019; Yang et al., 2020; Wang et al., 2020a), including value-based and policy gradient. Among these methods, VDN (Sunehag et al., 2017) and QMIX (Rashid et al., 2018) are representative methods that use value decomposition of the joint action-value function by adopting additivity and monotonicity structural constraints. Free from such structural constraints, QTRAN (Son et al., 2019) guarantees more general factorization than VDN and QMIX, however, such linear affine transformation fails to scale up in complex multi-agent scenarios, for example, StarCraft II environments.

However, most of works focus on decomposing the global Q value into individual Q values with different formulas and network architectures, either with structural constraints, for example additivity and monotonicity (Sunehag et al., 2017; Rashid et al., 2018), or with estimating values that forms a new global Q value estimation, for example representing Q values via summation of state-value and advantage or multi-head attention networks (Wang et al., 2020a; Yang et al., 2020). Current MARL methods neglect the limited representation of agent values, which fails to consider the agent-level impact of individuals to the whole system when transforming individual utilities to global values, leading to hostile training in complex scenarios. Typically, the problem of random cost underlying the nonstationarity of the environment, a.k.a risk-sensitive learning, which is very important for many real-world applications, has rarely been investigated. Current work are either confined in simple settings (Reddy et al., 2019) or have no general framework and convincing results in complex domains (Lyu & Amato, 2020). Further research should be done in risk-sensitive cooperative MARL. We propose RMIX and fill the gap in this field.

Recent advances in distributional RL (Bellemare et al., 2017; Dabney et al., 2018a;b) focuses on learning distribution over returns. However, with return distributions, these works still focus on either risk-neutral settings or with static risk level in single-agent setting, which neglects the ubiquitous significant risk-sensitive problems in many multi-agent real-world applications, including pipeline robots cooperation in factories, warehouse robots coordination, etc. This is very common in real-world applications, especially in highly dynamic tasks for example military action, resource allocation, finance portfolio management and Internet of Things (IoT), etc.

Chow & Ghavamzadeh (2014) proposed considered the mean-CVaR optimization problem in MDPs and proposed policy gradient with CVaR, and (García et al., 2015) presented a survey on safe RL, which have ignited the research on borrowing risk measures in RL (García et al., 2015; Tamar et al., 2015; Tang et al., 2019; Hiraoka et al., 2019; Majumdar & Pavone, 2020; Keramati et al., 2020; Ma et al., 2020). However, these works focus on single-agent settings, where there is one agent interacting with the environment compared with dynamic and non-stationary multi-agent environments. The merit of CVaR in optimization of MARL has yet to be discovered. We explore the CVaR risk measure in our methods and demonstrate the leading performance of our MARL methods.

B PROOFS

We present proofs on our propositions introduced in previous sections. The proposition and equations numbers are reused in restated propositions.

**Assumption 1.** The mean rewards are bounded in a known interval, i.e., \( r \in [-R_{\text{max}}, R_{\text{max}}] \).
Proposition 1. $\mathcal{T} : \mathcal{C} \mapsto \mathcal{C}$ is a $\gamma$-contraction.

Proof. We consider the sup-norm contraction,
\[
\left| \mathcal{T}C_1(s, u, \alpha_1) - \mathcal{T}C_2(s, u, \alpha_2) \right| \leq \gamma \left\| C_1(s, u, \alpha_1) - C_2(s, u, \alpha_2) \right\|_{\infty}
\]
\[\forall s \in \mathcal{S}, u \in \mathcal{U}, \alpha_i, i \in \{1, 2\} \in \mathcal{A}.\]

The sup-norm is defined as $\|C\|_{\infty} = \sup_{s \in \mathcal{S}, u \in \mathcal{U}, \alpha \in \mathcal{A}} |C(s, u)|$ and $C \in \mathbb{R}$.

With proposition 1, we can leverage TD learning \cite{SuttonBarto2018} to compute the maximal CVaR value of each agent, thus leading to the maximal global CVaR value. In some scenarios, where risk is not the primary concern for policy optimization, for example corridor scenario, where agents should learn to explore to win the game. RMIX will show less sample efficient to learn the optimal policy compared with its risk neutral variant, RMIX ($\alpha = 1$). As we can see in Figure 16 (b) section G.1.1, RMIX learns slower than RMIX ($\alpha = 1$) because it relies on the dynamic risk predictor while the predictor is trained together with the agent, it will take more samples in these environments. Interestingly, RMIX shows very good performance over other baselines.

Theorem 1. In decentralized execution, given $\alpha = \{\alpha_i\}_{i=0}^{n-1}$, we define the global $\arg \max$ performed on global CVaR $C_{\text{tot}}(\tau, u, \alpha)$ as:
\[
\arg \max_{u} C_{\text{tot}}(\tau, u, \alpha) = \left(\arg \max_{u_1} C_1(\tau_1, u_1, \alpha_1), \cdot \cdot \cdot, \arg \max_{u_n} C_n(\tau_n, u_n, \alpha_n)\right)
\]
where $\tau$ and $u$ are trajectories (up to that timestep) and actions of all agents, respectively. The individuals’ maximization operation over return distributions defined above satisfies IGM and allows each agent to participate in a decentralized execution solely by choosing greedy actions with respect to its $C_i(\tau_i, u_i, \alpha_i)$.

Proof. With monotonicity network $f_m$, in RMIX, we have
\[
C_{\text{tot}}(\tau, u, \alpha) = f_m(C_1(\tau_1, u_1, \alpha_1), \cdot \cdot \cdot, C_n(\tau_n, u_n, \alpha_n))
\]
Consequently, we have
\[
C_{\text{tot}}(\tau, \arg \max_{u'} C_i(\tau_i, u', \alpha_i))_{i=0}^{n-1} = f_m(\{\max_{u'} C_i(\tau_i, u', \alpha_i)\}_{i=0}^{n-1})
\]
By the monotonicity property of $f_m$, we can easily derive that if $j \in \{0, 1, \ldots, n - 1\}$, $u_j^* = \arg \max_{u'} C_j(\tau_j, u', \alpha_j)$, $\alpha^*_j \in \{0, 1\}$ is the optimal risk level given the current return distributions and historical return distributions, and actions of other agents are not the best action, then we have
\[
f_m(\{C_j(\tau_j, u_j, \alpha_j)\}_{i=0, i \neq j}^{n-1}) \leq f_m(\{C_j(\tau_j, u_j, \alpha_j)\}_{i=0}^{n-1}).
\]
So, for all agents, \( \forall j \in \{0, 1, \ldots, n - 1\} \), \( u_j^* = \arg \max_{u_j} C_j(\tau_j, u_j', \alpha_j) \), we have

\[
f_m(\{C_j(\tau_i, u_i, \alpha_i)\}_{i=0}^{n-1} \leq f_m(\{C_i(\tau_j, u_j, \alpha_j)\}_{i=0}^{n-1}) \\
\leq f_m(\{C_i(\tau_j, u_j^*, \alpha_j)\}_{i=0}^{n-1}) \\
= \max_{\{u_j\}_{i=0}^{n-1}} f_m(\{C_i(\tau_i, u_i, \alpha_i)\}_{i=0}^{n-1}).
\]

(15)

Therefore, we can get

\[
\max_{\{u_i, \alpha_i\}_{i=0}^{n-1}} f_m(\{C_i(\tau_i, u_i, \alpha_i)\}_{i=0}^{n-1}) = \max_{u, \alpha} C^{tot}(\tau, u, \alpha),
\]

(16)

which implies

\[
\max_{u} C^{tot}(\tau, u, \alpha) = C^{tot}(\tau, \{\arg \max_{u', \alpha'} C_i(\tau_i, u_i', \alpha_i)\}_{i=0}^{n-1}).
\]

(17)

\[
\square
\]

**Proposition 2.** For any agent \( i, i \in \{0, \ldots, n - 1\} \), \( \exists \lambda(\tau_i, u_i) \in (0, 1] \), such that \( C_i(\tau_i, u_i) = \lambda(\tau_i, u_i)E[Z_i(\tau_i, u_i)] \).

\[
\text{Proof.} \text{ We first provide that given a return distribution } Z, \text{ return random variable } Z \text{ and risk level } \alpha \in A, \text{ we can rewrite as } E[Z] < z < E[Z]. \text{ This can be easily proved by following Privault (2020)’s proof. Thus we can get } \Pi_i Z < E[Z], \text{ and there exists } \lambda(\tau_i, u_i) \in (0, 1], \text{ which is a value of agent’s trajectories, such that } \Pi_i Z_i(\tau_i, u_i) = \lambda(\tau_i, u_i)E[Z_i(\tau_i, u_i)]. \]

\[
\square
\]

Proposition 2 implies that we can view the CVaR value as truncated values of Q values that are in the lower region of return distribution \( Z_i(\tau_i, u_i) \). CVaR can be decomposed into two factors: \( \lambda(\tau_i, u_i) \) and \( E[Z_i(\tau_i, u_i)] \).

### C Additional Background

We introduce additional background on cooperative MARL algorithms, including QMIX, MAVEN and Gatten, for the convenience of readers who want to know more on these algorithms.

Q-based cooperative MARL is concerned with estimating an accurate action-value function to select the actions with maximum expected returns. The optimal Q-function is defined as (Rashid et al., 2018):

\[
Q^\text{tot}_\theta(s, u) := E \left[ \sum_{i=0}^{\infty} \alpha^i r(s_i, u_i, \theta), s_{t+1} \sim D' \left| s_t, u_t, \theta \right|, u_{t+1} = \arg \max_u Q^\text{tot}_\theta(s_{t+1}, \cdot) \right],
\]

\[
= r(s, u, \theta) + \gamma E \left[ \max_u Q^\text{tot}_\theta(s', \cdot) \right],
\]

where \( \theta \) is the parameters of \( Q^\text{tot} \) which can be model by deep neural networks working as the Q-function approximator. This Q-network can be trained by minimizing the loss function in a supervised-learning fashion as defined below:

\[
L(\theta) := E_{D'} \left[ \left( r_t + \gamma Q^\text{tot}_\theta(s_{t+1}, \arg \max_u Q^\text{tot}_\theta(s_{t+1}, \cdot)) - Q^\text{tot}_\theta(s_t, u_t) \right)^2 \right],
\]

where \( D' = (s_t, u_t, r_t, s_{t+1}) \) and \( D \) is the replay buffer. \( \tilde{\theta} \) indicates the parameters of the target network which is periodically copied from \( \theta \) for stabilizing training (Mnih et al., 2015). The network is trained in a centralized way with all partial observations accessible to all agents.
QMIX  QMIX (Rashid et al., 2018) is a well-known multi-agent Q-learning algorithm in the centralised training and decentralised execution paradigm, which restricts the joint action Q-values it can represent to be a monotonic mixing of each agent’s utilities in order to enable decentralisation and value decomposition:

\[ Q_{\text{mix}} := \{ Q_{\text{tot}} | Q_{\text{tot}}(\tau, u) = f_m (Q_1(\tau^1, u^1), \ldots, Q_n(\tau^n, u^n)), \frac{\partial f_m}{\partial Q_a} \geq 0, Q_a(\tau, u) \in \mathbb{R} \} \]

and the arg max operator is used to get the \( Q_{\text{tot}} \) for centralized training via TD loss similar to DQN (Mnih et al., 2015)

\[ \arg\max_u Q_{\text{tot}}(\tau, u) = \left( \begin{array}{c} \arg\max_u Q_1(\tau^1, u^1) \\ \vdots \\ \arg\max_u Q_n(\tau^n, u^n) \end{array} \right). \]

The architecture is shown in Figure 12. The monotonic mixing network \( f_m \) is parametrised as a feedforward network, of which non-negative weights are generated by hypernetworks (Ha et al., 2017) with the state as input.

MAVEN  MAVEN (Mahajan et al., 2019) (multi-agent variational exploration) overcomes the detrimental effects of QMIX’s monotonicity constraint on exploration via learning a diverse ensemble of monotonic approximations with the help of a latent space. It consists of value-based agents that condition their behaviour on the shared latent variable \( z \) controlled by a hierarchical policy that offloads \( \epsilon \)-greedy with committed exploration. Thus, fixing \( z \), each joint action-value function is a monotonic approximation to the optimal action-value function that is learnt with Q-learning.

Qatten  Qatten (Yang et al., 2020) explicitly considers the agent-level impact of individuals to the whole system when transforming individual \( Q_i \)’s into \( Q_{\text{tot}} \). It theoretically derives a general formula of \( Q_{\text{tot}} \) in terms of \( Q_i \), based on a multi-head attention formation to approximate \( Q_{\text{tot}} \), resulting in not only a refined representation of \( Q_{\text{tot}} \) with an agent-level attention mechanism, but also a tractable maximization algorithm of decentralized policies.
D PSEUDO CODE OF RMIX

Algorithm 1: RMIX

input: $K$, $\gamma$;
1 Initialize parameters $\theta$ of the network of agent, risk operator and monotonic mixing network;
2 Initialize parameters $\bar{\theta}$ of the target network of agent, risk operator and monotonic mixing network;
3 Initialize replay buffer $\mathcal{D}$;
4 for $e \leftarrow 0$ to $\text{MAX\_EPISODE}$ do
5 Start a new episode;
6 while $\text{EPISODE\_IS\_NOT\_TERMINATED}$ do
7 Get the global state $s^t$;
8 for agent $i \leftarrow 0$ to $n - 1$ do
9 Get observation $o^t_i$ from the environment;
10 Get action of last step $u^{t-1}_i$ from the environment;
11 Estimate the local return distribution $Z^t_i(o^t_i, u^{t-1}_i)$;
12 Predict the risk level $\alpha_i$:
13 \[
\arg \max_k \frac{\exp(f_{\text{emb}}(Z^t_i)^k, \phi^t_i))}{\sum_{k'=0}^{K-1} \exp(f_{\text{emb}}(Z^t_i)^{k'}, \phi^t_i))} / K, k \in \{1, \ldots, K\};
\]
14 Calculate CVaR values $C^t_i(o^t_i, u^{t-1}_i, \alpha_i) = \Pi_{\alpha_i} Z^t_i((o^t_i, u^{t-1}_i)$;
15 Get the action $u^t_i = \arg \max_u C^t_i(o^t_i, u^{t-1}_i, \alpha_i)$;
16 Concatenate $u^t_i$, $i \in \{0, \ldots, n - 1\}$ into $u^t$;
17 Execute $u^t_i$ into environment;
18 Receive global reward $r^t$ and observe a new state $s'$;
19 Store $(s^t, \{o^t_i\}_{i=0}^{n-1}, u^t, r^t, s')$ in replay buffer $\mathcal{D}$;
20 if UPDATE then
21 Sample a min-batch $\mathcal{D}'$ from replay buffer $\mathcal{D}$;
22 For each sample in $\mathcal{D}'$, calculate CVaR value $C^t_i$ by following steps in line 9-13;
23 Concatenate CVaR values $\{[C^t_0, \ldots, C^t_{n-1}]_0, \ldots, [C^t_{|\mathcal{D}'|-1}, \ldots, C^t_{|\mathcal{D}'|-1}]_{|\mathcal{D}'|-1}\}$;
24 For each $[C^t_0, \ldots, C^t_{n-1}]_j, j \in \{0, \ldots, |\mathcal{D}'|-1\}$, calculate $C^t_j$ in the mixing network;
25 Calculate the target value $y^t_{\text{tot}} = (r^t + \gamma \max_u C^t_{\text{tot}})$;
26 Calculate the TD loss $L_{\Pi}(\theta) := \mathbb{E}_{\mathcal{D}' \sim \mathcal{D}} [(y^t_{\text{tot}} - C^t_{\text{tot}})^2]$;
27 Update $\theta$ by minimizing the TD loss;
28 Update $\bar{\theta}$; $\bar{\theta} \leftarrow \bar{\theta}$;
29 return $\theta$;
E ADDITIONAL ENVIRONMENT INTRODUCTION

SMAC benchmark is a challenging set of cooperative StarCraft II maps for micromanagement developed by [Samvelyan et al.] (2019) built on DeepMind’s PySC2 (Vinyals et al., 2017). We introduce states and observations, action space and rewards of SMAC, and environmental settings of RMIX below.

States and Observations At each time step, agents receive local observations within their field of view. This encompasses information about the map within a circular area around each unit with a radius equal to the sight range. The sight range makes the environment partially observable for each agent. An agent can only observe other agents if they are both alive and located within its sight range. Hence, there is no way for agents to distinguish whether their teammates are far away or dead. The feature vector observed by each agent contains the following attributes for both allied and enemy units within the sight range: distance, relative x, relative y, health, shield, and unit type. All Protos units have shields, which serve as a source of protection to offset damage and can regenerate if no new damage is received. The global state is composed of the joint observations but removing the restriction of sight range, which could be obtained during training in the simulations. All features, both in the global state and in individual observations of agents, are normalized by their maximum values.

Action Space The discrete set of actions which agents are allowed to take consists of move[direction], attack[enemy id], stop and no-op. Dead agents can only take no-op action while live agents cannot. Agents can only move with a fixed movement amount 2 in four directions: north, south, east, or west. To ensure decentralization of the task, agents are restricted to use the attack[enemy id] action only towards enemies in their shooting range. This additionally constrains the ability of the units to use the built-in attack-move macro-actions on the enemies that are far away. The shooting range is set to 6 for all agents. Having a larger sight range than a shooting range allows agents to make use of the move commands before starting to fire. The unit behavior of automatically responding to enemy fire without being explicitly ordered is also disabled.

Rewards At each time step, the agents receive a joint reward equal to the total damage dealt on the enemy units. In addition, agents receive a bonus of 10 points after killing each opponent, and 200 points after killing all opponents for winning the battle. The rewards are scaled so that the maximum cumulative reward achievable in each scenario is around 20.

Environmental Settings of RMIX The difficulty level of the built-in game AI we use in our experiments is level 7 (very difficult) by default as many previous works did (Rashid et al., 2018; Mahajan et al., 2019; Yang et al., 2020). The scenarios used in Section 4 are shown in Table 1. We present the table of all scenarios in SMAC in Table 1 and the corresponding memory usage for training each scenario in Table 2. The Ally Units are agents trained by MARL methods and Enemy Units are built-in game bots. For example, 5m_vs_6m indicates that the number of MARL agent is 5 while the number of the opponent is 6. The agent (unit) type is marine. This asymmetric setting is hard for MARL methods.

---

7A type of unit (agent) in StarCraft II. Readers can refer to https://liquipedia.net/starcraft2/Marine_(Legacy_of_the_Void) for more information.
Table 1: SMAC Environments

<table>
<thead>
<tr>
<th>Name</th>
<th>Ally Units</th>
<th>Enemy Units</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3m</td>
<td>3 Marines</td>
<td>3 Marines</td>
<td>homogeneous &amp; symmetric</td>
</tr>
<tr>
<td>8m</td>
<td>8 Marines</td>
<td>8 Marines</td>
<td>homogeneous &amp; symmetric</td>
</tr>
<tr>
<td>25m</td>
<td>25 Marines</td>
<td>25 Marines</td>
<td>homogeneous &amp; symmetric</td>
</tr>
<tr>
<td>3s5z</td>
<td>3 Stalkers &amp; 5 Zealots</td>
<td>3 Stalkers &amp; 5 Zealots</td>
<td>heterogeneous &amp; symmetric</td>
</tr>
<tr>
<td>MMM</td>
<td>1 Medivac, 2 Marauders &amp; 7 Marines</td>
<td>1 Medivac, 2 Marauders &amp; 7 Marines</td>
<td>heterogeneous &amp; symmetric</td>
</tr>
<tr>
<td>5m_vs_6m</td>
<td>5 Marines</td>
<td>6 Marines</td>
<td>homogeneous &amp; asymmetric</td>
</tr>
<tr>
<td>8m_vs_9m</td>
<td>8 Marines</td>
<td>9 Marines</td>
<td>homogeneous &amp; asymmetric</td>
</tr>
<tr>
<td>10m_vs_11m</td>
<td>10 Marines</td>
<td>11 Marines</td>
<td>homogeneous &amp; asymmetric</td>
</tr>
<tr>
<td>27m_vs_30m</td>
<td>27 Marines</td>
<td>30 Marines</td>
<td>homogeneous &amp; asymmetric</td>
</tr>
<tr>
<td>3s5z_vs_3s6z</td>
<td>3 Stalkers &amp; 5 Zealots</td>
<td>3 Stalkers &amp; 6 Zealots</td>
<td>heterogeneous &amp; asymmetric</td>
</tr>
<tr>
<td>MMM2</td>
<td>1 Medivac, 2 Marauders &amp; 7 Marines</td>
<td>1 Medivac, 3 Marauders &amp; 8 Marines</td>
<td>heterogeneous &amp; asymmetric</td>
</tr>
<tr>
<td>1c3s5z</td>
<td>1 Colossi, 3 Stalkers &amp; 5 Zealots</td>
<td>1 Colossi, 3 Stalkers &amp; 5 Zealots</td>
<td>heterogeneous &amp; symmetric</td>
</tr>
<tr>
<td>2m_vs_1z</td>
<td>2 Marines</td>
<td>1 Zealot</td>
<td>micro-trick: alternating fire</td>
</tr>
<tr>
<td>3s_vs_5z</td>
<td>3 Stalkers</td>
<td>5 Zealots</td>
<td>micro-trick: kiting</td>
</tr>
<tr>
<td>6h_vs_8z</td>
<td>6 Hydralisks</td>
<td>8 Zealots</td>
<td>micro-trick: focus fire</td>
</tr>
<tr>
<td>corridor</td>
<td>6 Zealots</td>
<td>24 Zerglings</td>
<td>micro-trick: wall off</td>
</tr>
<tr>
<td>bane_vs_bane</td>
<td>20 Zerglings &amp; 4 Banelings</td>
<td>20 Zerglings &amp; 4 Banelings</td>
<td>micro-trick: positioning</td>
</tr>
</tbody>
</table>

F Additional Training Details

The baselines are list in table 3 as depicted below. To make a fair comparison, we use episode (single-process environment for training, compared with parallel) runner defined in PyMAML to run all methods. The evaluation interval is 10,000 for all methods. We use uniform probability to estimate $Z_i(\cdot, \cdot)$ for each agent. We use the other hyper parameters used for training in the original papers of all baselines. The metrics are calculated with a moving window size of 15. Experiments are carried out on NVIDIA Tesla V100 GPU 16G. We also provide memory usage of baselines (given the current size of the replay buffer) for training each scenario of SCII domain in SMAC.

We use the same neural network architecture of agent used by QMIX (Rashid et al., 2018). The trajectory embedding network $\phi_i$ is similar to the network of the agent.
Table 2: Memory usage (given the current size of the replay buffer) for the training of each method (exclude COMA, which is an on-policy method without using replay buffer) on scenarios of SCII domain in SMAC.

<table>
<thead>
<tr>
<th>Scenario Name</th>
<th>Memory Usage (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3m</td>
<td>2.7</td>
</tr>
<tr>
<td>2m_vs_1z</td>
<td>2.8</td>
</tr>
<tr>
<td>5m_vs_6m</td>
<td>3</td>
</tr>
<tr>
<td>3s_vs_5z</td>
<td>4</td>
</tr>
<tr>
<td>6h_vs_8z</td>
<td>4.6</td>
</tr>
<tr>
<td>8m</td>
<td>4.8</td>
</tr>
<tr>
<td>8m_vs_9m</td>
<td>4.9</td>
</tr>
<tr>
<td>3s5z</td>
<td>6.4</td>
</tr>
<tr>
<td>10m_vs_11m</td>
<td>7.1</td>
</tr>
<tr>
<td>3s5z_vs_3s6z</td>
<td>7.5</td>
</tr>
<tr>
<td>lc3s5z</td>
<td>8.6</td>
</tr>
<tr>
<td>MMM</td>
<td>8.7</td>
</tr>
<tr>
<td>MMM2</td>
<td>10.8</td>
</tr>
<tr>
<td>corridor</td>
<td>14.4</td>
</tr>
<tr>
<td>25m</td>
<td>27</td>
</tr>
<tr>
<td>27m_vs_30m</td>
<td>39.5</td>
</tr>
<tr>
<td>bane_vs_bane</td>
<td>41</td>
</tr>
</tbody>
</table>

Table 3: Baseline algorithms

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQL (Tampuu et al., 2017)</td>
<td>Independent Q-learning</td>
</tr>
<tr>
<td>VDN (Sunehag et al., 2017)</td>
<td>Value decomposition network</td>
</tr>
<tr>
<td>COMA (Foerster et al., 2017/a)</td>
<td>Counterfactual Actor-critic</td>
</tr>
<tr>
<td>QMIX (Rashid et al., 2018)</td>
<td>Monotonicity Value decomposition</td>
</tr>
<tr>
<td>QTRAN (Son et al., 2019)</td>
<td>Value decomposition with linear affine transform</td>
</tr>
<tr>
<td>MAVEN (Mahajan et al., 2019)</td>
<td>MARL with variational method for exploration</td>
</tr>
<tr>
<td>Qatten (Yang et al., 2020)</td>
<td>Multi-head attention for decomposing the global Q values</td>
</tr>
</tbody>
</table>

G. ADDITIONAL EXPERIMENTS ON SMAC

G.1 ABLATIONS

We use the same hyper-parameters in ablation study unless otherwise specified.

G.1.1 STATIC RISK LEVEL

We present more results of RMIX and RMIX (α = 1) in Figure 13, 14 and 15. In vary hard 3s5z_vs_3s6z, 6h_vs_8z games, RMIX and RMIX (α = 1) outperforms baselines. Surprisingly, RMIX is even slightly better in 6h_vs_8z where micro-trick (focus fire) is learned. In conclusion, RMIX is also capable of learning hard micro-trick tasks in StarCraft II. In asymmetric scenario 27m_vs_30m, RMIX shows leading performance as well.

We also conduct an ablation study with static risk level of α = 0.1 and α = 0.3 in RMIX-static. Obviously, as shown in Figure 16 with static risk level, RMIX-static shows steady progress over time, but its performance is lower than RMIX in 6h_vs_8z and 5m_vs_6m. On asymmetric scenario 5m_vs_6m, the converged performance of RMIX-static is 0.6, which is lower than RMIX’s (0.8).

In some scenarios, where risk is not the primal concern for policy optimization, for example corridor scenario, where agents should learn to explore to win the game. RMIX shows less sample efficient to learn the optimal policy compared with its risk neutral variant, RMIX (α = 1) in corridor as shown in Figure 16(b). RMIX learns slower than RMIX (α = 1) because it relies on the dynamic risk predictor while the predictor is trained together with the agent, it will take more samples in these environments. Interestingly, RMIX shows very good performance over other baselines.
Figure 13: \textit{test\_battle\_won\_mean} of RMIX, RMIX ($\alpha = 1$) and QMIX on 3s5z\_vs\_3s6z \textit{(heterogeneous and asymmetric scenario, very hard game)}.

Figure 14: \textit{test\_battle\_won\_mean} of RMIX, RMIX ($\alpha = 1$) and QMIX on 6h\_vs\_8z \textit{(heterogeneous and asymmetric scenario, very hard game)}.

Figure 15: \textit{test\_battle\_won\_mean} of RMIX, RMIX ($\alpha = 1$) and QMIX on 27m\_vs\_30m \textit{(homogeneous and asymmetric scenario, very hard game)}.
Figure 16: test_battle_won_mean of RMIX-static.
Table 4: Hyper-parameters

<table>
<thead>
<tr>
<th>hyper-parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch-size</td>
<td>32</td>
</tr>
<tr>
<td>Replay memory size</td>
<td>5,000</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate (lr)</td>
<td>5e-4</td>
</tr>
<tr>
<td>Critic lr</td>
<td>5e-4</td>
</tr>
<tr>
<td>RMSProp alpha</td>
<td>0.99</td>
</tr>
<tr>
<td>RMSProp epsilon</td>
<td>0.00001</td>
</tr>
<tr>
<td>Gradient norm clip</td>
<td>10</td>
</tr>
<tr>
<td>Action-selector</td>
<td>ε-greedy</td>
</tr>
<tr>
<td>ε-start</td>
<td>1.0</td>
</tr>
<tr>
<td>ε-finish</td>
<td>0.05</td>
</tr>
<tr>
<td>ε-anneal-time</td>
<td>50,000 steps</td>
</tr>
<tr>
<td>Target-update-interval</td>
<td>200</td>
</tr>
<tr>
<td>Evaluation interval</td>
<td>10,000</td>
</tr>
<tr>
<td>Number of atoms (m)</td>
<td>55</td>
</tr>
<tr>
<td>K</td>
<td>10</td>
</tr>
<tr>
<td>Runner episode</td>
<td></td>
</tr>
<tr>
<td>Training steps</td>
<td>1, 2, 3, 1.5, 2.5 and 8 millions</td>
</tr>
<tr>
<td>Discount factor (γ)</td>
<td>0.99</td>
</tr>
<tr>
<td>RNN hidden dim</td>
<td>64</td>
</tr>
</tbody>
</table>

G.2 ADDITIONAL RESULTS ANALYSIS

We provide additional results analysis of RMIX on corridor in Figure 17. There are 6 RMIX agents in corridor. For brevity of visualization, we use the data of 3 agents (agent 0, 1 and 3) to analyse the results and to demonstrate RMIX agents have learned to address time-consistency issue.
One episode of game starts, 6 Zealots consist of RMIX agents and 24 Zerglings compose enemies. In order to win the game, agent 3 draws the attention of enemies and go the other side of the battlefield. The rest agents combat with fewer number enemies. As being outnumbered, agent 3 quickly dies as shown in green line after agents die the $\alpha = 0$. Agent 0 is followed by agent 1 (low health value) in order to draw enemies to come over. To avoid being killed, $\alpha$ values are low which means the policy is risk-averse. In the other side, as there are many agents around and the number of enemies is small, the $\alpha$ values are over 0.8 (nearly risk-neutral). RMIX agents are going to win, agent 1 (low health value) walks outside the observation of enemies in order to survive. Agents are seeking and waiting for enemies to come in the center of the corridor. Reward is zero.

Figure 17: RMIX results analysis on corridor. We use trained model of RMIX and run the model to collect one episode data including game replay, states, actions, rewards and $\alpha$ values (risk level). We show rewards of one episode and the corresponding $\alpha$ value each agent predicts per time step in row one and row two. We provide description and analyses on how agents learn time-consistency $\alpha$ values for the rest rows. Pictures are screenshots from the game replay. Readers can watch the game replay via this anonymous link: [https://youtu.be/J-PG0loCDGk](https://youtu.be/J-PG0loCDGk). Interestingly, it also shows emergent cooperation strategies between agents at different time step during the episode, which demonstrate the superiority of RMIX.
G.3 Additional Results

We conduct experiments of RMIX, QMIX, MAVEN, Qatten, VDN, COMA and IQL on 17 SMAC scenarios. We show results of test\_battle\_won\_mean and test\_return\_mean of aforementioned methods in Figure [19] and [20] respectively. RMIX shows leading performance on most of scenarios, ranging from symmetric homogeneous scenarios to asymmetric heterogeneous scenarios. Surprisingly, RMIX also shows superior performance on scenarios where micro-trick should be learned to win the game.

In addition, we compare RMIX with QR-MIX [Hu et al., 2020]. Unlike QMIX, QR-MIX decomposes the estimated joint return distribution into individual Q values. We implement QR-MIX with PyMARL by using hyper-parameters in QR-MIX paper and train it on 3m (easy), 1c3s5z (easy), 5m\_vs\_6m (hard), 8m\_vs\_9m (hard), 10\_vs\_11m (hard), 27m\_vs\_30m (very hard), MMM2 (very hard), 3s5z\_vs\_3s6z (very hard), corridor (very hard) and 6h\_vs\_8z (very hard) with 3 random seeds for each scenario. Results are shown in Figure [18].

![Figure 18: test\_battle\_won\_mean of RMIX vs QR-MIX and QMIX.](attachment:image.png)
Figure 19: test_battle_won_mean of RMIX, MAVEN, QMIX, Qatten, IQL, QTRAN, VDN and COMA on 17 SMAC StarCraft II scenarios.
Figure 20: test_return_mean of RMIX, MAVEN, QMIX, Qatten, IQL, QTRAN, VDN, COMA on 17 SMAC StarCraft II scenarios.