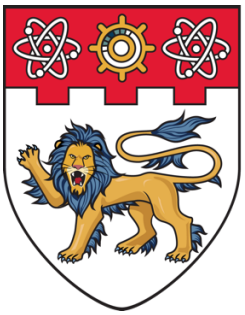


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

MULTI-AGENT REINFORCEMENT LEARNING FOR COMPLEX SEQUENTIAL DECISION-MAKING

QIU WEI

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

May 2023

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

13/04/2023

Date

QIU WEI

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accordance with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

09/05/2023

Date

Prof. Bo AN

Authorship Attribution Statement

This thesis contains material from four papers accepted at conferences in which I am listed as the first author and corresponding author.

Chapter 3 is published as **Wei Qiu**, Xinrun Wang, Runsheng Yu, Rundong Wang, Xu He, Bo An, Svetlana Obraztsova, Zinovi Rabinovich. RMIX: Learning Risk-Sensitive Policies for Cooperative Reinforcement Learning Agents. Advances in Neural Information Processing Systems. (**NeurIPS'21**).

The contributions of the co-authors are as follows:

- Prof. Bo An provided suggestions on the motivation, the experiments, and the writing.
- A/Prof. Svetlana Obraztsova participated in the discussion and provided suggestions on the experiments and the writing.
- Dr. Xinrun Wang proposed the idea of applying distributional RL to MARL and discussed ideas of the proposed method. He also wrote parts of the paper.
- I proposed applying risk-sensitive RL with distributional RL to MARL. I designed the algorithm, conducted the experiments, and wrote the paper.
- A/Prof. Zinovi Rabinovich participated in the discussion and provided critical comments on the algorithm, the experiments, and the writing.
- Runsheng Yu provided suggestions on the algorithm and the writing, especially the analysis of the algorithm.
- Rundong Wang provided suggestions on the algorithm, the experiments on StarCraft II, the writing of the method, and the experiments.
- Xu He provided suggestions on the validation of the algorithm and the writing of the section of the method.

Chapter 4 is published as **Wei Qiu**, Haipeng Chen, Bo An. Dynamic Electronic Toll Collection via Multi-Agent Deep Reinforcement Learning with Edge-based Graph Convolutional Network Representation. Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, pages 4568–4574, 2019. (**IJCAI'19**).

The contributions of the co-authors are as follows:

- Prof. Bo An provided suggestions on the motivation, the algorithm, the experiments, and the writing.
- Dr. Haipeng Chen provided suggestions on the algorithm and the experiments. He also participated in the writing of the paper, especially in improving the writing.
- I proposed the algorithm, conducted experiments, and wrote the paper.

Chapter 5 is published as **Wei Qiu**, Weixun Wang, Rundong Wang, Bo An, Yujing Hu, Lana Obraztsova, Zinovi Rabinovich, Jianye Hao, Yingfeng Chen, Changjie Fan. Off-Beat Multi-Agent Reinforcement Learning. Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems, May 2023, pages 2424–2426, 2023. (**AAMAS'23 Extended Abstract**).

The contributions of the co-authors are as follows:

- Prof. Bo An proposed the project and provided suggestions on the motivation, the experiments, and the writing.
- A/Prof. Svetlana Obraztsova provided suggestions on the experiments and the writing.
- I proposed the algorithm, conducted the experiments and wrote the paper.
- Dr. Yujing Hu proposed the initial project and provided suggestions on the experiments and the writing.
- Weixun Wang provided suggestions on the experiments and the writing.
- A/Prof. Zinovi Rabinovich provided suggestions on the algorithm, the experiments, and the writing.
- Rundong Wang participated in the discussion and provided suggestions on the algorithm and the experiments.
- Dr. Jianye Hao provided suggestions on the motivation and the writing of the paper.
- Dr. Yingfeng Chen and Dr. Changjie Fan provided provided computing resources.

Chapter 6 is published as **Wei Qiu**, Xiao Ma, Bo An, Svetlana Obraztsova, Shuicheng Yan, Zhongwen Xu. RPM: Generalizable Behaviors for Multi-Agent Reinforcement Learning. The Eleventh International Conference on Learning and Representation. (**ICLR'23**).

The contributions of the co-authors are as follows:

- Prof. Bo An provided critical comments on the motivation, the experiments, and the writing.
- A/Prof. Svetlana Obraztsova provided suggestions on the experiments and the writing.
- Dr. Zhongwen Xu proposed the idea of using historical policies.
- I proposed the problem, and the algorithm, conducted the experiments and wrote the paper.
- Dr. Xiao Ma participated in the discussion and provided valuable suggestions on the method.
- Prof. Shuicheng Yan participated in the discussion and provided valuable suggestions on the method.

13/04/2023

Date

QIU WEI

Acknowledgements

The PhD journey at Nanyang Technological University has been a meaningful and rewarding one for both my career and life. It has been a time of immense growth, discovery, and reflection, both personally and professionally. As I stand at the end of this journey, I am filled with a deep sense of gratitude for all those who have supported me along the way.

First and foremost, I am greatly indebted to my supervisors, Prof. Bo An and Prof. Lana Obraztsova, who, throughout my PhD journey at Nanyang Technological University, have provided me with insightful discussions, a suitable research environment, as well as the freedom that allows me to explore interesting research problems. They encouraged me to work on important topics. Not only did I learn about how to be professional, but also I learned from their methods and thoughts for tackling research problems. I could not have imagined having better advisors and mentors for my PhD journey.

I am also very fortunate to have met and learned from a group of talented people: Haipeng Chen, Jiuchuan Jiang, Mengchen Zhao, Youzhi Zhang, Xinrun Wang, Xu He, Lei Feng, Aye Phyu, Hongxin Wei, Rundong Wang, Yakub Cerny, Runsheng Yu, Yanchen Deng, Shuxin Li, Wanqi Xue, Feifei Lin, Zhuyun Yin, Xinyu Cai, Shuo Sun, Pengdeng Li, Shenggong Ji, Chaojie Wang, Shufeng Kong, Longtao Zheng and Weihao Tan. I would like to thank them for the joy, support, and excitement that they gave me.

I thank my Thesis Advisory Committee members: Prof. Zinovi Rabinovich and Prof. Xiaohui Bei. They helped me a lot and made my research journey easier at NTU. I had many fantastic discussions on my research projects with Prof. Rabinovich in his office. During my PhD, I had an exciting internship at Sea AI Lab and had many opportunities to collaborate with other great researchers from NTU and other institutes. I would like to thank Dr. Zhongwen Xu, Prof. Shuicheng

Yan, Dr. Tianbo Li, Dr. Haiyan Yin, Dr. Min Lin, Dr. Yangzihao Wang, Dr. Qian Liu, Dr. Tianyu Pang, Dr. Chao Du, Dr. Siyi Li, Chen Zhang, Xinyi Wan, Jiahao Ji, Xiangfan Li, Quanhong Fu, and other talented researchers and staffs I met at Sea AI Lab, who greatly inspired me and helped me a lot during my internship. I would like to thank Hang Xu, Tianze Luo, Qiuhao Zeng, Jianda Chen, Yi Ding, Su Zhang, Dr. Xinghua Qu, Lei Yuan, Weixun Wang, Dr. Yujing Hu, Hangtian Jia, Yang You, Hang Hua, Chenjia Bai, Tonghan Wang, Dr. Tabish Rashid, Dr. Bei Peng, Dr. Mingfei Sun, and Dr. Shuyue Hu for wonderful discussions on research problems. I would like to thank Jianxiong (Terry) Yin from NVAITC (NVIDIA AI Tech Center), who helped me a lot and provided GPU support for my research.

Last but not least, I would like to thank my family, my parents: Zhaonan Qiu and Jinhua Huang, and my brothers: Tao Qiu and Fei Qiu, for their unconditional love and support for me. Needless to say, this thesis would not have been possible without your encouragement. This thesis is dedicated to all of you. I love you.

Contents

Acknowledgements vi

List of Figures xi

List of Tables xvi

List of Author's Publications xvii

Symbols and Acronyms xix

Abstract xxvi

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Focus of This Thesis . . . 3
  - 1.2.1 Learning Risk-Sensitive Policies for Cooperative MARL . . . 4
  - 1.2.2 Learning to Scale Up with Cooperative MARL . . . 4
  - 1.2.3 Learning to Coordinate in Environments with Off-Beat Actions . . . 5
  - 1.2.4 Learning to Generalize for MARL Agents . . . 6
- 1.3 Main Contributions . . . 6
- 1.4 Thesis Organization . . . 8

2 Background 10
- 2.1 Markov Decision Process . . . 11
- 2.2 Reinforcement Learning . . . 12
  - 2.2.1 Temporal-Difference Learning . . . 14
  - 2.2.2 Q-Learning and DQN . . . 14
  - 2.2.3 Policy Gradient and PPO . . . 15
- 2.3 Decentralized Partially Observable Markov Decision Process . . . 16
- 2.4 Markov Games . . . 17
- 2.5 Multi-Agent Reinforcement Learning . . . 18
- 2.6 Chapter Summary . . . 19

3 Learning Risk-Sensitive Policies for Cooperative MARL **20**
3.1 Introduction . . . 21
3.2 Background . . . 22
3.3 Solution Method: RMIX . . . 24
3.3.1 CVaR of Return Distribution . . . 24
3.3.2 Risk Level Optimizer . . . 26
3.3.3 Training . . . 28
3.4 Experiments . . . 32
3.4.1 Experiment Setup . . . 32
3.4.1.1 Multi-Agent Cliff Navigation . . . 32
3.4.1.2 StarCraft II Scenarios . . . 33
3.4.2 Training Details . . . 35
3.4.3 Baseline Algorithms . . . 37
3.4.4 Experimental Results . . . 39
3.4.4.1 Multi-Agent Cliff Navigation . . . 39
3.4.4.2 StarCraft II . . . 41
3.4.5 Ablations . . . 42
3.4.6 Results Analysis . . . 44
3.5 Chapter Summary . . . 48

4 Learning to Scale Up with Cooperative MARL **49**
4.1 Introduction . . . 51
4.2 Electronic Toll Collection . . . 52
4.3 Problem Formulation . . . 52
4.3.1 Problem Setup . . . 52
4.3.2 A Finite Horizon MDP Formulation . . . 53
4.4 Solution Method: MARL-eGCN . . . 55
4.4.1 Deep PG-β with TD-learning . . . 55
4.4.2 Edge-Based Graph Convolutional Networks Representation . . . 56
4.4.3 MARL with eGCN . . . 57
4.5 Experimental Evaluation . . . 59
4.5.1 Experiment Setup . . . 59
4.5.2 Result Analysis . . . 61
4.5.3 Ablation Study and Robustness Test . . . 62
4.6 Chapter Summary . . . 64

5 Learning to Coordinate in Environments with Off-Beat Actions **66**
5.1 Introduction . . . 67
5.2 Related Works . . . 69
5.3 Off-Beat Actions and Off-Beat Dec-POMDP . . . 71
5.4 Solution Method: LeGEM . . . 72
5.4.1 Temporal Recency Reward Redistribution . . . 73
5.4.2 LeGEM-core: The Episodic Memory . . . 75

5.4.3 Searching for the Pivot Timesteps . . . 77
5.5 Experiments . . . 82
5.5.1 Experiment Setup . . . 83
5.5.2 Experiment Results . . . 85
5.6 Chapter Summary . . . 95

6 Learning to Generalize for MARL Agents 96
6.1 Introduction . . . 97
6.2 Related Works . . . 99
6.3 Problem Formulation . . . 101
6.4 Solution Method: RPM . . . 102
6.5 MARL Training . . . 105
6.6 Experiments . . . 106
6.6.1 Experimental Environment: Melting Pot . . . 106
6.6.2 Baselines . . . 112
6.6.3 Architectures . . . 113
6.6.4 Training Settings . . . 115
6.6.5 Experiment Results . . . 117
6.6.6 Ablation Study . . . 119
6.6.7 Case Study . . . 123
6.7 Chapter Summary . . . 125

7 Conclusions and Future Directions 126
7.1 Conclusions . . . 126
7.2 Future Directions . . . 127
7.2.1 Risk-Sensitive MARL . . . 127
7.2.2 MARL for Intelligent Traffic Systems . . . 127
7.2.3 Multi-Agent Coordination in OBMAS . . . 128
7.2.4 Generalizable Human-AI Coordination . . . 128

Bibliography 129

List of Figures

1.1 (a) A football match. (b) A StarCraft II scenario. (c) Robots in a warehouse. 1
1.2 The overall structure of this thesis. 9

2.1 The agent–environment interaction in a Markov decision process. . 11
2.2 A two-agent multi-agent system. Note that in Dec-POMDP, the two agents share the same reward since they should coordinate in the environment. However, in Markov games, agents receive different reward values because agents have different goals. They may cooperate at the beginning of the game and betray at the end of the game. 17

3.1 The illustration of Conditional Value at Risk (CVaR). 22
3.2 The framework (dotted arrow indicates that gradients are blocked during training). (a) Agent's policy network. (b) The overall architecture (agent network and mixer). (c) Risk operator. Each agent i applies an individual risk operator Π_{α_i} on its return distribution $Z_i(\cdot, \cdot)$ to calculate $C_i(\cdot, \cdot, \cdot)$ for execution given risk level α_i predicted by the dynamic risk level optimizer ψ_i. $\{C_i(\cdot, \cdot, \cdot)\}_{i=1}^{N}$ are fed into the mixer for centralized training. 25
3.3 Agent architecture. It has two Recurrent Neural Network (RNN) [1] models. One RNN is for the Q network (top) and another one is for the risk level optimizer (bottom). We implement the two RNN models with Gated Recurrent Unit (GRU) [2], which is efficient in practice. Note that $\tilde{Z}_i$ shares the same neural network layer with Z_i, while the RNNs of agent i and ψ function independently, without sharing neural network layers. 26
3.4 Risk level optimizer ψ_i. 27
3.5 Multi-Agent Cliff Navigation. 32
3.6 SMAC scenarios: 5m_vs_6m and MMM2. 33
3.7 The solid curves show the test return mean (averaged total rewards of evaluation episodes) on two MACN scenarios, MACN scenarios 1 and 2. The shades depict one standard deviation of the test return mean. 39
3.8 Test Win rates for six scenarios. 40
3.9 Test Win rates of RMIX, RDN, VDN and QMIX. 43

3.10 RMIX vs QMIX with reward noise. 43
3.11 Test return mean in MACN scenario 1. 44
3.12 Test return for six scenarios. 45
3.13 Test return of RMIX, RDN, VDN, QMIX in three scenarios. 46
3.14 Results analysis of RMIX on the corridor. There are 8 scenes as examples. Further discussion can be found in Section 3.4.6. 46
3.15 Two examples on CVaR calculation and action selection. 47

4.1 The architecture of the proposed MARL-eGCN with three modules, *i.e.*, the *traffic dynamics*, the *state representation*, and the *tolling agent*. At each time step, the state matrix is extracted from the *traffic dynamics* module and fed into the *state representation* (eGCN) module together with the adjacency matrix of the road network. Then, the output of eGCN is fed into the *tolling agent* module. The policies generate tolling actions, which are further input into traffic dynamics to change the traffic flow. The *actor* and *critic* of the *tolling agent* are updated according to the reward emitted by the *traffic dynamics*. 55
4.2 An example of zone partitions. 57
4.3 Map of Singapore: Singapore central region and entire Singapore planning areas. 60
4.4 Traffic throughput (95% confidence interval) and convergence comparison (values in thousands). 61
4.5 Ablation study (values in thousands). 62
4.6 Traffic throughput (in thousands) under various traffic conditions (Figures a, c, d share the same legend in Figure b). 63

5.1 **An illustrative scenario:** two-agent stag-hunter game, where two agents (hunters) have only partial observations, different durations of the shoot action, and cannot communicate. The goal is to catch the stag and they are rewarded when their shots hit the stag at the same time. Both agents can see the stag. As the shoot action durations of the two agents are different, to catch the stag, the two agents should shoot the arrow at different time steps given the distances. For hunter 0, the time step to shoot the arrow is 0, while for hunter 1, it is 5. At time step 9, the two arrows will hit the stag and all hunters will receive a positive shared reward. Though the scenario is easy for human beings, it is hard for MARL agents due to the action duration. **Experiment results:** in this scenario, the optimal policy for agent 0 is to shoot the arrow at time step 0 while the optimal policy for agent 1 is to shoot the arrow at time step 5. Such an asynchronous property of OBMAS motivates agents to learn tacit policies. The curves show that VDN and IQL fail to learn coordination policies even in this simple scenario. With our LeGEM, MARL methods gain enhanced performance as well as improved sample efficiency. 68

5.2 Reward redistribution for the example in Figure 5.1. In this example, hunter 0 takes the shooting action at time step 0, and hunter 1 takes the shooting action at time step 5. At the other time steps, both hunter 0 and hunter 1 take no-op actions. At time step 9, hunter 0 and hunter 1 receive the global reward of 15. LeGEM distributes the reward of 15 to time step 0 for agent 0 and time step 5 for agent 1, respectively. The final pivot time step is 5 via Equation 5.1. So, the trajectory is updated via the distribution of the reward. Then, we utilize TD-learning to train the MARL policy using the newly modified trajectory. 73
5.3 The maximum level of the graph is 7. Circles indicate the nodes and numbers indicate the visit count. 75
5.4 **Updating agent i's Φ_i:** Agent i's ϕ_i^t is updated with agent's trajectories τ_i^1 and then updated with τ_i^2. Solid arrows and circles indicate the pointers and nodes, respectively. Grey dotted lines indicate pointers to be created and grey circles with dotted outlines indicate nodes to be created. All the dotted elements (pointers and circles) consist of the newly added path in τ_i^1. All the created pointers and nodes will be added to ϕ_i^t. 76
5.5 An example of the search scheme (lines 6-13, Algorithm 4). r^4 is the off-beat reward at time step 4. Each agent searches its pivot time step $e_{t=4}^i (i \in \{0, 1\})$ from its own graph. 78
5.6 Our framework: LeGEM, the loss, and the agent's policy. There are two key components: centralized training and decentralized execution. 79
5.7 Stag-Hunter Game, Afforestation Game, and Quarry Game. 82
5.8 The test catch rate of our method vs MARL baselines in the Stag-Hunter Game. 86
5.9 The test task completion rate of our method vs MARL baselines in Quarry Game. 86
5.10 Performance of our method and other MARL methods in Stag-Hunt Game, Quarry Game, and Afforestation Game. 87
5.11 The test task completion rate of our method vs MARL baselines in Afforestation Game. 88
5.12 Results of QMIX, VDN, QTRAN and IQL with n-step return in Stag-Hunter Game. 88
5.13 Results of QMIX, VDN, QTRAN and IQL with TD(λ) in Stag-Hunter Game. 89
5.14 Results of QMIX, VDN, QTRAN and IQL with n-step return in Quarry Game. 89
5.15 Results of QMIX, VDN, QTRAN and IQL with TD(λ) in Quarry Game. 90
5.16 Results of QMIX, VDN, QTRAN and IQL with n-step return in Afforestation Game. 90

5.17 Results of QMIX, VDN, QTRAN and IQL with TD(λ) in Afforestation Game. All methods perform poorly and the test task completion rates are zero. 91
5.18 Results of MARL-RUDDER in Stag-Hunter Game. 92
5.19 Results of MARL-RUDDER in Afforestation Game. 93
5.20 Results of MARL-RUDDER in Quarry Game. 93

6.1 Two-Agent Hunting Game. (a) Training environment. Two agents (hunters) hunt in the environment. (b) After training in the training environment, all agents behave cooperatively to capture the stag. (c) In the new evaluation scenario, one agent is picked as the focal agent (in the magenta circle) and paired with a pre-trained agent (in the brown circle) that behaves in different ways to evaluate the performance of the selected agent. In conclusion, the conventional evaluation protocol fails to evaluate such behavior and current MARL methods easily fail to learn the optimal policy due to the lack of diversified multi-agent interaction data during training. 97
6.2 An example of our formulation. **Left:** All six agents' policies are trained with the MARL. **Right:** Two agents with policies π_{ϕ_1} and π_{ϕ_2} are picked as background agents, and the rest of the four agents (with new indices) are focal agents to be evaluated. The focal and the background agents constitute the evaluation scenario. 101
6.3 The workflow of RPM for a three-agent substrate. In the workflow, there are three agents in the substrate. Agent 3 is the background agent. Agents 1 and 2 are focal agents. 103
6.4 The neural network architecture of the policy of the agent is shown to the left. The green box to the lower left in the environment shows the agent's observation. 106
6.5 The networks of the policy (left) and the critic (right). 114
6.6 Melting Pot environments. 115
6.7 Evaluation results of RPM and baselines in 17 scenarios. The red dash horizontal lines indicate the results of random policy. The optimal (opt) values are shown in each sub-figure and were gathered from [3], which an exploiter generated. The exploiter was trained in the evaluation scenarios with RL methods, and the training time steps were 1,000 M. 118
6.8 Episode return in substrates. 119
6.9 Histograms of training episode returns. The long-tail patterns are found in CU, PC, PD, and RC. The histograms of CG and SH are skews. We can conclude that policies that have higher counts but with lower returns can be sampled more often in self-play, leading to the poor performance of generalization. 120
6.10 Ablation Study: the performance of RPM with 3 types of ψ and Random sampling (without ranks). 121
6.11 Ablation Study: the results of HFSP with different sampling ratios. 122

6.12 Results analysis. **(a)** The evaluation results of RPM on Stag Hunt (SH) 1; **(b)** The number of RPM keys during training; **(c)** The distribution of the keys of RPM during training; **(d)** The histogram of the keys of RPM at time step 200M during training. 123

List of Tables

3.1 SMAC Environments 35
3.2 Memory usage (given the current size of the replay buffer) for the training of each method (exclude COMA and DOP, which are on-policy methods without using replay buffers) on scenarios of SCII domain in SMAC. 36
3.3 Baseline algorithms 37
3.4 Test win rate of RMIX with varying risk levels. 42

5.1 Baseline algorithms. 82
5.2 Hyper-parameters in our experiments. 85
5.3 Results (mean and std) of QMIX and VDN with n-step return in Stag-Hunter Game. 91
5.4 Results (mean and std) of QMIX and VDN with TD(λ) in Stag-Hunter Game. 91
5.5 Results (mean and std) of RUDDER, QMIX-LeGEM and VDN-LeGEM in OBMAS. 94

6.1 Properties of Melting Pot environments. The first column shows the properties and the first row lists environments. ✓ mark indicates the environment possessing the corresponding property while ✗ mark stands for the environment that does not own the corresponding property. SH stands for Stag Hunt. PC stands for Pure Coordination. CU is Clean Up. PD stands for Prisoners' Dilemma. RC is Rational Coordination. CG stands for Chicken Game. 116
6.2 Hyper-parameters in our experiments. 116
6.3 The value of ψ 117
6.4 Ablation study: the averaged value of the last three evaluation episode returns. Curves are in Figure 6.10. 120
6.5 ψ values. ψ^* indicates the values of ψ used to get results in Figure 6.7. 121

List of Author's Publications

International Conferences

- Haipeng Chen, Bryan Wilder, **Wei Qiu**, Bo An, Eric Rice and Milind Tambe. A Learning Approach to Complex Contagion Influence Maximization. Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, 2023. (**IJCAI'23**) [4, 5][1]

- **Wei Qiu**, Xiao Ma, Bo An, Svetlana Obraztsova, Shuicheng Yan, Zhongwen Xu. RPM: Generalizable Multi-Agent Policies for Multi-Agent Reinforcement Learning. The Eleventh International Conference on Learning and Representation. (**ICLR'23**) [6]

- Rundong Wang, Longtao Zheng, **Wei Qiu**, Bowei He, Bo An, Zinovi Rabinovich, Yujing Hu, Yingfeng Chen, Tangjie Lv, Changjie Fan. Towards Skilled Population Curriculum for Multi-Agent Reinforcement Learning. arXiv preprint arXiv:2302.03429. [7]

- **Wei Qiu**, Weixun Wang, Rundong Wang, Bo An, Yujing Hu, Lana Obraztsova, Zinovi Rabinovich, Jianye Hao, Yingfeng Chen, Changjie Fan. Off-Beat Multi-Agent Reinforcement Learning. Proceedings of the Twenty-Second International Conference on Autonomous Agents and Multiagent Systems, May 2023, pages 2424–2426, 2023. (**AAMAS'23**, Extended Abstract). [8]

- Wanqi Xue, **Wei Qiu**✉, Zinovi Rabinovich, Bo An, Svetlana Obraztsova, and Yeo Chai Kiat. Mis-spoke or Mis-lead: Achieving Robustness in Multi-Agent Communicative Reinforcement Learning. Proceedings of the Twenty-First International Conference on Autonomous Agents and Multiagent Systems, May 2023, pages 1418–1426. (**AAMAS'22**) [9]

[1] A short extended abstract version was accepted by AAMAS 2023.

- **Wei Qiu**, Xinrun Wang, Runsheng Yu, Xu He, Rundong Wang, Bo An, Svetlana Obraztsova, and Zinovi Rabinovich. RMIX: Learning Risk-Sensitive Policies for Cooperative Reinforcement Learning Agents. Advances in Neural Information Processing Systems, pages 4568–4574, 2021. (**NeurIPS'21**) [10]

- Haipeng Chen, **Wei Qiu**, Bo An, and Milind Tambe. Contingency-Aware Influence Maximization: A Reinforcement Learning Approach. Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence, pages 1535–1545, 2021. (**UAI'21**) [11]

- Rundong Wang, Xu He, Runsheng Yu, **Wei Qiu**, Bo An, and Zinovi Rabinovich. Learning Efficient Multi-Agent Communication: An Information Bottleneck Approach. Proceedings of the Thirty-Seventh International Conference on Machine Learning, PMLR 119:9908–9918, 2020. (**ICML'20**) [12]

- **Wei Qiu**, Haipeng Chen, Bo An. Dynamic Electronic Toll Collection via Multi-Agent Deep Reinforcement Learning with Edge-based Graph Convolutional Network Representation. Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, pages 4568–4574, 2019. (**IJCAI'19**) [13]

Symbols and Acronyms

Claim on Notation Consistency

In this thesis, a consistent set of symbols and notations has been employed throughout each chapter for clarity and coherence in the presentation of multi-agent reinforcement learning (MARL) research. However, it is acknowledged that variations or notational differences may exist between chapters. These differences are intentional and have been introduced to accommodate specific nuances or requirements of individual chapters. The reader is encouraged to refer to the notation conventions outlined in the following for accurate interpretation. The overarching goal is to ensure clarity and understanding in the context of each chapter's content.

Symbols in Chapter 2

$\mathcal{N}$	The index set agents
$i \in \{1, ..., N\}$	Agent index
N	The Number of agents
$s \in \mathcal{S}$	State and state space
$\boldsymbol{s}$	The current global state
$o_i \in \mathcal{O}$	Agent i's observation and observation space
O	The observation function
$u_i \in \mathcal{U}$	Agent i's action and the action space
$\boldsymbol{u}$	The current action of all agents
$\mathcal{R}$	The reward space
r^t	Reward at time step t in Dec-POMDP
r_i^t	Agent i's reward at time step t in Markov Games
R	The reward function
γ	The discount factor
$\mathcal{P}$	The transition probability
V	Value in RL and MARL
Q_i	Agent i's Q value
Q^{tot}	The global Q value of all agents
Q^*	The optimal global Q value of all agents
θ	The the parameters of agent's network and Q^{tot}
$\bar{\theta}$	The the parameter of the target network
$\mathcal{D}$	The replay buffer
D'	A sample batch from the replay buffer
τ_i	Agent's action-observation-reward history

Symbols of Chapter 3

δ	The Dirac Delta function
δ_j^i	The j-th Dirac Delta function of agent i
Z_i^t	The return distribution of agent i at time step t
C_i^t	The Conditional Value at Risk of agent i at time step t.
α_i	The risk level of agent i
ψ_i	The risk level optimizer of agent i

Symbols of Chapter 4

$\mathcal{G}$	A directed road network
$\mathcal{Z}$	The set of zones
$\mathcal{E}$	The set of roads that connect the zones
$\mathcal{D}$	The set of origin-destination (OD) pairs
$\mathbf{s}^t$	The state at time step t
$\mathbf{s}_e^t$	The number of vehicles that travel to zone j at time step t on road e
$\mathbf{a}^t$	The action at time step t
a_e^t	The toll set by the transit authority on e
$\mathbf{f}$	The eGCN model

Symbols of Chapter 5

$\tilde{\boldsymbol{u}}$	The joint off-beat action
$\tilde{\boldsymbol{u}}_t$	The joint off-beat action at time step t
$\tilde{u}$	Agent's off-beat action
$\tilde{u}_i^t$	Agent i's off-beat action at time step t
$m_{\tilde{u}_i}$	The execution duration of agent i's action $\tilde{u}_i$
$\boldsymbol{m}$	The execution duration of $\tilde{\boldsymbol{u}}$
$\boldsymbol{m}_t$	The execution duration of $\tilde{\boldsymbol{u}}_t$
r^t	The global reward at time step t
$\hat{r}^{e_t}$	The redistributed reward
τ_i	Agent i's observation-action-reward trajectory
o_i^t	Agent i's observation at time step t
T	The length of the τ_i
ϕ_i^t	Agent i's LeGEMwhose maximum level is t
ω	The index of episode return
Ω	The length of the episode return list
Φ	The set of LeGEM
Φ_i	The set of agent i's LeGEM
$\phi_i^{t,\omega}$	Agent i's ω-th sub-graph of which maximum level is t
$\Phi_i^{t,\Omega}$	Agent i's set of sub-graphs of which maximum level is t
l	the index of the l-th level in the graph.
Λ	All the paths from node at level t to node at the level 0 for each agent
e_t	The pivot time step for r_t
e_t^i	Agent i's pivot time step for r_t
$\boldsymbol{e}_t^i$	The vector of agent i's pivot time step of each path in Λ for r_t

Symbols of Chapter 6

Ψ	The notation of ranked policy memory (RPM)
$\mathcal{G}$	The training substrate
$\mathcal{G}'$	The evaluation scenario

Acronyms

AC	Actor-Critic
AI	Artificial Intelligence
CNN	Convolutional Neural Network
CTDE	Centralised Training with Decentralised Execution
CVaR	Conditional Value at Risk
DETC	dynamic Electronic Toll Collection
Dec-POMDP	Decentralized Partially Observable Markov Decision Process
DQN	Deep Q Network
EM	Episodic Memory
GCN	Graph Convolutional Neural network
GRU	Gated Recurrent Unit
IQL	Independent Q-learning
MARL	Multi-Agent Reinforcement Learning
MAS	Multi-Agent Systems
MDP	Markov Decision Process
OBMAS	Off-Beat Multi-Agent Scenarios
PG	Policy Gradient
PI	Policy Iteration
PPO	Proximal Policy Optimization
QR-DQN	Quantile Regression Deep Q Network
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RTS	Real-Time Strategy
TD Learning	Temporal-difference Learning
VDN	Value decomposition network
VI	Value Iteration

Abstract

Many tasks involve multiple agents and require sequential decision-making policies to achieve common goals, such as football games, real-time strategy games and traffic light control in the road network. To obtain the policies of all agents, these problems can be modeled as multi-agent systems and solved with multi-agent reinforcement learning (MARL). However, optimizing policies in multi-agent scenarios is non-trivial due to complex multi-agent behaviors and the non-stationary nature of the environments' complex dynamics. Agents' behaviors and interactions with other agents cause the environment's states and agents' observations to change over time, making it challenging to develop effective policies that perform well over time. In addition, partial observability, where agents have limited or incomplete information about the environment, also complicates the problem. Moreover, the inherent uncertainty in the environment's dynamics makes decision-making unstable.

This doctoral thesis addresses these challenges by proposing novel MARL methods. These novel methods empower agents to learn efficient policies within dynamic and partially observable environments, especially environments where cooperation is required. In particular, we tackle the following four fundamental multi-agent research problems and propose a solution for each.

We start by studying the problem of learning risk-sensitive cooperative policies for agents in risky scenarios characterized by significant potential reward loss due to the execution of potential low-return actions. Particularly, we focus on environments where agent heterogeneity is prevalent within the team, and opponents may outnumber the RL agents. To tackle the problem, we propose RMIX to learn risk-sensitive cooperative policies for MARL. We first model distributions of individuals' Q values via distributional RL. Then we utilize the Conditional Value at Risk (CVaR) measure over the individual return distribution. We also propose a dynamic risk level optimizer to handle the temporal nature of the stochastic outcomes during execution. Empirically, RMIX shows leading performance over

state-of-the-art methods in various multi-agent risk-sensitive scenarios. It demonstrates enhanced coordination and reveals improved sample efficiency.

We then investigate the problem of learning scalable policies in dynamic Electronic Toll Collection (DETC) problems where the traffic network is large and dynamic. To this end, we propose a novel MARL approach for scaling up DETC by decomposing large states into smaller parts and learning a multi-agent policy for each decomposed state with the cooperative MARL method. Specifically, we decompose the graph network into smaller graphs and propose a novel edge-based graph convolutional neural network (eGCN) to extract the spatio-temporal correlations of the road network features. The extracted features are fed into the policy network of the cooperative MARL method. Experimental results show that such a divide-and-conquer approach can scale up to realistic-sized problems with robust performance and significantly outperform the state-of-the-art method.

Thirdly, we focus on learning efficient multi-agent coordination policies in scenarios where actions have durations. With action durations, the rewards are displaced, making training MARL policies challenging with temporal-difference learning. To address this problem, we propose a novel reward redistribution method built on our novel graph-based episodic memory called LeGEM-core to learn efficient multi-agent coordination in environments where off-beat actions are prevalent. Off-beat actions refer to actions that have action durations, during which the environmental changes are influenced by the execution of these actions. LeGEM-core memorizes agents' past experiences explicitly and enables credit assignment in MARL training. We name our solution method as LeGEM. We evaluate LeGEM on various multi-agent scenarios with off-beat actions, including the Stag-Hunter Game, Quarry Game, and Afforestation Game. Empirical results show that it significantly boosts multi-agent coordination in multi-agent environments with off-beat actions and achieves leading performance.

Lastly, we aim to learn generalizable policies that enable agents to coordinate with or compete with other agents that possess unseen policies during training. We propose RPM that learns generalizable policies for agents in evaluation scenarios where other agents behave differently. The main idea of RPM is to train MARL policies by gathering massive multi-agent interaction data. We first rank each agent's policies by its training episode return and then save the ranked policies in the memory; when an episode starts, each agent can randomly select a policy from memory as

the behavior policy. This novel self-play framework diversifies multi-agent interaction in the training data and improves the generalization performance of MARL. Experimental results on Melting Pot demonstrate that RPM enables agents to interact with unseen agents in multi-agent generalization evaluation scenarios and gain increased performance.

To conclude, this doctoral thesis investigates four fundamental multi-agent sequential decision-making research problems that are ubiquitous and unsolved. The proposed four MARL methodology solutions achieve efficient policy training and performance for agents in multi-agent environments with uncertainties raised by the potential loss of rewards, the issue of large state space, the action durations, and the lack of generalizability of MARL.

Chapter 1

Introduction

1.1 Background

Many entities, such as sports players in a match, video game players in an episode, cars in the traffic network, and robots in the warehouse, engage in a team or group to accomplish their goals. The way they interact with each other can be cooperative, competitive, or even mixed, and they require certain sequential decision-making strategies to achieve goals. For example, to win the promotion competition, eleven football players in the team could adopt a strategy and then cooperatively execute the strategy in the tournament (Figure 1.1 (a)). In StarCraft II video games, players in each group should cooperate with each other to combat the rest groups (Figure 1.1 (b)). Such complex coordination involves complex sequential decision-making at each time step. In a warehouse, there are thousands of storage racks and commodities. It is non-trivial for robots to coordinate with each other to conduct warehouse management and scheduling (Figure 1.1 (c)). These problems often contain a scenario with many entities, such as in Figure 1.1 (c), the warehouse

FIGURE 1.1: (a) A football match. (b) A StarCraft II scenario. (c) Robots in a warehouse.

is the scenario, and robots are entities that act in the warehouse. These multi-entity scenarios are multi-agent problems that require sequential decision-making strategies for agents to complete the assigned tasks. To obtain optimal strategies, these multi-agent problems can be modeled as multi-agent systems (MAS) [14] where each agent models each entity.

The field of multi-agent systems(MAS) [14, 15] is a subfield of artificial intelligence (AI) [16] that deals with the design and implementation of software agents that interact with each other in some specific environments. The term "agent" refers to an autonomous software entity that perceives its environment and acts upon it to achieve a goal or complete a set of tasks. In MAS, multiple agents coexist and interact, sharing information, coordinating their actions, and collectively solving complex problems that are difficult or impossible to solve by a single agent. MAS is particularly suitable for problem domains where the problem-solving task is sequential, distributed, uncertain, dynamic, and involves multiple stakeholders with different goals and preferences. MAS has been applied in a wide range of applications, including video games [17, 18], urban systems [19] and autonomous systems [20–22]. The design and implementation of MAS require advanced approaches that enable agents to learn efficient decision-making strategies via interacting with each other and the environment.

Recently, reinforcement learning (RL) methods [23, 24] are often applied to solve multi-agent problems, primarily because of their ability to scale effectively. Such a problem-solving paradigm is called multi-agent reinforcement learning (MARL) [25–27]. RL provides a framework for agents to learn optimal policies that maximize accumulated rewards in complex environments. With RL, MARL enables agents to learn from interacting with other agents and the environment through trial and error. By leveraging the power of deep neural networks [28] and RL [29], deep MARL is capable of acquiring advanced strategies and adapting to dynamic environments. This approach achieves superior performance by enabling complex coordination between agents in multi-agent scenarios, such as the coordination in video games [30] and traffic control in the traffic network [13].

MARL[1] has made remarkable advances in many domains, including autonomous systems [20–22] and real-time strategy (RTS) video games [17]. By virtue of the *centralized training with decentralized execution* (CTDE) [31] paradigm, which

[1]In the remainder of the thesis, we use MARL rather than deep MARL for conciseness.

aims to tackle the scalability and partial observability challenges in MARL, many CTDE-based MARL methods are proposed [26, 32–38]. In CTDE, an agent executes actions only via feeding its observations independently and optimizes its policy with access to global trajectories centrally. MARL offers several advantages over single-agent RL in multi-agent systems. Firstly, MARL can lead to better coordination and cooperation among agents [15, 26, 35], increasing the efficiency and robustness of MAS. Secondly, MARL is highly flexible and can be applied to a wide range of complex real-world scenarios with multiple interacting agents, leading to the improvement of efficiency not attained with single-agent RL [22]. Finally, MARL can capture the interaction between agents, leading to the emergence of adaptive and intelligent systems [39]. These advantages make MARL a practical approach for solving complex, multi-agent problems and can provide several benefits over single-agent RL in MAS.

1.2 Focus of This Thesis

However, optimizing policies in multi-agent scenarios is non-trivial due to complex multi-agent behaviors, and the non-stationary nature of the environments' complex dynamics. Agents' behaviors and interactions between agents cause the environment's states and agents' observations to change over time, making it challenging to develop effective policies that perform well over time. In addition, partial observability, where agents have limited or incomplete information about the environment, also complicates the problem. Moreover, the inherent uncertainty in the environment's dynamics makes decision-making unstable.

This doctoral thesis addresses these challenges by proposing novel MARL methods that enable agents to adapt to dynamic and partially observable environments and effectively learn efficient policies. In particular, we tackle the following four fundamental MARL research problems introduced in the following subsections 1.2.1, 1.2.2, 1.2.3 and 1.2.4. The first three research problems are mainly on learning multi-agent cooperative and coordinative policies in environments with risky outcomes (see Section 1.2.1), large state-action spaces (see Section 1.2.2), and action durations (see Section 1.2.3), respectively. The ultimate goal of MARL research is to train generalizable agents that are capable of cooperating and working with other agents that have unseen policies during training. We made our

first attempt at training agents that are generalizable in complex scenarios (see Section 1.2.4). In this section, we first introduce the background and challenges of four MARL research problems, which are the focus of this thesis. Then, we propose our method for each MARL research problem and summarize our main contributions in Section 1.3.

1.2.1 Learning Risk-Sensitive Policies for Cooperative MARL

Recently, centralized training with decentralized execution (CTDE) [31] has drawn enormous attention in MARL research. Empowered by CTDE, several MARL methods, including value-based and policy gradient-based, are proposed [26, 32, 33, 35]. 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 [26, 32, 33, 35, 36, 40].

However, such an expected, *i.e.*, risk-neutral, Q value is not sufficient even with CTDE due to the noisy nature of rewards and the inherent uncertainty in the environments, which is ubiquitous in many real-world multi-agent problems, such as multi-robot search and rescue [41]. These issues cause the failure of these methods to train cooperative agents in complex multi-agent environments. Specifically, these methods only learn the expected values over returns [26] and do not handle the high variance caused by events with extremely high/low rewards to agents but at small probabilities, which cause inaccurate/insufficient estimations of future returns. Therefore, instead of estimating risk-neural Q values, learning risk-sensitive Q values is essential for efficient multi-agent decision-making.

1.2.2 Learning to Scale Up with Cooperative MARL

Electronic Toll Collection (ETC) systems set tolls on different roads. Thus, vehicles are regulated to travel on less congested roads with lower tolls, making ETC a practical approach to mitigate traffic congestion. In recent years, there have been many dynamic ETC (DETC) pricing schemes [42–45] that calculate tolls based on real-time traffic dynamics. These schemes were proposed to overcome the limitations of traditional ETC schemes, which rely on predetermined prices and do not

adjust to changes in traffic conditions. By computing tolls based on current traffic flows, DETC schemes aim to improve traffic management and reduce congestion.

Despite that, devising a DETC scheme is still challenging and non-trivial due to its complex problem structure, including a large-scale road network, dynamic traffic flow, uncertainty in traffic demand, and its sequential decision-making problem. Recently, Chen et al. [44] made the first attempt at solving the DETC problem via RL. They formulated this sequential decision-making problem as a Markov Decision Problem (MDP) and proposed an RL method [44]. Despite its good performance in mitigating traffic congestion over other methods, a major limitation of the proposed method, PG-β, is that it is limited to a small-scale road network and cannot work under realistic-sized road networks. Therefore, scaling the current state-of-the-art DETC method to large-scale real-world road networks is vital. We focus on scaling up DETC to large-scale real-world road networks via MARL.

1.2.3 Learning to Coordinate in Environments with Off-Beat Actions

Despite the recent successes of MARL [26, 32, 33, 35, 38], learning effective multi-agent coordination policies for complex multi-agent systems remains challenging. One key challenge is the *off-beat* actions, *i.e.*, actions have execution durations, where the environmental changes are not synchronized with action execution during the execution durations. However, Dec-POMDP [15], which underpins many CTDE-based MARL methods, hinges on the assumption that actions are executed immediately, leading to catastrophic failure for centralized training on various off-beat multi-agent scenarios (OBMAS).

Two key obstacles cause training efficient multi-agent coordination policies in OB-MAS challenging: (i) The action durations are unknown to agents during executions, and communication is constrained and not always feasible; (ii) TD-learning [46] is impeded by the displaced rewards caused by off-beat actions. Thus, the temporal credit assignment is hampered in training. With off-beat actions, the nonstationarity issue, which mainly stems from rewards' time dependency on the agents' past actions, is exacerbated. We aim to tackle these problems in this thesis.

1.2.4 Learning to Generalize for MARL Agents

In recent years, much progress has been achieved in MARL research [17, 18, 47]. However, the MARL agents easily overfit the training environment and perform poorly in evaluation scenarios where other agents behave differently. Despite being crucial to real-world MARL applications, the generalizability problem [3] is largely ignored in current research.

Generalization may have different meanings from different perspectives and research directions. In supervised learning, many works focus on devising novel representations of neural networks [48, 49] to improve the generalization of deep neural network models. In RL, the generalized single agents are those who can generalize to different test environments rather than the training environments [49]. The generalization in MARL we study in this chapter is vital for MARL research. Generalization in MARL [3] means that an agent in multi-agent systems could cooperate or compete with agents possessing unseen policies. With generalizable policies, agents could team up with agents that have unseen policies to achieve a cooperative goal or win against other agents. We aim to train MARL agents who can adapt to new scenarios where other agents' policies are unseen during training.

1.3 Main Contributions

We propose one solution for each of the corresponding multi-agent research problems above. Our main contributions are summarized below.

- To learn risk-sensitive policies for efficient multi-agent coordination (introduced in Section 1.2.1), we propose RMIX, a novel cooperative risk-sensitive MARL method. Specifically, our contributions are in three folds: (i) we first learn the return distributions of individuals by using Dirac Delta functions 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 optimizer for CVaR calculation to handle the temporal nature of stochastic outcomes as well as tune the risk level during executions. The dynamic risk level optimizer 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) as our method focuses on optimizing the CVaR policies via CTDE, we finally optimize CVaR policies with CVaR values as target estimators in TD error via centralized training, and CVaR values are used as auxiliary local rewards to update local return distributions via Quantile Regression loss.

- To address the scaling-up issue in dynamic Electronic Toll Collection (DETC) (introduced in Section 1.2.2), we propose a novel MARL approach. We make several key contributions: (i) an enhancement over the state-of-the-art RL-based method with a deep neural network representation of the policy and value functions and a temporal difference learning framework to accelerate the update of target values; (ii) a novel edge-based graph convolutional neural network (eGCN) to extract the spatiotemporal correlations of the road network state features; (iii) a novel cooperative multi-agent reinforcement learning (MARL) that divides the whole road network into partitions according to their geographic and economic characteristics and trains a tolling agent for each partition. Experimental results show that such a divide-and-conquer approach can scale the RL method and significantly outperform the state-of-the-art method.

- To learn cooperative policies for agents in environments with off-beat actions (introduced in Section 1.2.3), we propose a novel reward redistribution method built on our novel *graph-based episodic memory* called LeGEM-core to learn efficient multi-agent coordination in environments where *off-beat* actions are prevalent, *i.e.*, all actions have execution durations, and during execution durations, the environmental changes are not synchronized with action execution. LeGEM-core memorizes agents' past experiences explicitly and enables credit assignment in MARL training. We name our solution method as LeGEM. Empirical results show that it significantly boosts multi-agent coordination in OBMAS and achieves leading performance.

- To improve the generalization performance of MARL agents in environments where other agents' policies are unseen (introduced in Section 1.2.4), we propose RPM, which aims to learn generalizable policies for MARL agents in evaluation scenarios where other agents behave differently. The main idea of

RPM is to train MARL policies via gathering massive multi-agent interaction data. We first rank each agent's policies by its training episode return and then *save the ranked policies in the memory*; when an episode starts, each agent can randomly select a policy from the RPM as the behavior policy. This novel self-play framework guarantees the diversity of multi-agent interaction in the training data. Experimental results on Melting Pot demonstrate that RPM enables MARL agents to interact with unseen agents and gain increased performance.

1.4 Thesis Organization

The organization of this thesis is as follows. Chapter 2 introduces the preliminaries of RL and MARL research. Chapter 3 introduces the solution to learning risk-sensitive policies for MARL. Chapter 4 introduces the solution to learning to scale up with cooperative MARL. Chapter 5 introduces the method of learning to coordinate in environments with off-beat actions for MARL agents. Chapter 6 introduces the method of learning generalizable policies for MARL. Chapter 7 concludes this thesis and discusses future works. The overall structure of this thesis is also depicted in Figure 1.2.

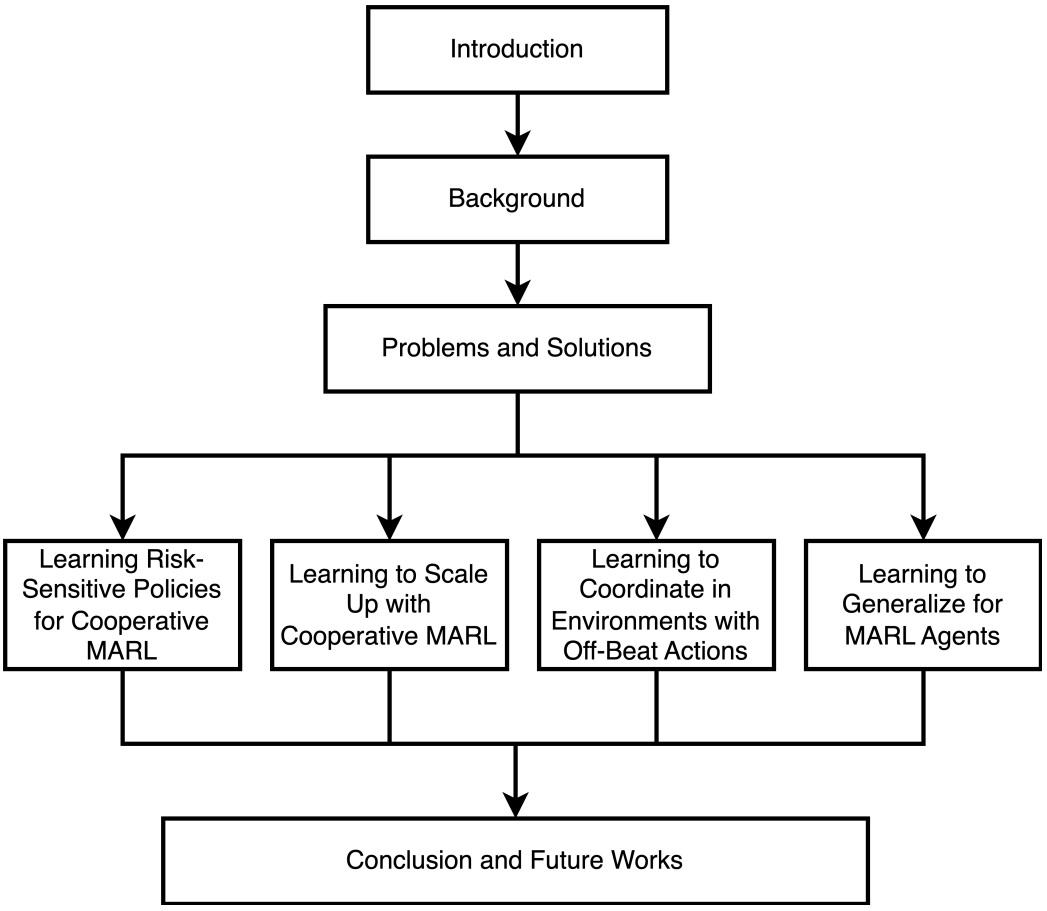


FIGURE 1.2: The overall structure of this thesis.

Chapter 2

Background

In this chapter, we present the background knowledge, mathematical formulations and objectives that are necessary for readers to follow our research works in the rest of this thesis. We first introduce the setting and problem-solving paradigm in the single-agent setting. In Section 2.1, we introduce Markov Decision Process (MDP) followed by Reinforcement Learning (RL) in Section 2.2. Then, we present the multi-agent setting and MARL. We introduce two popular multi-agent formulations, Decentralized Partially Observable Markov Decision Process (Dec-POMDP) in Section 2.3 and Markov Games in Section 2.4. We finally introduce Multi-Agent Reinforcement Learning (MARL) in Section 2.5. Note that the novel Dec-POMDP framework introduced in Chapter 5 is based on Dec-POMDP.

2.1 Markov Decision Process

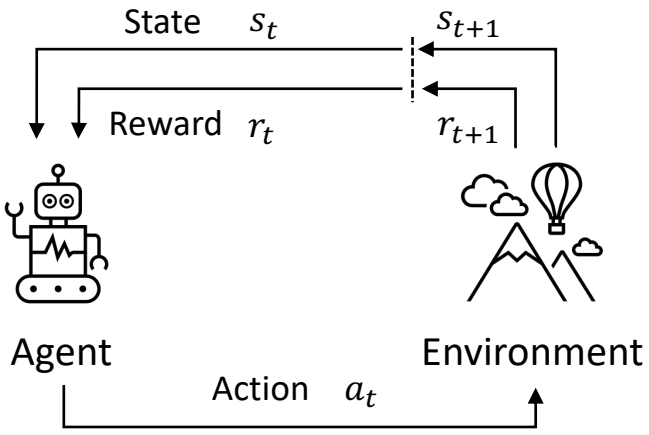


FIGURE 2.1: The agent–environment interaction in a Markov decision process.

Markov Decision Process (MDP) [50] is a widely used mathematical framework to model sequential decision-making problems in a stochastic environment. MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ represents the set of states, $\mathcal{A}$ represents the set of actions, $\mathcal{P}$ represents the transition probability function, $\mathcal{R}$ represents the reward function, and γ is a discount factor. At each time step, an agent observes a state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$ from a policy $\pi(s_t)$. Then, the environment transitions to a new state $s_{t+1} \in \mathcal{S}$ with probability $\mathcal{P}(s_{t+1}|s_t, a_t)$, and the agent receives a reward r_t according to the reward function $\mathcal{R}(s_t, a_t)$. The goal of the agent is to maximize the cumulative discounted reward G_t over a finite horizon starting from time step t:

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k \tag{2.1}$$

$$= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots + \gamma^{T-t} r_T$$

$$= r_t + \gamma \left(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T \right)$$

$$= r_t + \gamma G_{t+1} \tag{2.2}$$

where T is the time step when the decision horizon ends. Solving an MDP involves finding an optimal policy π^* that maximizes the cumulative discounted reward G_t. We illustrate the agent-environment interaction in a MDP in Figure 2.1.

2.2 Reinforcement Learning

There are many methods to solve MDPs. Dynamic programming methods are representative methods for solving MDPs, such as Value Iteration (VI) and Policy Iteration (PI) [29, 50]. However, one limitation of value iteration and policy iteration is that they require knowledge of the transition probabilities and reward functions of the environment, which may not be available or accurate in many real-world applications. In contrast, model-free RL[1] methods can learn directly from experience without requiring this information. In addition, solving VI and PI can be computationally expensive, especially for large state and action spaces. RL methods, such as Q-learning [51] and policy gradient method [52], are often more scalable and can handle high-dimensional state and action spaces. Furthermore, VI and PI assume that the MDP is stationary, which means that the transition probabilities and reward functions do not change over time. However, in many real-world applications, the environment may be non-stationary, while RL methods can adapt to such changes more effectively.

Over the past decade, by leveraging the Deep Neural Networks [28], Deep Reinforcement Learning (DRL) has made remarkable advances in many domains, including arcade video games [23, 53], complex continuous robot control [24, 54], the game of Go [55], magnetic control of tokamak plasmas [56] and matrix multiplication [57]. More concretely, the key ideas of RL are the value function $V^{\pi}(s)$ and the Q value function $Q^{\pi}(s,a)$, which are used to estimate the value of being in a particular state or taking a specific action in a state by following a policy π, respectively[2]. Formally, $V^{\pi}(s)$ is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t \mid s_t = s\right] = \mathbb{E}_{\pi}\left[\sum_{k=t}^{T} \gamma^{k-t} r_k \mid s_t = s\right], \text{ for all } s \in \mathcal{S} \tag{2.3}$$

where t is any time step and $\mathbb{E}_{\pi}[\cdot]$ denotes the expected value of the discounted accumulative rewards given that the agent follows policy π. $V^{*}(s)$ represents the optimal value function for a given state. It denotes the expected discounted cumulative rewards that an agent can achieve from a specific state s, following the

[1] Unless specified otherwise, we refer to RL as model-free RL where the transition function and the reward function are unknown to the agent.

[2] For brevity, we will omit the superscript π in Q value functions and value functions in the rest of this thesis.

optimal policy π^*. It is defined as $V^*(s) = \max_\pi V^\pi(s)$ for all $s \in \mathcal{S}$. Similarly, we define the Q value function:

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[G_t \mid s_t = s, a_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=t}^{T} \gamma^{k-t} r_k \mid s_t = s, a_t = a \right] \tag{2.4}$$

$Q^*(s,a)$ represents the optimal action-value function for a given state-action pair. It denotes the expected discounted cumulative rewards that an agent can achieve from taking action a in state s, and then following the optimal policy π^* thereafter. It is defined as $Q^*(s,a) = \max_\pi Q^\pi(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. Value functions satisfy recursive relationships similar to Equation 2.2. For any policy π and any state s, the following consistency condition holds between the value of s and the value of its successor states:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi \left[G_t \mid s_t = s \right] \\ &= \mathbb{E}_\pi \left[r_t + \gamma G_{t+1} \mid s_t = s \right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r \mathcal{P}\left(s', r \mid s, a\right) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid s_{t+1} = s' \right] \right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} \mathcal{P}\left(s', r \mid s, a\right) \left[r + \gamma V^\pi\left(s'\right) \right], \quad \text{for all} s \in \mathcal{S}. \end{aligned} \tag{2.5}$$

Once we have found the optimal value functions, V^* or Q^*, we can write Q^* in terms of V^* as follows:

$$Q^*(s,a) = \mathbb{E}\left[r_t + \gamma V^*\left(s_{t+1}\right) \mid s_t = s, a_t = a \right]. \tag{2.6}$$

Following the Bellman equation for state values (Equation 2.5), V^* and Q^* satisfy the Bellman optimality equations [29, 58]:

$$V^*(s) = \max_{a \in \mathcal{A}} Q^{\pi_*}(s,a) \tag{2.7}$$

$$\begin{aligned} &= \max_a \mathbb{E}_{\pi_*} \left[G_t \mid s_t = s, a_t = a \right] \\ &= \max_a \mathbb{E}_{\pi_*} \left[r_t + \gamma G_{t+1} \mid a_t = s, a_t = a \right] \\ &= \max_a \mathbb{E}\left[r_t + \gamma V^*\left(s_{t+1}\right) \mid s_t = s, a_t = a \right] \\ &= \max_a \sum_{s',r} \mathcal{P}\left(s', r \mid s, a\right) \left[r + \gamma V^*\left(s'\right) \right]. \end{aligned} \tag{2.8}$$

or

$$Q^*(s,a) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q^*\left(s_{t+1}, a'\right) \mid s_t = s, a_t = a\right]$$
$$= \sum_{s',r} \mathcal{P}\left(s', r \mid s, a\right)\left[r + \gamma \max_{a'} Q^*\left(s', a'\right)\right]. \tag{2.9}$$

2.2.1 Temporal-Difference Learning

Temporal-difference (TD) learning is central to RL [29, 46]. TD learning can learn policies from experience without a model of the environment's dynamic. The interaction between the agent and the environment generates the experiences. Unlike Monte Carlo methods that update the estimate based on the discounted accumulated rewards from the current time step to the end of the episode, TD learning updates its estimates based on the reward of the current time step and the other learned estimates. Formally, its updated rule of V is:

$$V\left(s_t\right) := V\left(s_t\right) + \alpha\left[r_t + \gamma V\left(s_{t+1}\right) - V\left(s_t\right)\right], \tag{2.10}$$

where $\alpha \in (0, 1]$ is a constant value that determines how much the estimated value is updated with each iteration. In Monte Carlo, the target is $G_t = \sum_{t=0}^{\infty} \gamma^t r_t$, whereas the target for the TD update is $r_t + \gamma V\left(s_{t+1}\right)$. The TD-error is defined as $\delta_t \doteq r_t + \gamma V\left(S_{t+1}\right) - V\left(S_t\right)$. This method is a special case of TD(λ) and n-step TD [29]. We call it TD(0) or one-step TD.

2.2.2 Q-Learning and DQN

Q-learning [51] is one of the early milestones in RL. It is defined as:

$$Q\left(s_t, a_t\right) := Q\left(s_t, a_t\right) + \alpha\left[\left(r_t + \gamma \max_{a} Q\left(s_{t+1}, a\right)\right) - Q\left(s_t, a_t\right)\right], \tag{2.11}$$

where the learned action-value function, Q, directly approximates Q^* via the `max` operator over the Q in the TD target. While taking action, it adopts ϵ-greedy for exploration. The agent has an $\epsilon \in [0, 1]$ probability of selecting an action randomly sampled from the action space and a $1 - \epsilon$ probability of selecting an action that has the largest Q value. Mnih et al. proposed DQN [23], which used a deep neural

network to parameterize the $Q(s, a; \theta)$ function with parameters θ. To find the optimal θ, DQN minimizes the following regression loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}\left[\left(y^{\mathrm{DQN}} - Q(s, a; \theta)\right)^2\right], \tag{2.12}$$

where $y^{\mathrm{DQN}} = r + \gamma \max_{a'} Q\left(s', a'; \theta_i^-\right)$. θ_i^- is the parameter of the target Q and copied periodically from θ. There are many variants of DQN, such as C51 [59], Rainbow [60], QR-DQN [61], and IQN [62]. These methods achieved human-level performance in Atari games [63]. We will not introduce them in this thesis due to the page limit. For more information, readers can read their papers.

2.2.3 Policy Gradient and PPO

In this section, we introduce policy gradient methods [52, 64]. Unlike value-based RL methods that optimize the policy via optimizing the Q or V, policy gradient methods directly optimize the policy $\pi(s; \theta)$ via conducting gradient ascent on a performance measure $\mathcal{J}(\theta)$, *e.g.*, V^{π_θ}, the true value function for π_θ. By deriving the gradient of $\mathcal{J}(\theta)$,

$$\nabla \mathcal{J}(\theta) = \mathbb{E}_\pi\left[G_t^\infty \frac{\nabla \pi\left(a_t \mid s_t; \theta\right)}{\pi\left(a_t \mid s_t; \theta\right)}\right], \tag{2.13}$$

we can get the REINFORCE [52] policy gradient method. REINFORCE estimates the gradient of the expected return using the Monte Carlo return $G_t^\infty = \sum_{k=t}^{\infty} \gamma^{k-t} r_k$. The resulting gradient estimates can have high variance, leading to slow convergence and instability in the learning process.

In Actor-Critic methods [24, 54, 65–67], the actor is the policy, and the critic is the value function. The gradient is estimated using the critic rather than G_t, which updates the policy. The TD error $r_t + \gamma V(s_{t+1}) - V(s_t)$ is used to replace the critic in the gradient. The critic is trained with TD learning.

Proximal Policy Optimization (PPO) [24] is an Actor-Critic method. It works by optimizing a surrogate objective function that approximates the expected improvement in the policy. This objective function is a clipped surrogate objective that constrains the size of the policy update to prevent large changes that could result in unstable learning. The clipping function ensures that the new policy does not

deviate too much from the old policy, which improves the stability and robustness of the learning process.

Formally, PPO updates the parameter θ_{k+1} of its policy at iteration $k+1$ by taking multiple steps of stochastic gradient ascent via:

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_{s,a\sim\pi_{\theta_k}} \left[\mathcal{L}\left(s, a, \theta_k, \theta\right)\right], \tag{2.14}$$

where $\mathcal{L}$ is defined as:

$$\mathcal{L}(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \text{clip}(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1 \pm \epsilon) A^{\pi_{\theta_k}}(s, a)\right), \tag{2.15}$$

where $\text{clip}(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1 \pm \epsilon) = \text{clip}(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1 + \epsilon, 1 - \epsilon)$ and ϵ is a hyperparameter that determines the extent to which the new policy is allowed to deviate from the old policy during the update step.

PPO has become a popular choice for solving a wide range of RL problems, including complex tasks with high-dimensional states and action spaces. It has also achieved state-of-the-art results in many benchmark environments, such as the Atari games [63] and OpenAI Gym [68].

2.3 Decentralized Partially Observable Markov Decision Process

A fully cooperative MARL problem can be described as a decentralized partially observable Markov decision process (Dec-POMDP) [15], which can be formulated as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{U}, \mathcal{P}, R, \Upsilon, O, \mathcal{N}, \gamma \rangle$, where $\boldsymbol{s} \in \mathcal{S}$ denotes the state of the environment (we illustrate it in Figure 2.2). Each agent $i \in \mathcal{N} := \{1, ..., N\}$ chooses an action $u_i \in \mathcal{U}$ at each time step[3], giving rise to a joint action vector, $\boldsymbol{u} := [u_i]_{i=1}^N \in \mathcal{U}^N$. $\mathcal{P}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{u}) : \mathcal{S} \times \mathcal{U}^N \times \mathcal{S} \mapsto \mathcal{P}(\mathcal{S})$ is a Markovian transition function. Every agent shares the same joint reward function $R(\boldsymbol{s}, \boldsymbol{u}) : \mathcal{S} \times \mathcal{U}^N \mapsto \mathcal{R}$, and $\gamma \in [0, 1)$ is the discount factor. Due to partial observability, each agent has individual partial observation $\upsilon \in \Upsilon$ of the state, according to the observation function

[3]In multi-agent systems and MARL, the notation for actions differs from that in MDP and RL in Sections 2.1 and 2.2, aligning with the prevalent convention in related MARL literature [26, 35]. This notation will be consistently used throughout the remainder of this thesis.

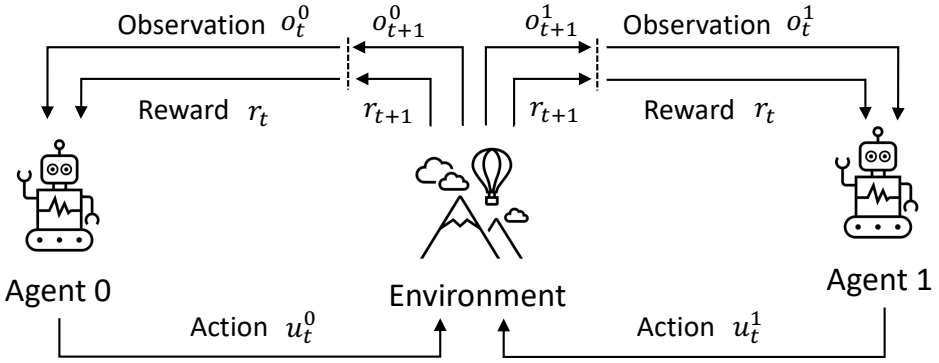


FIGURE 2.2: A two-agent multi-agent system. Note that in Dec-POMDP, the two agents share the same reward since they should coordinate in the environment. However, in Markov games, agents receive different reward values because agents have different goals. They may cooperate at the beginning of the game and betray at the end of the game.

$O(\boldsymbol{s}, i) : \mathcal{S} \times \mathcal{N} \mapsto \Upsilon$. Each agent learns its own policy $\pi_i(u_i|\tau_i) : \mathcal{T} \times \mathcal{U} \mapsto [0, 1]$ given its action-observation history $\tau_i \in \mathcal{T} := (\Upsilon \times \mathcal{U})$. In Dec-POMDP, the reward at each time step is shared among all agents. Thus, the credit assignment issue between agents is one of the key research challenges. Many MARL methods were proposed to tackle it [26, 33, 35–37, 40, 69], among which QMIX [26], QPLEX [40] and HAPPO [37] are representative methods.

2.4 Markov Games

We use a tuple $\mathcal{G} = \langle \mathcal{N}, \mathcal{S}, \mathcal{U}, \mathcal{O}, \mathcal{P}, R, \gamma, \rho \rangle$ to represent Markov Games [70]. $\mathcal{N}$ is a set of agents with the size $|\mathcal{N}| = N$; $\mathcal{S}$ is a set of states; $\boldsymbol{u} = \times_{i=1}^{N} u_i \in \mathcal{U}$ is a set of joint actions with $u_i \in \mathcal{U}_i$ denoting the action for an agent i; $\mathcal{O} = \times_{i=1}^{N} \mathcal{O}_i$ is the observation set, with $\mathcal{O}_i$ denoting the observation set of the agent i; the observation of the agent i at time step t is a partial observation of the state s_t; $\mathcal{P} : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$ is the transition function and $R = \times_{i=1}^{N} r_i$ is the reward function where $r_i : \mathcal{S} \times \mathcal{U} \to \mathbb{R}$ specifies the reward for the agent i given the state and the joint action; γ is the discount factor; the initial states are determined by a distribution $\rho : \mathcal{S} \to [0, 1]$. Given a state $s \in \mathcal{S}$, each agent $i \in \mathcal{N}$ chooses its action u_i and obtains the reward $r(s, \boldsymbol{u})$ with the private observation $o_i \in \mathcal{O}_i$, where $\boldsymbol{u} = \{u_i\}_{i=1}^{N}$ is the joint action. The joint policy of agents is denoted as $\boldsymbol{\pi_\theta} = \{\pi_{\theta_i}\}_{i=1}^{N}$ where $\pi_{\theta_i} : \mathcal{S} \times \mathcal{U}_i \to [0, 1]$ is the policy for the agent i. The objective of each agent is to maximize its total return $R_i = \sum_{t=0}^{\infty} \gamma^t r_i^t$. In Markov Games settings, agents can

cooperate [25, 71, 72], compete with other agents for resources or rewards [18, 73] or even betray the cooperation [74]. Many MARL methods solve Markov Games problems [25, 71, 75–81], among which MADDPG [25], MF-Q [71] and PSRO [75] are representative methods.

2.5 Multi-Agent Reinforcement Learning

MARL[4] aims to solve multi-agent systems problems with RL. Each agent's policy π_i is optimized by maximizing the following objective:

$$\mathcal{J}(\pi_i) \triangleq \mathbb{E}_{s_{0:\infty}\sim\rho_{\mathcal{G}}^{0:\infty}, a_{0:\infty}^{i}\sim\pi_i}\left[\sum_{t=0}^{\infty}\gamma^t r_t^i\right],$$

where $\mathcal{J}(\pi_i)$ is a performance measure for policy gradient RL methods [52, 54, 66]. Each policy's Q value Q_i is optimized by minimizing the following regression loss [23] with TD-learning [46]:

$$\mathcal{L}(\theta_i) \triangleq \mathbb{E}_{\mathcal{D}'\sim\mathcal{D}}\left[\left(y_t^i - Q_{\theta_i}^i\left(\boldsymbol{s}_t, \boldsymbol{u}_t, s_t^i, u_t^i\right)\right)^2\right],$$

where $y_t^i = r_t^i + \gamma \max_{\boldsymbol{u}'} Q_{\bar{\theta}_i}^i\left(\boldsymbol{s}_{t+1}, \boldsymbol{u}', s_t^i, u^{i,\prime}\right)$. θ_i are the parameters of the agents. $\bar{\theta}_i$ is the parameter of the target Q^i and periodically copied from θ. $\mathcal{D}$ is the replay buffer that stores all agents' past transitions where each transition contains the global state $\boldsymbol{s}$, all agents' actions $\boldsymbol{u}$ and the global reward r. $\mathcal{D}'$ contains transitions that are sampled from the replay buffer $\mathcal{D}$ for TD learning. The number of sampled transitions is far smaller than the number of transitions in the replay buffer.

[4]Unless specified otherwise, we refer to MARL as model-free MARL where the transition function and the reward function are unknown to the agents. In RL, model-based RL methods [82–84] are efficient and have gained great success. However, they are not scalable and still not able to gain convincing results due to non-stationary multi-agent environments. Furthermore, model-free MARL methods are scalable and computationally flexible to multi-agent scenarios. Therefore, we focus on the model-free MARL research problems in this thesis.

2.6 Chapter Summary

This chapter introduces the background of the proposed MARL methods in this thesis. In particular, we introduce the background of single-agent RL, which consists of the Markov Decision Process, Temporal-difference Learning, Q-Learning, DQN, Policy Gradient, and PPO. We then introduce the background of multi-agent systems, including the Decentralized Partially Observable Markov Decision Process, Markov Games, and MARL.

Chapter 3

Learning Risk-Sensitive Policies for Cooperative MARL

Current value-based MARL methods optimize individual Q values to guide individuals' behaviors for optimal policies via centralized training with decentralized execution (CTDE). However, such expected, *i.e.*, risk-neutral, Q value is not sufficient even with CTDE due to the randomness of rewards and the uncertainty in environments, which is ubiquitous in many real-world multi-agent problems, such as multi-robot search and rescue [41]. These issues cause the failure of these methods to train coordinating agents in complex multi-agent environments. To address these issues, this chapter proposes RMIX[1], a novel cooperative MARL method with the Conditional Value at Risk (CVaR) measure over the learned distributions of individuals' Q values. Specifically, we first learn the return distributions of individuals to calculate CVaR for decentralized execution analytically. Then, to handle the temporal nature of the stochastic outcomes during executions, we propose a dynamic risk level optimizer for risk level tuning. Finally, we optimize the CVaR policies with CVaR values used to estimate the target in TD error during centralized training and the CVaR values are used as auxiliary local rewards to update the local distribution via Quantile Regression loss. Empirically, RMIX outperforms many state-of-the-art methods on various multi-agent risk-sensitive navigation scenarios and challenging StarCraft II cooperative tasks, demonstrating enhanced coordination and revealing improved sample efficiency.

[1]The work in this chapter has been published in [10].

3.1 Introduction

Recently, many researchers put their efforts into extending the RL methods into multi-agent systems (MASs) to solve multi-agent problems [17, 19, 19, 21] (see Chapter 1) and proposed many novel MARL methods. Most of these methods focus on decomposing the global Q value into individual Q values with different constraints and network architectures. However, they ignore the fact that such expected, *i.e.*, risk-neutral, Q value is not sufficient as optimistic actions executed by some agents can impede team coordination, such as imprudent actions in hostage rescue operations. In complex environments, agents can encounter states with extremely high/low rewards while MARL methods cannot handle the noisy nature of rewards, which leads to the failure of learning multi-agent cooperation. Even further, given that the environment is nonstationary from the perspective of each agent, decision-making over the agent's return distribution [59, 85] takes events of potential return into account, which makes agents able to address uncertainties in the environment compared with simply taking the expected values for execution. However, current MARL methods did not extensively investigate these aspects.

Motivated by the previous reasons, we intend to extend the risk-sensitive RL [86–89] ("Risk" refers to the uncertainty of future outcomes [85]) to MARL, 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) [90]. Among these risk measures, CVaR has been gaining popularity due to both theoretical and computational advantages [90, 91]. However, there are two main obstacles: (i) most of the previous works focus on risk-neutral or static risk levels in the single-agent settings, ignoring the randomness of reward and the temporal structure of agents' trajectories [85, 87, 92, 93]; (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 have sub-optimal behaviors in MARL.

In this chapter, we propose RMIX, a novel cooperative risk-sensitive MARL method. Specifically, our contributions are in three folds: (i) we first learn the return distributions of individuals by using Dirac Delta functions 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 optimizer for CVaR calculation to handle the temporal nature of stochastic outcomes as well as tune the risk level during executions. The dynamic risk level optimizer 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) as our method focuses on optimizing the CVaR policies via CTDE, we finally optimize CVaR policies with CVaR values as target estimators in TD error via centralized training, and CVaR values are used as auxiliary local rewards to update local return distributions via Quantile Regression loss. These also allow our method to achieve temporally extended exploration and enhanced temporal coordination, which are keys to solving complex multi-agent tasks. Empirically, we show that RMIX outperforms many state-of-the-art methods on various multi-agent risk-sensitive navigation scenarios and challenging StarCraft II cooperative tasks, demonstrating enhanced coordination and revealing improved 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.

3.2 Background

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

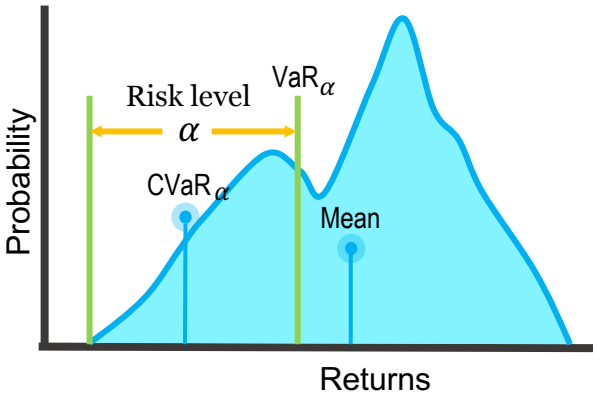


FIGURE 3.1: The illustration of Conditional Value at Risk (CVaR).

Conditional Value at Risk. Conditional Value at Risk (CVaR) is a coherent risk measure and enjoys computational properties [90] that are derived for loss distributions in discreet decision-making in finance. It has gained popularity in various engineering and finance applications. CVaR (as illustrated in Figure 3.1) is the expectation of values that are less equal than the α-percentile value of the distribution over returns. Formally, let $X \in \mathcal{X}$ be a bounded random variable with cumulative distribution function $F(x) = \mathscr{P}[X \leq x]$ 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 $\text{CVaR}_\alpha(X) := \sup_\nu \left\{\nu - \frac{1}{\alpha}\mathbb{E}[(\nu - X)^+]\right\}$ [94] when X is a discrete random variable. Correspondingly, $\text{CVaR}_\alpha(X) = \mathbb{E}_{X\sim F}[X|X \leq F^{-1}(\alpha)]$ [95] when X has a continuous distribution. The α-percentile value is the value at risk (VaR). For ease of notation, this chapter writes CVaR as a function of the CDF F, $\text{CVaR}_\alpha(F)$.

Risk-sensitive Reinforcement Learning. Risk-sensitive Reinforcement Learning (RL) uses risk criteria over policy/value, which is a sub-field of the Safety RL [96]. Von Neumann and Morgenstern [97] 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 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 [85, 98] and power formula [85]. However, few works have been done in MARL, and they cannot be easily extended. The proposed method fills this gap.

Centralized Training with Decentralized Execution. Centralized Training with Decentralized Execution (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 [26, 33, 35] 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 Individual-Global-Max (IGM) [35] principle where the optimal joint actions across all agents are equivalent to the collection of individual optimal actions of each agent. To achieve scalability, existing CTDE methods typically learn a shared local value or policy network for all agents.

Distributional Reinforcement Learning. Distributional RL methods [59, 61] apply a sample-based approximation to distributional versions of the usual Bellman operators. For example, one can define a distributional Bellman operator [59] as $\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ as

$$\mathcal{T}^\pi Z_\pi(s,a) \overset{D}{:=} R(s,a) + \gamma P^\pi Z(s,a) \tag{3.1}$$

where $\overset{D}{=}$ denotes equality in distribution, and the transition operator is defined as $P^\pi Z(s,a) \overset{D}{:=} Z(s',a')$ with $s' \sim P(\,\cdot\,|s,a)$, $a' \sim \pi(s)$. The optimality version $\mathcal{T}$ is similarly any $\mathcal{T}Z = \mathcal{T}^\pi Z$ where π is an optimal policy w.r.t. expected return. Note that this is not necessarily unique when there are multiple optimal policies. The distributional Bellman operator has a convergence guarantee [59, 99], which laid the theoretical foundation for follow-up distributional RL methods, such as QR-DQN [61], FQF [100] and NC-QR-DQN [101].

3.3 Solution Method: RMIX

This section presents the framework RMIX in Figure 3.2, 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, this chapter first introduces the CVaR operator to analytically calculate the CVaR value with the modeled individual distribution of each agent in Section 3.3.1 and then proposes the dynamic risk level optimizer to alleviate time-consistency issue in Section 3.3.2. Finally, this chapter provides the details of centralized training of RMIX in Section 3.3.3.

3.3.1 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 parameterized return distribution [90]. However, the sampling method is usually computationally expensive [92]. Therefore, we let each agent learn a return distribution parameterized by a mixture of Dirac Delta (δ) functions [2], which is demonstrated to be

[2]The Dirac Delta is a *Generalized function* in the theory of distributions and not a function given the properties of it. We use the name *Dirac Delta function* by convention.

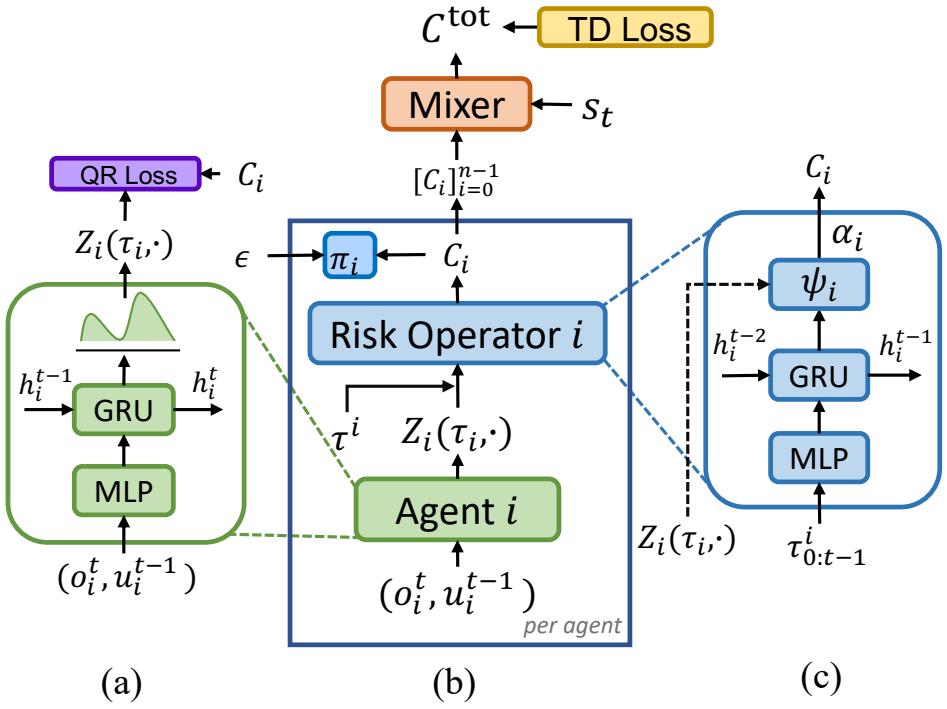


FIGURE 3.2: The framework (dotted arrow indicates that gradients are blocked during training). (a) Agent's policy network. (b) The overall architecture (agent network and mixer). (c) Risk operator. Each agent i applies an individual risk operator Π_{α_i} on its return distribution $Z_i(\cdot,\cdot)$ to calculate $C_i(\cdot,\cdot,\cdot)$ for execution given risk level α_i predicted by the dynamic risk level optimizer ψ_i. $\{C_i(\cdot,\cdot,\cdot)\}_{i=1}^N$ are fed into the mixer for centralized training.

highly expressive and computationally efficient [59]. By following [59], we define the parameterized return distribution of each agent i at time step t as:

$$Z_i^t(\tau_i, u_i^{t-1}) = \sum_{j=1}^{M} \mathscr{P}_j(\tau_i, u_i^{t-1}) \delta_j^i(\tau_i, u_i^{t-1}) \tag{3.2}$$

where M is the number of Dirac Delta functions. $\delta_j^i(\tau_i, u_i^{t-1})$ is the j-th Dirac Delta function of agent i and indicates the estimated return which can be computed by neural networks in practice. $\mathscr{P}_j(\tau_i, u_i^{t-1})$ is the corresponding probability of the estimated return given local observations and actions. τ_i and u_i^{t-1} are trajectories (up to that time step) and actions of agent i, respectively. With the individual return distribution $Z_i^t(\tau_i, u_i^{t-1}) \in \mathcal{Z}$ and cumulative distribution function (CDF) $F_{Z_i(\tau_i, u_i^{t-1})}$, we define the CVaR operator Π_{α_i}, at a risk level α_i ($\alpha_i \in (0,1]$ and $i \in [1,\dots,N]$), over return as[3] $C_i^t(\tau_i, u_i^{t-1}, \alpha_i) = \Pi_{\alpha_i^t} Z_i^t(\tau_i, u_i^{t-1}) := \text{CVaR}_{\alpha_i^t}(F_{Z_i^t(\tau_i, u_i^{t-1})})$

[3]We will omit the superscript of t in Z_i^t in the rest of this chapter for notation brevity.

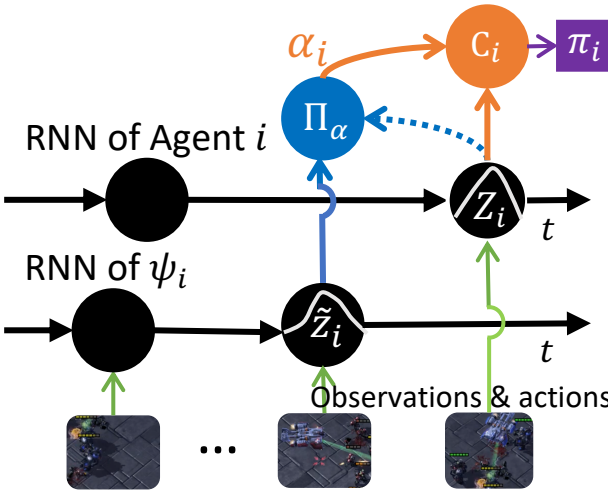


FIGURE 3.3: Agent architecture. It has two Recurrent Neural Network (RNN) [1] models. One RNN is for the Q network (top) and another one is for the risk level optimizer (bottom). We implement the two RNN models with Gated Recurrent Unit (GRU) [2], which is efficient in practice. Note that $\tilde{Z}_i$ shares the same neural network layer with Z_i, while the RNNs of agent i and ψ function independently, without sharing neural network layers.

where $C \in \mathcal{C}$. As we use CVaR on return distributions, it corresponds to risk-neutrality (expectation, $\alpha_i = 1$) and indicates the improving degree of risk-aversion ($\alpha_i \to 0$). CVaR_{α_i} can be estimated in a nonparametric way given the ordering of Dirac Delta functions $\{\delta_j^i\}_{j=1}^m$ [102] by leveraging the individual distribution:

$$\text{CVaR}_{\alpha_i} = \sum_{j=1}^{M} \mathscr{P}_j \delta_j^i \mathbf{1}\left\{\delta_j^i \leq \hat{v}_{M,\alpha_i}\right\}, \tag{3.3}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function and $\hat{v}_{M,\alpha_i}$ is estimated value at risk from $\hat{v}_{M,\alpha_i} = \delta^i_{\lfloor M\times(1-\alpha_i)\rfloor}$ with $\lfloor\cdot\rfloor$ 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 $\arg\max_{u_i} C_i(u_i|\tau_i, u_i^{t-1}, \alpha_i)$.

3.3.2 Risk Level Optimizer

The values of risk levels, *i.e.*, $\alpha_i \in (0, 1]$ ($i \in [1, \ldots, N]$), in Equation 3.3 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, especially in environments where agents have partial observations and should cooperate to complete tasks, which makes it hard to tune the best risk

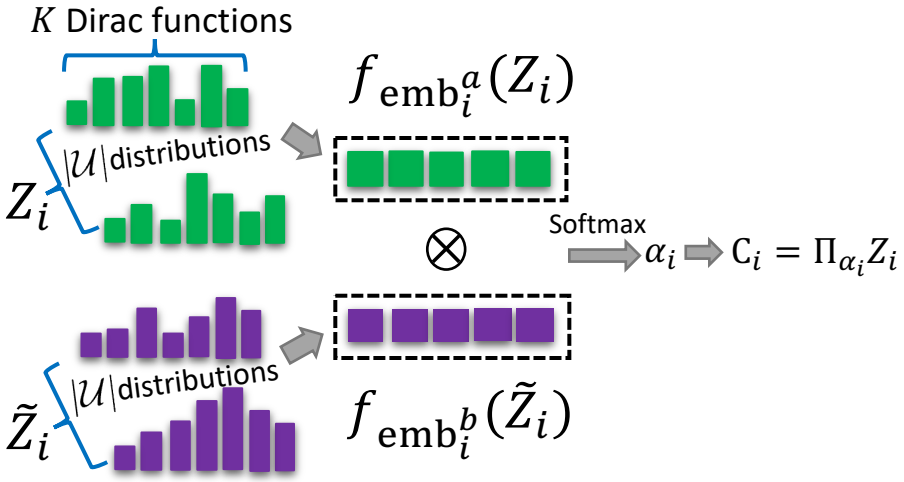


FIGURE 3.4: Risk level optimizer ψ_i.

level and impede centralized training in the evolving multi-agent cooperation environments. Furthermore, learning a return distribution for each time step incurs high computational costs and lacks scalability. Therefore, we propose the dynamic risk level optimizer, which determines the risk levels of agents by explicitly taking into account the temporal nature of the stochastic outcomes, such as other agents' behaviors and the presence of opponents in the agent's observation, to alleviate time-consistency issue [91, 103] and stabilize the centralized training. Specifically, we represent the risk operator Π_α by a deep neural network, which calculates the CVaR value (see Equation 3.3 in Section 3.3.1) with dynamic risk level output by the risk level optimizer.

We show the architecture of agent i in Figure 3.3 (agent network and risk operator with risk level optimizer ψ_i, as shown in Figure 3.2) and illustrate how ψ_i works with agent i for CVaR calculation in practice in Figure 3.4. Each agent has a risk level optimizer $\psi_i (i \in [1, \cdot, N])$. At time step t, agent i's return distribution is Z_i and its last return distribution at time step $t-1$ is $\tilde{Z}_i$. Then we conduct the inner product to measure the discrepancy between $f_{\text{emb}_i^a}(Z_i)^k$, the embedding of individual return distribution Z_i, and $f_{\text{emb}_i^b}(\tilde{Z}_i)^k$, the embedding of distribution $\tilde{Z}_i$. For agent i, using the same underlying GRU [2] network to get the $\tilde{Z}_i$ can harm the performance of the agents because the deep learning optimizer should optimize the parameters of the GRU, the distribution Z_i and the dynamic risk levels (see Figure 3.3) simultaneously in one update path with TD learning. To mitigate this

issue, we feed agent i's trajectories $\phi_i(\tau_i^{0:t-1}, u_i^{t-1})$ into another GRU network and its output is fed into the neural network layer of agent i's Z_i. Thus, we can get $\tilde{Z}_i$.

In ψ_i for agent i, we discretize the risk level, whose range is $(0, 1]$, evenly into K levels for the convenience of computing. The probability of the k-th risk level α_i^k is defined as:

$$\mathscr{P}(\alpha_i^k) = \frac{\exp\left(\left\langle f_{\text{emb}_i^a}(Z_i)^k, f_{\text{emb}_i^b}(\tilde{Z}_i)^k \right\rangle\right)}{\sum_{k'=0}^{K-1} \exp\left(\left\langle f_{\text{emb}_i^a}(Z_i)^{k'}, f_{\text{emb}_i^b}(\tilde{Z}_i)^{k'} \right\rangle\right)}. \tag{3.4}$$

Then we get the $k^* \in [1, \ldots, K]$ with the maximal probability by $\arg\max$ and normalize it into $(0, 1]$, thus $\alpha_i = k^*/K$. α_i is a scalar value and used to calculate the CVaR value (see Equation 3.3 in Section 3.3.1) of each action-return distribution. Finally, we obtain C_i and the policy π_i as illustrated in Figure 3.3. During training, $f_{\text{emb}_i^a}$ updates its weights and the gradients of $f_{\text{emb}_i^a}$ are blocked (the dotted arrow in Figure 3.3) 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 [104, 105] because the agent's current return distribution and its return distribution of the previous time step are separate inputs of their embeddings and there is no *key*, *query* and *value* weight matrices. The dynamic risk level optimizers allow agents to determine the risk level dynamically based on historical return distributions and it is a hyperparameter (e.g., α) tuning strategy as well.

3.3.3 Training

We train RMIX by addressing the two challenging issues: credit assignment and return distribution learning.

Loss Function. As there is only a global reward signal and agents have no access to individuals' rewards, we first utilize the monotonic mixing network (f_m) from QMIX to do Credit assignment. f_m enforces a monotonicity constraint on the relationship between C^{tot} and each C_i for RMIX:

$$\frac{\partial C^{\text{tot}}}{\partial C_i} \geq 0, \forall i \in \{1, 2, \ldots, N\}, \tag{3.5}$$

where $C^{\text{tot}} = f_m(C_1(\cdot,\cdot,\cdot),\cdots,C_N(\cdot,\cdot,\cdot))$ and $C_i(\tau_i, u_i, \alpha_i)$ is the individual CVaR value of agent i. To ease the confusion, the C^{tot} is not the global CVaR value as modeling global return distribution as well as local distribution are challenging with the credit assignment issue in Dec-POMDP problems, and the risk level values are locally decided and used during training. Then, to maximize the CVaR value of each agent, we define the risk-sensitive Bellman operator $\mathcal{T}$:

$$\mathcal{T}C^{\text{tot}}(\boldsymbol{s},\boldsymbol{u}) := \mathbb{E}\left[R(\boldsymbol{s},\boldsymbol{u}) + \gamma \max_{\boldsymbol{u}'} C^{\text{tot}}(\boldsymbol{s}',\boldsymbol{u}')\right] \tag{3.6}$$

The risk-sensitive Bellman operator $\mathcal{T}$ operates on the C^{tot} and the reward, which can be proved to be a contracting operation, as shown in Proposition 3.1.

Proposition 3.1. *$\mathcal{T} : \mathcal{C} \mapsto \mathcal{C}$ is a γ-contraction.*

Proof. We consider the sup-norm contraction,

$$\left|\mathcal{T}C_{(1)}(\boldsymbol{s},\boldsymbol{u}) - \mathcal{T}C_{(2)}(\boldsymbol{s},\boldsymbol{u})\right| \leq \gamma \left\|C_{(1)}(\boldsymbol{s},\boldsymbol{u}) - C_{(2)}(\boldsymbol{s},\boldsymbol{u})\right\|_\infty \quad \forall \boldsymbol{s} \in \mathcal{S}, \boldsymbol{u} \in \mathcal{U}. \tag{3.7}$$

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

In $\{C_i\}_{i=1}^N$, the risk level is fixed and can be considered implicit input. Given two different return distributions $Z_{(1)}$ and $Z_{(2)}$, we prove:

$$\begin{aligned}
\left|\mathcal{T}C_{(1)} - \mathcal{T}C_{(2)}\right| &\leq \max_{\boldsymbol{s},\boldsymbol{u}} \left|\left[\mathcal{T}C_{(1)}\right](\boldsymbol{s},\boldsymbol{u}) - \left[\mathcal{T}C_{(2)}\right](\boldsymbol{s},\boldsymbol{u})\right| \\
&= \max_{\boldsymbol{s},\boldsymbol{u}} \left|\gamma \sum\nolimits_{\boldsymbol{s}'} \mathcal{P}(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{u}) \left(\max_{\boldsymbol{u}'} C_{(1)}(\boldsymbol{s}',\boldsymbol{u}') - \max_{\boldsymbol{u}'} C_{(2)}(\boldsymbol{s}',\boldsymbol{u}')\right)\right| \\
&\leq \gamma \max_{\boldsymbol{s}'} \left|\max_{\boldsymbol{u}'} C_{(1)}(\boldsymbol{s}',\boldsymbol{u}') - \max_{\boldsymbol{u}'} C_{(2)}(\boldsymbol{s}',\boldsymbol{u}')\right| \\
&\leq \gamma \max_{\boldsymbol{s}',\boldsymbol{u}'} \left|C_{(1)}(\boldsymbol{s}',\boldsymbol{u}') - C_{(2)}(\boldsymbol{s}',\boldsymbol{u}')\right| \\
&= \gamma \left\|C_{(1)} - C_{(2)}\right\|_\infty
\end{aligned} \tag{3.8}$$

This further implies that

$$\left|\mathcal{T}C_{(1)} - \mathcal{T}C_{(2)}\right| \leq \gamma \left\|C_{(1)} - C_{(2)}\right\|_\infty \quad \forall \boldsymbol{s} \in \mathcal{S}, \boldsymbol{u} \in \mathcal{U}. \tag{3.9}$$

□

With proposition 3.1, we can leverage the TD learning [29] to compute the maximal CVaR value of each agent, thus leading to the maximal global CVaR value. Following the CTDE convention, we define the TD loss of RMIX:

$$\mathcal{L}_{\Pi}(\theta) := \mathbb{E}_{\mathcal{D}' \sim \mathcal{D}} \left[(y_t^{\text{tot}} - C^{\text{tot}}(\boldsymbol{s}_t, \boldsymbol{u}_t))^2 \right] \tag{3.10}$$

where $y_t^{\text{tot}} = \left(r_t + \gamma \max_{\boldsymbol{u}'} C_{\bar{\theta}}^{\text{tot}}(\boldsymbol{s}_{t+1}, \boldsymbol{u}') \right)$, and $(y_t^{\text{tot}} - C_{\theta}^{\text{tot}}(\boldsymbol{s}_t, \boldsymbol{u}_t))$ is the CVaR TD error of RMIX for updating CVaR values. θ is the parameters of C^{tot} which can be modeled by a deep neural network and $\bar{\theta}$ indicates the parameters of the target network which is periodically copied from θ for stabilizing training [23]. While training, gradients from Z_i are blocked to avoid changing the weights of the agents' network from the dynamic risk level optimizer.

Local Return Distribution Learning. The CVaR estimation relies on accurately updating the local return distribution and the update is non-trivial. However, unlike many deep learning [106] and distributional RL methods [59, 61] where the label and local reward signal are accessible, in the problem of RMIX, the exact rewards for each agent are unknown, which is very common in real-world problems. To address this issue, we first consider CVaR values as dummy rewards of each agent due to its property of modeling the potential loss of return and then leverage the Quantile Regression (QR) loss used in Distributional RL [61] to explicitly update the local distribution decentrally. More concretely, QR aims to estimate the quantiles of the return distribution by minimizing the quantile regression loss between $Z_i(\tau_i, u_i)$ and its target distribution $\hat{Z}_i(\tau_i, u_i) = C_i(\tau_i, u_i, \alpha_i) + \gamma Z_i(\tau_i', u_i')$. Formally, the quantile distribution is represented by a set of quantiles $\tau_j = \frac{j}{K}$ and the quantile regression loss for the Q network is defined as:

$$\mathcal{L}_{QR} = \frac{1}{N} \sum\nolimits_{i=1}^{N} \sum\nolimits_{j=1}^{K} \mathbb{E}_{\hat{Z}_i \sim Z_i} [\rho_{\tau_j}(\hat{Z}_i - Z_i)] \tag{3.11}$$

where $\rho_\tau(\nu) = \nu(\tau - \mathbf{1}\{\nu < 0\})$. Huber loss [107] smoothly transitions from L1 loss (mean absolute error) for large errors to L2 loss (mean squared error) for small errors, making it less sensitive to outliers compared to mean squared error (MSE). Therefore, we use it in Quantile Regression (QR) loss. We name the loss function

as quantile Huber loss $\rho_\tau(\nu) = \mathcal{L}_\kappa(\nu)|\tau - \mathbf{1}\{\nu < 0\}|$ where $\mathcal{L}_\kappa(\nu)$ is defined as:

$$\mathcal{L}_\kappa(\nu) = \begin{cases} \frac{1}{2}\nu^2, & \text{if } |\nu| \geq \kappa, \\ \kappa(|\nu| - \frac{1}{2}\kappa), & \text{otherwise.} \end{cases} \tag{3.12}$$

Note that Risk-sensitive RL and Distributional RL are two orthogonal research directions as discussed in Sections 3.1 and 3.2.

Algorithm 1: RMIX

Input: initialize parameters $\bar{\theta}$ and θ of the network and the target network of agents, risk operator, monotonic mixing network and replay buffer $\mathcal{D}$;
for $e \in \{1, \ldots, MAX\_EPISODE\}$ **do**
  while *EPISODE_NOT_DONE* **do**
    Observe the global state $\boldsymbol{s}^t$;
    for *agent* $i \in \{1, \ldots, N\}$ **do**
      Observe o_i^t and get action u_i^{t-1} ;
      Predict the risk level α_i (Equation 3.4);
      Calculate CVaR values (Equation 3.3);
      Get the action u_i^t;
    Concatenate u_i^t, $i \in [1, .., N]$ into $\mathbf{u}_t$;
    Execute $\boldsymbol{u}_i^t$ into environment;
    Receive r^t and observe a new state $\boldsymbol{s}'$;
    Store $(\boldsymbol{s}^t, \{o_i^t\}_{i=1}^N, \boldsymbol{u}^t, r^t, \boldsymbol{s}')$ into $\mathcal{D}$;
    if *UPDATE* **then**
      Sample a min-batch $\mathcal{D}'$ from $\mathcal{D}$;
      For each sample in $\mathcal{D}'$, calculate CVaR value C_i (Equation 3.3);
      Concatenate CVaR values $\{[\{C_i^1\}_{i=1}^N]_1, \ldots, [\{C_i^{|\mathcal{D}'|}\}_{i=1}^N]_{|\mathcal{D}'|}\}$;
      For each $[\{C_i^j\}_{i=1}^N]_{0,j\in[1,\ldots,|\mathcal{D}'|]}$, calculate C_j^{tot} via the mixing network;
      Update θ by minimizing the TD loss (Equation 3.10);
      if *UPDATE* **then**
        Update Z_i (Equation 3.11);
      Update $\bar{\theta}$: $\bar{\theta} \leftarrow \theta$;
Output: A well-trained policy for each agent.

Training. Finally, we train RMIX in an end-to-end manner where each agent shares a single agent network and a risk predictor network to solve the lazy-agent issue [33]. ψ_i is trained together with the agent network via the loss defined in Equation 3.10. During training, f_{emb_i} updates its weights while the gradients of f_{emb_i} are blocked in order to prevent changing the weights of the return distribution in agent i. In fact, agents only use CVaR values for execution and the risk level optimizer only predicts the α; thus the increased network capacity is mainly from

the local return distribution and the CVaR operator. The proposed framework is flexible and can be easily used in many cooperative MARL methods. We present the proposed framework in Figure 3.2. The pseudo-code of RMIX is in Algorithm 1.

3.4 Experiments

We empirically evaluate our method on multi-agent risk-sensitive navigation scenarios and StarCraft II (SCII) cooperative scenarios. Especially, we are interested in the robust cooperation between agents and agents' learned risk-sensitive policies in complex cooperative scenarios.

3.4.1 Experiment Setup

3.4.1.1 Multi-Agent Cliff Navigation

We customize the cliff walking environment [29] in the single-agent domain and develop Multi-Agent Cliff Navigation (MACN) for multi-agent navigation with risk. In MACN, there are two agents whose task is to complete the navigation from the starting position to the goal. At each time step, each agent observes an observation with a dimension of 3×3. Agents can take an action at each time step and the action set is: $\{\text{UP}, \text{DOWN}, \text{LEFT}, \text{RIGHT}\}$. The two agents share the team reward. As depicted in Figure 3.5, there are some regions that are dangerous and agents will be rewarded with a -100 reward when any agent steps into these regions, and consequently the episode ends. Agents will receive a -1 reward at

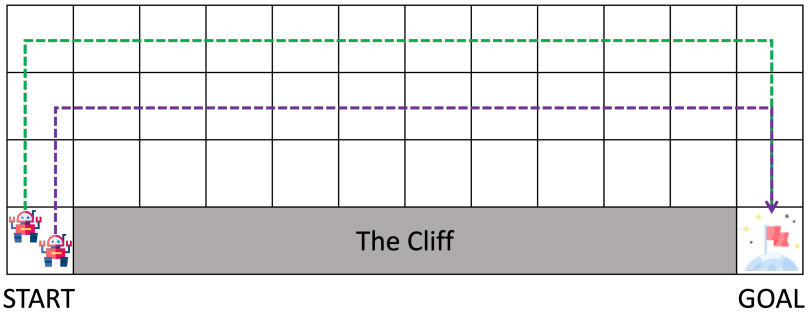


FIGURE 3.5: Multi-Agent Cliff Navigation.

(a) 5m_vs_6m (b) MMM2

FIGURE 3.6: SMAC scenarios: 5m_vs_6m and MMM2.

each time step when they are at the safe region. When one agent reaches the goal, agents will be rewarded with a -0.5 reward. If the two agents arrive at the goal at the same time, agents will be rewarded with a 0 reward and the episode ends. When only one agent arrives at the goal, the two agents will be rewarded with a -0.5 reward without extra penalty. The objective of the agents is to maximize the accumulated rewards in each episode. We report the test return mean (averaged total rewards of evaluation episodes) along with one standard deviation of the test return mean (shaded in figures).

3.4.1.2 StarCraft II Scenarios

We consider SMAC [30], a challenging set of cooperative StarCraft II (SCII) maps for micromanagement, as our testbed. The enemy units are controlled by SCII built-in AI and each of the ally units is controlled by a learning agent. The version of the SCII simulator is 4.10. We evaluate our method for every 10,000 training steps during training by running 32 episodes in which agents trained with our method battle with built-in game bots. We report the mean test win rate (percentage of episodes won by MARL agents) along with one standard deviation of the test win rate (shaded in figures). We present the results of our method and baselines on 6 scenarios: 1c3s5z (*easy*, *symmetric*) and MMM2 (*super hard*, *symmetric* and *homogeneous*), 5m_vs_6m (*super hard*, *asymmetric* and *homogeneous*), 8m_vs_9m (*easy*, *asymmetric* and *homogeneous*), 10m_vs_11m (*easy*, *asymmetric* and *homogeneous*) and corridor (*super hard*, *asymmetric* and *homogeneous*). An asymmetric scenario indicates the number of the ally units and that of the opponent are not

equal. A heterogeneous scenario means that all ally units are from different categories whose action and observation spaces are different while heterogeneous agents belong to the same category.

States and Observations of StarCraft II. At each time step, agents get local observations within their field of view, which contains information (relative x, relative y, distance, health, shield, and unit type) about the map within a circular area for both allied and enemy units and makes the environment partially observable for each agent. The global state is composed of joint observations, which could be used during training. All features, both in the global state and in individual observations of agents, are normalized by their maximum values.

Action Space of StarCraft II. actions are in the discrete space. Agents are allowed to make move[direction], attack[enemy id], stop, and no-op. The no-op action is only legal action for dead agents. Agents can only move in four directions: north, south, east, or west. The shooting range is set to 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.

Reward Function of StarCraft II. At each time step, the agents receive a shared reward equal to the total damage dealt to the enemy agents. 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 StarCraft II. 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 [26, 108]. We present the table of all scenarios in SMAC in Table 3.1 and the corresponding memory usage for training each scenario in Table 3.2. 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 agents is 5 while the number of the opponent is 6. The agent (unit) type is *marine*[4]. This asymmetric setting is hard for MARL methods. An asymmetric scenario indicates the number of the ally units and that of the opponent are not equal. A heterogeneous scenario means that ally units are from different categories whose action and observation spaces are different while heterogeneous agents belong

[4]A type of unit (agent) in StarCraft II. Readers can refer to `https://liquipedia.net/starcraft2/Marine_(Legacy_of_the_Void)` for more information

to the same category. The version of the SCII simulator is 4.10. We evaluate our method for every 10,000 training steps during training by running 32 episodes in which agents trained with our method battle with built-in game bots. We report the mean test win rate (percentage of episodes win of MARL agents) along with one standard deviation of the test win rate (shaded in figures).

TABLE 3.1: SMAC Environments

Name	**Ally Units**	**Enemy Units**	**Type**
5m_vs_6m	5 Marines	6 Marines	homogeneous & asymmetric
8m_vs_9m	8 Marines	9 Marines	homogeneous & asymmetric
10m_vs_11m	10 Marines	11 Marines	homogeneous & asymmetric
MMM2	1 Medivac, 2 Marauders & 7 Marines	1 Medivac, 3 Marauders & 8 Marines	heterogeneous & asymmetric
1c3s5z	1 Colossi & 3 Stalkers & 5 Zealots	1 Colossi & 3 Stalkers & 5 Zealots	heterogeneous & symmetric
corridor	6 Zealots	24 Zerglings	micro-trick: wall off

3.4.2 Training Details

We use the same neural network architecture used by QMIX [26]. The trajectory embedding network ϕ_i is similar to the network of the agent. The last layer of the risk level optimizer generates the local return distribution and shares the same weights with the last layer of the agent network. The QR loss is minimized to periodically (empirically every 50 time steps) update the weights of the agent as simultaneously updating the local distribution and the whole network can impede training. The QR loss is used for updating the local distribution while the TD loss of centralized training is for the agent network weights learning and credit assignment. In the QR loss, the $C_i(\cdot, \cdot, \cdot)$ is considered as scalar value, and gradients from $C_i(\cdot, \cdot, \cdot)$ are blocked. As the QR update relies on accurate estimation of the dummy reward C_i, we start to update C_i when the test winning rate is over 35%, which means the agents have grasped some strategies to win the game.

TABLE 3.2: Memory usage (given the current size of the replay buffer) for the training of each method (exclude COMA and DOP, which are on-policy methods without using replay buffers) on scenarios of SCII domain in SMAC.

Scenario	**Memory Usage (GB)**
5m_vs_6m	3
8m_vs_9m	4.9
10m_vs_11m	7.1
1c3s5z	8.6
MMM2	10.8
corridor	14.4

StarCraft II. To make a fair comparison, we use `episode` (single-process environment for training, compared with `parallel`) runner defined in PyMARL to run all methods. We use `num_circle=1` to train QPLEX for fair training at the sample level for each update. We use Adam to optimize the CVaR and QR loss as suggested by QR-DQN [61] with a learning rate of 5×10^{-4}. The batch size is 32 and the replay buffer size is 5000. We use $K = 10$. We set $M = 100$ for 5m_vs_6m and 8m_vs_9m. To trade off the experimental performance and the computation overhead, we use $M = 55$ for the scenario of the corridor, and $M = 65$ for 10m_vs_11m, 1c3s5z and MMM2. In order to explore, we use ϵ-greedy with ϵ annealed linearly from 1.0 to 0.05 over 50K time steps from the start of training and keep it constant for the rest of the training for all methods. The discount factor $\gamma = 0.99$ and we follow the default hyper-parameters used in the original papers of all baselines. The evaluation interval is $10,000$ for all methods. We use uniform probability to estimate $Z_i(\cdot, \cdot)$ for each agent. Experiments are carried out on NVIDIA Tesla V100 GPU 16G and NVIDIA GeForce RTX 3090 24G. We also provide memory usage of baselines (given the current size of the replay buffer) for training each scenario of the SCII domain in SMAC as shown in Table 3.2.

MACN. We follow the same settings used in SCII to train RMIX and baselines in MACN scenarios. $M = 100$ and $K = 10$.

TABLE 3.3: Baseline algorithms

Algorithms	**Brief Description**
IQL [109]	Independent Q-learning
VDN [33]	Value decomposition network
COMA [32]	Counterfactual Actor-critic
QMIX [26]	Monotonicity Value decomposition
QTRAN [35]	Value decomposition with linear affine transform
LH-IQN [111]	Likelihood Hysteretic with IQN (independent learning)
DOP [36]	Policy Gradient with a linear sum of individual Q values
WQMIX [110]	Weighted TD-error of QMIX in centralized training.
QPLEX [40]	Using a dueling network to represent the Q^{tot}

3.4.3 Baseline Algorithms

We compare three categories of MARL methods: (i) Value-based methods: IQL [109], VDN [33], QMIX [26], QTRAN [35], WQMIX [110], QPLEX [40]; (ii) Policy gradient methods: COMA [32] and DOP [36]; (iii) Distributional RL-based methods. Among these methods, QPLEX and WQMIX are state-of-the-art value-based MARL methods, and DOP is a state-of-the-art policy gradient MARL method. We summarize all baselines in Table 3.3. We implement our method on PyMARL [30] and use five random seeds to train each method. We carry out experiments on NVIDIA Tesla V100 GPU 16G.

IQL [109]. IQL is an independent Q-learning method for multi-agent RL. Each agent learns its independent Q values via DQN [23].

VDN [33]. To address the credit assignment issue in Dec-POMDP MARL methods, VDN utilizes a linear combination of individual Q values to represent the $Q^{\text{tot}}(\boldsymbol{\tau}, \boldsymbol{u})$ as $Q^{\text{tot}} = \sum_{i=1}^{N} Q_i(\tau_i, u_i)$. Then, TD-learning can be used to train Q^{tot}.

QMIX [26]. Unlike the linear combination used in VDN, QMIX considers the monotonic constraint on the relationship between Q^{tot} and Q_i, that is

$$\frac{\partial Q^{\text{tot}}(\boldsymbol{\tau}, \boldsymbol{u})}{\partial Q_i(\tau_i, u_i)} \geq 0, \forall i \in \{1, 2, \ldots, N\}$$

where $Q^{\text{tot}(\boldsymbol{\tau}, \boldsymbol{u})} = f_m(Q_1(\tau_1, u_1), \ldots, Q_N(\tau_N, u_N))$ and f_m is a mixing network built on top of a hyperntework [112] with $(Q_1(\tau_1, u_1), \ldots, Q_N(\tau_N, u_N))$ as inputs. QMIX outperforms VDN on various hard SCII challenges.

QTRAN [35]. QTRAN relaxes the constraints of VDN and QMIX and factorize the $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ with transformation:

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

where $V_{\text{jt}}(\boldsymbol{\tau}) = \max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - \sum_{i=1}^{N} Q_i\left(\tau_i, u_i\right)$. QTRAN outperforms QMIX and VDN on matrix game and struggles on high-dimension SCII tasks.

QPLEX [40]. Wang et al. utilize the dueling structure $Q = V + A$ [113] and propose the following factorization:

$$\begin{aligned} &\text{Joint Dueling:} \quad Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) = V_{tot}(\boldsymbol{\tau}) + A_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) \text{ and } V_{tot}(\boldsymbol{\tau}) = \max_{\boldsymbol{u}'} Q_{tot}\left(\boldsymbol{\tau}, \boldsymbol{u}'\right) \\ &\text{Individual Dueling:} \quad Q_i\left(\tau_i, a_i\right) = V_i\left(\tau_i\right) + A_i\left(\tau_i, a_i\right) \text{ and } V_i\left(\tau_i\right) = \max_{a_i'} Q_i\left(\tau_i, a_i'\right) \end{aligned}$$

WQMIX [110]. QMIX restricts the joint action Q-values to be a monotonic mixing of each agent's Q-values. However, such restriction prevents it from representing value functions in which nonmonotonicity occurs. To address this issue, Rashid et al. [110] propose the following QMIX operator Π_w by imposing weights $w(\boldsymbol{\tau}, \boldsymbol{u})$ on the TD-error:

$$\Pi_w Q := \underset{q \in \mathcal{Q}^{mix}}{\operatorname{argmin}} \sum_{\mathbf{u} \in \mathbf{U}} w(\boldsymbol{\tau}, \boldsymbol{u})(Q(\boldsymbol{\tau}, \boldsymbol{u}) - q(\boldsymbol{\tau}, \boldsymbol{u}))^2$$

LH-IQN [111]. LH-IQN was built on top of Hysteretic Q-Learning [114] and Lenient Q-Learning [115] with IQN to trains independent learners. However, it performs poorly in complex scenarios, the reason maybe LH-IQN is an independent learning method similar to IQL.

COMA [32]. Foerster et al. consider the credit assignment problem in policy graident methods and propose the following counterfactual advantage function:

$$A^i(\boldsymbol{\tau}, \mathbf{u}) = Q^{\text{tot}}(\boldsymbol{\tau}, \mathbf{u}) - \sum_{u'^i} \pi^i\left(u'^i \mid \tau^i\right) Q^{\text{tot}}\left(\boldsymbol{\tau}, \left(\mathbf{u}^{-i}, u'^i\right)\right)$$

where $A^i\left(\boldsymbol{\tau}, u^i\right)$ is a separate baseline for each agent that uses the centralized critic to reason about counterfactuals where only agent i's action changes.

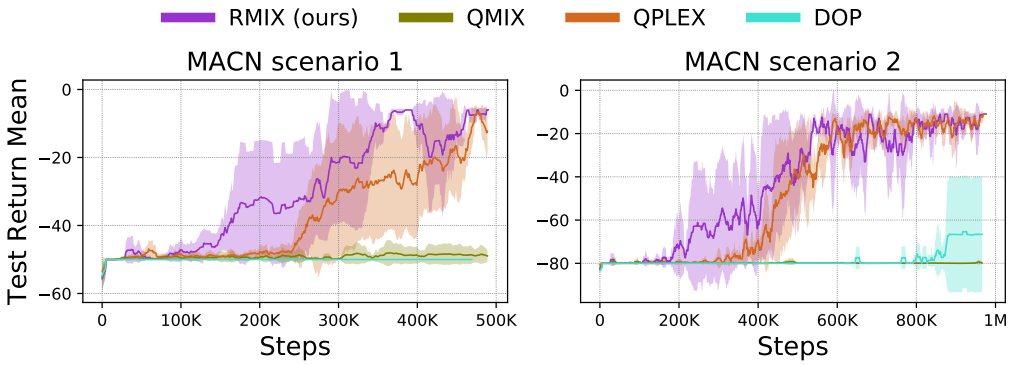


FIGURE 3.7: The solid curves show the test return mean (averaged total rewards of evaluation episodes) on two MACN scenarios, MACN scenarios 1 and 2. The shades depict one standard deviation of the test return mean.

DOP [36]. Wang et al. find the centralized-decentralized mismatch (CDM) in MADDPG [25]. Inspired by the successes of value decomposition in VDN and QMIX, they propose the Decomposed Off-Policy policy gradient (DOP) method which decomposes the centralized Q value with a linear combination of individual Q values:

$$Q^{\text{tot}}(\boldsymbol{\tau}, \mathbf{u}) = \sum_i k_i(\boldsymbol{\tau}) Q_i(\boldsymbol{\tau}, u_i) + b(\boldsymbol{\tau})$$

where k_i is the weight and b is the bias which takes the global observation-action histories as inputs.

3.4.4 Experimental Results

In this subsection, we first showcase the learned risk-sensitive policies of RMIX in grid-world MACN scenarios and then present the improved complex coordination of RMIX in SCII scenarios.

3.4.4.1 Multi-Agent Cliff Navigation

We conduct experiments on two multi-agent cliff navigation scenarios to showcase the gained performances of RMIX and the learned risk-sensitive policy. MACN scenario 1 has a width of 7 and a height of 4 while MACN scenario 2 has a width of 12 and a height of 4. The reward functions of the two scenarios are the same and not changed (see Section 3.4.1.1). The baselines are QMIX, QPLEX, and DOP, which

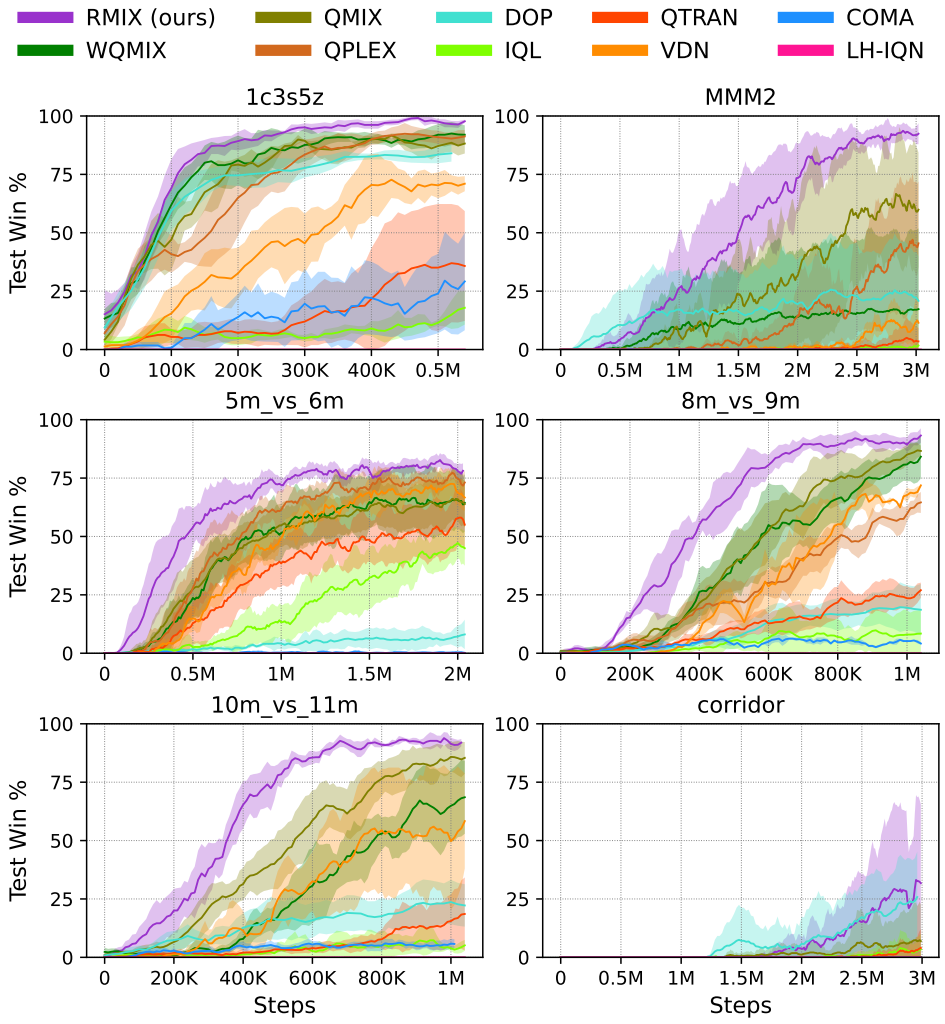


FIGURE 3.8: Test Win rates for six scenarios.

are well-known risk-neutral MARL methods. As shown in Figure 3.7, trained with risk-sensitive policies, RMIX demonstrates better sample efficiency compared with other risk-neutral, *i.e.*, expected-Q, baseline methods in two MACN scenarios with high risk. QPLEX outperforms QMIX and DOP and shows converged performance due to the dueling network [113] and attention architecture [116]. In Section 3.4.5, we illustrate experimental results on less risky MACN scenarios.

3.4.4.2 StarCraft II

Comparison with Value-based MARL Baselines. As depicted in Figure 3.8, RMIX outperforms QMIX, WQMIX, IQL, QTRAN, and VDN in all *asymmetric*/*symmetric* and *homogeneous*/*heterogeneous* scenarios, demonstrating its superiority on improving multi-gent coordination in complex scenarios. RMIX even outperforms QPLEX on MMM2, 5m_vs_6m, and corridor, and also shows competitive performance on 8m_vs_9m and 10m_vs_11m. Note that QPLEX was built on top of QMIX and used a dueling network and advantage functions to tackle the credit assignment problem in Dec-POMDP. Unlike RMIX, which only leverages the merit of CVaR, the overall structure of QPLEX is more complex and benefits much more from increased model capacities than RMIX. RMIX improves coordination in a sample-efficient way via risk-sensitive policies. 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.

Comparison with Policy Gradient MARL Baseline. Our method also outperforms COMA and DOP as illustrated in Figure 3.8. COMA aims to address multi-agent credit assignment issues by utilizing counterfactual values. DOP learns the centralized Q value for policy learning with a linear combination of individual Q values. The two methods are on-policy methods and are sample-inefficient.

Comparison with Distributional/Risk-sensitive MARL Baseline. Although there are few practical methods for risk-sensitive multi-agent reinforcement learning, we also conduct experiments to compare our method with LH-IQN [111], which is a risk-sensitive method built on a distributional RL method called implicit Quantile network (IQN). We can find that our method outperforms LH-IQN in many scenarios. The performance of LH-IQN is not good in StarCraft II scenarios because it is an independent MARL method and the IQN was used to guide the update of the Lenient Q-Network.

TABLE 3.4: Test win rate of RMIX with varying risk levels.

	1c3s5z	**5m_vs_6m**
RMIX	$\mathbf{97.10 \pm 2.56\%}$	$\mathbf{80.01 \pm 5.57\%}$
RMIX_0.1	$93.48 \pm 1.67\%$	$60.65 \pm 6.07\%$
RMIX_0.3	$95.47 \pm 2.84\%$	$69.16 \pm 4.74\%$
RMIX_0.5	$95.15 \pm 1.63\%$	$70.75 \pm 4.73\%$
RMIX_0.7	$95.89 \pm 2.15\%$	$74.88 \pm 8.48\%$
RMIX_1.0	$89.70 \pm 8.22\%$	$77.56 \pm 4.11\%$
QMIX	$88.34 \pm 2.64\%$	$64.75 \pm 8.69\%$

3.4.5 Ablations

RMIX consists of two components: the CVaR policies and the risk level optimizer. The CVaR policies are different from vanilla Q values and the risk level optimizer is proposed to model the temporal structure and as an α-finding strategy for hyperparameter tuning. Our ablation studies serve to answer the following questions: **Q1:** Can RMIX with static α also work? **Q2:** Can the risk level optimizer learn α values and fast learn a good policy compared with RMIX with static α? **Q3:** Can our framework be applied to other methods? **Q4:** Is our method robust to the randomness of rewards? **Q5:** How does our method perform in less risky scenarios?

To answer **Q1** and **Q2**, we conduct an ablation study by fixing the risk level in RMIX with the value of $\{0.1, 0.3, 0.5, 0.7, 1.0\}$ and compare with RMIX and QMIX on 1c3s5z (0.5 million time steps) and 5m_vs_6m (2 million time steps). As illustrated in Table 3.4, with static α values, RMIX is capable of learning good performance over QMIX, which demonstrates the benefits of learning risk-sensitive MARL policies in complex scenarios where the potential of loss should be taken into consideration in coordination. With risk level optimizer, RMIX outperforms RMIX with static α values, illustrating that agents have captured the temporal features of scenarios and possessed the α value tuning merit.

We answer **Q3** to show that our proposed method is applicable in other value-based MARL methods, we then apply additivity of individual CVaR values to represent the global CVaR value as $C^{\text{tot}}(\boldsymbol{\tau}, \boldsymbol{u}) = C_1(\tau_1, u_1, \alpha_1) + \cdots + C_N(\tau_N, u_N, \alpha_N)$. Following the training of RMIX, we name this method Risk Decomposition Network (RDN). We use an experiment setup of VDN and train RDN on 3 SMAC scenarios. With CVaR policies, RDN outperforms VDN on 10m_vs_11m and converges faster

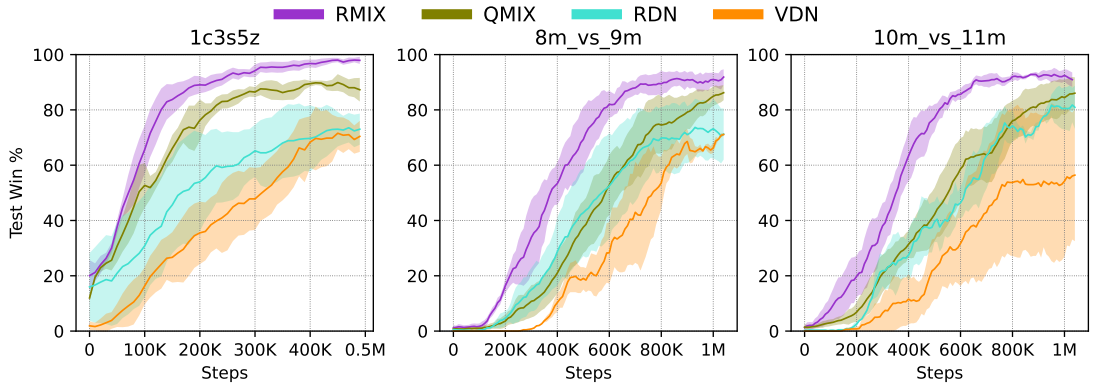


FIGURE 3.9: Test Win rates of RMIX, RDN, VDN and QMIX.

than VDN on 1c3s5z and 8m_vs_9m, as depicted in Figure 3.9, demonstrating that our framework can be applied to VDN and outperforms VDN.

We answer **Q4** by directly injecting Gaussian noise $\mathcal{N}(0, \zeta)$ to the reward function, where $\zeta \in \{0.1, 0.3, 0.5, 0.7, 1.0\}$. The reward function in SMAC is deterministic. However, each agent owns a stochastic policy which makes the nonstationarity of the environment and leads to the randomness of reward, especially at the early training stage where agents explore. It has also been claimed by distributional RL [59, 85] in a single-agent setting. Results in Figure 3.10 show that RMIX is robust to random noise in the reward function.

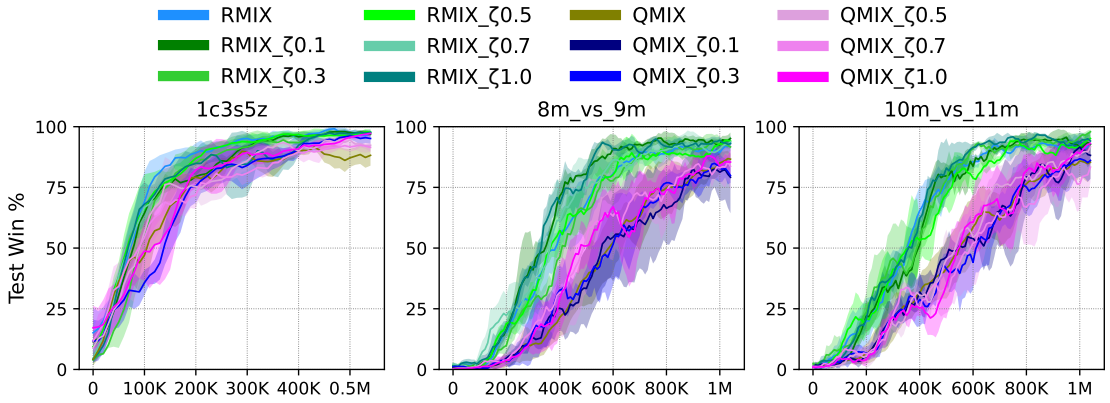


FIGURE 3.10: RMIX vs QMIX with reward noise.

We answer **Q5** by conducting experiments in MACN scenarios with low risk. We select MACN scenario 1 as the evaluation scenario where the width is 7 and the height is 4. To make the scenario less risky, we change the punishment of stepping

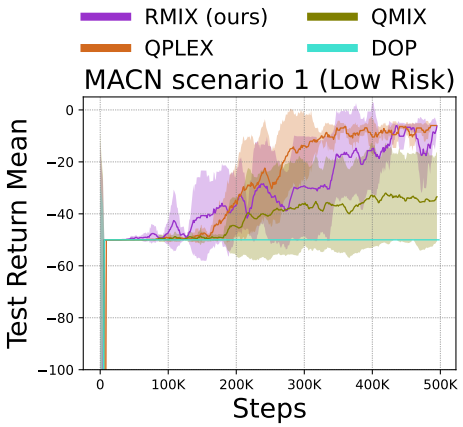


FIGURE 3.11: Test return mean in MACN scenario 1.

into the dangerous region, a.k.a cliff, from the value of -100 to -50. We present the averaged test return value of RMIX, QMIX, QPLEX, and DOP in Figure 3.11. We can find that in scenarios with low risk, unlike results in Figure 3.7, RMIX shows comparable converged performance with QPLEX. The variance of the average test return of RMIX is larger than that of QPLEX. We can also find the improved performance of QMIX in this low-risk scenario. We can conclude that in low-risk scenarios, RMIX can also show comparable performance compared with risk-neutral MARL methods.

We show the test return value in Figure 3.12 and RMIX outperforms baseline methods as well. As shown in Figure 3.13, RDN shows a convincing test return value over VDN in three scenarios.

3.4.6 Results Analysis

We are interested in finding if the risk level optimizer can predict temporal risk levels. We use the trained model of RMIX and run the model to collect one episode data including game replay, states, actions, rewards, and α values. As shown in Figure 3.14, the first row shows the rewards of one episode and the second row shows the α value each agent predicts per time step. There are eight scenes in the third row to show how agents learn time-consistency α values. Scenes in Figure 3.14 are screenshots from the game replay.

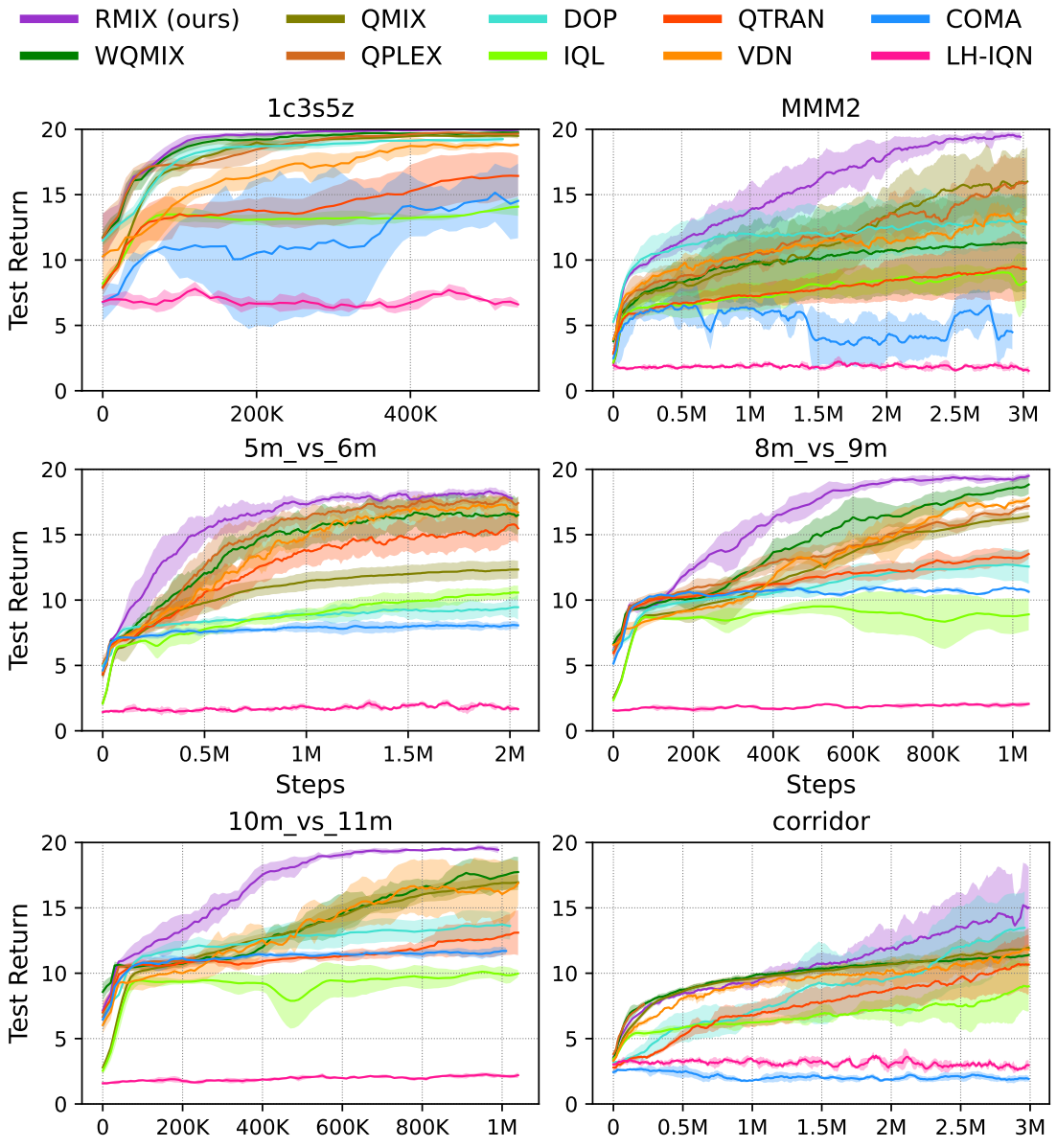


FIGURE 3.12: Test return for six scenarios.

We use trajectories of agents 0, 1, and 3 as examples in Figure 3.14. The number in the circle indicates the index of the agent. Scene (1): one episode starts, 6 Zealots consist of RMIX agents and 24 Zerglings compose enemies; Scene (2): to win the game, agent 3 draws the attention of enemies and goes to the other side of the battlefield. The α value is 0.3 at step 2. Many enemies are chasing agent 3. The rest agents are combating fewer number enemies; Scene (3): at step 14, agent 3 is at the corner of the battlefield the α is decreasing. As being outnumbered, agent 3 quickly dies and the α is zero; In Scene (4) agents 0 and 1 show similar α values as

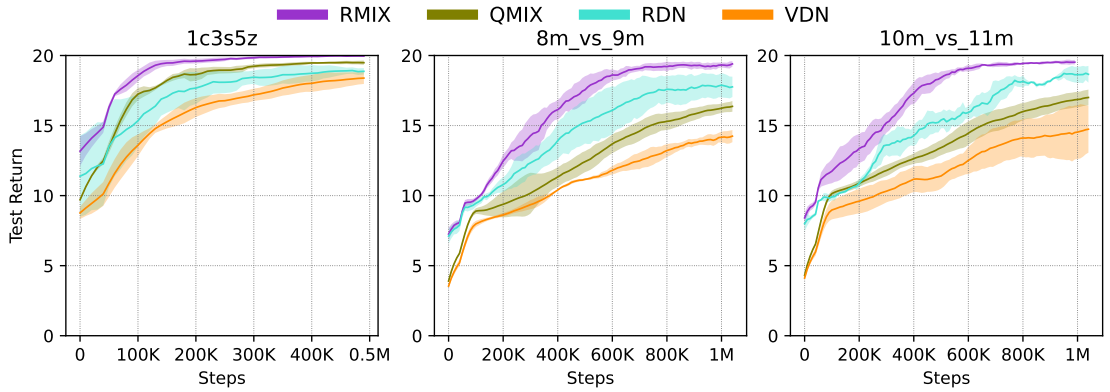


FIGURE 3.13: Test return of RMIX, RDN, VDN, QMIX in three scenarios.

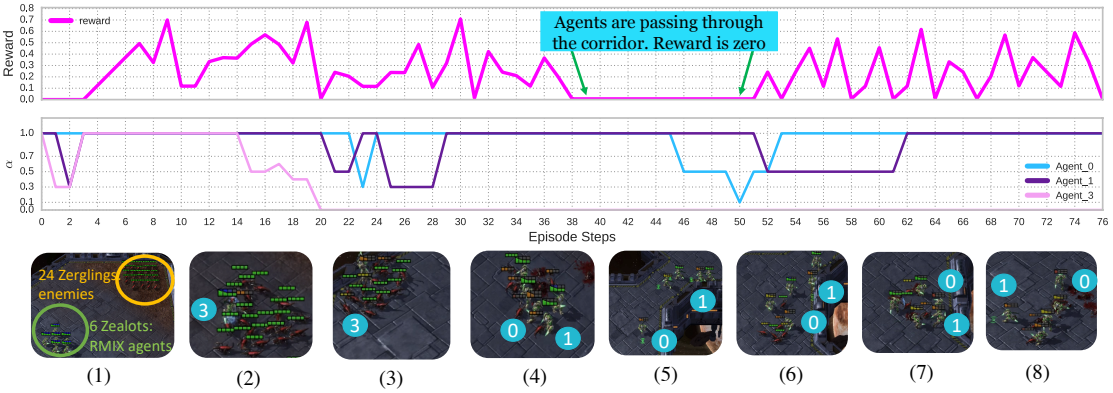
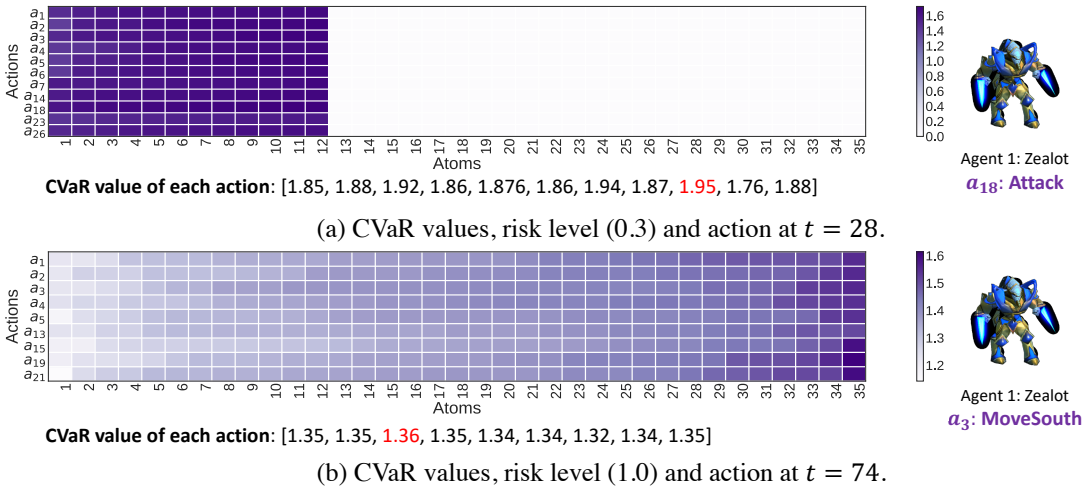


FIGURE 3.14: Results analysis of RMIX on the corridor. There are 8 scenes as examples. Further discussion can be found in Section 3.4.6.

they are walking around and fighting with enemies from time steps 22 to 30. Then agents kill enemies around and plan to go to the other side of the battlefield to win the game; Scene (5): agent 0 (low health value) is walking through the corridor alone to draw enemies to come over. To avoid being killed, α values are low (steps 46-50) which means the policy is risk-averse. From step 38 and step 51, the reward is zero; Scene (6): agent 0 is facing many enemies and luckily its teammates are coming to help. So, the α is increasing (step 50); Scene (7): as there are many teammates around and the number of enemies is small, agents are going to win. Agents 0 and 1 walk outside the range of the observations of enemies to survive. The α values of agent 0 and agent 1 are 1 (risk-neutral); Scene (8): agents win the game. The video is available on this link: `https://youtu.be/5yBEdYUhysw`. Interestingly, the result shows emergent cooperation strategies between agents at



(a) CVaR values, risk level (0.3) and action at $t = 28$.

(b) CVaR values, risk level (1.0) and action at $t = 74$.

FIGURE 3.15: Two examples on CVaR calculation and action selection.

different steps during the episode, which demonstrates the superiority of RMIX.

We showcase how agents apply the risk level to select actions. We collect the executed actions, the predicted risk levels and the local distributions of agent 1 (Zealot) of time steps 28 and 74. Rewards and risk levels have already been shown in Figure 3.14. As depicted in Figure 3.15, there are 35 Dirac delta atoms of the local distributions as shown in the heat map. The x-axes denote each Dirac function. The y-axes stand for available actions at the current time step for agent 1 while the size of the action space is 30. At time step 28, as shown in Figure 3.15 (a), the predicted risk level is 0.3. We sort values in the heat map of each action in ascending order and then apply Equation 3.3 to get CVaR values of each action. Values in white cells in the heat map (Figure 3.15 (a)) are excluded while calculating the CVaR value. At time step 28, because many opponents are attacking agent 1, the risk level is 0.3 (means agent 1 is going to encounter potential reward loss) and agent 1's action is Attack. In Figure 3.15 (b), at time step 74, as agents are going to win the game, agent 1 does not need to get involved in the combat, agent 1's risk level is 1.0 and it takes a_3, *i.e.*, MoveSouth, to get out of the battlefield to avoid being killed by the opponent. The heat map illustrates that all the return values are used for CVaR calculation.

3.5 Chapter Summary

In this chapter, we propose RMIX, a novel and practical MARL method with CVaR over the learned distributions of individuals' Q values as risk-sensitive policies for cooperative agents. Empirically, we show that our method outperforms baseline methods on many challenging StarCraft II tasks, reaching convincing performance and enhanced coordination as well as improved sample efficiency. Our work provides a practical framework for future risk-sensitive MARL research. In addition to the algorithmic contribution, our method could help the development of various real-world applications where risk-sensitive decision-making is desperately needed. For example, the driving policy should be risk-sensitive to any potential accidents and uncertainties in automatic driving scenarios. Besides training risk-sensitive policies for automatic driving, the proposed method is applicable for multi-robot rescue where imprudent decisions can lead to the failure of the mission.

The proposed method trains MARL policies in a risk-sensitive manner with CVaR. However, the learned policy does not guarantee the exact safety bound. We adopt the Quantile-Regression [61] method to estimate the local return distribution accurately and efficiently train the CVaR values, which is not scalable for scenarios that have many agents and large action space. The aforementioned limitations motivate us to develop novel risk-sensitive MARL methods in the future.

Risk-sensitive policy learning is vital for many real-world multi-agent applications, especially in risky tasks, for example, autopilot vehicles and finance portfolio management. For future work, better risk measurement together with accurate spatial-temporal trajectory representation can be investigated. Also, learning to model other agents' risk levels and reach a consensus with communication can be another direction for enhancing multi-agent coordination. Our method is built on a value-based MARL method and devising risk-sensitive policy gradient MARL methods is our future direction too.

Chapter 4

Learning to Scale Up with Cooperative MARL

This chapter investigates the Dynamic Electronic Toll Collection (DETC) problem and proposes a cooperative MARL solution. Over the past decades, Electronic Toll Collection (ETC) systems have proven the capability of alleviating traffic congestion in urban areas. DETC was recently proposed to further improve the efficiency of ETC, where tolls are dynamically set based on traffic dynamics. However, computing the optimal DETC scheme is computationally difficult and existing approaches are limited to small-scale or partial road networks, which significantly restricts the adoption of DETC. To this end, this chapter considers computing the optimal DETC scheme as a multi-agent problem. Then we propose a novel multi-agent reinforcement learning (RL) approach[1] to obtain the optimal DETC scheme with a divide-and-conquer manner. We make several key contributions: (i) an enhancement over the state-of-the-art RL-based method with a deep neural network representation of the policy and value functions and a temporal difference learning framework to accelerate the update of target values; (ii) a novel edge-based graph convolutional neural network (eGCN) to extract the spatiotemporal correlations of the road network state features; (iii) a novel cooperative multi-agent reinforcement learning (MARL) that divides the whole road network into partitions according to their geographic and economic characteristics and trains a tolling agent for each partition. Experimental results show that such a divide-and-conquer approach

[1]The work in this chapter has been published in [13].

can scale up to realistic-sized problems with robust performance and significantly outperform the state-of-the-art method.

4.1 Introduction

All over the world, traffic congestion is becoming a severe issue in urban areas. ETC systems are one of the most effective approaches introduced by transportation authorities to mitigate traffic congestion. By setting different tolls on different roads, vehicles are regulated to travel on less congested roads with lower tolls. However, traditional ETC prices are pre-defined based on historical data and do not adapt to real-time traffic conditions. The true merit of ETC has not been fully leveraged. To this end, many dynamic pricing schemes have been proposed to tackle these issues [42–45] where tolls are computed based on traffic dynamics.

Nonetheless, devising a DETC scheme is still challenging and non-trivial due to its complex problem structure, including a large-scale road network, dynamic traffic flow, uncertainty in traffic demand, and its sequential decision-making problem. Conventionally, this sequential decision-making problem can be formulated as a Markov Decision Problem (MDP) and solved by Reinforcement Learning (RL) methods. Recently, Chen et al. [44] made the first attempt at solving the DETC problem via reinforcement learning. The solution method, PG-β, used the policy gradient method with Beta distribution as the policy distribution. Despite its good performance in mitigating traffic congestion over other methods, a major limitation of PG-β, is that it is limited to a small-scale road network and cannot work under realistic-sized road networks. To this end, this chapter aims to close the gap by scaling up the current state-of-the-art DETC method to large-scale real-world road networks with multi-agent deep reinforcement learning and various other novel techniques exploiting the problem characteristics.

The key contributions are summarized as follows: (i) to improve the performance of PG-β, we propose DPG-β, which employs deep neural networks to better represent the policy and value functions of the tolling agent, together with a temporal difference learning to accelerate the update of target values; (ii) naive feature representation with the deep neural network has only limited improvements over the state-of-the-art methods as it fails to exploit the graph nature of the road network. We apply a novel edge-based graph convolutional neural network (eGCN) to capture the features of spatio-temporal correlations of the road network; (iii) the dimension of the state space is equal to the number of roads, while the action space is the number of tolled roads. For large-scale road networks, training

RL models becomes extremely challenging due to huge state and action spaces. Hence, we first consider the DTEC problem as a multi-agent problem. We then propose a novel cooperative multi-agent reinforcement learning (MARL) algorithm that decomposes the state and action spaces into sub-spaces and solves DETC in a divide-and-conquer fashion; (iv) extensive experimental evaluations were conducted in the real-world road network of Singapore. Empirically, The proposed approach can scale up to a realistic-sized road network with robust performance and significantly outperform the state-of-the-art RL-based methods.

4.2 Electronic Toll Collection

Originally aimed to alleviate the delay on toll roads and have the merit of collecting tolls without cash and without requiring cars to stop, Electronic Toll Collection (ETC) systems have been widely deployed in highways and arterial roads in urban areas all over the world. For example, ETC has been implemented and deployed in Norway's three major cities: Bergen (1986), Oslo (1990), and Trondheim (1991), and in Singapore (1998) as a congestion pricing system to alleviate traffic congestion in central areas and restricted zones for expressways and arterial roads. Based on a pay-as-you-use principle, motorists are charged when they pass through ETC gantries during tolling hours. To alleviate traffic congestion, the toll rate varies based on traffic conditions and periods, which motivates drivers to travel on less congested roads.

4.3 Problem Formulation

4.3.1 Problem Setup

Formally, the city road network can be abstracted as a directed road network $\mathcal{G} = (\mathcal{Z}, \mathcal{E}, \mathcal{D})$ in which $\mathcal{Z}$ is the set of zones, $\mathcal{E}$ is the set of roads that connect the zones and $\mathcal{D}$ is the set of origin-destination (OD) pairs which define optional paths and traffic demand of both zones.

We denote an OD pair as a tuple $\langle z_k, z_j, q^t_{k,j}, P_{k,j} \rangle$, where zone $z_k \in \mathcal{Z}$ is the origin, zone z_j is the destination, $q^t_{k,j}$ is the traffic demand at time step t between z_k and z_j,

and the $P_{k,j}$ denotes all the acyclic paths from z_k to z_j. There are H time steps for daily-basis DETC planning. Following the *travel time model* [117], the travel time on road $e \in \mathcal{E}$ at time step t is $T_e^t = T_e^0[1 + \gamma\,(s_e^t/C_e)^\xi]$, where T_e^0 is the free-flow travel time, s_e^t is the number of vehicles on road e, and C_e is the capacity of road e. γ and ξ are constants that measure to which extent congestion affects travel time. The travel cost of a path $p \in P_{k,j}$ can be defined as $c_{k,j,p}^t = \sum_{e \in p}(a_e^t + \omega T_e^t)$ where a_e^t is the toll imposed on road e and ω is a constant for the value of time. We use the widely-adopted *stochastic user equilibrium* (SUE) model [118, 119] as the traffic equilibrium in this chapter, where the portion of traffic demand from zone k to zone j via path p at time step t is defined as $x_{k,j,p}^t = \frac{\exp\{-\omega' c_{k,j,p}^t\}}{\sum_{p' \in P_{k,j}} \exp\{-\omega' c_{k,j,p'}^t\}}$ where ω' is a constant which reveals vehicles' sensitivity to travel cost. We formulate the problem setup as an MDP in section 4.3.2.

4.3.2 A Finite Horizon MDP Formulation

Conventionally, an MDP is defined as a tuple $\langle S, A, r, T \rangle$, which consists of a set of states S, a set of actions A, a reward function $r : S \times A \to r$ and a transition function $T : S \times A \to S$. The goal of the agent for each time step t is to maximize the expected accumulated reward $\sum_{t'=t}^{\infty} \lambda^{t'-t} r^{t'}$ where $\lambda \in [0, 1]$ is the discount factor. It can be achieved by learning an optimal policy $\pi : S \to A$ which outputs an action a given a state s. However, the number of vehicles that reach the destinations depends on the OD demands at each time step due to traffic dynamics. Consequently, the value of a state changes over time, and the value of an action is also time-dependent. Due to its advantages of formulating sequential decision-making problems, we use finite horizon MDP to model DETC, where we maintain and update a value function and policy function for each time step $t = 0, 1, ..., H$. We define the corresponding elements for the MDP formulation used in this chapter.

State and Action. We define the state at time step t, as $\mathbf{s}_e^t = \langle s_{e,j}^t \rangle$ where $s_{e,j}^t$ denotes the number of vehicles that travel to zone j at time step t on road e, and $\mathbf{s}^t = \langle \mathbf{s}_e^t \rangle$ is the state matrix of $\mathcal{G}$ at time step t. The action at time step t is defined as $\mathbf{a}^t = \langle a_e^t \rangle$, where a_e^t is the toll set by the transit authority on e that has an ETC gantry.

State Transition. The state of the next time step $t + 1$ can be formed as $s_{e,j}^{t+1} = s_{e,j}^t - s_{e,j,out}^t + s_{e,j,in}^t$, where $s_{e,j}^{t+1}$ and $s_{e,j}^t$ denote the number of vehicles on road e that

travel to zone j at time step $t+1$ and t respectively. $s^t_{e,j,out}$ is the number of vehicles that go to zone j and exit road e at time step t. It can be defined as $s^t_{e,j,out} = s^t_{e,j} \cdot \frac{v^t_e \cdot \tau}{L_e} = \frac{s^t_{e,j} \cdot \tau}{T^0_e[1+\gamma(s^t_e/C_e)^\xi]}$, where L_e is the length of road e, $v^t_e = \frac{L_e}{T^t_e} = \frac{L_e}{T^0_e[1+\gamma(s^t_e/C_e)^\xi]}$ is average travel speed of road e, and τ is the time interval between time steps. Similarly, $s^t_{e,j,in}$ is the number of vehicles that go to zone j and enter road e at time step t. It can be defined as $s^t_{e,j,in} = \sum_{D_k=e^- \bigcap e \in p \in P_{k,j}} (q^t_{k,j} + \bar{q}^t_{k,j}) \cdot x^t_{k,j,p}$, where $q^t_{k,j}$ is the traffic demand which comes from zone k to zone j as introduced in Section 4.3.1. $\bar{q}^t_{k,j}$ is traffic demand which denotes the number of vehicles that come from zone k's neighboring roads and head for zone j during the period of $t-1$. It can be defined as $\bar{q}^t_{k,j} = \sum_{e^+=k} s^t_{e,j,out}$. Thus,

$$s^{t+1}_{e,j} = s^t_{e,j} - \frac{s^t_{e,j} \cdot \tau}{T^0_e[1+\gamma(s^t_e/C_e)^\xi]} + \sum_{D_k=e^- \bigcap e \in p \in P_{k,j}} (q^t_{k,j} + \bar{q}^t_{k,j}) \cdot x^t_{k,j,p}. \tag{4.1}$$

Reward Function. The target of the MDP is to maximize the accumulated reward. We define the reward $r(s^t) = \sum_{e \in \mathcal{E}} \sum_{z_j=e^+} \frac{s^t_{e,j}\tau}{T^0_e[1+\gamma(s^t_{e,j}/C_e)^\xi]}$ as the number of vehicles which arrive at destinations at time step t, where e^+ is the ending point of the road e and τ is the length of the time interval.

Policy and Value Function. At time step t, a policy $\pi^t(\mathbf{a}^t|\mathbf{s}^t)$ is the conditional probability of taking action $\mathbf{a}^t$ provided state $\mathbf{s}^t$. The value function at time step t can be defined as $v^t(\mathbf{s}^t) = \sum_{t'=t}^{H} \lambda^{t'} r^t$.

Challenges. There are three key challenges in the formulation of ETC: 1) the state space is discrete and high dimensional (w.r.t. the number of roads), 2) the action space is also continuous and high dimensional (w.r.t. the number of tolled roads), and 3) the action space is bounded. The dynamic programming (DP) method is a classic method to solve MDPs. However, solving the formulated MDP with DP can be computationally difficult. Vanilla reinforcement learning methods, such as Q-learning and SARSA [51, 120], also fail under the problem setup because of the large scale and continuous state-action spaces. While policy gradient methods [52, 54, 121] work well under large-scale MDPs with continuous action space, these methods do not perform well under the bounded action space. Although PG-β [44] can solve DETC, it is limited to partial road networks.

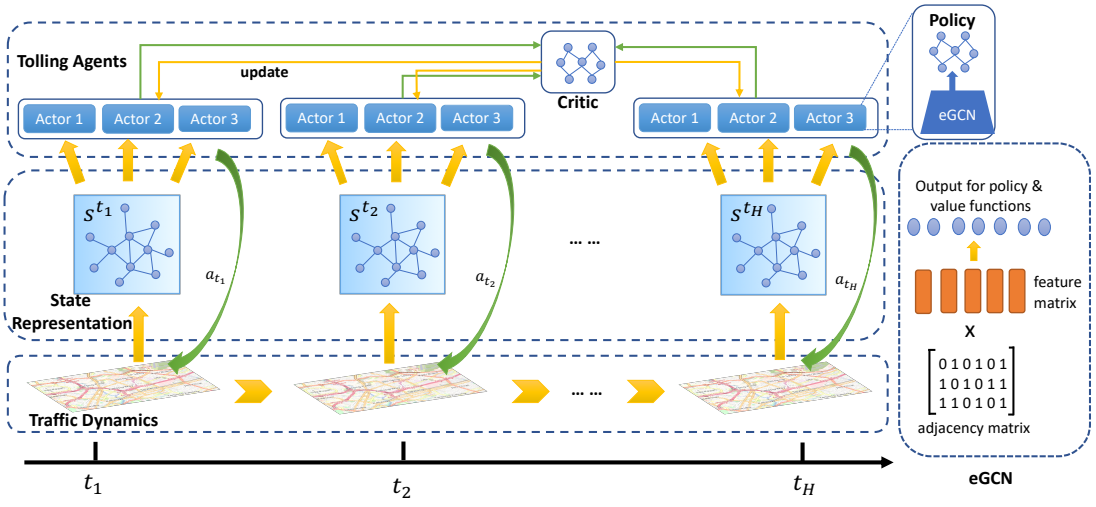


FIGURE 4.1: The architecture of the proposed MARL-eGCN with three modules, *i.e.*, the *traffic dynamics*, the *state representation*, and the *tolling agent*. At each time step, the state matrix is extracted from the *traffic dynamics* module and fed into the *state representation* (eGCN) module together with the adjacency matrix of the road network. Then, the output of eGCN is fed into the *tolling agent* module. The policies generate tolling actions, which are further input into traffic dynamics to change the traffic flow. The *actor* and *critic* of the *tolling agent* are updated according to the reward emitted by the *traffic dynamics*.

4.4 Solution Method: MARL-eGCN

In this section, we introduce the improved PG-β with a deep neural network to better represent the policy and value networks of the tolling agents, together with a temporal difference learning to boost the update of target values. To extract the spatio-temporal correlations of the state features for the road network, we propose a novel edge-based graph convolutional neural network, which is adapted to the problem structure. To tackle the large state-action space in DETC, we decompose the state and action space and solve DETC with the proposed MARL model.

4.4.1 Deep PG-β with TD-learning

The linear representation of PG-β fails to capture the inner correlation of state and the value function updating method, (*i.e.*, Monte Carlo Method), of PG-β is not efficient because it updates the value function until the end of an episode. We propose DPG-β, which employs deep neural networks to replace the linear representation of PG-β to better represent the policy and value functions of the

tolling agent, together with a temporal difference (TD) learning in place of the Monte Carlo method to accelerate the update of target values. In DPG-β, we use two fully connected layers with tanh nonlinearity by adopting the idea of target network [23], we define the TD-error loss function at time step t as

$$\mathcal{L}^t(\phi^t) = \mathbb{E}_{\mathbf{s}^{t+1}\sim T}\left[\left(Q_{dpg}^{t+1}\left(\mathbf{s}^t, \mathbf{a}^t\right) - y\right)^2\right], \tag{4.2}$$

where ϕ^t is the parameters of the network at time step t and T is the state transition function. $y = r + \lambda \cdot Q_{dpg}^{t+1}(\mathbf{s}^{t+1}, \mathbf{a}^{t+1})$ is the TD-target value. Q_{dpg}^{t+1} is the value function at time step t for policy π^{t+1}. The TD-error is applied to update the policy function, the TD-target is applied to update the value function, and the policy distribution is the Beta distribution.

4.4.2 Edge-Based Graph Convolutional Networks Representation

Naive feature representation with the deep neural networks has only limited improvements over the state-of-the-art methods as it fails to exploit the graph nature of the road network. As a result, we adopt a graph convolutional networks (GCNs) framework to represent the value and policy functions in the DETC problem. Moreover, the states define the traffic volume of roads (edges) and vehicles transit from roads to roads in a road network and impact the price of tolled roads in DETC. Unlike ordinary GCNs [122–125] which take nodes as inputs, we design a novel edge-based graph convolutional neural network (eGCN) on DETC road network to extract the spatio-temporal correlations of the state (edge) features. Both the feature matrix and adjacency matrix in the problem of DETC is defined on edges instead of nodes.

We define eGCN as $\mathbf{f}(\mathbf{s}, \mathbf{V})$ where $\mathbf{s} \in \mathbb{R}^{|E|\times|Z|}$ is the state matrix of a given road network, and $\mathbf{s}^t$ is state matrix at time step t which is defined earlier at section 4.3.2. $\mathbf{V}$ is the adjacency matrix of the edges. We first apply an embedding layer to each layer to obtain a latent matrix $\mathbf{Y}$ and apply a non-linear activation function to $\mathbf{Y}$. We feed a state matrix $\mathbf{s}^t$ at time step t to eGCN. Formally, we adopt the formulation proposed by Kipf and Welling [124] to model the proposed eGCN with

the following layer-wise formulation:

$$\mathbf{Y}^{(l+1)} = \sigma(\tilde{\mathbf{M}}^{-\frac{1}{2}}\tilde{\mathbf{V}}\tilde{\mathbf{M}}^{-\frac{1}{2}}\mathbf{Y}^{(l)}\mathbf{\Phi}^{(l)}), \tag{4.3}$$

where $\tilde{\mathbf{V}} = \mathbf{V} + \mathbf{I}$, $\mathbf{V}$ is the adjacency matrix defined on the edges, and $\mathbf{I}$ is an identity matrix to account for the effect of a node itself. $\mathbf{M}$ is a diagonal node-degree matrix utilized to normalize $\tilde{\mathbf{V}}$. $\mathbf{Y}^l$ and $\mathbf{Y}^{(l+1)}$ are the outputs of the l-th and the $(l+1)$-th layers, respectively. $\mathbf{\Phi}^l$ is a trainable weight matrix for the l-th layer, and $\sigma(\cdot)$ is the activation function. The final output of eGCN is put into the neural network of policy and value function as demonstrated in Figure 4.1.

4.4.3 MARL with eGCN

It is very challenging to train one single agent under the large state space and large and bounded-continuous action space or training such an agent may lead to sub-optimal performance due to the huge amount of parameters to be optimized. A key observation is that traffic and road networks in metropolises can be partitioned by geographical distance, demographic distribution, and electoral divisions.

As shown in Figure 4.2, there are 4 partitions and each partition has some nodes (zones). The magenta lines in the figure are tolled roads. Therefore, the pricing scheme in each partition is largely based on the traffic condition within the partition, and, correspondingly, the pricing scheme between partitions is dominated by traffic conditions between neighboring partitions. Inspired by this urban characteristic, we propose a multi-agent reinforcement learning (MARL) solution by dividing the planning areas into N partitions. Each partition is treated as one

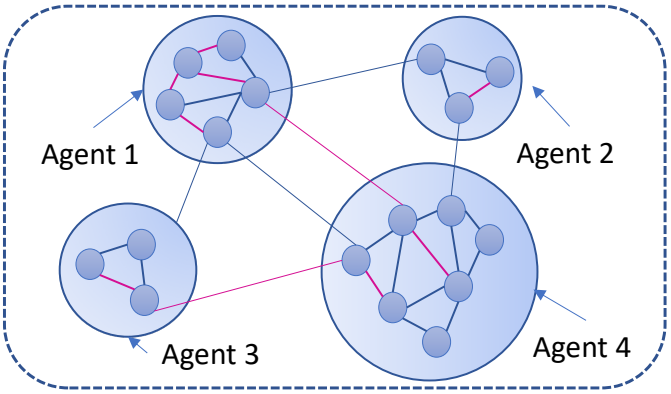


FIGURE 4.2: An example of zone partitions.

agent, and all the agents cooperate by sharing states with each other to solve the formulated MDP introduced in section 4.3.2. We adopt the framework of centralized training with decentralized execution. Figure 4.1 illustrates the architecture of MARL-eGCN.

More concretely, we employ a general and fast policy gradient algorithm, *actor critic* where *actor* is the policy function and critic is the value function, to build the proposed MARL model with eGCN. We use $\theta^t = \left\{\theta_t^i, ..., \theta_t^N\right\}$ to denote policy parameters where $t \in [1, ..., H]$ and use ϕ to denote the parameter matrix of the value networks. We define the state of agent i of partition i at time step t as $\hat{\mathbf{s}}^{i,t} = \langle \mathbf{s}^{i,t}, s_{e,j'}^{i,t} \rangle$, where $\mathbf{s}^{i,t} = \langle \mathbf{s}_e^{i,t} \rangle$. $s_{e,j}^{i,t}$ denotes the number of vehicles on road e of partition i, which travel to zone j of partition i at time step t. $s_{e,j'}^{i,t}$ denotes the number of vehicles on road e of partition i, which head for zone j' of other partitions at time step t. The global state $\mathbf{s}^t$ at time step t is a global status of the traffic condition. The action for agent i at time step t is $\mathbf{a}^{i,t} = \langle a_e^{i,t} \rangle$ and the global action at time step t is $\mathbf{a}^t$, which is a concatenated vector of actions of all agents. The reward r_i^t for each agent i given $\hat{\mathbf{s}}^{i,t}$ at time step t is the number of vehicles that arrive at their destinations.

Then we build a local adjacency matrix $\mathbf{V}_i$ with size $K \times N$ for edges in partition i with its K neighbors where each row is a binary vector indicating neighbors of agent i. Finally the individual input for agent i is $\rho_t^i = \mathbf{V}_i \times \mathbf{f}(\mathbf{s}^t, \mathbf{V})$. We use Beta distribution as the policy distribution for the bounded and continuous action space, then we define the set of policies for each agent as $\pi = \{\pi_i\}_{i=1,\cdots,N}$. Hence, we can derive the gradient of the performance measure $\mathbf{J}$ for the policy function of agent i as:

$$\nabla_{\theta_t^i} \mathbf{J}(\theta_t^i) = \mathop{\mathbb{E}}_{\mathbf{s}_{t+1} \sim T} [\nabla_{\theta_t^i} \log \pi_i(\mathbf{a}^{i,t} | \rho_t^i) \cdot Q^t(\rho_t^i, \mathbf{a}^{i,t})], \tag{4.4}$$

where $Q^t(\rho_t^i, \mathbf{a}^{i,t})$ is the centralized action-value function that inputs ρ_t^i that output by eGCN at time step t as shown in line 4 in Algorithm 2. ρ_t^i is the spatio-temporal state for agent i related to the partition of agent i, which is different from the input for vanilla actor-critic. We define the loss function of the value function as follows:

$$\mathcal{L}^t(\phi^t) = \mathbb{E}_{\mathbf{s}_{t+1} \sim T} \left[\left(Q^t(\rho_t^i, \mathbf{a}^{i,t}) - y \right)^2 \right], \tag{4.5}$$

where ϕ is the parameter matrix of $\mathcal{L}$ and $y = r_i + \lambda \cdot Q^t(\rho_t^i, \mathbf{a}^{i,t})$ is the TD-target value of agent i, which is estimated by the value network and Q^t is the value

Algorithm 2: MARL-eGCN

input: $\theta_t^i \in \mathbb{R}$, $j \in [1, ..., N]$ and ϕ

1 **for** $e \leftarrow 0$ **to** *MAX-EPISODE-NUMBER* **do**
2 **while** $t = 1 <$ *MAX-EPISODE-LENGTH* **do**
3 **for** *agent* i *in* N **do**
4 $\rho_t^i = \mathbf{V}_i \times \mathbf{f}(\mathbf{s}^t, \mathbf{V})$;
5 $\mathbf{a}^{i,t} = \pi(\rho_t^i, \theta_t^i)$;
6 Concatenate $\mathbf{a}^{i,t}$, $i \in [1, .., N]$ into $\mathbf{a}_t$;
7 Take $\mathbf{a}^t$ into traffic road graph and get $\hat{\mathbf{s}}^{i,t+1}$;
8 **foreach** $i \leftarrow 0$ **to** N **do** get r_t^i;
9 **for** $i \leftarrow 0$ **to** N **do**
10 $y_t^i = \sum_{t'=t}^{H} r_{t'}^i$;
11 Update critic by minimizing the loss: $\mathcal{L}(\phi) = y_t^i - Q(\rho_t^i, \mathbf{a}_t)$;
12 Update actor using the policy gradient:
 $\nabla_{\theta_t^i}\mathbf{J} = \nabla_{\theta_t^i}\pi_i(\mathbf{a}^{i,t}|\rho_t^i) \cdot Q(\rho_t^i, \mathbf{a}^{i,t})$;
13 $\hat{\mathbf{s}}^{i,t} = \hat{\mathbf{s}}^{i,t+1}$;
14 **return** θ_t^i, $i \in [1, ..., N]$;

function at time step t. We call the MARL model MARL-eGCN. Algorithm 2 illustrates the detailed description of MARL-eGCN where embedded features are output by eGCN in line 4, the policy outputs action vector for each agent in line 5 and the parameters of *actor* and *critic* are updated in lines 11 and 12.

4.5 Experimental Evaluation

In this section, we evaluate our proposed approach in the real-world road networks of Singapore.

4.5.1 Experiment Setup

We first introduce our problem scenarios, and the compared methods, parameters of the model, and the training settings.

Problem Scenarios. As shown in Figure 4.3, we choose the road network of Singapore's central region (Central Net) and the road network of Singapore's all planning areas (Whole Net) as our experimental scenarios. There are 11 zones (planning areas) in Central Net and over 33 zones in Whole Net. We create an

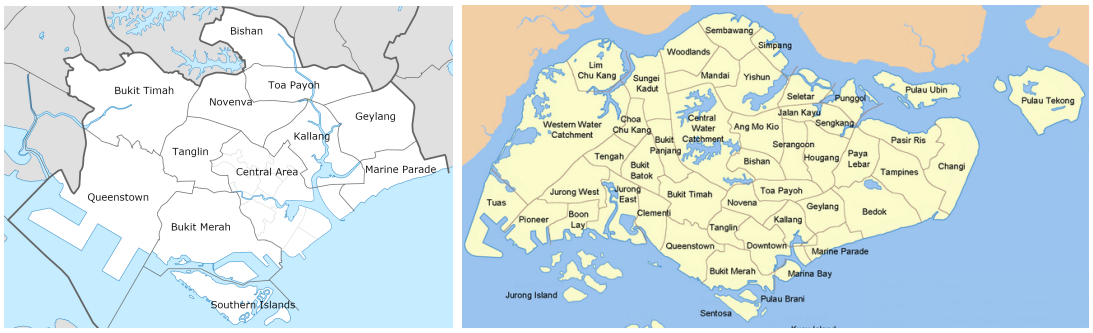


FIGURE 4.3: Map of Singapore: Singapore central region and entire Singapore planning areas.

abstract road network for both the Central Net and Whole Net based on the distribution of ETC gantries, the arterial roads, and the highway network of Singapore. We merge some zones that have low populations and small acreage into their neighboring zones, and thus we get 11 zones and 40 edges (roads) for Central Net and 33 zones and 124 edges for Whole Net. We estimate the OD demand for the two road networks by using population data of each zone and Annual Vehicle Statistics 2017 published by the Singapore government [126]. We first obtain the total number of vehicles as 961,842 and the population of Singapore as 5,612,300, and thus we get the per-person vehicle ownership rate as 0.171. Then we obtain the population of each zone and finally estimate the number of vehicles in each zone. We set $\gamma = 0.15$ and $\xi = 4$ according to [117], and we set the toll range $[0, 6]$ for each arterial road based on the current ETC scheme in Singapore. The time horizon H is set as 6 and each time step is 2 hours.

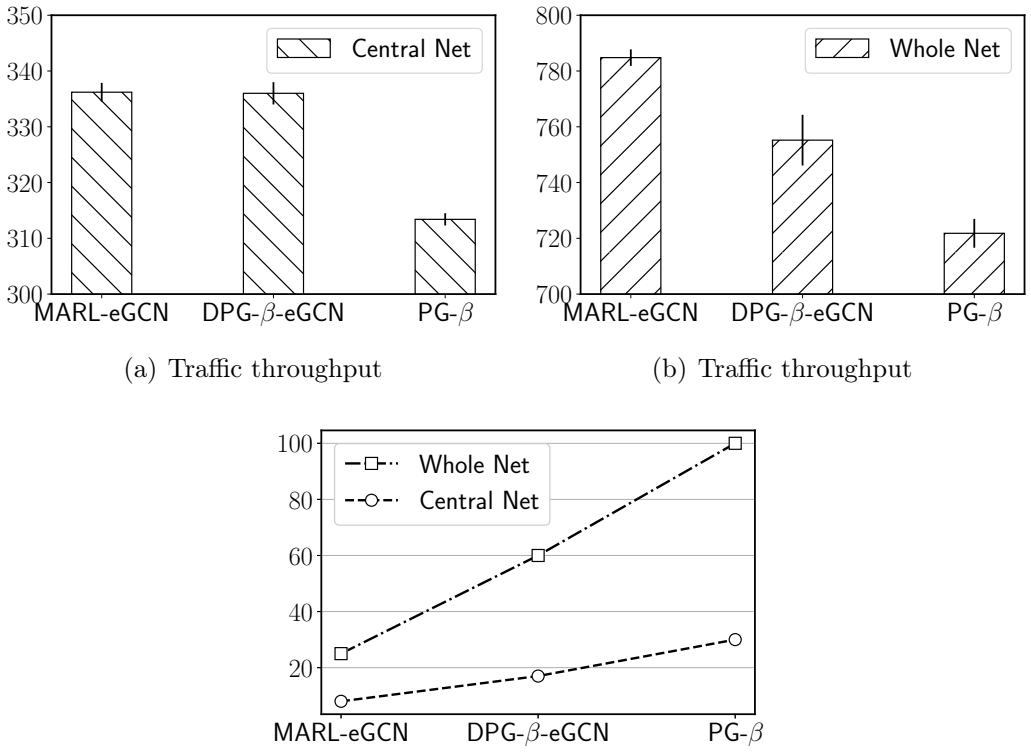
Compared Methods. The methods that we evaluated include (i) PG-β, (ii) DPG-β, (iii) DPG-β-eGCN, which combines DPG-β with eGCN, (iv) MARL-eGCN and (v) MARL (MARL-eGCN without eGCN). The first 3 methods are single-agent methods, while the last two are multi-agent methods. For multi-agent methods, we train 2 agents on Central Net and 4 agents on Whole Net, respectively.

Parameters of Model & Training Settings. The discount factor λ of PG-β is 1 (as that in [44]) and set as 0.9 for the other models. There are 2 dense layers for all neural networks each with 128 neurons for single-agent models and each with 64 neurons for all multi-agent models. The learning rates of policy and value function for single agent models are 0.0001 and 0.0005 respectively. The learning rates of policy and value function for multi-agent models are 0.001 and 0.0005 respectively. We use tanh activation function for all neural networks. We use 2 layers for eGCN,

with a shape of $(|\mathcal{Z}|, 16)$ and $(16, 16)$, respectively. We use *Adam* [127] to optimize deep neural networks. We train PG-β for 500,000 episodes and the other models for 100,000 episodes. We implement the traffic dynamics with C++ and create an extension file for Python with Boost.Python[2] to speed up the Python code. All models are implemented by Python and run on a 64-bit server with 500 GB RAM and 80 Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz processors.

4.5.2 Result Analysis

We use the number of vehicles that arrive at their destinations (*i.e.*, traffic throughput) and the number of episodes for convergence as the evaluation criteria.



(a) Traffic throughput

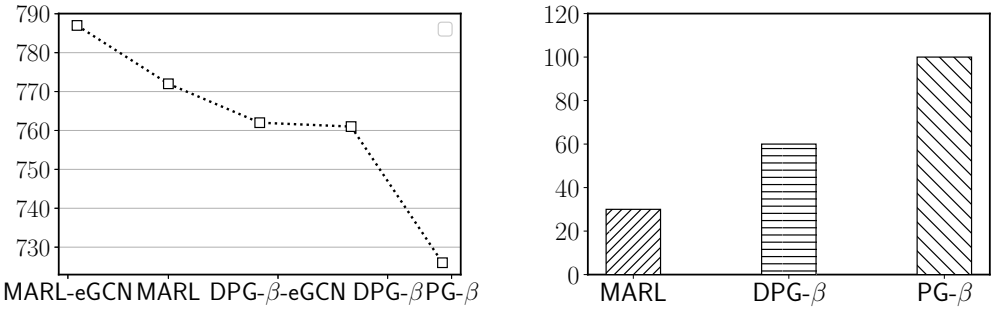
(b) Traffic throughput

(c) Compare convergence

FIGURE 4.4: Traffic throughput (95% confidence interval) and convergence comparison (values in thousands).

Central Net. We first compare MARL-eGCN, DPG-β-eGCN, and PG-β on the road network of Singapore's central region. As illustrated in Figure 4.4, under

[2] `https://www.boost.org/doc/libs/1_68_0/libs/python/doc/html/index.html`



(a) Traffic throughput

(b) Compare convergence

FIGURE 4.5: Ablation study (values in thousands).

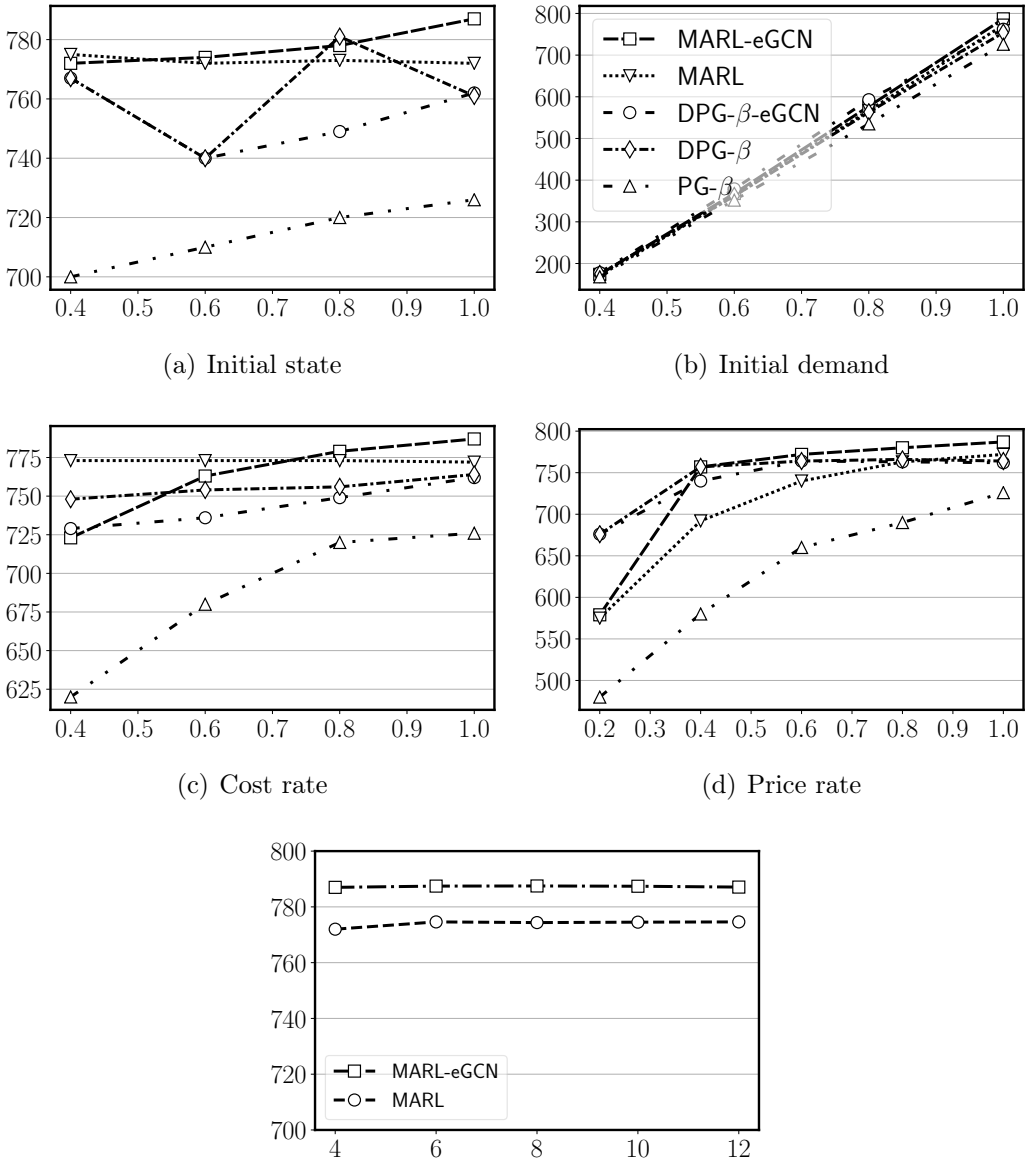
the scenario of Central Net, both DPG-β-eGCN and MARL-eGCN significantly outperform PG-β. We notice that for a small-scale road network like Central Net, the multi-agent (MARL-eGCN) and single-agent (DPG-β-eGCN) models have similar traffic throughput. In terms of training efficiency, we can see that compared with PG-β, DPG-β-eGCN uses around 50% of episodes to converge, while MARL-eGCN uses only 27.7%.

Whole Net. To further demonstrate the advantage of MARL-eGCN in large-scale road networks, we conduct experiments on the Whole Network. As shown in Figure 4.4, MARL-eGCN and DPG-β-eGCN are superior to PG-β. Compared with PG-β, MARL-eGCN gains 8.4% larger traffic throughput and uses only 25% of episodes to get stable results. The results demonstrate that MARL-eGCN can get better traffic throughput with fewer training episodes.

4.5.3 Ablation Study and Robustness Test

As MARL and eGCN are the two key components of MARL-eGCN, we conduct an ablation study on Whole Net to investigate the contribution of these components. The traffic throughput results are shown in Figure 4.5(a), and the convergence comparison results are shown in Figure 4.5(b).

First, by comparing MARL-eGCN with MARL, we can see that after removing eGCN, MARL gets 2% lower traffic throughput and 20% larger number of episodes for convergence, compared with MARL-eGCN. Then, we remove both MARL and eGCN, and compare MARL-eGCN with DPG-β. Results show that MARL-eGCN



(a) Initial state

(b) Initial demand

(c) Cost rate

(d) Price rate

(e) Number of agents

FIGURE 4.6: Traffic throughput (in thousands) under various traffic conditions (Figures a, c, d share the same legend in Figure b).

outperforms DPG-β with 3.5% improvement in terms of traffic throughput and only half number of training episodes for convergence.

To compare tolling schemes under different traffic settings, we vary one parameter and keep other parameters fixed for further evaluation. Figure 4.6 depicts traffic throughput obtained from different pricing schemes under different parameter settings where the x-axis is the value of the parameter under evaluation and the y-axis is the traffic throughput (in thousands). As shown in Figure 4.6(a), the traffic throughput increases linearly w.r.t the increasing initial state except for PG-β and its variants that are more unstable under various initial states. In Figure 4.6(b), all models are sensitive to the changing initial demand, which is intuitive because OD demand is the main factor that impacts the traffic throughput. Similarly in Figure 4.6(c), with a higher cost rate, traffic throughput increases nearly for all tolling schemes among which PG-β is more sensitive to the cost rate. Under different maximum price rates, in Figure 4.6(d), MARL-eGCN outperforms other methods. By adding more agents, as illustrated in Figure 4.6(e), we found that MARL-eGCN gets the largest traffic throughput under 8 agents and traffic throughput decreases with over 8 agents, while for MARL, the optimal number of agents is 6. This is because more agents make learning harder due to feature redundancy and larger coordination overhead, while fewer agents cannot leverage the power of multi-agent learning. In general, our approach MARL-eGCN outperforms existing tolling methods under all settings and is consistently better than the state-of-the-art method PG-β. Compared with a single super agent which may lead to sub-optimal behaviors in some environments, our MARL approach can be more general and adaptive to various scenarios.

4.6 Chapter Summary

To scale up the state-of-the-art RL-based approaches to the DETC problem, this chapter proposes a novel learning architecture called MARL-eGCN. The two intrinsic ideas of MARL-eGCN are: (i) we design a problem-specific cooperative multi-agent RL framework to decompose the huge state and action spaces into sub-spaces and solve the DETC problem in a divide-and-conquer manner; (ii) we devise an edge-based GCN representation of the value and policy functions for the tolling agents, to exploit the inter-correlations among edges in the road network.

By performing extensive experimental evaluations on a real-world traffic network in Singapore, we show that MARL-eGCN significantly outperforms the state-of-the-art approaches in terms of both traffic throughput and training efficiency and can handle real-world-sized DETC problems. Our method can be applied to tackle complex sequential decision-making problems where the problem structures can be decomposed into many partitions, such as optimizing the order dispatch in online car-hailing and improving the throughput of network traffic in large-scale cloud systems.

Our method requires domain knowledge to create partitions of the road network, which is a main limitation. We have evaluated our method in a real-world traffic network in Singapore. However, the performance of our method in other cities is still yet to be evaluated. We are also interested in evaluating the generalization performance of our method. That is, we train our method in the road network of a city and deploy it to other cities, which is crucial for the massive deployment of MARL methods. We leave them to the future work.

Chapter 5

Learning to Coordinate in Environments with Off-Beat Actions

While Multi-Agent Reinforcement Learning (MARL) has seen success with centralized training with decentralized execution (CTDE), efficient coordination in complex multi-agent systems remains difficult due to action durations. This chapter[1] delves into the problem of learning efficient coordination with model-free cooperative MARL in multi-agent environments where *off-beat* actions are prevalent, *i.e.*, all actions have execution durations. During execution durations, the environmental changes are influenced by, but not synchronized with action execution. Existing MARL methods often assume immediate feedback of the action execution, leading to failures in multi-agent scenarios with off-beat actions. The primary challenges include the complexity augmentation of off-beat actions during decentralized execution, unknown action durations for agents, and the temporal credit assignment issue in TD-learning. To tackle these issues, this chapter proposes a novel leveled graph-based temporal recency reward redistribution scheme for TD-learning, designed to learn multi-agent coordinative policies efficiently in OBMAS. Empirical evaluations on various games demonstrate significant improvements in multi-agent coordination and overall performance.

[1]The work in this chapter has been published in [8].

5.1 Introduction

In Multi-Agent Reinforcement Learning (MARL), multiple agents act interactively and complete tasks in a sequential decision-making manner with Reinforcement Learning (RL). It has made remarkable advances in many domains, including autonomous systems [20, 22] and real-time strategy (RTS) video games [17]. By the virtue of the centralized training with decentralized execution (CTDE) [31] paradigm, which aims to tackle the scalability and partial observability challenges in MARL, many CTDE-based MARL methods are proposed [26, 32, 33, 35, 38]. With these methods, an agent executes actions only via feeding its individual observations independently and optimizes its own policy with access to global trajectories centrally.

Despite the recent successes of MARL, learning effective multi-agent coordination policies for complex multi-agent systems remains challenging. One key challenge is the *off-beat* actions, *i.e.*, actions have execution durations[2] and during the execution durations, the environmental changes are influenced by, but not synchronized with, action execution (an illustrative example is shown in Figure 5.1). However, Dec-POMDP [15], which underpins many CTDE-based MARL methods, hinges on the assumption that actions are executed immediately, leading to catastrophic failure for centralized training on various off-beat multi-agent scenarios (OBMAS). To fill this gap, we study MARL in settings where off-beat actions are prevalent. Such a setting is very common in many real-world problems. For example, in traffic light control, traffic lights in the conjunctions of the road network may have different execution durations.

Training multi-agent coordination policies in OBMAS efficiently is challenging: (i) agents' actions can have different execution durations, which augments the order of complexity of OBMAS during decentralized execution, resulting in failure of the coordination; (ii) the action durations are unknown to agents during individual executions, and communication is constrained and not always feasible, making it non-trivial to coordinate; (iii) TD-learning [46] is the core method for optimizing policies in RL [23, 130]. By bootstrapping from the most recent estimate of the value function, it alleviates the RL credit assignment problem, which

[2]In the RL literature [128, 129], action execution durations are called *delays of actions*. In this chapter, we use the term *execution durations*, which is self-consistent with off-beat actions defined in Section 5.3.

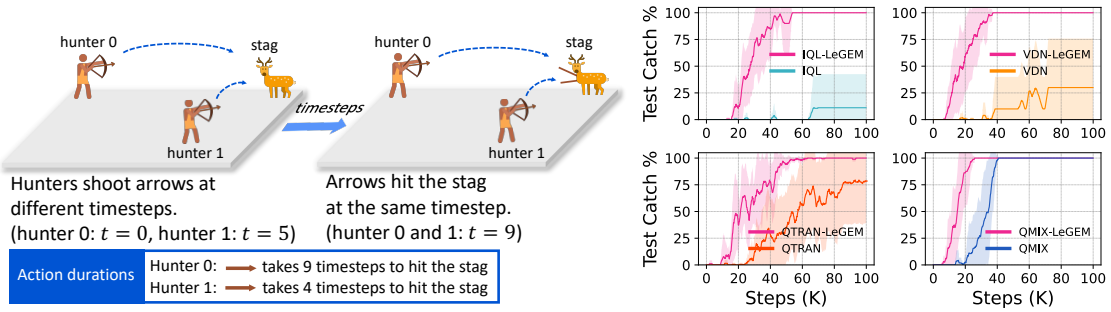


FIGURE 5.1: **An illustrative scenario:** two-agent stag-hunter game, where two agents (hunters) have only partial observations, different durations of the shoot action, and cannot communicate. The goal is to catch the stag and they are rewarded when their shots hit the stag at the same time. Both agents can see the stag. As the shoot action durations of the two agents are different, to catch the stag, the two agents should shoot the arrow at different time steps given the distances. For hunter 0, the time step to shoot the arrow is 0, while for hunter 1, it is 5. At time step 9, the two arrows will hit the stag and all hunters will receive a positive shared reward. Though the scenario is easy for human beings, it is hard for MARL agents due to the action duration. **Experiment results:** in this scenario, the optimal policy for agent 0 is to shoot the arrow at time step 0 while the optimal policy for agent 1 is to shoot the arrow at time step 5. Such an asynchronous property of OBMAS motivates agents to learn tacit policies. The curves show that VDN and IQL fail to learn coordination policies even in this simple scenario. With our LeGEM, MARL methods gain enhanced performance as well as improved sample efficiency.

involves distributing an action's contribution to the reward over the temporal interval. However, the temporal credit assignment issue in TD-learning hampers MARL training due to the displaced rewards in multi-agent experience replay. With off-beat actions, the nonstationarity issue, which mainly stems from rewards' time dependency on the agents' past actions, is exacerbated.

While action durations are ubiquitous, existing works only focus on single-agent settings, *i.e.*, delay, in RL. Many approaches [128, 131, 132] augment the state space with actions to be executed into the environment. However, such a state-augmentation trick leads to exponentially increasing training samples with the growing action duration, making training intractable [133]. Chen et al. [134] extend the delayed MDP [128] and propose Delayed Markov Game for MARL. However, on one hand, such a state-augmentation treatment is confined to short delays, *e.g.*, one time step delay; on the other hand, the delayed time step of the actions is privileged information, which is not available in many scenarios. Recent works on macro-actions [135, 136] introduce asynchronous actions by designing high-level

macro-actions with prior environment knowledge in the robotics domain. Macro-actions are different from options in hierarchical RL [137] in that the former is manually designed. Macro-actions design requires a thorough understanding of the environment, including environment dynamics, reward function, and action semantics. When there are many environments, designing macro-actions is not practical.

This chapter aims to achieve efficient multi-agent coordination in OBMAS. We first propose off-beat actions and off-beat Dec-POMDP. Then, to train multi-agent coordination policies, we propose our novel leveled graph-based temporal recency reward redistribution scheme for TD-learning, which is built on our novel episodic memory and addresses the challenging temporal credit assignment problem raised by the off-beat actions. Specifically, each agent maintains graph-based episodic memories by utilizing agents' individual trajectories. During centralized training, given an observation and an action, each agent searches for its pivot time step from its episodic memory. The pivot time step of each agent is the time step wherein the off-beat reward relates to the time step where the off-beat action was executed. The pivot time steps of each agent are ranked and the final pivot time step will be chosen by recency and later used for reward redistribution and target estimation in TD-learning. We evaluate our method in the Stag-Hunter Game, the Quarry Game, and the Afforestation Game. Empirical results show that our method significantly boosts multi-agent coordination and achieves leading performance.

5.2 Related Works

Action Delay in RL. Conventionally, the execution of actions in RL is instantaneous and the execution duration is neglected. Katsikopoulos and Engelbrecht [138] propose the delayed MDP where actions have delays and Walsh et al. [131] propose a model-based method for the delayed MDP. To optimize the delayed MDP, many RL approaches [128, 131, 132, 139] augment the state space with the queuing actions to be executed. However, this state-augmentation trick is intractable [133]. Chen et al. [134] extend the delayed MDP [128] and propose a delayed Markov game. However, the state-augmentation treatment is confined to short delays in single-agent RL. Recently, Bouteiller et al. [129] applied a replay buffer correction method. However, the delayed time step is privileged information and may change

over time [129]. It is not available for agents in many scenarios. Simply applying this single-agent trajectory correction in MARL cannot attain satisfactory performance due to off-beat actions.

Credit Assignment in RL. Credit assignment [46, 137] tackles long-horizon sequential decision-making problem by distributing the contribution of every single step over the temporal interval. TD learning [29] is the most established credit assignment method, which is the basis of many RL methods. RUDDER [140] redistributes the episodic return to key time steps in the episode [141–143]. Klissarov and Precup [144] propose a reward propagation method via graph convolutional neural network. Another line of works utilizes episodic memory (EM) [145–147] to recall key events and aggregate information of the past for decision-making. However, applying EM to MARL in OBMAS is non-trivial because (i) OBMAS is non-stationary from the perspective of each agent and (ii) novel structures are needed to address the credit assignment issue caused by displaced rewards. It is worth noting that delayed rewards in RL [142, 148] are fundamentally different from off-beat actions in our setting. In RL, the delayed rewards are commonly raised by multi-timestep decision-making, not the action duration at each time step. In our setting, there is a very definite point where this reward is realized by the environment due to the action durations.

MARL. Many MARL methods focus on factorizing the global Q value to train agents' policies via CTDE [26, 32, 33, 35, 38, 40]. However, these existing works assume actions are executed synchronously. Messias et al. [149] propose an event-driven, asynchronous formulation of the multi-agent POMDP. However, the assumption of free communication [12] is unrealistic and the asynchronous execution [150] is confined to the design of events and did not propose methods for solving the challenging credit assignment issue in OBMAS. Recently, Amato et al. [151] and Xiao et al. [135] propose macro-actions, which are similar to hierarchical methods [152] and manually designed via exploiting domain knowledge of the environments. However, these works mainly focus on macro-action selection during multi-timestep decision-making and assume the environment can use pre-defined methods for state transition manually. Unfortunately, it is not feasible to design macro-actions without looking into the environmental dynamics. In contrast, off-beat actions are primitive actions, and our method does not exploit any domain

knowledge of the environment. Besides that, the credit assignment is still a challenge for macro-actions methods in OBMAS and the asynchronous[3] nature of off-beat actions undermines the temporal credit assignment in TD-learning, causing the failure of multi-agent coordination.

5.3 Off-Beat Actions and Off-Beat Dec-POMDP

In this section, we first define off-beat actions[4] for multi-agent scenarios; then we propose the Off-Beat Dec-POMDP.

Definition 5.1 (Off-Beat Actions)**.** Off-beat action $\tilde{u} \in \mathcal{U}$ characterizes OBMAS where the action $\tilde{u}_i$ taken by agent i has execution duration $m_{\tilde{u}_i} \sim A(m|\tilde{u}_i, i)$, $A \in \mathcal{A}$, $m \in \{0, 1, 2, \cdots, M\}$ and $M \leq T$, where T is the maximum duration and A is the action duration distribution. It is a distribution and takes $\tilde{u}_i$ and the index of the agent as parameters. A can be either stochastic or deterministic. The joint off-beat action is $\tilde{\boldsymbol{u}} = [\tilde{u}_i]_{i=1}^N$. The execution duration is decided when the action is committed to the environment. Thus, the execution duration of an action $\tilde{\boldsymbol{u}}_t$ initiated at time step t is $\boldsymbol{m}_t = \{m^t_{\tilde{u}^t_i}\}_{i=1}^N$.

Note that for each agent, $m^t_{\tilde{u}^t_i}$ [5] can be different. At time step t, there are at least 1 action[6] and at most N actions being initiated (committed to the environment for execution), leading to asynchronicity of the joint actions. Next, we formulate the Off-Beat Dec-POMDP for OBMAS.

[3] We clarify the term *asynchronous*: actions that are simultaneously committed into the environment by all agents will not complete their respective action durations at the same time in future time steps.

[4] Asynchronicity is prevalent in real-world multi-agent scenarios, including asynchronicity in observations, actions and communication, etc. In this chapter, we focus on the asynchronicity of actions in multi-agent scenarios. For brevity, we name the asynchronicity of actions in MARL as *off-beat*.

[5] We will omit t in the rest of this chapter for brevity.

[6] We note that agents have a special NO-OP action available.

Definition 5.2 (Off-Beat Dec-POMDP)**.** Off-Beat Dec-POMDP extends Dec-POMDP, such that
(1) state space is $\mathcal{S}$; (2) joint action space is $\mathcal{U}^N$; (3) action duration space is $\mathcal{A}^N$;
(4) transition function is $\mathcal{P}(\boldsymbol{s}'|\boldsymbol{s}, \tilde{\boldsymbol{u}}, \boldsymbol{m}) : \mathcal{S} \times \mathcal{U}^N \times \mathcal{S} \times \mathcal{A}^N \mapsto [0, 1]$, and $\boldsymbol{m}$ is the action durations of the joint action;
(5) the reward function is $R(\boldsymbol{s}, \tilde{\boldsymbol{u}}, \boldsymbol{m}) : \mathcal{S} \times \mathcal{U}^N \times \mathcal{A}^N \mapsto \mathcal{R}$;
(6) we call a reward r as off-beat reward when any its $m_{\tilde{u}_i} \geq 1$, $m_{\tilde{u}_i} \in \boldsymbol{m}$, and $r \neq 0$.

In OBMAS, at each time step t, the environment receives actions that agents initiate for execution in the environment. The initiated actions $\tilde{\boldsymbol{u}}_t$ are instantaneous actions inferred by agents' policies given individuals' observations. The joint reward is the consequence of the committed joint actions of the current time step and previous time steps, depending on the actions' durations. We discuss some properties of Off-Beat Dec-POMDP below.

Remark 5.1. *When the durations for all actions are identical, we call off-beat Dec-POMDP as delayed Dec-POMDP and there is no off-beat action in it.*

In delayed Dec-POMDP, actions have the same delayed time steps, which is different from off-beat actions, where actions have different action durations or delays.

Remark 5.2. *There exists $\tilde{\boldsymbol{u}}$ that is synchronous since the durations of agents' actions can be $m = 0$. When m of all actions is zero, off-beat Dec-POMDP reduces to Dec-POMDP.*

Remark 5.3 (Non-Markovian Reward). *With off-beat actions, the Markovian property of the reward function $R(\boldsymbol{s}, \tilde{\boldsymbol{u}}, \boldsymbol{m})$ does not hold.*

With off-beat actions, the shared rewards can be readily displaced, causing non-Markovian rewards.

5.4 Solution Method: LeGEM

The key challenge in learning efficient multi-agent coordination for Off-Beat MARL is the temporal credit assignment problem raised by the off-beat actions. We propose LeGEM, which has three components: (i) a novel temporal recency reward

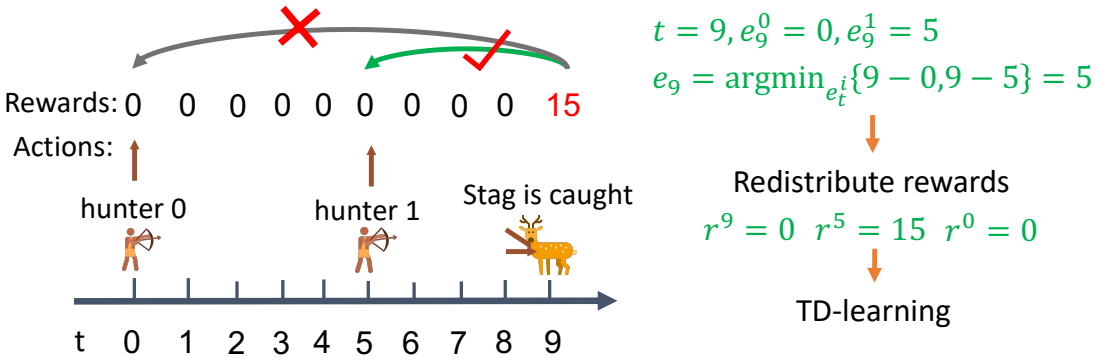


FIGURE 5.2: Reward redistribution for the example in Figure 5.1. In this example, hunter 0 takes the shooting action at time step 0, and hunter 1 takes the shooting action at time step 5. At the other time steps, both hunter 0 and hunter 1 take no-op actions. At time step 9, hunter 0 and hunter 1 receive the global reward of 15. LeGEM distributes the reward of 15 to time step 0 for agent 0 and time step 5 for agent 1, respectively. The final pivot time step is 5 via Equation 5.1. So, the trajectory is updated via the distribution of the reward. Then, we utilize TD-learning to train the MARL policy using the newly modified trajectory.

redistribution method; (ii) novel searching schemes for the reward redistribution method; (iii) a novel leveled graph-based episodic memory named LeGEM-core. We first introduce our reward redistribution method in Section 5.4.1, followed by LeGEM-core in Section 5.4.2. LeGEM-core provides graph structures and empowers the searching schemes (in Section 5.4.3) for our reward redistribution method.

5.4.1 Temporal Recency Reward Redistribution

The key obstacle impeding learning efficient multi-agent coordination with TD-learning for Off-Beat MARL is the displaced rewards caused by off-beat actions. The TD error is the difference between the TD target and the prediction. TD targets can be estimated with n-step target, TD(λ) and other techniques [29]. To train MARL policies in OBMAS, one should accurately estimate the TD target where the reward plays the key role [153]. Unfortunately, current n-step target and TD(λ) methods cannot accurately estimate TD targets for Off-Beat MARL. We resolve the aforementioned issue by redistributing rewards to agents' pivot time steps (we will introduce the method for searching agent's povit time steps in

Section 5.4.3). The *pivot time step* of each agent is the time step when the off-beat action was executed and later the off-beat action triggers the off-beat reward. For example, in Figure 5.1, when two hunters receive the reward at time step 9, the pivot time steps $e^i_{t=9}$ ($i \in \{0, 1\}$) are 0 and 5 for hunter 0 and hunter 1, respectively.

More concretely, our key idea is that we can pull the reward of the agent's off-beat action back to the time step when it was committed into the environment, which can dramatically enhance learning despite the long-term reward delays incurred by off-beat actions. The time step to which the reward should be distributed is called the *final pivot time step*. We denote the *final pivot time step* at time step t as e_t. However, for a shared off-beat reward at time step t, each agent's pivot time step e^i_t can be different. It is vital to address the inter-agent credit assignment and decide the final pivot time step e_t. We can get e_t via proximity:

$$e_t = \arg\min_{e^i_t} \{t - e^i_t\}^N_{i=1} \tag{5.1}$$

In the example of Figure 5.1, the final pivot time step e_9 for time step 9 is $e_{t=9} = 5$. Note that proximity is effective for MARL training as credits will be backpropagated to time steps that are far away from the e_t via Bellman Equation and RNN-based policy networks [26]. Then, we can utilize it to update the reward in each transition $(\boldsymbol{s}^{e_t}, \tilde{\boldsymbol{u}}^{e_t}, r^{e_t}, \boldsymbol{s}^{e_t+1})$:

$$\hat{r}^{e_t} = \mathbb{1}(e_t \geq t) \cdot r^{e_t} + \mathbb{1}(e_t < t) \cdot r^t, \tag{5.2}$$

where $\mathbb{1}(\cdot)$ is the indicator function. r^{e_t} is replaced with $\hat{r}^{e_t}$.

This update rule is conducted iteratively from $t = 0$ to $t = T-1$. To stabilize learning and circumvent the overestimation of the TD target, r^t is also updated after Equation 5.2 via $(1-\mathbb{1}(e_t < t) \cdot (1-\beta)) \cdot r^t$. It also avoids aggregated biased/wrong estimation of TD target being backpropagated in the Bellman equation. β is a very small positive hyperparameter. Therefore, we can utilize $\hat{r}_{e_t}$ for centralized training in TD-learning:

$$\mathcal{L}^{\text{TD}}(\theta) := \mathbb{E}_{\mathcal{D}' \sim \mathcal{D}}[(\hat{y}^{e_t} - Q^{\text{tot}}_\theta(\boldsymbol{s}^{e_t}, \tilde{\boldsymbol{u}}^{e_t}))^2], \tag{5.3}$$

where $\hat{y}^{e_t} = \hat{r}^{e_t} + \gamma \max_{\tilde{\boldsymbol{u}}'} Q^{\text{tot}}_{\bar{\theta}}(\boldsymbol{s}^{e_t+1}, \tilde{\boldsymbol{u}}')$. We can incorporate it into MARL methods and present the experimental results in Section 5.5. We also present an example in Figure 5.2 to illustrate the workflow of our reward redistribution method.

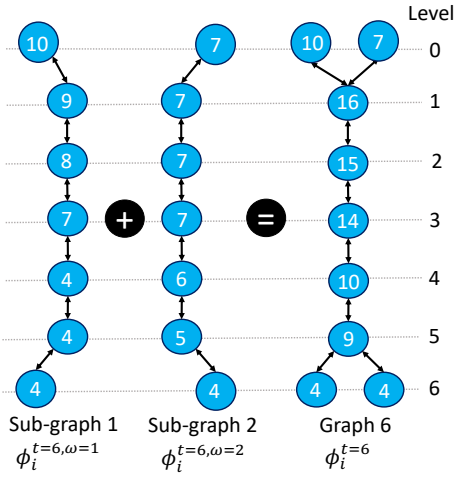


FIGURE 5.3: The maximum level of the graph is 7. Circles indicate the nodes and numbers indicate the visit count.

5.4.2 LeGEM-core: The Episodic Memory

The reward redistribution method introduced in Section 5.4.1 relies on certain structures to search the pivot time step e_t^i for agent i. Episodic memories can explicitly recall the past and identify key states that lead to future rewards [146]. We propose our novel leveled graph-based episodic memory, called LeGEM-core. Since the action durations are unknown to agents, we can search the final pivot time step e_t from LeGEM-core (in Section 5.4.3). LeGEM-core memorizes each agent's past trajectories which are an agent's partial observations and unilateral actions. During training, each agent i collects its individual trajectories τ_i. We then define τ_i of agent i as $\tau_i = [(o_i^0, \tilde{u}_i^0, r^0), \cdots, (o_i^{T-1}, \tilde{u}_i^{T-1}, r^{T-1})]$, where T is the length of the trajectory and the triplet $(o_i^t, \tilde{u}_i^t, r^t)$ represents the observation, action and reward of time step t. Note that r^t is globally shared between agents.

Graphs and Sub-Graph. Each agent's episodic memory (EM) has T graphs categorized by the length of the episode. Each graph consists of many sub-graphs that are categorized by the episode return. We define the graph of agent i's EM as a directed graph $\phi_i^t \in \Phi_i$ where Φ_i is the set of graphs of agent i and ϕ_i^t is the t-th graph of Φ_i, $t \in \{0, \cdots, T-1\}$. To model an agent's behavior explicitly and make the trajectories easy to represent, we create T graphs for each agent and let $\Phi_i = \{\phi_i^t\}_{t=0}^{T-1}$ where T is the maximum depth of all graphs and the maximum length of the episode as well. The maximum level of ϕ_i^t is $t+1$. The graph consists of nodes that are connected by edges. Each node contains the key, the

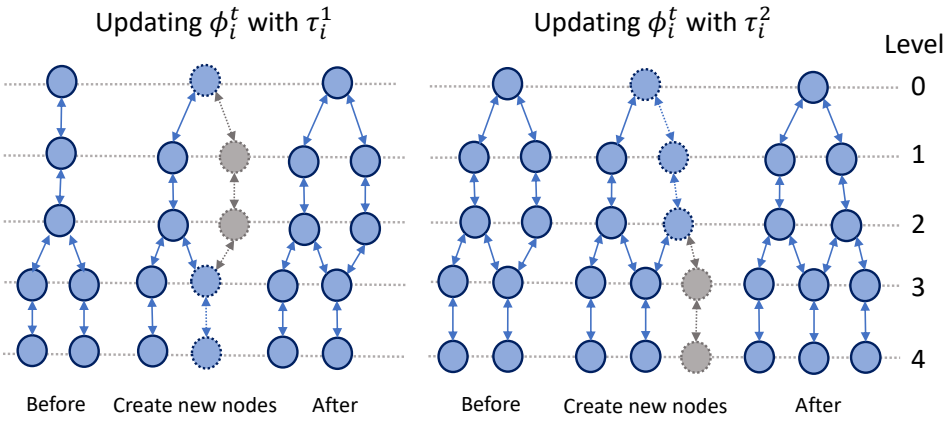


FIGURE 5.4: **Updating agent i's Φ_i:** Agent i's ϕ_i^t is updated with agent's trajectories τ_i^1 and then updated with τ_i^2. Solid arrows and circles indicate the pointers and nodes, respectively. Grey dotted lines indicate pointers to be created and grey circles with dotted outlines indicate nodes to be created. All the dotted elements (pointers and circles) consist of the newly added path in τ_i^1. All the created pointers and nodes will be added to ϕ_i^t.

visit count, and pointers connecting the precursors (nodes at the previous level) and the successors (nodes at the next level). Besides the ϕ_i^t, we define the sub-graph set of ϕ_i^t as $\Phi_i^{t,\Omega} = \{\phi_i^{t,\omega}\}_{\omega=0}^{\Omega-1}$ by using the discretized episode return and there are Ω sub-graphs. $\phi_i^{t,\omega}$ is the ω-th sub-graph whose episode return is the ω-th item in $[0, \cdots, \boldsymbol{r}^{t,i}]$ where $\boldsymbol{r}^{t,i}$ is the maximum discretized episode return of ϕ_i^t. We define the key $(o_i^t, \tilde{u}_i^t)$ using agent i's observation o_i^t and action $\tilde{u}_i^t$ at time step t. In Figure 5.3, we provide an example to showcase the relationship between sub-graphs and the graph of the EM. For complex and continuous state scenarios, for example, StarCraft II scenarios, we use SimHash [154] to discretize the key $(o_i^t, \tilde{u}_i^t)$, which has been widely used in commercial search engines and RL [155], and is not environment-dependent, *i.e.*, no domain knowledge is required. The visit count of the node indicates the total number of visits made by agent i to the node. The initial value of the visit count is 1. Nodes are bidirectionally connected for ease of searching.

Creating and Updating Graphs. Given τ_i of length T, if the node is already in the graph at level t, we then increase the visit count by 1. Otherwise, we create a new node for level t of the graph and update its pointers. Meanwhile, sub-graphs will be also created and updated. The process of creating and updating LeGEM is intuitive. Algorithm 3 shows the whole procedure to construct the graph. To illustrate it, we provide an example in Figure 5.4 to show how to

construct the graph. Figure 5.3 shows the relationship between sub-graphs and the graphs. It is worth noting that τ_i is generated via the interaction of the agent with the environment, and no additional interaction is needed to collect τ_i. τ_i is saved in the experience replay for MARL training. The visit count is vital for searching the pivot time steps (it will be introduced in the following subsection, Section 5.4.3). We use ϵ-greedy for agents to explore the environment and collect individual trajectories.

Algorithm 3: `UpdateLeGEM`

Input: Agent i's $\{\tau_i^d\}_{d=1}^{D}$ and Φ_i.
for $d \leftarrow 1$ ***to*** D **do**
  Get $\phi_i^l \leftarrow \Phi_i$`[length(`$\tau_i^d$`)-1]`; `// length(`$\tau_i^d$`)-1 equals` l
  Calculate the discretized episode reward $\boldsymbol{r}^{l,i}$;
  Get the index ω by using $\boldsymbol{r}^{l,i}$;
  Get $\phi_i^{l,\omega} \leftarrow \Phi_i^{l,\Omega}$`[`$\omega$`]`;
  for $t \leftarrow 0$ ***to*** length$(\tau_i^d) - 1$ **do**
    if $(o_i^t,\, u_i^t) \in \phi_i^l$ **then**
      `// There is no need to update the node of sub-graph` $\phi_i^{l,\omega}$ `as it shares the same node with` ϕ_i^l
      $\psi \leftarrow \phi_i^l$`.getNode(`$o_i^t,\, u_i^t$`)`;
      ψ`.visitCount++`;
    else
      $\psi \leftarrow$ `newNode(`o_i^t, u_i^t, r^t`)`;
      ϕ_i^l`.append(`ψ`)`;
      ϕ_i^l`.updatePointers(`ψ`)`; `// Sub-graph` $\phi_i^{l,\omega}$ `shares the same node with` ϕ_i^l
      $\phi_i^{l,\omega}$`.append(`ψ`)`;
      $\phi_i^{l,\omega}$`.updatePointers(`ψ`)`;
Return: Φ_i.

5.4.3 Searching for the Pivot Timesteps

With LeGEM-core, the structured past experiences of an agent, we can use it to search the agent's pivot time step when the action that triggered the rewarded state was executed. We propose our search scheme for our reward redistribution method.

Search Scheme. For agent i, given τ_i, the corresponding graph is $\phi_i^l = \Phi_i$`[`l`]` ($l =$ `length(`τ_i`)-1`) and $\phi_i^{l,\omega} = \Phi_i^{l,\Omega}$`[`$\omega$`]`, and episode return is $\boldsymbol{r}^{l,i}$. Agent i searches

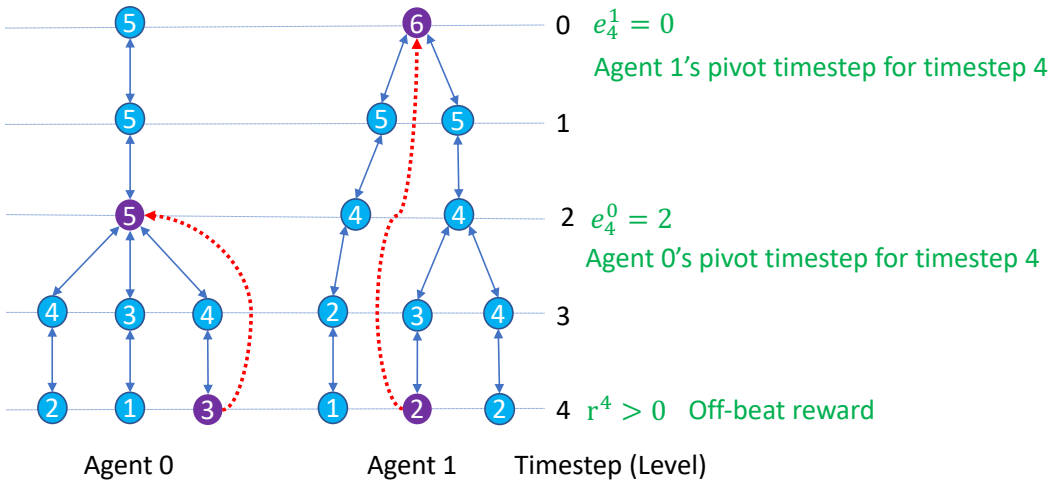


FIGURE 5.5: An example of the search scheme (lines 6-13, Algorithm 4). r^4 is the off-beat reward at time step 4. Each agent searches its pivot time step $e^i_{t=4} (i \in \{0, 1\})$ from its own graph.

Algorithm 4: Search Scheme

Input: ω, Λ, τ_i, $\boldsymbol{r}^{l,i}$ and Φ_i;
Initialize: $\boldsymbol{e}^i_t$: an array whose initial values are all t and its size is the number of paths in Λ;
$\phi_i^{l,\omega} \leftarrow \Phi_i^{l,\Omega}[\omega]$;
vc $\leftarrow$ `VisitCount`(Λ); (Algorithm 6)
foreach *j-th* path $\Lambda[j] \in \Lambda$ **do**
    $e_t^{i,j,\downarrow} \leftarrow$ `UL`$(\Lambda[j], \text{vc}, \tau_i)$; (Algorithm 7)
    $e_t^{i,j,\uparrow} \leftarrow$ `LU`$(\Lambda[j], \text{vc}, \tau_i)$; (Algorithm 8)
    if $e_t^{i,j,\downarrow} \neq -1$ **then**
        $\boldsymbol{e}^i_t[j] \leftarrow e_t^{i,j,\downarrow}$;
    else if $e_t^{i,j,\uparrow} \neq -1$ **then**
        $\boldsymbol{e}^i_t[j] \leftarrow e_t^{i,j,\uparrow}$;
    else
        $\boldsymbol{e}^i_t[j] \leftarrow t$; `// Pivot time step not found`
$e^i_t \leftarrow$ `GetTheMode`$(\boldsymbol{e}^i_t)$; (Algorithm 9)
Return: e^i_t.

from the node (the key is $(o_i^t, \tilde{u}_i^t)$ and $o_i^t \in \tau_i$, $u_i^t \in \tau_i$) at level t in sub-graph $\phi_i^{l,\omega}$ to find the pivot time step e^i_t for r^t. Concretely, we propose a bi-directional search method. The forward search is called the Low-Up (LU) search, which traverses from the given node at level t upwards to the node at level 0. The backward search is named the Up-Low (UL) search, which traverses from the node at level 0 downwards to the given node at level t. LU traversing ends when the pattern of increasing visit count ends, and the corresponding level is the candidate pivot

Algorithm 5: `SearchPivotTimesteps` (ρ)

Input: τ, Φ, and `Search` (the search scheme);
Initialize: κ: an empty list to store pivot time steps;
`// Length of` τ `and` τ_i `are equal.`
$l \leftarrow$ `length(`τ_i`)-1;`
for $t \leftarrow 0$ ***to*** `length(`τ`)-1` **do**
  if $r^t \neq 0$ ($r^t \in \tau$) **then**
    `// Off-beat reward`
    for $i \leftarrow 1$ ***to*** N **do**
      Get τ_i from τ;
      $\phi_i^l \leftarrow \Phi_i[l]$;
      $\psi \leftarrow \phi_i^l$`.getNode(`$o_i^t, \tilde{u}_i^t$`)`;
      Find all the paths Λ from node ψ to the node at level 0;
      Get the discretized episode return $\boldsymbol{r}^{l,i}$;
      Get the index ω with $\boldsymbol{r}^{l,i}$;
      $e_t^i \leftarrow$ `Search(`$\omega, \Lambda, \tau_i, \boldsymbol{r}^{l,i}, \Phi_i$`)`;
    Get e_t (Equation 5.1) and append e_t to κ;
Return: κ.

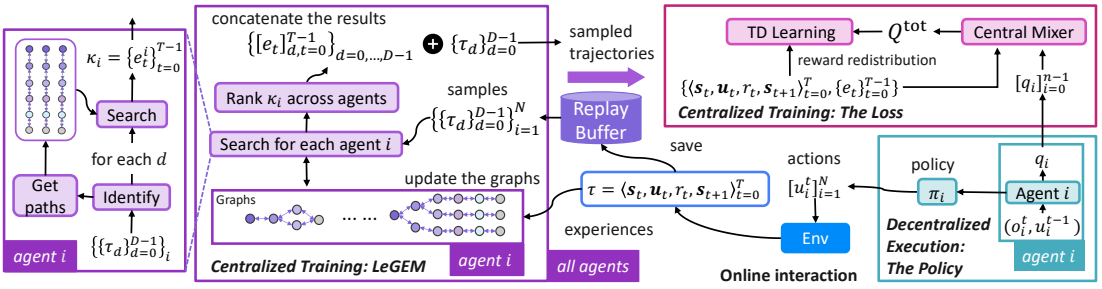


FIGURE 5.6: Our framework: LeGEM, the loss, and the agent's policy. There are two key components: centralized training and decentralized execution.

time step. On the contrary, UL traversing ends when the pattern of decreasing visit count ends and the corresponding level is the candidate pivot time step. In Algorithm 4, we first get visit count (line 4) and then apply UL traversing (line 6) and LU traversing (line 7). We get the results (line 14) by selecting the pivot time step with the maximum count. UL traversing has a higher search priority than its counterpart. The reason is that there exists a pattern in which the visit count decreases from the node at level 0, and such a pattern ends at the pivot time step. In practice, it works well in scenarios whose trajectories are single-off-beat-reward trajectories (there is only one off-beat reward) and the accuracy of the search scheme is over 90% in grid world scenarios. Algorithms 6, 7, 8 and 9 are intuitive and readers can easily understand them literally. We present them in the

following. An example of Algorithm 4 is shown in Figure 5.5.

During training, given the off-beat reward (line 5, Algorithm 5), each agent searches from its EM to find the pivot time step (line 13, Algorithm 5) and the reward is redistributed for MARL training (see Section 5.4.1). We also provide a pictorial view of our framework in Figure 5.6 to show the whole pipeline.

Algorithm 6: `VisitCount`

Input: Λ.
Initialize: vc $\leftarrow$ `[]`, an empty vector to store visit count mask for each path in Λ.
foreach *j-th* path Λ`[`j`]` $\in \Lambda$ **do**
    pathVC $\leftarrow$ `sort(set([node.visitCount for node in` Λ`[`j`]]))`;
    Create an empty look-up table tb $\leftarrow$ `{}`;
    foreach *index* $k \in$ pathVC **do**
        tb`[`pathVC`[`k`]]` $\leftarrow k$;
    Create an empty vector ls $\leftarrow$ `[]`;
    foreach *node* $\in$ *trajectory* Λ`[`j`]` **do**
        ls`.append(`tb`[`node`.visitCount]);`
    vc`.append(`ls`);`
Return: vc.

Algorithm 7: `UL`

Input: Λ`[`j`]`, vc, τ_i.
Initialize: $e_t^{i,j,\downarrow} \leftarrow -1$, res $\leftarrow$ `[]// Initialize the return value and an empty vector`
vc $\leftarrow$ vc`[:`t`]`; `// Slicing the visit count`
left $\leftarrow 0$, right $\leftarrow 1$; `// Create two pointers`
while right $<$ `size(`vc`)` **do**
    if vc`[`right`]` $=$ vc`[`left`]` **then**
        right++; `// Non-decreasing pattern`
    else if vc`[`right`]` $>$ vc`[`left`]` **then**
        res`.append(`right`);`
        left $\leftarrow$ right;
        right++;
    else
        `break; // Break the while loop`
if `size(`res`)` $= 0$ **then**
    $e_t^{i,j,\downarrow} \leftarrow -1$;
else
    $e_t^{i,j,\downarrow} \leftarrow$ res`[-1]; // Get the last value of res`
Return: $e_t^{i,j,\downarrow}$.

Algorithm 8: `LU`

Input: $\Lambda[j]$, vc, τ_i.
Initialize: $e_t^{i,j,\uparrow} \leftarrow -1$, res $\leftarrow$ []. `// Initialize the return value and an empty vector`
vc $\leftarrow$ `reverse(`vc`[:`t`])`; `// Slicing the visit count and then reverse`
left $\leftarrow 0$, right $\leftarrow 1$; `// Create two pointers`
while right $<$ `size(`vc`)` **do**
    if vc[right] $=$ vc[left] **then**
        right++; `// Non-increasing pattern`
    else if vc[right] $>$ vc[left] **then**
        res`.append(`right`)`;
        left $\leftarrow$ right;
        right++;
    else
        `break; // Break the while loop`
if `size(`res`)` $= 0$ **then**
    $e_t^{i,j,\uparrow} \leftarrow -1$;
else
    $e_t^{i,j,\uparrow} \leftarrow$ res`[-1]`; `// Get the last value of res`
    $e_t^{i,j,\uparrow} \leftarrow$ `size(`vc`)` $- e_t^{i,j,\uparrow} - 1$; `// Get the right time step as vc is reversed`
Return: $e_t^{i,j,\uparrow}$.

Algorithm 9: `GetTheMode`

Input: $\boldsymbol{e}_t^i$. `// Receives a vector of pivot time steps`
`// Get the pivot time step with the maximum count`
`// Since getValueOfMaxCount( · ) is easy to implement, for brevity, we do not present its implementation here`
$e_t^i \leftarrow$ `getValueOfMaxCount(`$\boldsymbol{e}_t^i$`)`;
Return: e_t^i.

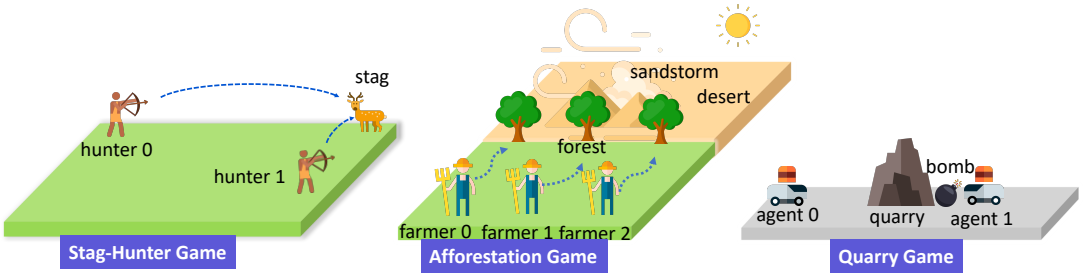


FIGURE 5.7: Stag-Hunter Game, Afforestation Game, and Quarry Game.

5.5 Experiments

We perform experiments on various OBMAS. We aim to answer the following questions: **Q1:** Can LeGEM improve the multi-agent coordination of MARL methods in OBMAS? **Q2:** Can LeGEM outperform previous parameterized episodic memory for MARL? **Q3:** Can bootstrapping methods help to improve the performance of MARL? **Q4:** Can LeGEM outperform the multi-agent exploration and multi-agent risk-sensitive methods? **Q5:** Can RUDDER address the issue caused by the displaced rewards?

TABLE 5.1: Baseline algorithms.

Categories	Methods
MARL (**Q1**)	QMIX[26] VDN [33] IQL [109] QTRAN [35] QPLEX [40]
EM (**Q2**)	EMC [156]
Bootstrap (**Q3**)	N-step Return [29] λ-Return [29]
Ex-Risk (**Q4**)	MAVEN [108] EMC [156] RMIX [10]
Reward Shaping (**Q5**)	RUDDER [140]

5.5.1 Experiment Setup

Scenarios. We conduct experiments on Stag-Hunter Game, Quarry Game, and Afforestation Game (Figure 5.7) where off-beat action are introduced. The setting of them is simple but contains necessary elements for the research of MARL. It is challenging for agents to learn coordinative policies in them. We introduce them in the following.

- **Stag-Hunter Game.** As depicted in Figure 5.7, there are n agents whose action durations are different; the task is to catch the stag by shooting it simultaneously for all agents. Agents cannot move and the distance between the agent and the stag is different. The stag will escape when hit by $j \in \{1, \cdots, n-1\}$ arrows. In this case, agents will receive a positive reward given the number of arrows that successfully shoot the stag. In Section 5.5, the environmental dimension of the Stag-Hunter Game is 15×15 and the maximum number of time steps is 14. At each time step, each agent can only observe its position and the position of the stag. It cannot observe the position of other agents. Agents can select `SHOOT` or `NOOP` actions. `SHOOT` means shooting the arrow and `NOOP` means no actions to be executed. For agent 0, the action duration of its `SHOOT` action is 14 while the action duration of agent 1 `SHOOT` action is 6. When agent 1 shoots the arrow at time step 0, all agents will receive a positive reward at the end of the episode, making it challenging for TD learning to calculate the exact contribution of each agent.

- **Quarry Game.** There are n agents in a quarry, as shown in Figure 5.7. Agents' task is to complete the n-explosive installation task, and only when all the explosives detonate will agents receive the optimal positive reward. After the installation, agents should go to the safe zones. Otherwise, agents will die and receive a negative reward when the explosive detonates. Agents will receive a medium-level reward given the number of detonated explosives. The explosive has a different period to detonate after the installation. At each time step, each agent can observe its position, the position of the quarry, the position of the explosive set by the agent (if any), and the time seconds left for the explosive set by the agent (if any). The agent can select `MOVE_LEFT`, `MOVE_RIGHT`, `NOOP` or `INSTALL_EXPLOSIVE` actions at each time step. Note that each agent cannot observe the status of the other agents and others'

explosives. The episode ends after the maximum time steps or the explosive detonates. To complete the task, agents should place the explosive at the right time step and return to the safe zone.

- **Afforestation Game.** In Figure 5.7, there are n farmer agents in the farm. To the north of the farm, there is a desert. In the early spring, strong sandstorms may gust from the north and destroy the farm. In order to protect the farm, farm agents should plant trees in the north of the farm. Only when trees are tall enough can they protect the farm. Trees can have different durations to grow, making the off-beatness of the planting action. Agents receive the optimal positive reward when there are n trees that can protect the farm before the sandstorm. Note that agents have partial observations and will receive a reward of -0.1 at each time step in all scenarios shown in Figure 5.7. At each time step, each agent can observe its position, the position of the trees planted by itself, and the status (position and the age of the tree) of the sandstorm (if any). The agent cannot observe the other agents' positions and trees planted by other agents. Agents can take `MOVE_NORTH`, `MOVE_SOUTH`, `NOOP` or `PLANT_TREE` actions. Similar to Quarry Game and Stag-Hunter, agents should take the right action at the right time step to complete the task. At the last time step of the episode, the sandstorm will gust and if there are enough trees to protect the farm, the task will be a success. Note that agents should return to the safe zone when the sandstorm comes. Otherwise, agents will receive a fraction of the punishment. Agents will receive rewards when they fail to return to the safe zone. The reward is proportional to the number of agents who fail to return to the safe zone.

Training Settings. We implement our method on PyMARL [30] and use 10 random seeds to train each method on all environments. We use open-source code of baselines publicly by the corresponding authors on GitHub in all experiments. We list all baselines in Table 5.1, including the corresponding research questions to be answered. We use the default settings of PyMARL in our research, including the relay buffer, the mixing network, and the training hyperparameters. To explore, we use ϵ-greedy with ϵ annealed linearly from 1.0 to 0.05 over 50K time steps from the start of training and keep it constant for the rest of the training for all methods. The discount factor $\gamma = 0.99$ for all methods. We follow the optimal hyper-parameters used in the original papers of all methods in our experiments. We

TABLE 5.2: Hyper-parameters in our experiments.

Hyper-parameter	**Value**
Optimizer	RMSProp
Learning rate	5e-4
RMSProp alpha	0.99
RMSProp epsilon	0.00001
Gradient norm clip	10
Batch size	32
Replay buffer size	5,000
Exploration method	ϵ-greedy
ϵ-start	1.0
ϵ-finish	0.05
ϵ-anneal time	50,000 steps
γ	0.99
β	0.00001
Evaluation interval	10,000
Target update interval	200

carry out experiments on NVIDIA A100 Tensor Core GPU and NVIDIA GeForce RTX 3090 24G. We resort to mean-std values as our performance evaluation measurement. We use $\beta = 0.00001$. To create sub-graphs in LeGEM, we first calculate the episode return and keep 1 decimal of it. We then use this episode return to create each sub-graph. We list some important hyper-parameters in Table 5.2.

5.5.2 Experiment Results

We answer **Q1**. With LeGEM, MARL methods get enhanced performances as shown in Figures 5.8, 5.9 and 5.11. We are also interested in finding if LeGEM could reinforce the performance of simple methods. As depicted in Figure 5.10, with LeGEM, both VDN and QMIX outperform QPLEX, which is a state-of-the-art MARL method. We present the results of MARL methods on the Afforestation Game. In Figure 5.11, we can find that with LeGEM, all four methods get improved performance. We also compare QMIX-LeGEM and VDN-LeGEM with EMC, MAVEN, QPLEX, and RMIX. Despite the simple structure of VDN, VDN-LeGEM performs well and even outperforms QMIX-LeGEM, demonstrating comparable performance with RMIX as depicted in Figure 5.10. In the Afforestation Game, agents will receive a reward when they fail to return to the safe zone. The reward is proportional to the number of agents who fail to return to the safe zone.

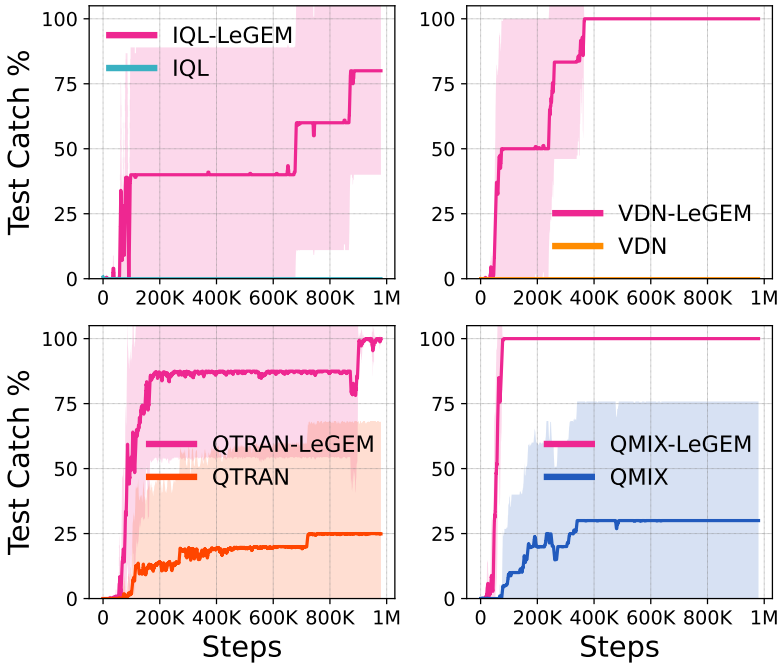


FIGURE 5.8: The test catch rate of our method vs MARL baselines in the Stag-Hunter Game.

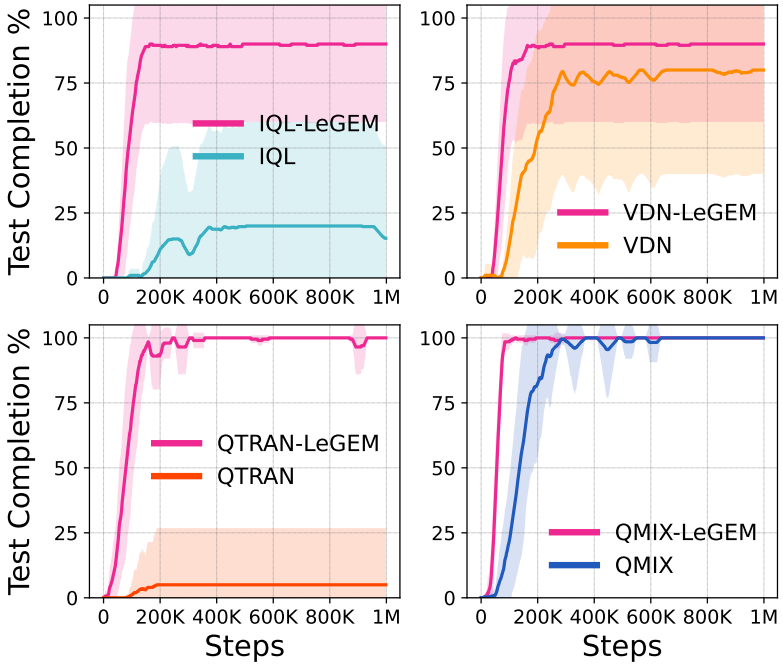


FIGURE 5.9: The test task completion rate of our method vs MARL baselines in Quarry Game.

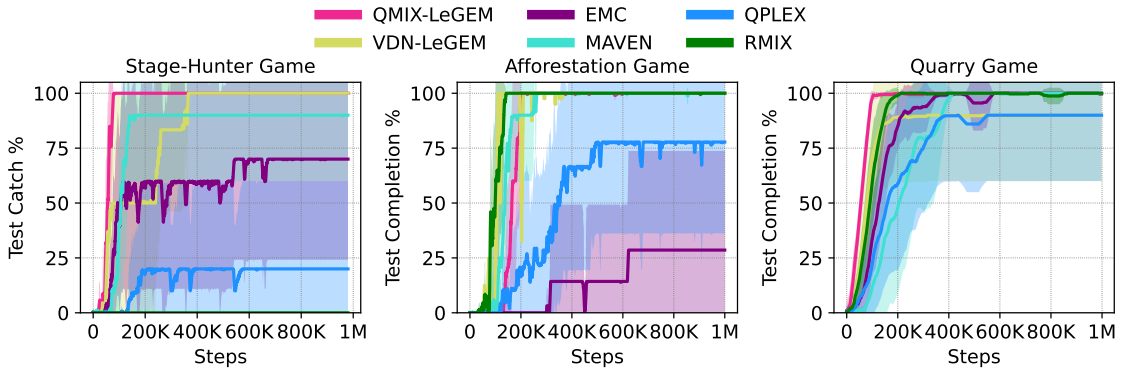


FIGURE 5.10: Performance of our method and other MARL methods in Stag-Hunt Game, Quarry Game, and Afforestation Game.

Such a clear and simple reward rule (*i.e.*, a "hint" for agents) makes learning much easier than that on Quarry and Stag-Hunter Game. This is the main reason why RMIX performs well. We can also find that MAVEN is also showing good performance due to its latent space learning model, which can efficiently learn the environment dynamics of the Afforestation Game. QMIX-LeGEM also shows good performance and it outperforms EMC and QPLEX.

We answer **Q2** by presenting the performance curves of EMC in Figure 5.10. EMC is an episodic memory MARL method with curiosity-driven exploration. It utilizes the episodic memory from RL [147]. With LeGEM, QMIX outperforms EMC. EMC performs poorly in the Stag-Hunter Game and Afforestation Game.

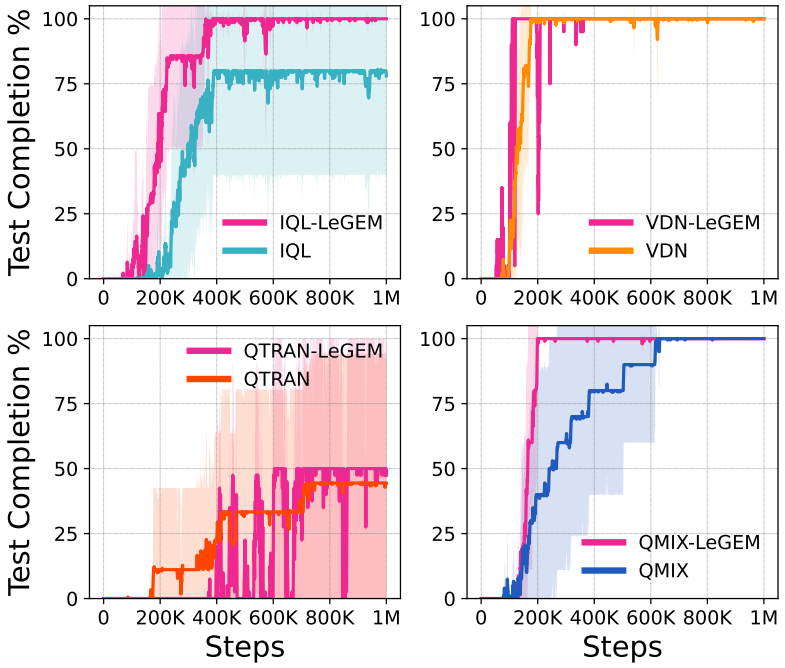


FIGURE 5.11: The test task completion rate of our method vs MARL baselines in Afforestation Game.

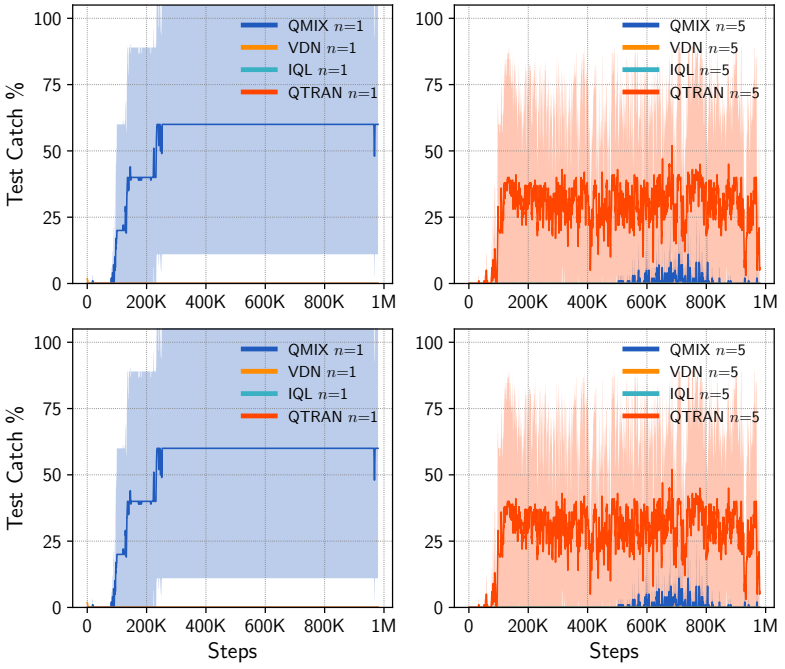


FIGURE 5.12: Results of QMIX, VDN, QTRAN and IQL with n-step return in Stag-Hunter Game.

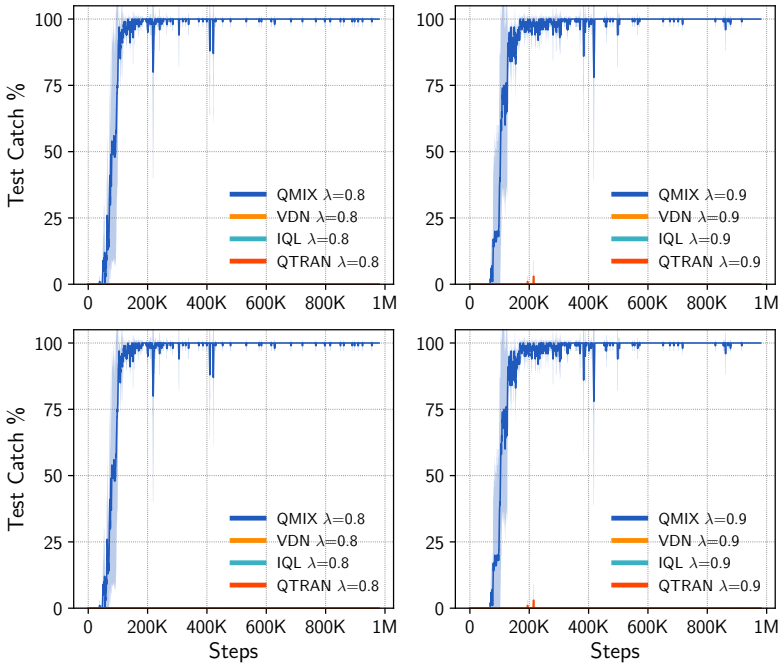


FIGURE 5.13: Results of QMIX, VDN, QTRAN and IQL with TD(λ) in Stag-Hunter Game.

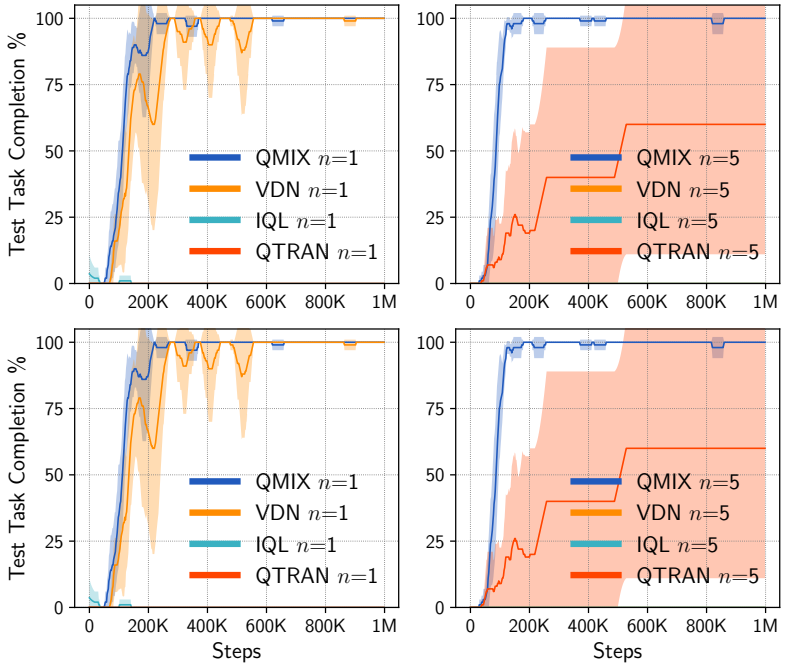


FIGURE 5.14: Results of QMIX, VDN, QTRAN and IQL with n-step return in Quarry Game.

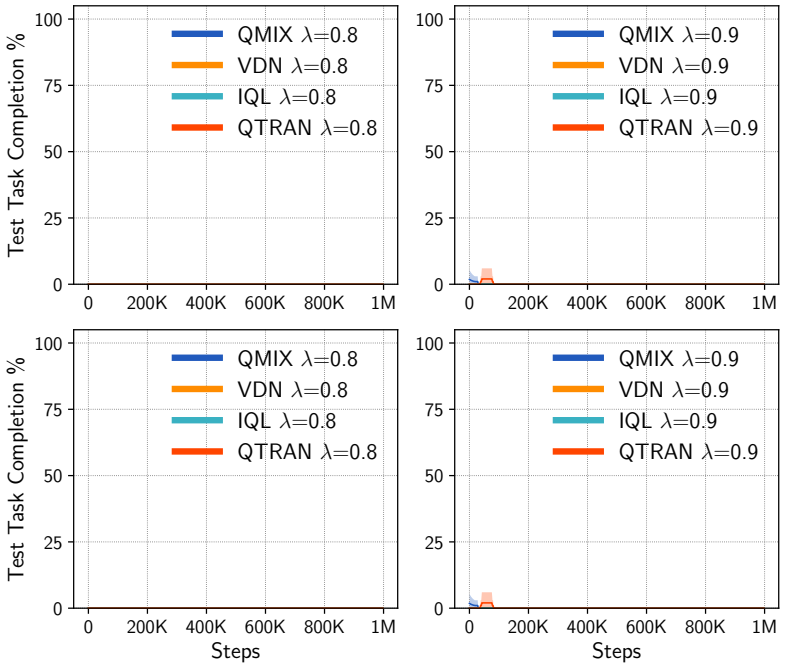


FIGURE 5.15: Results of QMIX, VDN, QTRAN and IQL with TD(λ) in Quarry Game.

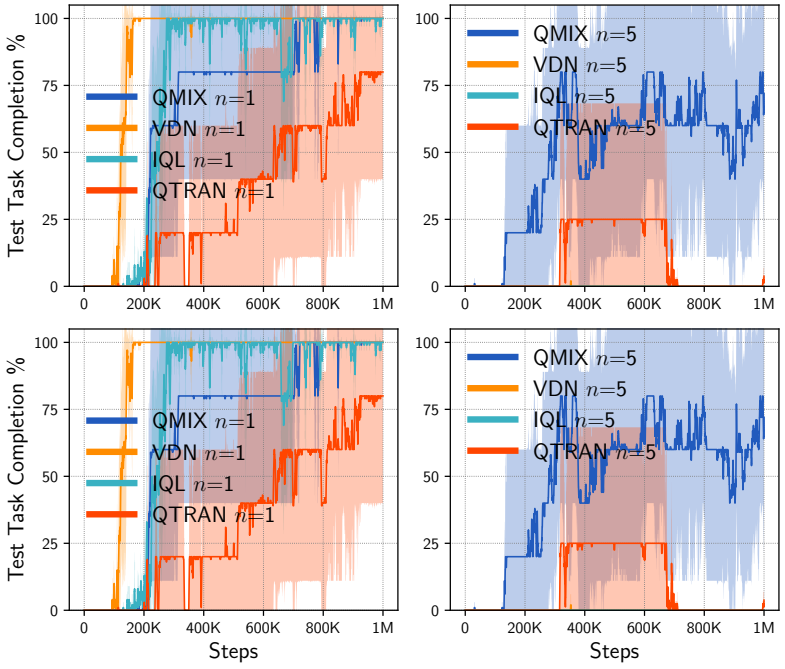


FIGURE 5.16: Results of QMIX, VDN, QTRAN and IQL with n-step return in Afforestation Game.

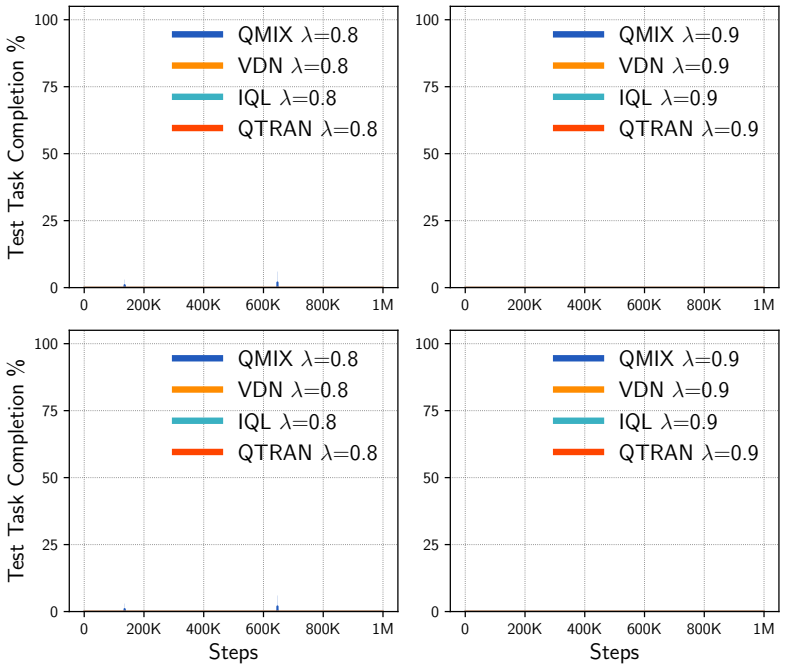


FIGURE 5.17: Results of QMIX, VDN, QTRAN and IQL with TD(λ) in Afforestation Game. All methods perform poorly and the test task completion rates are zero.

To answer **Q3**, we use n-step return and TD(λ) to estimate the TD-target. As shown in Table 5.3, with n-step return, both QMIX and VDN fail to learn good policies even with $n = 15$. Surprisingly, with TD(λ), QMIX can achieve good performance with $\lambda \in \{0.8, 0.9, 0.99, 1\}$ as shown in Table 5.4. However, we cannot find such an outcome in VDN and there is no guarantee of good results when using TD(λ). We provide additional experiment results on n-step return and TD(λ).

TABLE 5.3: Results (mean and std) of QMIX and VDN with n-step return in Stag-Hunter Game.

$n =$	1	5	10	15
QMIX	$60.0 \pm 40\%$	0 ± 0	0 ± 0	0 ± 0
VDN	0 ± 0	0 ± 0	0 ± 0	0 ± 0

TABLE 5.4: Results (mean and std) of QMIX and VDN with TD(λ) in Stag-Hunter Game.

$\lambda =$	0.8	0.9	0.99	1
QMIX	$100 \pm 0\%$	$100 \pm 0\%$	$89 \pm 10\%$	$61 \pm 37\%$
VDN	0 ± 0	0 ± 0	0 ± 0	0 ± 0

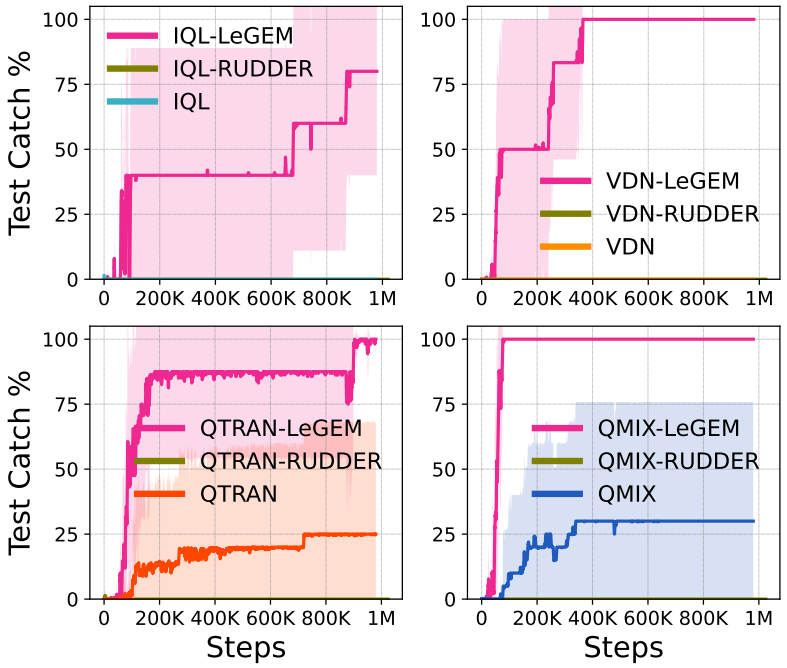


FIGURE 5.18: Results of MARL-RUDDER in Stag-Hunter Game.

As illustrated in Figures 5.12 and 5.13, QMIX can attain acceptable performance with some specific values of n and λ in Stag-Hunter Game. However, there are no convincing improvements in the performance of VDN, QTRAN, and IQL. On Quarry Game (Figures 5.14 and 5.15) and Afforestation Game (Figures 5.16 and 5.17), we can find that TD(λ) cannot help to improve the performance of MARL methods. We can conclude that n-step and TD(λ) have limited ability to improve the performance of MARL methods on OBMAS.

To answer **Q4**, we provide results of exploration methods of MARL and risk-sensitive MARL method. MAVEN utilizes mutual information to learn latent space for exploration and RMIX aims to learn risk-sensitive policies for MARL. In the Stag-Hunter Game, RMIX performs poorly as shown in Figure 5.10, mainly because the potential loss of reward is displaced by off-beat actions. Overall, MAVEN is stabler than EMC and RMIX. QMIX-LeGEM is stable in all scenarios and outperforms MAVEN. With LeGEM, even simple methods such as VDN can perform well and outperform many MARL methods with advanced components. Indeed, exploration in OBMAS is beneficial for multi-agent learning. However, the challenging temporal credit assignment issue cannot be easily addressed merely with exploration in OBMAS.

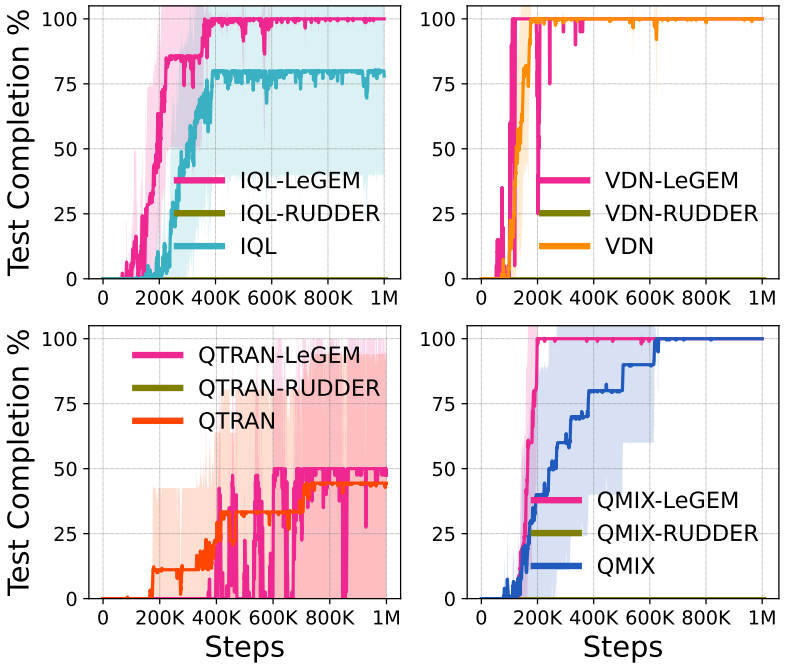


FIGURE 5.19: Results of MARL-RUDDER in Afforestation Game.

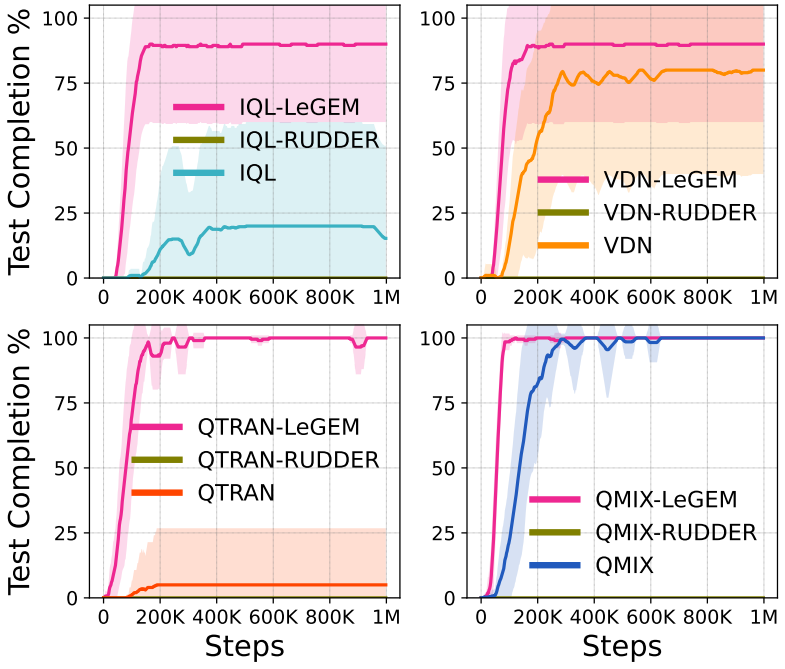


FIGURE 5.20: Results of MARL-RUDDER in Quarry Game.

TABLE 5.5: Results (mean and std) of RUDDER, QMIX-LeGEM and VDN-LeGEM in OBMAS.

	RUDDER	QMIX-LeGEM	VDN-LeGEM
Stag-Hunter Game	0 ± 0	$100 \pm 0\%$	$100 \pm 0\%$
Quarry Game	0 ± 0	$100 \pm 0\%$	$87 \pm 20\%$
Afforestation Game	0 ± 0	$100 \pm 0\%$	$100 \pm 0\%$

We answer **Q5**. RUDDER [140] is a popular method for reward shaping and long-term reward delay problems. Despite its merit, RUDDER performs poorly in OBMAS as summarized in Table 5.5. The main reason is that it redistributes the reward to the time step the reward was returned to the agents, and cannot estimate the pivot time step since the action durations are unknown to the agents. We present detailed results of QMIX-RUDDER, VDN-RUDDER, IQL-RUDDER, and QTRAN-RUDDER on Stag-Hunter Game, Quarry Game, and Afforestation Game via incorporating RUDDER [140] into QMIX, VDN, IQL and QTRAN. As rewards in off-beat Dec-POMDP are not episodic rewards [142], we convert rewards into episodic rewards by using total rewards of the episode as the reward of the last time step of the episode (rewards at other time steps in the episode are zero) in MARL-RUDDER methods. We use 10 random seeds for training each MARL method with RUDDER. On Stag-Hunter Game, Quarry Game, and Afforestation Game, QMIX-RUDDER, VDN-RUDDER, IQL-RUDDER and QTRAN-RUDDER all perform poorly as shown in Figures 5.18, 5.19 and 5.20. The performances are all zero. There are three reasons caused the poor performance: (i) RUDDER cannot capture the association between off-beat actions and off-beat rewards, making it challenging to detect the pivot time steps and redistribute the episodic reward to pivot time steps; (ii) RUDDER conducts the contribution analysis by estimating the reward of each time step via regression. Due to the sparsity of off-beat rewards and the estimation error of RUDDER, RUDDER redistributes the reward to time steps around the pivot time step, rendering the failure of TD learning; (iii) Our setting is a partially observable multi-agent setting, simply redistributing rewards without considering the multi-agent setting can redistribute the reward to the wrong time steps. In TD learning, the estimation of the TD target is essential to the learning of the policy (or the Q value). However, with off-beat actions, n-step return and TD(λ) fail in Off-Beat MARL as shown in the following figures. It is not surprising to see that RUDDER also fails in Off-Beat MARL.

5.6 Chapter Summary

This chapter investigates model-free MARL with off-beat actions. To address challenges in OBMAS, we first propose Off-Beat Dec-POMDP. Then, we propose a new class of episodic memory, LeGEM, for model-free MARL algorithms. LeGEM addresses the challenging temporal credit assignment problem raised by off-beat actions in TD-learning via the novel reward redistribution scheme. Empirical results show that our method significantly boosts multi-agent coordination and achieves leading performance as well as improved sample efficiency.

Searching from a graph-structured episodic memory takes much overhead in LeGEM, which is the limitation of our method. Scaling up LeGEM to complex OBMAS is our future direction. Recently, there has been a growing interest in model-based planning [84]. Leveraging LeGEM for model-based planning is also our future work. This chapter focuses on Dec-POMDP-based MARL methods. We leave it to future work to investigate off-beat actions in frameworks like Markov Game [70] and MMDP [157]. We are also interested in finding the merit of our method in real-world problems in our future work, such as scheduling [158] with off-beat settings.

Chapter 6

Learning to Generalize for MARL Agents

Despite the recent advancement in MARL, the MARL agents easily overfit the training environment and perform poorly in evaluation scenarios where other agents behave differently, which is the key issue of the successful deployment of MARL. Obtaining generalizable policies for MARL agents is thus necessary but challenging mainly due to complex multi-agent interactions. In this chapter[1], we model the MARL problem with Markov Games and propose a simple yet effective method, called ranked policy memory (RPM), *i.e.*, to maintain a look-up memory of policies to achieve good generalizability. The main idea of RPM is to train MARL policies via gathering massive multi-agent interaction data. In particular, we first rank each agent's policies by its training episode return, *i.e.*, the episode return of each agent in the training environment; we then save the ranked policies in the memory; when an episode starts, each agent can randomly select a policy from the RPM as the behavior policy. Each agent uses the behavior policy to gather multi-agent interaction data for MARL training. This innovative self-play framework guarantees the diversity of multi-agent interaction in the training data. Experimental results on Melting Pot demonstrate that RPM enables MARL agents to interact with unseen agents in multi-agent generalization evaluation scenarios and complete given tasks.

[1]The work in this chapter has been published in [6].

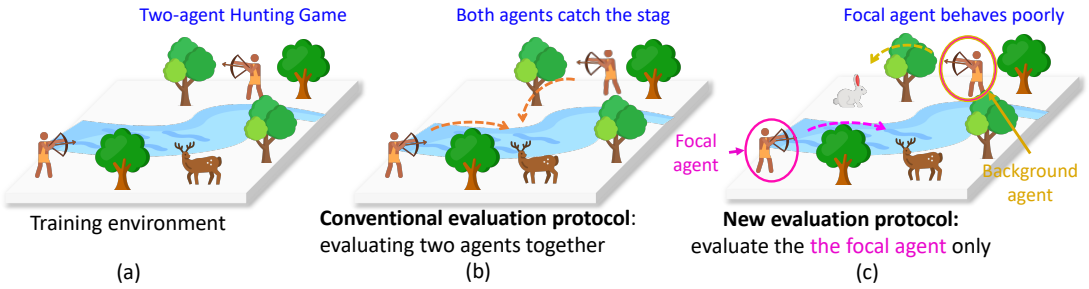


FIGURE 6.1: Two-Agent Hunting Game. (a) Training environment. Two agents (hunters) hunt in the environment. (b) After training in the training environment, all agents behave cooperatively to capture the stag. (c) In the new evaluation scenario, one agent is picked as the focal agent (in the magenta circle) and paired with a pre-trained agent (in the brown circle) that behaves in different ways to evaluate the performance of the selected agent. In conclusion, the conventional evaluation protocol fails to evaluate such behavior and current MARL methods easily fail to learn the optimal policy due to the lack of diversified multi-agent interaction data during training.

6.1 Introduction

In MARL [27], each agent acts decentrally and interacts with other agents to complete given tasks or achieve specified goals via reinforcement learning (RL) [29]. In recent years, much progress has been achieved in MARL research [17, 18, 47]. However, the MARL agents trained with current methods tend to suffer poor generalizability [159] in the new environments. The generalizability issue is critical to real-world MARL applications [3], but is mostly neglected in current research.

In this work, we aim to train MARL agents who can adapt to new scenarios where other agents' policies are unseen during training. We illustrate a two-agent hunting game as an example in Figure 6.1. The game's objective for two agents is to catch the stag together, as one agent acting alone cannot catch the stag and risks being killed. They may perform well in evaluation scenarios similar to the training environment, as shown in Figure 6.1 (a) and (b), respectively, but when evaluated in scenarios different from the training ones, these agents often fail. As shown in Figure 6.1 (c), the learning agent (called the focal agent following [3]) is supposed to work together with the other agent (called the background agent also following [3]) that is pre-trained and can capture the hare and the stag. In this case, the focal agent would fail to capture the stag without help from its teammate. The

teammate of the focal agent may be tempted to catch the hare alone and not cooperate, or may only choose to cooperate with the focal agent after capturing the hare. Thus, the focal agent should adapt to their teammate's behavior to catch the stag. However, the policy of the background agent is unseen to the focal agent during training. Therefore, without generalization, the agents trained as Figure 6.1 (left) cannot achieve an optimal policy in the new evaluation scenario.

Inspired by the fact that human learning is often accelerated by interacting with individuals of diverse skills and experiences [160, 161], we propose a novel method aimed at improving the generalization of MARL through the collection of diverse multi-agent interactions. Concretely, we first model the MARL problem with Markov Games [70] and then propose a simple yet effective method called ranked policy memory (RPM) to attain generalizable policies. The core idea of RPM is to maintain a look-up memory of policies during training for the agents. In particular, we first evaluate the trained agents' policies after each training update. We then rank the trained agents' policies by the training episode returns and save them in the memory. In this way, we obtain various levels, *i.e.*, the performance of the policies. When starting an episode, the agent can access the memory and load a randomly sampled policy to replace the current behavior policy. The new ensemble of policies enables the agents in self-play to collect diversified experiences in the training environment. These diversified experiences contain many novel multi-agent interactions that can enhance the extrapolation capacity of MARL, thus boosting the generalization performance. We note that an easy extension by incorporating different behavior properties as the keys in RPM could potentially further enrich the generalization but it is left for future work.

We implement RPM on top of the state-of-the-art MARL algorithm MAPPO [162]. To verify its effectiveness, we conduct large-scale experiments with the Melting Pot [3], which is a well-recognized benchmark for MARL generalization evaluation. The experiment results demonstrate that RPM significantly boosts the performance of generalized social behaviors and outperforms many baselines in a variety of multi-agent generalization evaluation scenarios.

6.2 Related Works

Recent advances in MARL [27, 163] have demonstrated its success in various complex multi-agent domains, including multi-agent coordination [25, 26, 36], real-time strategy (RTS) games [17, 18, 164], social dilemma [74, 165–167], multi-agent communication [168, 169], asynchronous multi-agent learning [8, 151], open-ended environment [170], autonomous systems [21, 171] and game theory equilibrium solving [47, 75]. Despite strides made in MARL, training generalizable behaviors in MARL is yet to be investigated.

Generalization in RL [172–175] has achieved much progress in domain adaptation [176] and procedurally generated environments [48, 177, 178] in recent years. However, there are few works of generalization in MARL domains [167, 179–181]. Recently, [167] proposed a hierarchical MARL method for agents to play against opponents it hasn't seen during training. However, the evaluation scenarios are only limited to simple competitive scenarios. [180] investigated the generalization in MARL empirically and proposed theoretical findings based on successor features [182, 183]. However, no technique to achieve generalization in MARL has been proposed in [180].

Several recent studies [184–186] have used population-based training to enhance multi-agent interaction by improving multi-agent diversity. Fictitious Co-Play (FCP) proposed in [184] aims to learn policies for a two-agent cooperative game. It is a two-stage method. In the first stage, n agents are trained independently with different random seeds via self-play. It needs n seeds and much more extra computation. In the second stage, an FCP agent is trained by interacting with the trained policies of n agents. Another extra run of training is also needed. However, our method, RPM, is an end-to-end training method for social dilemmas, competitive, cooperative, and even mixed environments with more than two agents. It needs only one run training (*i.e.*, end-to-end one step) by utilizing fictitious self-play to sample n policies for each agent without maintaining n populations of agents. Lupu et al. [185] considered the problem of zero-shot coordination and proposed a method to achieve diversity via the population-based training (PBT) method. In contrast, our work aims to achieve the generalization of coordination, competition, and social dilemmas in multi-agent systems via our novel ranked policy memory method. The proposed method in [185] cannot be applied to competition

and social dilemma scenarios. Besides that, PBT needs much more computation, which could be computationally expensive for large-scale multi-agent scenarios. Lupu et al. [185] also pointed out that "self-play (SP) agents control their own trajectory distribution during training, each policy typically only performs well on this exact distribution." We believe that is the key issue of the self-play method. The issue can be alleviated by introducing ranks, and each agent loads its previous policy for multi-agent self-play. The RPG [186] method is a population-based training method without self-play. It is highly dependent on the randomized reward function. For simple grid world scenarios in [186], conducting randomized reward function perturbations is not challenging. However, it is non-trivial to find proper reward function perturbations for complex scenarios.

Ad-hoc team building [187, 188] models the multi-agent problem as a single-agent learning task. In ad-hoc team building, one ad-hoc agent is trained by interacting with agents that have fixed pretrained policies, and the non-stationarity issue is not severe. However, in our formulation, non-stationarity is the main obstacle to MARL training. In addition, there is only one ad-hoc agent evaluated by interacting agents that are unseen during training, while there can be more than one focal agent in our formulation as defined in Definition 6.2, thus making our formulation general and challenging. There has been a growing interest in applying self-play to solve complex games [72, 189–191]; however, its value in enhancing the generalization of MARL agents has yet to be examined.

Meta-learning in MARL [192, 193] aims to address the non-stationarity issue in MARL, a well-known issue that has been extensively studied. To address the issue, the two works adopted the learning-to-learn framework. Typically, Al-Shedivat et al. [192] considers the problem of continuous adaptation in non-stationary environments where agents have few-shot interactions, *i.e.*, the agent must learn from only a limited amount of experience that it can collect before its environment changes. The MAML framework was used to address the issue. Kim et al. [193] proposed a novel meta-multiagent policy gradient theorem that directly accounts for the non-stationary policy dynamics inherent to multiagent learning settings. The proposed meta-multiagent policy gradient theorem explicitly models the learning procedure of the other agent (peer learning) in two-agent settings by considering the sequential dependence of the future parameters of other agents on the meta-agent i's parameter. While the previous work [192] ignored it. However, it would

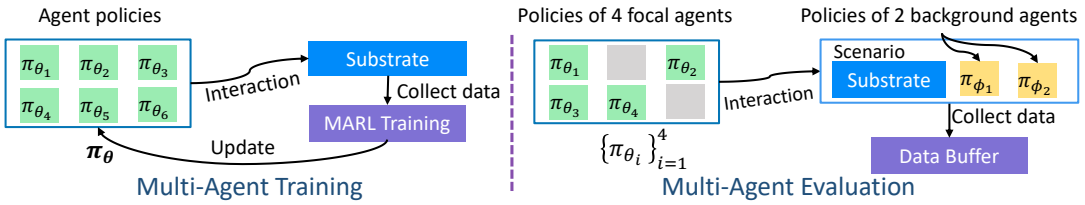


FIGURE 6.2: An example of our formulation. **Left:** All six agents' policies are trained with the MARL. **Right:** Two agents with policies π_{ϕ_1} and π_{ϕ_2} are picked as background agents, and the rest of the four agents (with new indices) are focal agents to be evaluated. The focal and the background agents constitute the evaluation scenario.

be difficult to train 3+ meta-agents simultaneously in complex scenarios due to the infinite recursion problem associated with the meta-learning framework in [193]. The two works consider two-agent settings. However, solving the non-stationarity issue in scenarios with more than two agents is more challenging than in two-agent settings. The performance of the two works in these scenarios is yet to be investigated. Furthermore, adapting the meta-learning framework in MARL to improve the generalization in MARL is non-trivial.

6.3 Problem Formulation

We introduce the formulation of MARL for training and evaluation of our problem. Our goal is to improve the generalizability of MARL policies in scenarios where the policies of agents or opponents are unseen during training while the physical environment is unchanged. Following [3], the training environment is defined as *substrate*. Each substrate is an N-agent partially observable Markov game $\mathcal{G}$. Each agent optimizes its policy π_{θ_i} via the following protocol.

Definition 6.1 (Multi-Agent Training)**.** There are N agents act in the substrate, which is denoted as $\mathcal{G}$. Each agent receives partial environmental observation not known to other agents and aims to optimize its policy π_{θ_i} by optimizing its accumulated rewards: $\sum_{t=0}^{\infty} \gamma^t r_t^i$. The performance of the joint policy $\boldsymbol{\pi_\theta} = \{\pi_{\theta_i}\}_{i=1}^N$ is measured by the mean individual return: $\bar{R}(\boldsymbol{\pi_\theta}) = \frac{1}{N}\sum_{i=1}^N R(\pi_{\theta_i}; \mathcal{G})$. $R(\pi_{\theta_i}; \mathcal{G})$ measures the episode return of policy π_{θ_i} in game $\mathcal{G}$ for agent i.

In order to evaluate the trained MARL policies in evaluation scenario $\mathcal{G}'$, we follow the evaluation protocol defined by [3]:

Definition 6.2 (Multi-Agent Evaluation)**.** There are M $(1 \leq M \leq N-1)$ focal agents that are selected from N agents. The focal agents are agents to be evaluated in evaluation scenarios. They are paired with $N-M$ background agents whose policies $\boldsymbol{\pi_\phi} = \{\pi_{\phi_j}\}_{j=1}^{N-M}$ were pre-trained with pseudo rewards in the same physical environment where the policies $\boldsymbol{\pi_\theta}$ are trained. To measure the generalized performance in evaluation scenarios, we use the mean individual return of focal agents as the performance measure: $\bar{R}(\{\pi_\theta\}_{i=1}^{M}) = \frac{1}{M}\sum_{i=1}^{M} R(\pi_{\theta_i}; \mathcal{G}')$.

We show an example of our formulation in Figure 6.2. Note that the focal agents cannot utilize the interaction data collected during evaluation to train or finetune their policies. Without training the policies of focal agents with the collected trajectories during evaluation, the focal agents should behave adaptively to interact with the background agents to complete challenging multi-agent tasks. It is also worth noting that the ad-hoc team building [187, 188] is different from our formulation both in the training and evaluation. We discuss the differences in the related works section (in Paragraph 3, Section 6.2).

In MARL, the focal agents need to adaptively interact with background agents to complete given tasks. Formally, we define the objective for optimizing the performance of the focal agents without exploiting their trajectories in the evaluation scenario for training the policies $\{\pi_{\theta_j}\}_{j=1}^{M}$:

$$\max \mathcal{J}(\{\pi_{\theta_j}\}_{j=1}^{M}) \triangleq \max \mathbb{E}_{s_{0:\infty}\sim\rho_{\mathcal{G}'}^{0:\infty}, a_{0:\infty}^{j}\sim\{\pi_{\theta_j}\}_{j=1}^{M}} \left[\sum_{t=0}^{\infty} \gamma^t \frac{1}{M} \sum_{j=1}^{M} r_t^j \middle| \mathcal{G}' \right]. \tag{6.1}$$

6.4 Solution Method: RPM

To improve the generalization of MARL, agents in the substrate must cover as much as multi-agent interactions, *i.e.*, data, that resemble the unseen multi-agent interactions in the evaluation scenario. However, current training paradigms, like independent learning [109] and centralized training and decentralized execution (CTDE) [31], cannot give diversified multi-agent interactions, as the agents' policies

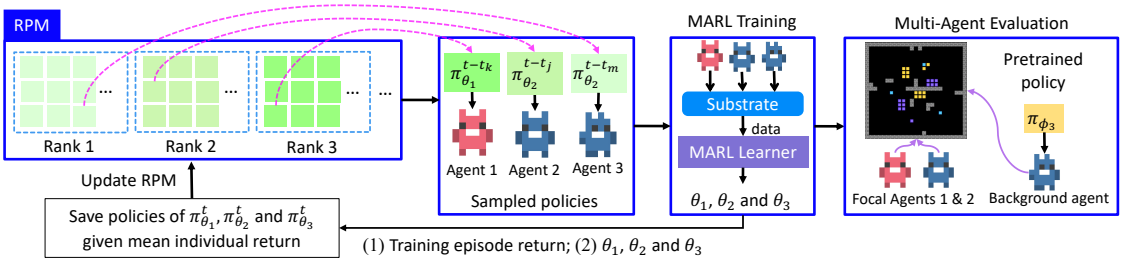


FIGURE 6.3: The workflow of RPM for a three-agent substrate. In the workflow, there are three agents in the substrate. Agent 3 is the background agent. Agents 1 and 2 are focal agents.

are trained at the same pace. To this end, we propose a Ranked Policy Memory (RPM) method to provide diversified multi-agent behaviors.

RPM Building & Updating. We denote an RPM with Ψ, which consists of $|R_{\max}|$ entries, *i.e.*, ranks, where $|R_{\max}|$ is the maximum training episode return (the episode return in the substrate). When an agent is acting in the substrate, it will receive the training episode return R of all agents with policies $\{\pi^i_\theta\}^N_{i=1}$. Then $\{\pi^i_\theta\}^N_{i=1}$ are saved into Ψ by appending agents' policies into the corresponding memory slot, $\Psi[r_e]$.`add`$(\{\pi^i_e\}^N_{i=1})$. To avoid there being too many entries in the policy memory caused by continuous episode return values, we discretize the training episode return. Each discretized entry κ covers a range of $[\kappa, \kappa+\psi)$, where $\psi > 0$, and it can be either an integer or a float number. For the training episode return R, the corresponding entry κ can be calculated by:

$$\kappa = \begin{cases} \lfloor R/\psi \rfloor \times \mathbf{1}\{(R \bmod \psi) \neq 0\} \times \psi, & \text{if } R \geq 0, \\ \lfloor R/\psi \rfloor \times \psi, & \text{otherwise.} \end{cases} \tag{6.2}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function, and $\lfloor \cdot \rfloor$ is the floor function. Intuitively, discretizing R saves memory and memorizes policies of similar performance into the same rank. Therefore, diversified policies can be saved to be sampled for agents.

RPM Sampling. The memory Ψ stores diversified policies with different levels of performance. We can sample various policies of different ranks and assign each policy to each agent in the substrate to collect multi-agent trajectories for training. These diversified multi-agent trajectories can resemble trajectories generated by the interaction with agents possessing unknown policies in the evaluation scenario. At the beginning of an episode, we first randomly sample N keys with replacements and then randomly sample one policy for each key from the corresponding list. All

agents' policies will be replaced with the newly sampled policies for multi-agent interactions in the substrate, thus generating diversified multi-agent trajectories.

Algorithm 10: MARL with RPM

1 **Input**: Initialize $\boldsymbol{\pi_\theta}$, Ψ, $\mathcal{D}$, $\mathcal{G}$ and $\mathcal{G}'$;
2 **Input**: Initialize behavior policy $\boldsymbol{\pi_{\theta_b}} \leftarrow \boldsymbol{\pi_\theta}$;
3 **for** each update **do**
4 **if** `RPM sampling` **then**
5 $\boldsymbol{\pi_{\theta_b}} \leftarrow$ `SamplingRPM(`Ψ`)`;
6 $\mathcal{D} \leftarrow$ `GatherTrajectories(`$\boldsymbol{\pi_{\theta_b}}, \mathcal{G}$`)`;
7 $\boldsymbol{\pi_\theta} \leftarrow$ `MARLTrainig(`$\boldsymbol{\pi_\theta}, \mathcal{D}$`)`;
8 $\Psi \leftarrow$ `UpdateRPM(`$\boldsymbol{\pi_\theta}, \Psi, \mathcal{G}$`)`;
9 $\bar{R} \leftarrow$ `Evaluate(`$\boldsymbol{\pi_\theta}, \mathcal{G}'$`)`;
10 $\boldsymbol{\pi_{\theta_b}} \leftarrow \boldsymbol{\pi_\theta}$;
11 **Output**: $\boldsymbol{\pi_\theta}$.

The Workflow of RPM. We showcase an example of the workflow of RPM in Figure 6.3. There are three agents in training. Agents sample policies from RPM. Then all agents collect data in the substrate for training. The training episode return is then used to update RPM. During the evaluation, agents 1 and 2 are selected as focal agents and agent 3 is selected as the background agent. We present the pseudo-code of MARL training with RPM in Algorithm 10. In Lines 4-5, the $\boldsymbol{\pi_{\theta_b}}$ is updated by sampling policies from RPM. Then, new trajectories of $\mathcal{D}$ are collected in line 6. $\boldsymbol{\pi_\theta}$ is trained in line 7 with MARL method by using the newly collected trajecotries and $\boldsymbol{\pi_{\theta_b}}$ is updated with the newly updated $\boldsymbol{\pi_\theta}$. RPM is updated in line 8. After that, the performance of $\boldsymbol{\pi_\theta}$ is evaluated in the evaluation scenario $\mathcal{G}'$ and the evaluation score $\bar{R}$ is returned in line 9.

Discussion. RPM leverages agents' previously trained models in substrates to cover as many patterns of multi-agent interactions as possible to achieve generalization of MARL agents when paired with agents with unseen policies in evaluation scenarios. It uses the self-play framework for data collection. Self-play [72, 189, 190, 194] maintains a memory of the opponent's previous policies for acquiring equilibria. RPM differs from other self-play methods in four aspects: (i) self-play utilizes the agent's previous policies to create fictitious opponents when the real opponents are not available. By playing with the fictitious opponents, many fictitious data are generated for training the agents. In RPM, agents load their previous policies to diversify the multi-agent interactions, such as multi-agent coordination and social dilemmas, and all agents' policies are trained by utilizing the diversified

multi-agent data. (ii) Self-play does not maintain explicit ranks for policies while RPM maintains ranks of policies. (iii) Self-play was not introduced for the generalization of MARL while RPM aims to improve the generalization of MARL. In Section 6.6, we also present the evaluation results of a self-play method.

6.5 MARL Training

We incorporate RPM into the MARL training pipeline. We take MAPPO [162] for instantiating our method, which is a multi-agent variant of PPO [24] and outperforms many MARL methods [26, 40, 110] in various complex multi-agent domains. In MAPPO, a central critic is maintained for utilizing the concealed information of agents to boost multi-agent learning due to non-stationarity. RPM introduces a novel method for agents to collect experiences/trajectories $\boldsymbol{\tau} = \{\tau_i\}_{i=1}^N$. Each agent optimizes the following objective:

$$\mathcal{J}(\theta_i) = \mathbb{E}\left[\texttt{min}\left(\eta_i^t\left(\theta_i^t\right)\cdot A_i^t, \texttt{clip}\left(\eta_i^t\left(\theta_i^t\right), 1-\epsilon, 1+\epsilon\right)\cdot A_i^t\right)\right], \tag{6.3}$$

where $\eta_i^t(\theta_i^t) = \frac{\pi_{\theta_i^t}(u_i^t|\tau_i^t)}{\pi_{\theta_i^{\text{old}}}(u_i^t|\tau_i^t)}$ denotes the important sampling weight. The $\texttt{clip}(\cdot)$ clips the values of θ^i that are outside the range $[1-\epsilon, 1+\epsilon]$ and ϵ is a hyperparameter. A_i^t is a generalized advantage estimator (GAE) [65]. To optimize the central critic $V_\psi(\{o_i^t, u_i^t\}_{i=1}^N)$, we mix agents' observation-action pairs and output an N-head vector where each value corresponds to the agent's value:

$$\mathcal{L}(\psi) := \mathbb{E}_{\mathcal{D}'\sim\mathcal{D}}\left[\left(y_t - V_{\bar{\psi}}(\{o_i^t, u_i^t\}_{i=1}^N)\right)^2\right], \tag{6.4}$$

where $y_t = \left[\sum_{l=0}^{k-1}\gamma^l r_i^{t+l} + \gamma^k V_{\bar{\psi}}(\{o_i^{t+k}, u_i^{t+k}\}_{i=1}^N)\texttt{[}i\texttt{]}\right]_{i=1}^N$ is a vector of k-step returns, and $\mathcal{D}'$ is a sample from the replay buffer $\mathcal{D}$. In complex scenarios, *e.g.*, Melting Pot, with an agent's observation as input, its action would not impact other agents' return, since the global states contain redundant information that deteriorates multi-agent learning. We present the whole training process, the network architectures of the agent, and the central critic in the following section.

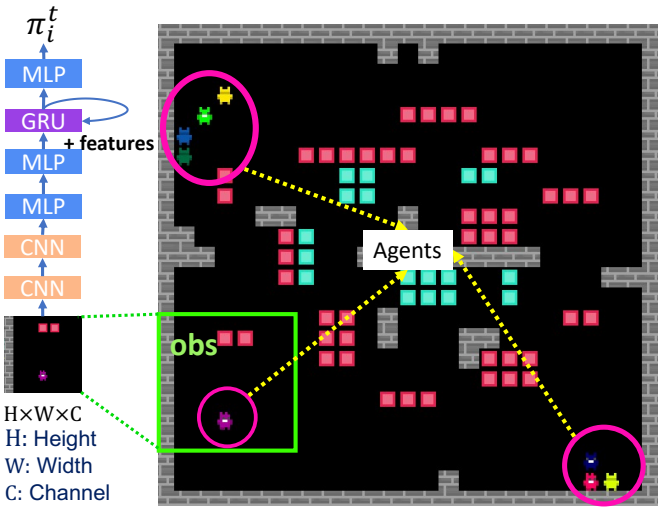


FIGURE 6.4: The neural network architecture of the policy of the agent is shown to the left. The green box to the lower left in the environment shows the agent's observation.

6.6 Experiments

In this section, to verify the effectiveness of RPM in improving the generalization of MARL, we conduct extensive experiments on Melting Pot and present the empirical results. We first introduce Melting Pot, baselines, and experiment setups. Then we present the main results of RPM. To demonstrate that ψ is important for RPM, we conducted ablation studies. We finally showcase a case study to visualize RPM. To sum up, we answer the following questions: **Q1**: Is RPM effective in boosting the generalization performance of MARL agents? **Q2**: How does the value of ψ impact RPM training? **Q3**: Does RPM gather diversified policies and trajectories?

6.6.1 Experimental Environment: Melting Pot

To demonstrate that RPM enables MARL agents to learn generalizable behaviors, we carry out extensive experiments on DeepMind's Melting Pot [3]. Melting Pot is a suite of testbeds for the generalization of MARL methods. It proposes a novel evaluation pipeline for the evaluation of the MARL method in various domains. That is, all MARL agents are trained in the substrate; during evaluation, some agents are selected as the focal agents and the rest agents become the background

agents (pre-trained policies of MARL models will be loaded); the evaluation scenarios share the same physical properties as the substrates. Melting Pot environments possess many properties, such as temporal coordination and free riding as depicted in Table 6.1. An agent performing well in such environments indicates that its behaviors exhibit these properties. In each substrate, episodes last 1000 or 2000 steps. The agent has a partial observation of the environment. Each observation has a size of 11×11 sprites where each sprite's size is 8×8 pixels. The agent is in the lower center of the observation. It can observe 9 rows in front of itself. It takes 1 row and 1 row left behind. There are 5 columns on either side of the agent. Thus, in RGB pixels, the size of each observation is $88 \times 88 \times 3$. All agents use RGB pixel representations as their inputs. In Figure 6.4, the agent's observation is shown in the green box to the lower left of the state (*i.e.*, the whole image). The agent is in the lower middle of the observation. The deep neural network architecture of the agent's policy is shown on the left. We introduce the neural network architecture design, MARL training and hyperparameters in the following.

We introduce substrates and evaluation scenarios used in the experiments. Agents' actions in all substrates and scenarios are: `forward`, `backward`, `strafe left`, `strafe right`, `turn left`, `turn right`. Unless otherwise stated, each episode lasts 1000 steps.

Chicken Game. In this environment, there are 8 agents in the substrate. Agents move around the environments[2] and collect resources of 2 different colors. Each agent carries an inventory $\rho = (\rho_1, \rho_2)$ with the count of resources collected since the last respawn. Due to partial observability, agents can only observe their inventory[3]. The more resources of a given type an agent picks up, the more committed the agent becomes to the pure strategy corresponding to that resource[4]. The agent can zap the other agent via its zapping beam for interaction. When an interaction occurs, a traditional matrix game is started. Here, in this environment, it is a Chicken Game [195] where both agents trying to exploit the other leads to the worst payoff, *i.e.*rewards, for both. Gathering red resources makes the agent's strategy towards committing 'hawk' while collecting the green resources pushes the agent toward

[2]There are two categories of environments: substrates and evaluation scenarios.

[3]It also applies to other environments where agents have inventories

[4]It also applies for other environments where matrix games should be resolved when two-agent interactions occur.

playing 'dove'. The payoff matrix for row and column players is:

$$\Phi_{\text{row}} = \Phi_{\text{col}}^{\text{T}} = \begin{bmatrix} 3 & 2 \\ 5 & 0 \end{bmatrix}.$$

Chicken Game (CG) 1. The task and the payoff matrix in this scenario are the same as in Chicken Game. In this scenario, one focal agent is joining seven background agents. Unlike the focal agent, which can play any strategy, the background agents were pretrained with pseudo rewards to play 'dove'. The best strategy for the focal agent is to commit to a 'hawk' strategy.

Chicken Game (CG) 2. The task and the payoff matrix in this scenario are the same as in Chicken Game. In this scenario, one focal agent joins seven background agents. The background agents were trained alongside agents that play pure 'hawk' and 'dove' strategies but were not given any pseudo rewards. They learned to play 'hawk' to defect other agents who were collecting dove resources during the interaction.

Chicken Game (CG) 3. The task and the payoff matrix in this scenario are the same as in Chicken Game. In this scenario, two focal agents join five background agents. The background agents play the 'dove' strategy unless and until they are defected on by a partner who commits a 'hawk'. Subsequently, they will commit 'hawk' in each encounter for the remainder of the episode.

Stag Hunt. Similar to Chicken Game, there are 8 agents in this environment. Each agent collects resources that represent 'hare' (in red color) or 'stag' (in green color) and compares inventories in an interaction, *i.e.*, encounter. The results of solving the encounter are the same as the classic Stag Hunt matrix game. In this environment, agents are facing tension between the reward for the team and the risk for the individual. The matrix for the interaction is:

$$\Phi_{\text{row}} = \Phi_{\text{col}}^{\text{T}} = \begin{bmatrix} 4 & 0 \\ 2 & 2 \end{bmatrix}.$$

Stag Hunt (SH) 1. In this environment, one agent interacts with seven pretrained agents. All background agents were trained to play the 'stag' strategy during the interaction. The optimal policy for the focal agent is also to play 'stag'.

Stag Hunt (SH) 2. In this environment, one agent interacts with seven pretrained agents. All background agents were trained to play the 'hare' strategy during the interaction. The optimal policy of the focal agent is also to play 'hare'.

Stag Hunt (SH) 3. In this environment, two agents interact with six pretrained agents. All background agents were trained to be reciprocators. They play stag. They are triggered to play hare when their interaction partners play hare for the remainder of the episode.

Clean Up. There are seven agents in the environment. Agents are rewarded (+1) for collecting apples. In the environment, there is an orchard and a river. Agents should clean the river frequently to reduce pollution for the irrigation of the orchard. The cleanliness of the river has an inverse effect on the rate at which apples grow in the orchard. When the cleanliness rate reaches a certain threshold, apples stop growing. Agents have the option to take the `clean` action, which results in the removal of a small quantity of pollution around the agent from the river. So, agents should move to clean the river without any rewards. Consequently, agents should maintain the public good of orchard regrowth by cleaning the river. This dynamic creates a conflict between the short-term individual incentive for agents to maximize their rewards by remaining in the orchard and the long-term collective interest in maintaining a clean river.

Clean Up (CU) 1. In this evaluation scenario, three focal agents join four background agents. All background agents have been trained to behave altruistically, *i.e.*, always cleaning the river without consuming apples. Thus, the optimal policy for the focal agent is to collect as many apples as possible without moving out of the orchard to clean the river.

Clean Up (CU) 2. In this evaluation scenario, three focal agents join four background agents. All background agents start out cleaning in the first 250 steps. They alternate cleaning with eating every 250 steps. Focal agents should learn to take turns with the background agents.

Pure Coordination. In this environment, eight agents cannot be identified as individuals because they all have the same appearance. Agents collect resources of three different colors. So, the size of the agent's inventory is 3. To maximize the reward, all agents should collect the same colored resource when the encounter

occurs. The matrix for the interaction is:

$$\Phi_{\text{row}} = \Phi_{\text{col}}^{\text{T}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Pure Coordination (PC) 1. In this evaluation scenario, there are seven focal agents and one background agent. The background agent has been trained to target one particular resource out of three colors of resources. Focal agents should observe other agents to see the resources other agents are collecting and then decide the right color to pick. This scenario aims to evaluate whether agents' coordination is not disrupted by the presence of unfamiliar/unknown other agents who have a special preference for one particular colored resource.

Pure Coordination (PC) 2. In this evaluation scenario, there is one focal agent and seven background agents. The background agents target resource B.

Pure Coordination (PC) 3. In this evaluation scenario, there is one focal agent and seven background agents. The background agents target resource C.

Prisoners' Dilemma. Eight agents collect colored resources that represent 'defect' (red) or 'cooperate' (green). Agents compare their inventories in an encounter where a classic Prisoner's Dilemma matrix game is resolved. Agents face a tension between the reward for the group and the reward for the individual. The matrix for the interaction is:

$$\Phi_{\text{row}} = \Phi_{\text{col}}^{\text{T}} = \begin{bmatrix} 3 & 0 \\ 4 & 1 \end{bmatrix}.$$

Prisoners Dilemma (PD) 1. In this evaluation scenario, one focal agent joins seven background agents. All background agents will play cooperative strategies, *i.e.*, collecting 'cooperate' resources and rarely collecting 'defect'). The optimal policy for the focal agent is to identify such a pattern and then collect 'defect' resources.

Prisoners Dilemma (PD) 2. In this evaluation scenario, one focal agent joins seven background agents. The background agents are conditional cooperators. They collect 'cooperate' resources and cooperate with interaction partners. They stop cooperative strategies when they have been defected by their partners twice.

After that, they collect 'defect' resources and strike back for the remainder of the episode.

Prisoners Dilemma (PD) 3. In this evaluation scenario, one focal agent joins seven background agents. The background agents are conditional cooperators. They collect 'cooperate' resources and cooperate with interaction partners. They stop cooperative strategies when they have been defected by their partners twice. After that, they collect 'defect' resources and strike back for the remainder of the episode. The optimal strategy for the focal agents is to cooperate when the episode nearly ends and then defect only once as there is no time for the background agents to revenge.

Rational Coordination. The environment setting is the same as Pure Coordination, except that different colored resources are of different values. Agents should find the optimal color to maximize the group reward. The matrix for the interaction is:

$$\Phi_{\text{row}} = \Phi_{\text{col}}^{\text{T}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 2 & 3 \end{bmatrix}.$$

Rational Coordination (RC) 1. In this evaluation scenario, there are seven focal agents and one background agent. The background agent has been trained to target one particular resource out of three colors of resources. This scenario is similar to **Pure Coordination 1** since it aims to evaluate that agents' coordination is not disrupted by the presence of unfamiliar other agents who have a special preference for one particular colored resource. However, this scenario is more challenging than **Pure Coordination 1**. While focal agents' choices are better than miscoordination, some choices are better than coordinating for the focal agents.

Rational Coordination (RC) 2. In this evaluation scenario, there are seven focal agents and one background agent. The background agent has been trained to target resource A. Because both resources B and C are better than resource A for every agent, it is not rational for all agents to coordinate on resource A.

Rational Coordination (RC) 3. In this evaluation scenario, there are four focal agents and four background agents. The background agent has been trained to target different resources, making them uncoordinated. In this case, it is rational

that the focal agents should learn to observe partners' preferences and interact with familiar individuals who collect resources C.

6.6.2 Baselines

We introduce baselines trained and evaluated in the experiment in detail. Baselines are MAPPO [162], MAA2C [196], OPRE [167], RandNet [48] and HFSP [72, 189].

MAPPO. MAAPO is an extension of PPO [24] for multi-agent RL. Following the CTDE [31] training and execution paradigm, agents take actions independently during execution and agents' policies are trained via sharing information (*e.g.*,) with other agents. In MAPPO, there are N policies $\{\pi_i\}_{i=1}^N$ for each agent i. A central critic is maintained by feeding all agents' observations and actions $\{o_i^t, u_i^t\}_{i=1}^N$. Although the global state contains all agents' observations, it contains redundant information that deteriorates the central critic learning with TD-learning [46]. Note that all baselines that have a central critic takes all agents' observations and actions $\{o_i^t, u_i^t\}_{i=1}^N$ as the input.

MAA2C. MAA2C is a multi-agent RL variant of A2C [197]. MAA2C adopts the same training and execution paradigm used in MAPPO. Similar to A2C, TD error is used as the advantage in MAA2C for training agents' policies via maximizing policy gradient loss.

OPRE. We build OPRE [167] on top of MAPPO. The key idea behind OPRE is to re-use the same latent space to factorize the policy via creating a hierarchical policy structure:

$$\pi_i(u_i|o_i^{\leq t}, o') = \sum_z q(z|o')\eta(u_i|o_i^{\leq t}, z)$$

where o' is $\{o_i^t, u_i^t\}_{i=1}^N$ and $\eta(u_i|o_i^{\leq t}, z)$ is a mixture component of the policy, *i.e.*, an option. $o_i^{\leq t}$ can be represented via recurrent neural networks [1, 198]. Note that the 'option' here differs from the *option* in hierarchical RL [137, 152]. In OPRE, the option has no explicit probability distribution of entering an option and no explicit probability distribution of exiting the current option. The behavior policy is defined as:

$$\mu_i(o_i^{\leq t}) = \sum_z p(z|o')\eta(u_i|o_i^{\leq t}, z)$$

Then $p(z|o')$ can be trained via $\texttt{KL}(q||p)$ together with the policy and the central critic in an end-to-end manner. We use the default hyperparameters used in OPRE in our experiments. The number of options is 16.

RandNet. Lee et al. [48] proposed RandNet for improving the generalization of RL in unseen environments, especially environments with new textures and layouts. RandNet utilizes a single-layer convolutional neural network (CNN) as a random network, where its output has the same dimension as the input. To reinitialize the parameters of the random network, RandNet utilizes the following mixture of distributions: $P(\phi) = \alpha\mathbb{I}(\phi = \mathbf{I}) + (1-\alpha)\mathcal{N}\left(\mathbf{0}; \sqrt{\frac{2}{n_{\text{in}}+n_{\text{out}}}}\right)$ where $\mathbf{I}$ is an identity kernel, $\alpha \in [0, 1]$ is a positive constant, $\mathcal{N}$ stands for the normal distribution. n_{in} and n_{out} are the number of input and output channels, respectively. We use RandNet in the policy network and the critic network of MAPPO. We use the default hyperparameters used in RandNet in our experiments.

HFSP. Self-play [72, 189, 190, 194] has been studied for obtaining equilibria via creating fictitious plays by sampling agents' past policies. HFSP is a heuristic fictitious self-play method. HFSP uses the MARL framework of MAPPO. Like RPM, it maintains a memory to save all the policies after each training step. HFSP agents have a probability of 0.7 to sample the lasted policies and a probability of 0.3 to sample previous policies. RPM can be considered as a ranked self-play by sampling policies with a hierarchy.

6.6.3 Architectures

We first introduce the neural network architecture of the policy, the critic and the training pipeline for all methods, and the hyperparameters used in the neural network architectures. RPM and all baselines use the same network architecture.

Actor Network. The actor network consists of a convolutional neural network (CNN) with two layers. The two CNN layers use the ReLU activation function. The first and the second layer have 16 and 32 output channels, 8 and 4 kernel shapes, and 8 and 1 strides, respectively. An MLP follows the two CNN layers with two layers with 64 neurons each. The MLP uses the ReLU action function. It is then followed by a GRU [198] with 128 units. The input of the GRU is the concatenation of the output of the MLP and the features (such as the agent's

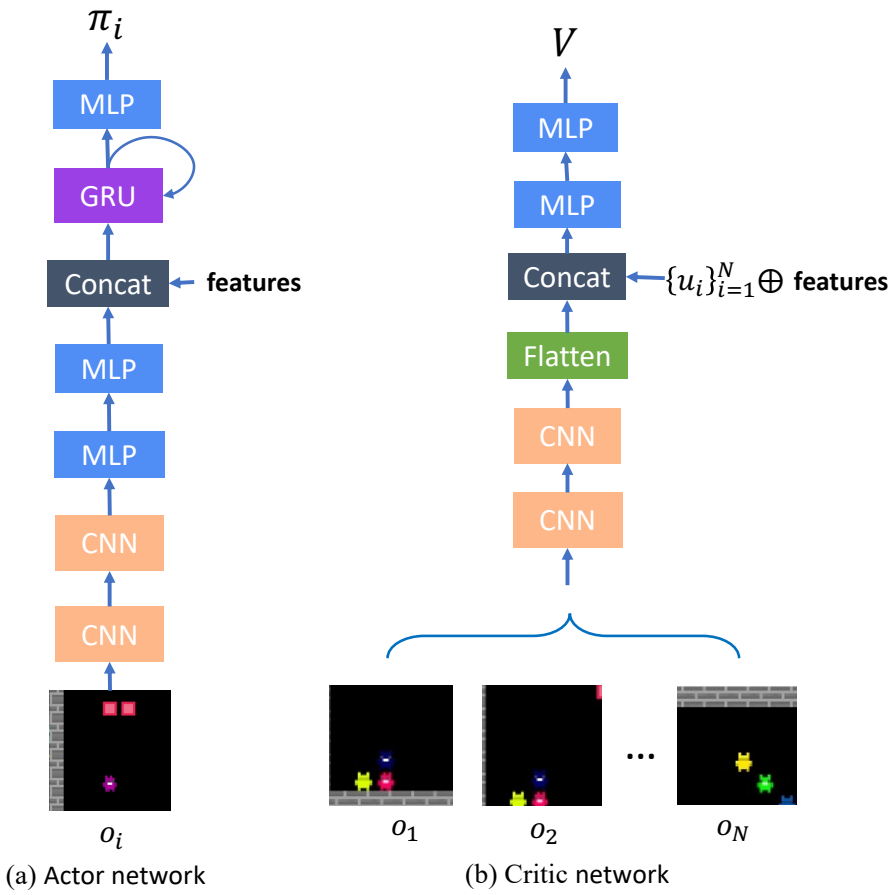


FIGURE 6.5: The networks of the policy (left) and the critic (right).

position, orientation, and inventory). The output of the GRU is fed into the MLP, and it outputs the policy π_i for agent i.

Critic Network. The critic network is shared by all agents. The critic network consists of a CNN with two layers. The two CNN layers use the ReLU activation function. The first and the second layer have 16 and 32 output channels, 8 and 4 kernel shapes, and 8 and 1 strides, respectively. The CNN is then followed by a concatenation of all agents' actions and features (such as the agent's position, orientation, and inventory). The concatenation is then fed into an MLP with two layers with 64 neurons. The MLP uses the ReLU action function. The MLP outputs the value, a vector with the dimension of N. We take all agents' observations as a batch and feed them into the CNN. We then flatten the CNN's output and feed it with agents' actions and features as inputs to the MLP network to get the value vector for all agents.

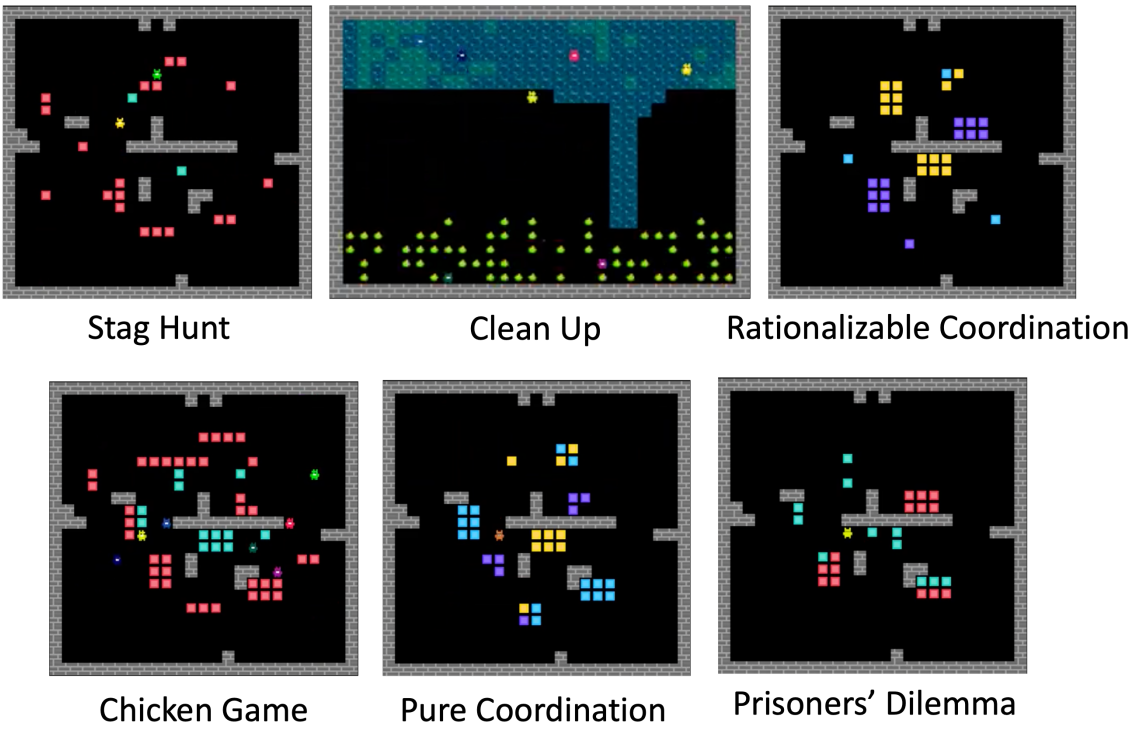


FIGURE 6.6: Melting Pot environments.

6.6.4 Training Settings

We implement our method with Python and PyTorch. The learner is implemented with EPyMARL [196] and the actors that collect experiments are implemented with Ray [199]. We train agents in Melting Pot substrates for 200 million frames with 3 random seeds for RPM and 4 seeds for baselines. We randomly sample policies from RPM. The discount factor $\gamma = 0.99$ and we follow the default hyper-parameters used in the original papers of all methods in our research. We carry out experiments on NVIDIA A100 Tensor Core GPU. We resort to mean-std values as our performance evaluation measurement. We use Adam as our optimizer. We list some important hyper-parameters in Table 6.2. We list values of ψ used in our experiments in Table 6.3.

Baselines. Our baselines are MAPPO [162], MAA2C [196], OPRE [167], heuristic fictitious self-play (HFSP) [164, 200] and RandNet [48]. MAPPO and MAA2C are MARL methods that achieved outstanding performance in various multi-agent scenarios [196]. OPRE was proposed for the generalization of MARL. RandNet is a general method for the generalization of RL by introducing a novel component in

TABLE 6.1: Properties of Melting Pot environments. The first column shows the properties and the first row lists environments. ✓ mark indicates the environment possessing the corresponding property while ✗ mark stands for the environment that does not own the corresponding property. SH stands for Stag Hunt. PC stands for Pure Coordination. CU is Clean Up. PD stands for Prisoners' Dilemma. RC is Rational Coordination. CG stands for Chicken Game.

	SH	**PC**	**CU**	**PD**	**RC**	**CG**
Temporal Coordination	✗	✗	✓	✗	✗	✗
Reciprocity	✓	✓	✓	✓	✗	✓
Deception	✓	✗	✓	✓	✗	✓
Fair Resource Sharing	✗	✗	✓	✗	✗	✗
Convention Following	✓	✓	✓	✗	✓	✓
Task Partitioning	✗	✗	✓	✓	✗	✗
Trust & Partnership	✓	✗	✗	✗	✗	✓
Free Riding	✗	✗	✓	✗	✗	✗

TABLE 6.2: Hyper-parameters in our experiments.

hyper-parameter	**Value**
Optimizer	Adam
Learning rate	1e-4
Adam betas	$(0.9, 0.999)$
Adam epsilon	1e-8
Adam weight decay	0
Gradient norm clip	10
Batch size	60
Replay buffer size	600
γ	0.99
Evaluation interval	1,000
Target update interval	200
p	0.5

the convolutional neural network. HFSP is a general self-play method for obtaining equilibria in competitive games, we use it by using the policies saved by RPM.

Training Setup. We use 6 representative substrates (Figure 6.6) to train MARL policies and choose some evaluation scenarios from each substrate as our evaluation testbed. The properties of the environments are listed in Table 6.1. We train agents in Melting Pot substrates for 200 million frames with 3 random seeds for all methods. Our training framework is distributed with 30 CPU actors to collect experiences and 1 GPU for the learner to learn policies, similar to the framework

TABLE 6.3: The value of ψ

Melting Pot Substrate	**The value of ψ**
Stag Hunt	1
Pure Coordination	0.01
Clean Up	1
Prisoners' Dilemma	0.02
Rational Coordination	0.2
Chicken Game	1

used in IMPALA [130]. To improve the efficiency and save memory, we use parameter sharing [26, 40, 162], *i.e.*, all agents share a policy network. We adopt the CTDE framework to train the policies and the critic. We implement our actors with Ray [199] and the learner with EPyMARL [196]. We use mean-std to measure the performance of all methods. The bold lines in all figures are mean values, and the shades stand for the standard deviation. Due to a limited computation budget, it is redundant to compare our method with other methods, such as QMIX [26] and MADDPG [25] as MAPPO outperforms them. All experiments are conducted on NVIDIA A100 GPUs.

6.6.5 Experiment Results

To answer **Q1**, we present the evaluation results of 17 Melting Pot evaluation scenarios in Figure 6.7. Our method can boost MARL in various evaluation scenarios, which have different properties, as shown in Table 6.1. In Chicken Game (CG) 1-2 (the number stands for the number of the evaluation scenario of Chicken Game), RPM outperforms its counterparts by a convincing margin. HFSP performs no better than RPM. RandNet gets around 15 evaluation mean returns on Chicken Game (CG) 1. MAA2C and OPRE perform nearly random (the red dashed lines indicate the random result) in the two scenarios. In Pure Coordination (PC) 1-3, Rational Coordination (PC) 1-3 and Prisoners' Dilemma (PD) 1-3, most baselines perform poorly. In Stag Hunt (SH) 1-3 and Clean Up (CU) 1-2, MAPPO and MAA2C perform unsatisfactorily. We can also find that HFSP even gets competitive performance in Stag Hunt (SH) 1-3. However, HFSP performs poorly in Pure Coordination (PC) 1-3, Rational Coordination (RC) 1-3, and Prisoners' Dilemma (PD) 1-3. Therefore, the vanilla self-play method cannot directly be applied to

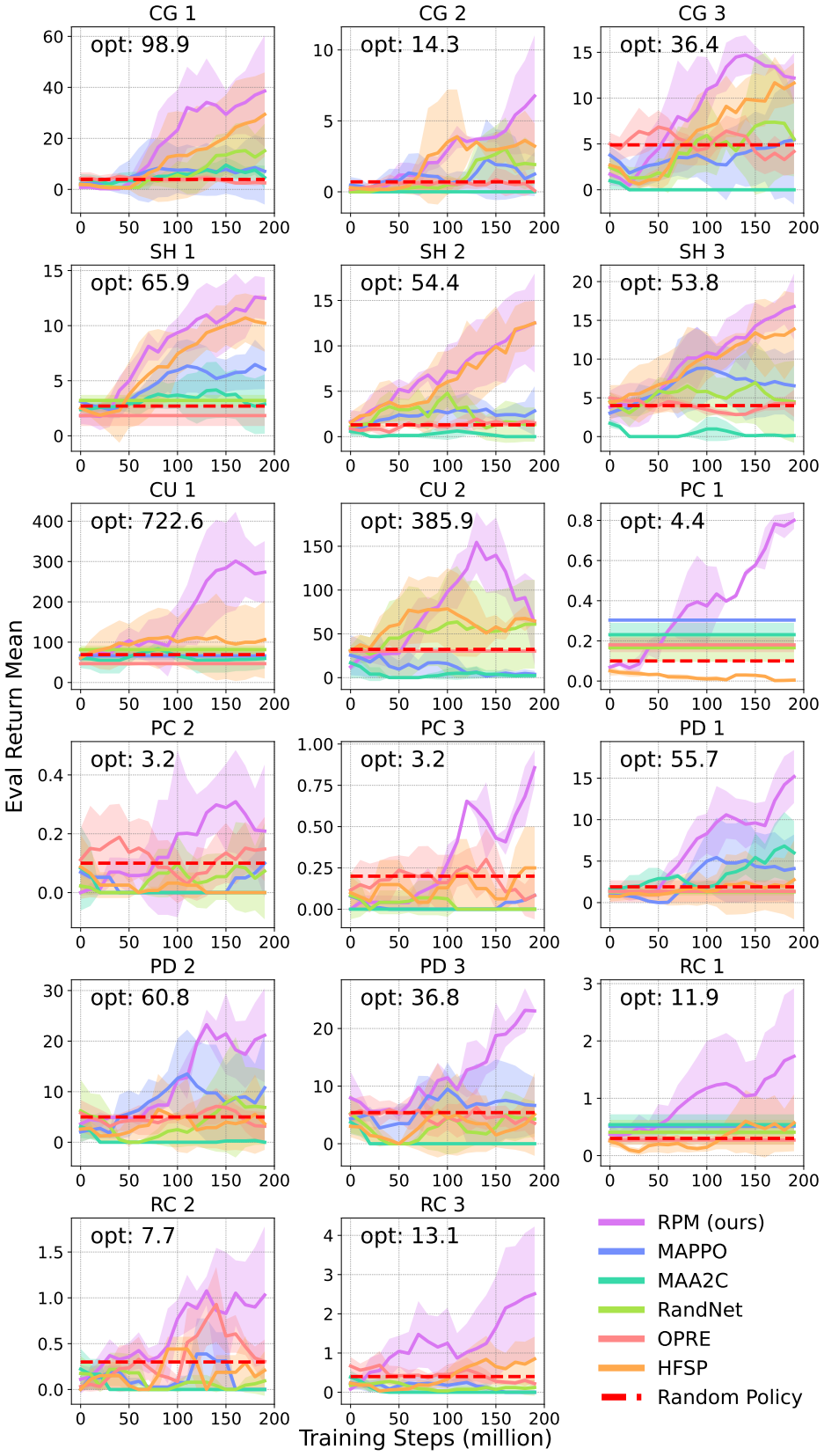


FIGURE 6.7: Evaluation results of RPM and baselines in 17 scenarios. The red dash horizontal lines indicate the results of random policy. The optimal (opt) values are shown in each sub-figure and were gathered from [3], which an exploiter generated. The exploiter was trained in the evaluation scenarios with RL methods, and the training time steps were 1,000 M.

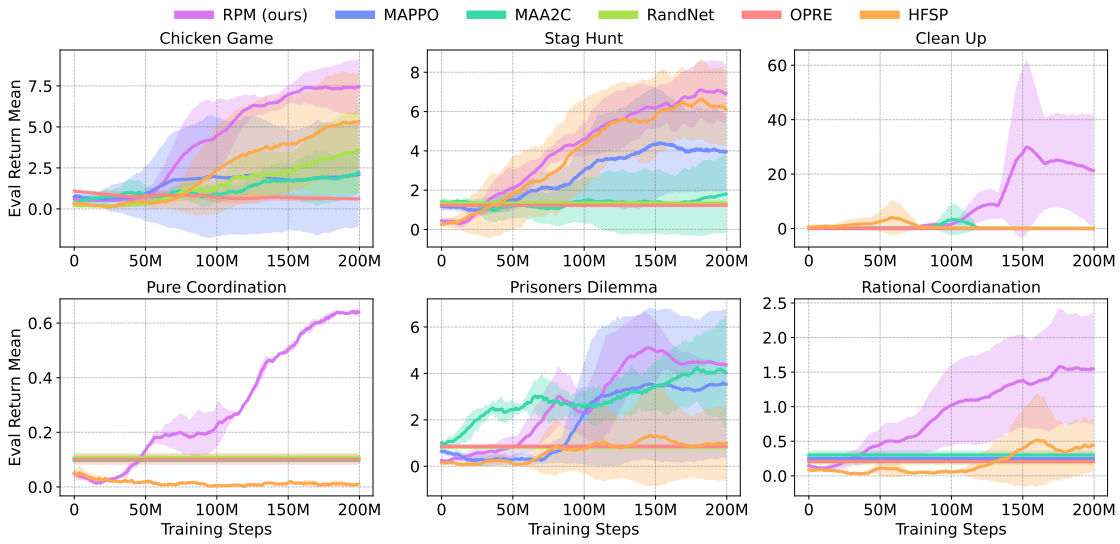


FIGURE 6.8: Episode return in substrates.

improve the generalization of MARL methods. In summary, RPM boosts the performance of MAPPO in all evaluation scenarios. We depict the episode return within the substrate. During training, the MARL methods are evaluated in the substrate. Figure 6.8 demonstrates that despite the environments being distinct, RPM also demonstrates leading performance. Once the agents in the substrate achieve a satisfactory episode return, the trained policy will be saved at the appropriate rank. In turn, it improves the performance of RPM in the evaluation scenario by collecting diverse data on multi-agent interactions.

To answer **Q2**, we present results of the impact of ψ and the sampling ratio in HFSP in the following.

6.6.6 Ablation Study

The Impact of ψ. To investigate which value of ψ has the greatest impact on RPM performance, we conduct ablation studies by (i) removing ranks and sampling from the checkpoint directly; and (ii) reducing the number of ranks by changing the value of ψ. As shown in Figure 6.10, without ranks (sampling policies without ranks randomly), RPM cannot attain stable performance in some evaluation scenarios. Especially in Pure Coordination (PC) 1-3, the result is low and has a large variance. In RPM, choosing the right interval ψ can improve the performance, as shown in the

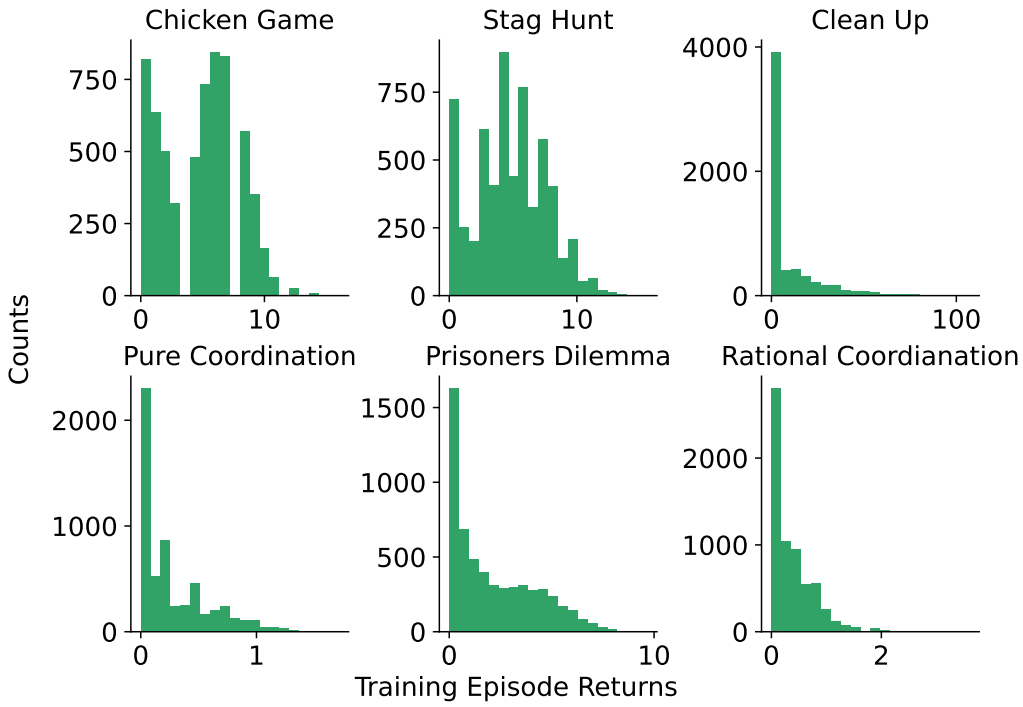


FIGURE 6.9: Histograms of training episode returns. The long-tail patterns are found in CU, PC, PD, and RC. The histograms of CG and SH are skews. We can conclude that policies that have higher counts but with lower returns can be sampled more often in self-play, leading to the poor performance of generalization.

TABLE 6.4: Ablation study: the averaged value of the last three evaluation episode returns. Curves are in Figure 6.10.

Eval Scenarios	**RPM**	**Random**	**Types of ψ**		
			1	2	3
Pure Coordination 1	0.78	0.18	0.33	0.39	0.42
Pure Coordination 2	0.23	0.16	0.24	0.17	0.27
Pure Coordination 3	0.70	0.19	0.37	0.33	0.42
Prisoners' Dilemma 1	13.90	10.11	10.70	8.70	3.20
Prisoners' Dilemma 2	19.60	10.41	13.76	17.74	14.96
Prisoners' Dilemma 3	22.31	10.28	19.80	11.74	9.76

TABLE 6.5: ψ values. ψ^* indicates the values of ψ used to get results in Figure 6.7.

Eval Scenarios	ψ^*	**Types of ψ**		
		1	2	3
Pure Coordination 1	0.01	0.1	0.5	1
Pure Coordination 2	0.01	0.1	0.5	1
Pure Coordination 3	0.01	0.1	0.5	1
Prisoners' Dilemma 1	0.02	0.2	1	5
Prisoners' Dilemma 2	0.02	0.2	1	5
Prisoners' Dilemma 3	0.02	0.2	1	5

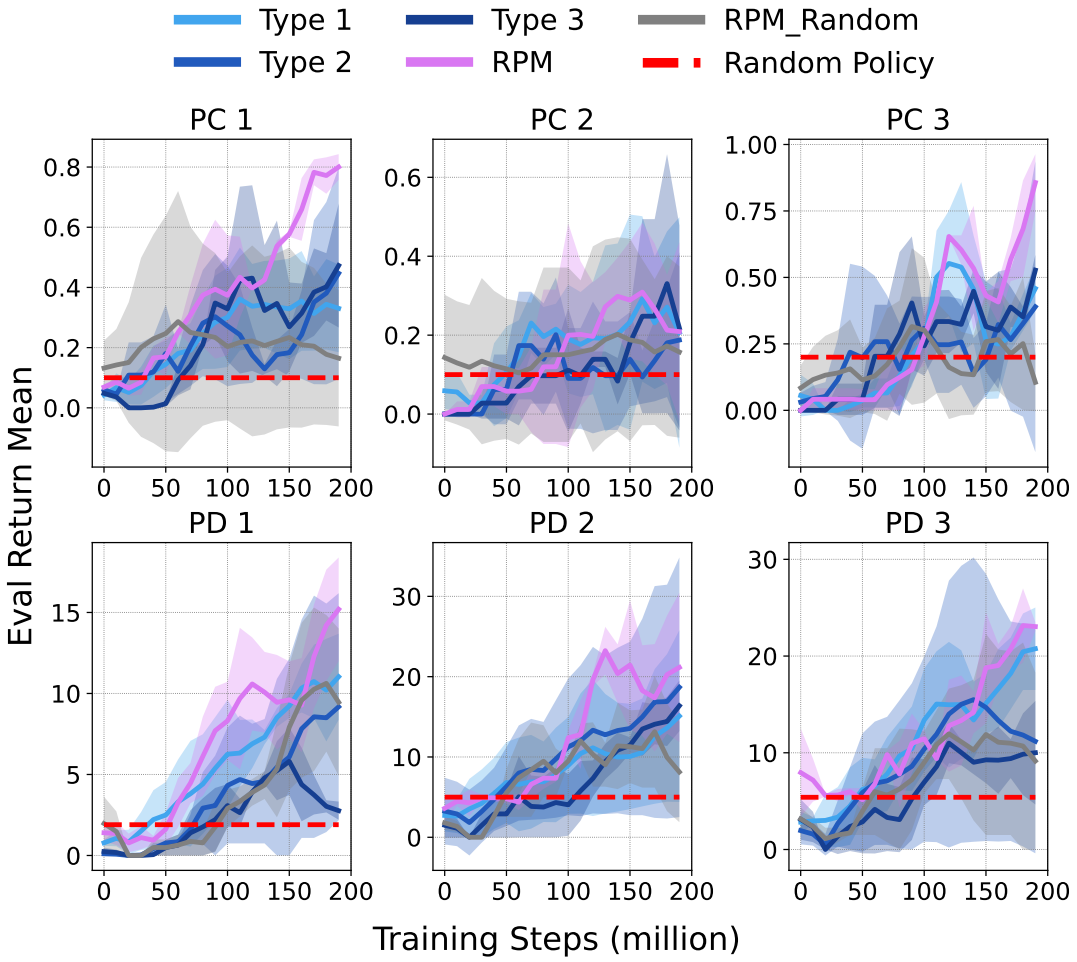


FIGURE 6.10: Ablation Study: the performance of RPM with 3 types of ψ and Random sampling (without ranks).

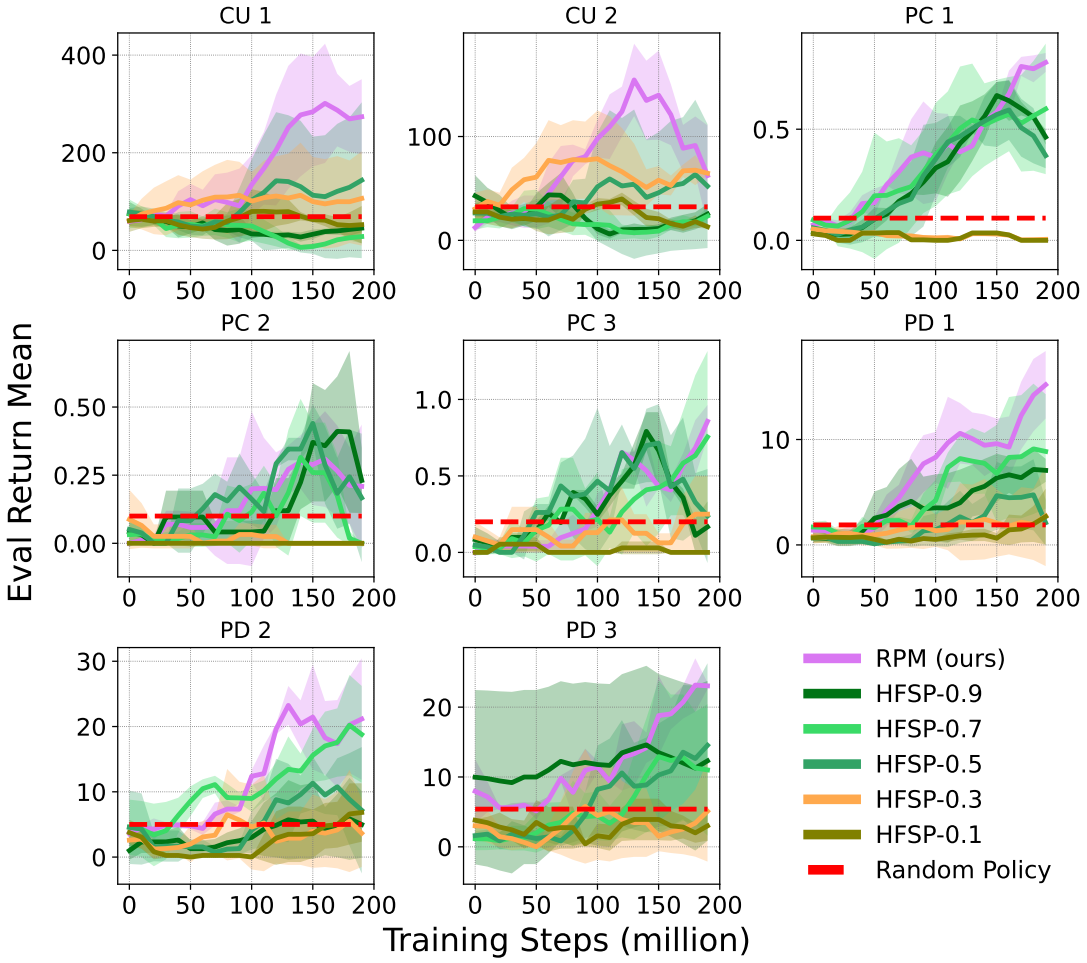


FIGURE 6.11: Ablation Study: the results of HFSP with different sampling ratios.

results of Pure Coordination (PC) 1-3 and Prisoners' Dilemma (PD) 1-3, showing that the value of ψ is important for RPM. We summarize the results and values of ψ in Tables 6.4 and 6.5.

The Sampling Ratio in HFSP HFSP shows comparable results in some scenarios in Figure 6.7. In Figure 6.7, the sampling ratio of HFSP is 0.3. We are interested in studying the impact of the sampling ratio in HFSP on evaluation performance. We conduct experiments in CU 1 and 2, PC 1 and 3, and PD 1 and 3. The sampling ratio list is $[0.9, 0.7, 0.5, 0.3, 0.1]$. We use the default training setup and use 3 random seeds. HFSP shows comparable results in PC 2 and 3, but its performances are poor in CU 1 and 2 and PD 2 and 3. As shown in Figure 6.11, HFSP heavily relies on the sampling ratio. HFSP should be carefully tuned on each substrate to attain good performance, which is not feasible. In contrast, RPM is

FIGURE 6.12: Results analysis. **(a)** The evaluation results of RPM on Stag Hunt (SH) 1; **(b)** The number of RPM keys during training; **(c)** The distribution of the keys of RPM during training; **(d)** The histogram of the keys of RPM at time step 200M during training.

stable (the sampling ratio is 0.5) on all substrates. HFSP can also perform well in substrates such as PC and PD, where the return-checkpoint count distribution is more uniform. The absence of ranks leads to the frequent sampling of policies with high count values in substrates that have skewed return-checkpoint count distribution, thereby reducing the diversity of training data. Such distributions typically comprise a large number of policies with suboptimal performance.

6.6.7 Case Study

We showcase how RPM helps to train the focal agents to choose the right behaviors in the evaluation scenario after training in the substrate. To illustrate the trained performance of RPM agents, we use the RPM agent trained on Stag Hunt and run the evaluation on Stag Hunt 1. In Stag Hunt, there are 8 agents. Each agent collects

resources that represent 'hare' (red) or 'stag' (green) and compares inventories in an interaction, *i.e.*, encounter. The results of solving the encounter are the same as the classic Stag Hunt matrix game. In this environment, agents are facing tension between the reward for the team and the risk for the individual. In Stag Hunt 1, One focal agent interacts with seven pretrained background agents. All background agents were trained to play the 'stag' strategy during the interaction[5]. The optimal policy for the focal agent is also to play 'stag'. However, it is challenging for agents to detect other agents' strategies since such behavior may not persist in the substrate. Luckily, RPM enables focal agents to behave correctly in this scenario.

To answer **Q3**, we present the analysis of RPM on the substrate Stag Hunt and its evaluation scenario SH 1 in Figure 6.12. We can find that in Figure 6.12 (b), the number of the keys in RPM is growing monotonically during training and the maximum number of the keys in RPM is over 20, showing that agents trained with RPM discover many novel patterns of multi-agent interaction and new keys are created and the trained models are saved in RPM. Meanwhile, the evaluation performance is also increasing in SH 1 as depicted in Figure 6.12 (a). In Figure 6.12 (c), it is interesting to see that the distribution of the keys of RPM is expanding during training. In the last 25 million training steps, the last distribution of RPM keys covers all policies of different performance levels, ranging from 0 to 14. By utilizing RPM, agents can collect diversified multi-agent trajectories for multi-agent training. Figure 6.12 (d) demonstrates the final histogram of RPM keys after training. There are over 600 trained policies that have a small value of keys. Since agents should explore the environment at the early stage of training, it is reasonable to find that many trained policies of RPM keys have low training episode returns. After 50 million training steps, RPM has more policies with higher training episode returns. Note that the maximum training episode return of RPM keys is over 14 while the maximum mean evaluation return of RPM shown in Figure 6.12 (a) is around 14.

Our experiments show that training policies with good performance in the substrate is crucial for improving generalization performance in the evaluation scenarios. When MARL agents perform poorly in the substrate, the evaluation performance will also be inferior or random, making it hard to have diversified policies. We show the results in Figure 6.8.

[5]This preference was trained with pseudo rewards by [3] and the trained models are available at this link: `https://github.com/deepmind/meltingpot`

6.7 Chapter Summary

In this chapter, we consider the problem of achieving generalizable behaviors in MARL. We first model the problem with Markov Game. To train agents that can interact with agents that possess unseen policies. We propose a simple yet effective method, RPM, to collect diversified multi-agent interaction data. We save policies in RPM by ranking the training episode return. Empirically, RPM significantly boosts the performance of MARL agents in various Melting Pot evaluation scenarios.

RPM's performance is dependent on the appropriate value of ψ. Several attempts may be needed to determine the correct value of ψ for RPM. We are interested in discovering broader measures for ranking policies that do not explicitly consider the training episode return. Recently, there has been a growing interest in planning in RL, especially with model-based RL. We are interested in exploring the direction of applying planning and opponent/teammate modeling for attaining generalized MARL policies for future work. Agents are engaged in complex interactions in multi-agent scenarios. Devising novel self-play methods is our future direction. RPM relies on self-play, which works for homogeneous and symmetric scenarios. We leave it for future work on improving the generalization of MARL agents in heterogeneous scenarios.

Chapter 7

Conclusions and Future Directions

7.1 Conclusions

In this thesis, we investigated four fundamental MARL research problems that are essential for sequential decision-making in multi-agent scenarios. We aim to address these problems with Deep MARL. To this end, we first proposed a novel risk-sensitive MARL method to learn risk-sensitive policies for improving multi-agent coordination in environments with uncertainty, such as enhancing multi-agent coordination in StarCraft II micromanagement scenarios, in Chapter 3. We investigated the problem of scaling up DETC and cast the problem as a MARL problem, and we proposed a novel MARL to scale up DETC in a divide-and-conquer manner in Chapter 4. We considered the problem of multi-agent coordination in environments with off-beat actions, which was rarely studied within the MARL community. To learn efficient multi-agent coordination in OBMAS, we proposed LeGEM to boost multi-agent coordination via a novel reward redistribution method in Chapter 5. We finally studied the problem of improving the generalization of MARL methods and proposed a novel method called RPM in Chapter 6. RPM significantly improved the evaluation performance of MARL in Melting Pot evaluation scenarios.

In summary, we focused on obtaining efficient sequential decision-making with deep MARL. To this end, we proposed four novel deep MARL methods. Deep MARL has emerged as a powerful approach for solving complex problems in multi-agent systems. Compared to other methods, deep MARL is better suited to capturing the complex, dynamic interactions between agents in environments. By training

agents to make decisions based on their observations, deep MARL can enable the emergence of sophisticated collective behaviors that would be difficult to achieve with other techniques.

7.2 Future Directions

While we have made advanced progress in MARL research, some critical challenges remain to be investigated. Specifically, we are interested in investigating finding novel methods to improve our proposed method. In addition, we will focus on human-AI coordination.

7.2.1 Risk-Sensitive MARL

Risk-sensitive policy learning is vital for many real-world multi-agent applications, especially in risky tasks, for example, autopilot vehicles, military action, resource allocation, finance portfolio management and the Internet of Things (IoT). For future work, better risk measurement, together with accurate spatial-temporal trajectory representation, can be investigated. Also, learning to model other agents' risk levels and reach a consensus with communication can be another direction for enhancing multi-agent coordination.

7.2.2 MARL for Intelligent Traffic Systems

Traffic congestion is still a noticeable issue in metropolitans. The heterogeneous nature of road networks, where various kinds of traffic lights, ETC systems, roads, public transportation systems and public policies, etc., constitute a complex traffic system, which hinders the application of DETC systems. We will improve DETC systems by considering the heterogeneous nature of road networks and apply our method to other cities' road networks to gain generalization for complex tolling control as well as learn transfer ability to reduce deployment costs.

7.2.3 Multi-Agent Coordination in OBMAS

Searching from graph structures takes much overhead in LeGEM. Scaling up our method to complex OBMAS is our future direction. One possible direction is applying Transformers [116] to memory building as Transformers are easy to parallel and efficient in learning and inference. Another direction is building hierarchical memory structures for temporal abstraction. LeGEMfocuses on the Dec-POMDP framework. We leave it to future work to investigate off-beat actions in frameworks like Markov game [70] and MMDP [157].

7.2.4 Generalizable Human-AI Coordination

ChatGPT [201, 202] represents a significant milestone in developing AI agents with natural language processing (NLP). As a large language model (LLM) [203–208] trained on vast amounts of data, ChatGPT can generate human-like natural language responses to various questions and prompts. Such an ability to converse with humans fosters greater collaboration and understanding between humans and AI, as users can engage with the system more naturally without needing to learn specialized commands or interfaces. Typically, it opens up new possibilities for generalizable human-AI coordination, from personalized chatbots and virtual assistants to more sophisticated applications in video games, robotics, education, healthcare, and customer service. As such, LLMs like ChatGPT are likely to play a significant role in shaping the future of human-AI coordination and paving the way for more advanced and intelligent systems that can better serve and enhance human activities.

With LLMs, we will focus on creating generalizable autonomous agents that can communicate with human beings and serve them in many scenarios, including video games, personal assistants in smartphones, personal assistants in smart home systems, and even unseen scenarios. For example, in video games, a well-trained MARL agent can communicate with various human players unseen before. The MARL agent can fulfill human players' desires, such as collecting resources for human players and assisting human players in a battle during the game. Such an agent can significantly enhance human players' user experience in video games and further advance the application of MARL methods to many real-world problems.

Bibliography

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. xi, 26, 112

[2] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for sttaistical machine translation. In *EMNLP*, 2014. xi, 26, 27

[3] Joel Z Leibo, Edgar A Dueñez-Guzman, Alexander Vezhnevets, John P Agapiou, Peter Sunehag, Raphael Koster, Jayd Matyas, Charlie Beattie, Igor Mordatch, and Thore Graepel. Scalable evaluation of multi-agent reinforcement learning with melting pot. In *ICML*, pages 6187–6199, 2021. xiv, 6, 97, 98, 101, 102, 106, 118, 124

[4] Haipeng Chen, Bryan Wilder, Wei Qiu, Bo An, Eric Rice, and Milind Tambe. A learning approach to complex contagion influence maximization. In *AAMAS*, pages 2622–2624, 2023. xvii

[5] Haipeng Chen, Bryan Wilder, Wei Qiu, Bo An, Eric Rice, and Milind Tambe. A learning approach to complex contagion influence maximization. In *IJCAI*, 2023. xvii

[6] Wei Qiu, Xiao Ma, Bo An, Svetlana Obraztsova, Shuicheng YAN, and Zhongwen Xu. RPM: Generalizable multi-agent policies for multi-agent reinforcement learning. In *ICLR*, 2023. URL `https://openreview.net/forum?id=HnSceSzlfrY`. xvii, 96

[7] Rundong Wang, Longtao Zheng, Wei Qiu, Bowei He, Bo An, Zinovi Rabinovich, Yujing Hu, Yingfeng Chen, Tangjie Lv, and Changjie Fan. Towards skilled population curriculum for multi-agent reinforcement learning, 2023. xvii

[8] Wei Qiu, Weixun Wang, Rundong Wang, Bo An, Yujing Hu, Svetlana Obraztsova, Zinovi Rabinovich, Jianye Hao, Yingfeng Chen, and Changjie Fan. Off-beat multi-agent reinforcement learning. In *AAMAS*, pages 2424–2426, 2023. xvii, 66, 99

[9] Wanqi Xue, Wei Qiu, Bo An, Zinovi Rabinovich, Svetlana Obraztsova, and Chai Kiat Yeo. Mis-spoke or mis-lead: Achieving robustness in multi-agent communicative reinforcement learning. In *AAMAS*, 2022. xvii

[10] Wei Qiu, Xinrun Wang, Runsheng Yu, Rundong Wang, Xu He, Bo An, Svetlana Obraztsova, and Zinovi Rabinovich. RMIX: Learning risk-sensitive policies for cooperative reinforcement learning agents. In *NeurIPS*, 2021. xviii, 20, 82

[11] Haipeng Chen, Wei Qiu, Han-Ching Ou, Bo An, and Milind Tambe. Contingency-aware influence maximization: A reinforcement learning approach. In *UAI*, pages 1535–1545, 2021. xviii

[12] Rundong Wang, Xu He, Runsheng Yu, Wei Qiu, Bo An, and Zinovi Rabinovich. Learning efficient multi-agent communication: An information bottleneck approach. In *ICML*, pages 9908–9918, 2020. xviii, 70

[13] Wei Qiu, Haipeng Chen, and Bo An. Dynamic electronic toll collection via multi-agent deep reinforcement learning with edge-based graph convolutional networks. In *IJCAI*, pages 4568–4574. AAAI Press, 2019. xviii, 2, 49

[14] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008. 2

[15] Frans A Oliehoek, Christopher Amato, et al. *A Concise Introduction to Decentralized POMDPs*, volume 1. Springer, 2016. 2, 3, 5, 16, 67

[16] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010. 2

[17] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. 2, 6, 21, 67, 97, 99

[18] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364 (6443):859–865, 2019. 2, 6, 18, 97, 99

[19] Arambam James Singh, Akshat Kumar, and Hoong Chuin Lau. Hierarchical multiagent reinforcement learning for maritime traffic management. In *AAMAS*, 2020. 2, 21

[20] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial Informatics*, 9(1):427–438, 2012. 2, 67

[21] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Guided deep reinforcement learning for swarm systems. *In AAMAS 2017 Autonomous Robots and Multirobot Systems (ARMS) Workshop*, 2017. 21, 99

[22] Ming Zhou, Jun Luo, Julian Villella, Yaodong Yang, David Rusu, Jiayu Miao, Weinan Zhang, Montgomery Alban, Iman Fadakar, Zheng Chen, et al. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving. *arXiv preprint arXiv:2010.09776*, 2020. 2, 3, 67

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 2, 12, 14, 18, 30, 37, 56, 67

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 2, 12, 15, 105, 112

[25] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NIPS*, pages 6379–6390, 2017. 2, 18, 39, 99, 117

[26] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *ICML*, pages 4295–4304, 2018. 3, 4, 5, 16, 17, 23, 34, 35, 37, 67, 70, 74, 82, 99, 105, 117

[27] Yaodong Yang and Jun Wang. An overview of multi-agent reinforcement learning from game theoretical perspective. *arXiv preprint arXiv:2011.00583*, 2020. 2, 97, 99

[28] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. 2, 12

[29] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018. 2, 12, 13, 14, 30, 32, 70, 73, 82, 97

[30] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019. 2, 33, 37, 84

[31] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008. 2, 4, 67, 102, 112

[32] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI*, 2018. 3, 4, 5, 37, 38, 67, 70

[33] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017. 4, 5, 17, 23, 31, 37, 67, 70, 82

[34] Weixun Wang, Tianpei Yang, Yong Liu, Jianye Hao, Xiaotian Hao, Yujing Hu, Yingfeng Chen, Changjie Fan, and Yang Gao. Action semantics network: Considering the effects of actions in multiagent systems. In *ICLR*, 2020.

[35] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *ICML*, pages 5887–5896, 2019. 3, 4, 5, 16, 17, 23, 37, 38, 67, 70, 82

[36] Yihan Wang, Beining Han, Tonghan Wang, Heng Dong, and Chongjie Zhang. DOP: Off-policy multi-agent decomposed policy gradients. In *ICML*, 2021. URL `https://openreview.net/forum?id=6FqKiVAdI3Y`. 4, 37, 39, 99

[37] Jakub Grudzien Kuba, Ruiqing Chen, Munning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*, 2021. 17

[38] Ling Pan, Tabish Rashid, Bei Peng, Longbo Huang, and Shimon Whiteson. Regularized softmax deep multi-agent Q-learning. In *NeurIPS*, volume 34, 2021. 3, 5, 67, 70

[39] David Ha and Yujin Tang. Collective intelligence for deep learning: A survey of recent developments. *arXiv preprint arXiv:2111.14377*, 2021. 3

[40] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. QPLEX: Duplex dueling multi-agent q-learning. In *ICLR*, 2021. 4, 17, 37, 38, 70, 82, 105, 117

[41] Joseph L Baxter, EK Burke, Jonathan M Garibaldi, and Mark Norman. Multi-robot search and rescue: A potential field based approach. *Autonomous robots and agents*, pages 9–16, 2007. 4, 20

[42] Kuilin Zhang, Hani S Mahmassani, and Chung-Cheng Lu. Dynamic pricing, heterogeneous users and perception error: Probit-based bi-criterion dynamic stochastic user equilibrium assignment. *Transportation Research Part C: Emerging Technologies*, 27:189–204, 2013. 4, 51

[43] Guni Sharon, Josiah P Hanna, Tarun Rambha, Michael W Levin, Michael Albert, Stephen D Boyles, and Peter Stone. Real-time adaptive tolling scheme for optimized social welfare in traffic networks. In *AAMAS*, pages 828–836, 2017.

[44] Haipeng Chen, Bo An, Guni Sharon, Josiah P Hanna, Peter Stone, Chunyan Miao, and Yeng Chai Soh. DyETC: Dynamic electronic toll collection for traffic congestion alleviation. In *AAAI*, pages 757–765, 2018. 5, 51, 54, 60

[45] Hamid Mirzaei, Guni Sharon, Stephen Boyles, Tony Givargis, and Peter Stone. Link-based parameterized micro-tolling scheme for optimal traffic management. In *AAMAS*, pages 2013–2015, 2018. 4, 51

[46] Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 1984. 5, 14, 18, 67, 70, 112

[47] Julien Perolat, Bart de Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. Mastering the game of stratego with model-free multiagent reinforcement learning. *Science*, 2022. 6, 97, 99

[48] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *ICLR*, 2019. 6, 99, 112, 113, 115

[49] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021. 6

[50] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014. 11, 12

[51] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992. 12, 14, 54

[52] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992. 12, 15, 18, 54

[53] Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022. 12

[54] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016. 12, 15, 18, 54

[55] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017. 12

[56] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022. 12

[57] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022. 12

[58] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957. 13

[59] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *ICML*, 2017. 15, 21, 24, 25, 30, 43

[60] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2017. 15

[61] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *AAAI*, pages 2892–2901, 2018. 15, 24, 30, 36, 48

[62] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *ICML*, pages 1096–1105, 2018. 15

[63] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. 15, 16

[64] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 12, 1999. 15

[65] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016. 15, 105

[66] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, pages 1587–1596, 2018. 18

[67] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, pages 1861–1870, 2018. 15

[68] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. 16

[69] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. Credit assignment for collective multiagent rl with global rewards. In *NIPS*, volume 31, 2018. 17

[70] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994. 17, 95, 98, 128

[71] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In *ICML*, pages 5571–5580. PMLR, 2018. 18

[72] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In *ICLR*, 2020. 18, 100, 104, 112, 113

[73] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *AAMAS*, 2018. 18

[74] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *ICML*, 2019. 18, 99

[75] Marc Lanctot, Vinicius Zambaldi, Audrūnas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *NIPS*, 2017. 18, 99

[76] Sriram Srinivasan, Marc Lanctot, Vinicius Zambaldi, Julien Pérolat, Karl Tuyls, Rémi Munos, and Michael Bowling. Actor-critic policy optimization in partially observable multiagent environments. In *NIPS*, volume 31, 2018.

[77] Jiaming Song, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon. Multi-agent generative adversarial imitation learning. In *NIPS*, volume 31, 2018.

[78] Aditya Grover, Maruan Al-Shedivat, Jayesh Gupta, Yuri Burda, and Harrison Edwards. Learning policy representations in multiagent systems. In *ICML*, pages 1802–1811, 2018.

[79] Jiachen Yang, Alireza Nakhaei, David Isele, Kikuo Fujimura, and Hongyuan Zha. CM3: Cooperative multi-goal multi-stage multi-agent reinforcement learning. In *ICLR*, 2020.

[80] Paul Muller, Shayegan Omidshafiei, Mark Rowland, Karl Tuyls, Julien Perolat, Siqi Liu, Daniel Hennes, Luke Marris, Marc Lanctot, Edward Hughes, et al. A generalized training approach for multiagent learning. In *ICML*, 2019.

[81] Kaiqing Zhang, Sham Kakade, Tamer Basar, and Lin Yang. Model-based multi-agent rl in zero-sum markov games with near-optimal sample complexity. In *NeurIPS*, volume 33, pages 1166–1178, 2020. 18

[82] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2019. 18

[83] Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *ICLR*, 2020.

[84] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. 18, 95

[85] Will Dabney, Georg Ostrovski, David Silver, and Remi Munos. Implicit quantile networks for distributional reinforcement learning. In *ICML*, 2018. 21, 23, 43

[86] Yinlam Chow and Mohammad Ghavamzadeh. Algorithms for cvar optimization in MDPs. In *NIPS*, pages 3509–3517, 2014. 21

[87] Ramtin Keramati, Christoph Dann, Alex Tamkin, and Emma Brunskill. Being optimistic to be conservative: Quickly learning a cvar policy. In *AAAI*, pages 4436–4443, 2020. 21

[88] Junyu Zhang, Amrit Singh Bedi, Mengdi Wang, and Alec Koppel. Cautious reinforcement learning via distributional risk in the dual domain. *arXiv preprint arXiv:2002.12475*, 2020.

[89] Cristian Bodnar, Adrian Li, Karol Hausman, Peter Pastor, and Mrinal Kalakrishnan. Quantile qt-opt for risk-aware vision-based robotic grasping. In *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020. doi: 10.15607/RSS.2020.XVI.075. 21

[90] R Tyrrell Rockafellar and Stanislav Uryasev. Conditional value-at-risk for general loss distributions. *Journal of Banking & Finance*, 26(7):1443–1471, 2002. 21, 23, 24

[91] Andrzej Ruszczyński. Risk-averse dynamic programming for markov decision processes. *Mathematical Programming*, 125(2):235–261, 2010. 21, 27

[92] Yichuan Charlie Tang, Jian Zhang, and Ruslan Salakhutdinov. Worst cases policy gradients. In *CoRL*, pages 1078–1093, 2020. 21, 24

[93] Xiaoteng Ma, Qiyuan Zhang, Li Xia, Zhengyuan Zhou, Jun Yang, and Qianchuan Zhao. Distributional soft actor critic for risk sensitive learning. *arXiv preprint arXiv:2004.14547*, 2020. 21

[94] R Tyrrell Rockafellar, Stanislav Uryasev, et al. Optimization of conditional value-at-risk. *Journal of Risk*, 2:21–42, 2000. 23

[95] Carlo Acerbi and Dirk Tasche. On the coherence of expected shortfall. *Journal of Banking & Finance*, 26(7):1487–1503, 2002. 23

[96] Javier García et al. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(42):1437–1480, 2015. 23

[97] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior, 2nd rev.* Princeton university press, 1947. 23

[98] Yinlam Chow, Aviv Tamar, Shie Mannor, and Marco Pavone. Risk-sensitive and robust decision-making: a cvar optimization approach. In *NIPS*, pages 1522–1530, 2015. 23

[99] Mark Rowland, Marc G Bellemare, Will Dabney, Rémi Munos, and Yee Whye Teh. An analysis of categorical distributional reinforcement learning. *arXiv preprint arXiv:1802.08163*, 2018. 24

[100] Derek Yang, Li Zhao, Zichuan Lin, Tao Qin, Jiang Bian, and Tie-Yan Liu. Fully parameterized quantile function for distributional reinforcement learning. In *NeurIPS*, pages 6193–6202, 2019. 24

[101] Fan Zhou, Jianing Wang, and Xingdong Feng. Non-crossing quantile regression for distributional reinforcement learning. In *NeurIPS*, volume 33, pages 15909–15919, 2020. 24

[102] Ravi Kumar Kolla, LA Prashanth, Sanjay P Bhat, and Krishna Jagannathan. Concentration bounds for empirical conditional value-at-risk: The unbounded case. *Operations Research Letters*, 47(1):16–20, 2019. 26

[103] Dan A Iancu, Marek Petrik, and Dharmashankar Subramanian. Tight approximations of dynamic risk measures. *Mathematics of Operations Research*, 40(3):655–682, 2015. 27

[104] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *ICML*, pages 2961–2970, 2019. 28

[105] Yaodong Yang, Jianye Hao, Ben Liao, Kun Shao, Guangyong Chen, Wulong Liu, and Hongyao Tang. Qatten: A general framework for cooperative multiagent reinforcement learning. *arXiv preprint arXiv:2002.03939*, 2020. 28

[106] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014. 30

[107] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics: Methodology and distribution*, pages 492–518. Springer, 1992. 30

[108] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. MAVEN: Multi-agent variational exploration. In *NeurIPS*, pages 7613–7624, 2019. 34, 82

[109] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PLoS ONE*, 12(4), 2017. 37, 82, 102

[110] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation. *arXiv preprint arXiv:2006.10800*, 2020. 37, 38, 105

[111] Xueguang Lyu and Christopher Amato. Likelihood quantile networks for coordinating multi-agent reinforcement learning. In *AAMAS*, pages 798–806, 2020. 37, 38, 41

[112] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In *ICLR*. OpenReview.net, 2017. 37

[113] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *ICML*, pages 1995–2003, 2016. 38, 40

[114] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 64–69. IEEE, 2007. 38

[115] Gregory Palmer, Karl Tuyls, Daan Bloembergen, and Rahul Savani. Lenient multi-agent deep reinforcement learning. In *AAMAS*, pages 443–451, 2018. 38

[116] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017. 40, 128

[117] BPR. Traffic assignment manual. *US Department of Commerce*, 1964. 53, 60

[118] Hong K Lo and WY Szeto. A methodology for sustainable traveler information services. *Transportation Research Part B: Methodological*, 36(2):113–130, 2002. 53

[119] Hai-Jun Huang and Zhi-Chun Li. A multiclass, multicriteria logit-based traffic equilibrium assignment model under atis. *European Journal of Operational Research*, 176(3):1464–1477, 2007. 53

[120] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using Connectionist Systems*, volume 37. University of Cambridge, 1994. 54

[121] Vijaymohan R Konda and Vivek S Borkar. Actor-critic–type learning algorithms for markov decision processes. *SIAM Journal on Control and Optimization*, 38(1):94–123, 1999. 54

[122] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015. 56

[123] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, pages 2014–2023, 2016.

[124] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2016. 56

[125] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016. 56

[126] LTA. Annual motor vehicle population by vehicle quota categories. https://data.gov.sg/dataset/annual-motor-vehicle-population-by-vehicle-quota-category, 2017. Accessed: 2016-04-12. 60

[127] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Computer Science*, 2014. 61

[128] Simon Ramstedt and Chris Pal. Real-time reinforcement learning. In *NeurIPS*, volume 32, pages 3073–3082, 2019. 67, 68, 69

[129] Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame, Christopher Pal, and Jonathan Binas. Reinforcement learning with random delays. In *ICLR*, 2020. 67, 69, 70

[130] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *ICML*, pages 1407–1416, 2018. 67, 117

[131] Thomas J Walsh, Ali Nouri, Lihong Li, and Michael L Littman. Learning and planning in environments with delayed feedback. In *AAMAS*, volume 18, pages 83–105. Springer, 2009. 68, 69

[132] Ted Xiao, Eric Jang, Dmitry Kalashnikov, Sergey Levine, Julian Ibarz, Karol Hausman, and Alexander Herzog. Thinking while moving: Deep reinforcement learning with concurrent control. In *ICLR*, 2019. 68, 69

[133] Esther Derman, Gal Dalal, and Shie Mannor. Acting in delayed environments with non-stationary Markov policies. In *ICLR*, 2020. 68, 69

[134] Baiming Chen, Mengdi Xu, Zuxin Liu, Liang Li, and Ding Zhao. Delay-aware multi-agent reinforcement learning for cooperative and competitive environments. *arXiv e-prints*, pages arXiv–2005, 2020. 68, 69

[135] Yuchen Xiao, Joshua Hoffman, and Christopher Amato. Macro-action-based deep multi-agent reinforcement learning. In *CoRL*, pages 1146–1161, 2020. 68, 70

[136] Yuchen Xiao, Joshua Hoffman, Tian Xia, and Christopher Amato. Multi-agent/robot deep reinforcement learning with macro-actions (student abstract). In *AAAI*, volume 34, pages 13965–13966, 2020. 68

[137] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999. 69, 70, 112

[138] Konstantinos V Katsikopoulos and Sascha E Engelbrecht. Markov decision processes with delays and asynchronous cost collection. *IEEE Transactions on Automatic Control*, 48(4):568–574, 2003. 69

[139] Yufeng Yuan and Rupam Mahmood. Asynchronous reinforcement learning for real-time control of physical robots. *arXiv preprint arXiv:2203.12759*, 2022. 69

[140] Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. RUDDER: Return decomposition for delayed rewards. In *NeurIPS*, volume 32, 2019. 70, 82, 94

[141] Tanmay Gangwani, Yuan Zhou, and Jian Peng. Learning guidance rewards with trajectory-space smoothing. In *NeurIPS*, volume 33, pages 822–832, 2020. 70

[142] Zhizhou Ren, Ruihan Guo, Yuan Zhou, and Jian Peng. Learning long-term reward redistribution via randomized return decomposition. In *ICLR*, 2022. 70, 94

[143] David Raposo, Sam Ritter, Adam Santoro, Greg Wayne, Theophane Weber, Matt Botvinick, Hado van Hasselt, and Francis Song. Synthetic returns for long-term credit assignment. *arXiv preprint arXiv:2102.12425*, 2021. 70

[144] Martin Klissarov and Doina Precup. Reward propagation using graph convolutional networks. In *NeurIPS*, volume 33, 2020. 70

[145] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *ICML*, pages 2827–2836, 2017. 70

[146] Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. Optimizing agent behavior over long time scales by transporting value. *Nature Communications*, 10 (1):1–12, 2019. 75

[147] Guangxiang Zhu, Zichuan Lin, Guangwen Yang, and Chongjie Zhang. Episodic reinforcement learning with associative memory. In *ICLR*, 2020. 70, 87

[148] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016. 70

[149] João V Messias, Matthijs TJ Spaan, and Pedro U Lima. Multiagent pomdps with asynchronous execution. In *AAMAS*, pages 1273–1274, 2013. 70

[150] Shayegan Omidshafiei, Ali-Akbar Agha-Mohammadi, Christopher Amato, and Jonathan P How. Decentralized control of partially observable markov decision processes using belief space macro-actions. In *ICRA*, pages 5962–5969, 2015. 70

[151] Christopher Amato, George Konidaris, Leslie P Kaelbling, and Jonathan P How. Modeling and planning with macro-actions in decentralized pomdps. *Journal of Artificial Intelligence Research*, 64:817–859, 2019. 70, 99

[152] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, volume 31, 2017. 70, 112

[153] David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, 2021. 73

[154] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth Annual ACM Symposium on Theory of Computing*, pages 380–388, 2002. 76

[155] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *NIPS*, volume 30, 2017. 76

[156] Lulu Zheng, Jiarui Chen, Jianhao Wang, Jiamin He, Yujing Hu, Yingfeng Chen, Changjie Fan, Yang Gao, and Chongjie Zhang. Episodic multi-agent reinforcement learning with curiosity-driven exploration. In *NeurIPS*, volume 34, 2021. 82

[157] Craig Boutilier. Planning learning and coordination in multiagent decision processes. In *Proceedings of the 6th Conference on the Theoretical Aspects of Rationality and Knowledge*, pages 195–210, 1996. 95, 128

[158] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM special interest group on data communication*, pages 270–288, 2019. 95

[159] Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: how do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795, 2020. 97

[160] Andrew N Meltzoff, Patricia K Kuhl, Javier Movellan, and Terrence J Sejnowski. Foundations for a new science of learning. *Science*, 325(5938): 284–288, 2009. 98

[161] Michael Tomasello. *Origins of human communication*. MIT press, 2010. 98

[162] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021. 98, 105, 112, 115, 117

[163] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pages 321–384, 2021. 99

[164] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 99, 115

[165] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *AAMAS*, 2017. 99

[166] Weixun Wang, Jianye Hao, Yixi Wang, and Matthew Taylor. Towards cooperation in sequential prisoner's dilemmas: a deep multiagent reinforcement learning approach. *arXiv preprint arXiv:1803.00162*, 2018.

[167] Alexander Vezhnevets, Yuhuai Wu, Maria Eckstein, Rémi Leblond, and Joel Z Leibo. Options as responses: Grounding behavioural hierarchies in multi-agent reinforcement learning. In *ICML*, pages 9733–9742, 2020. 99, 112, 115

[168] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *NIPS*, pages 2137–2145, 2016. 99

[169] Lei Yuan, Jianhao Wang, Fuxiang Zhang, Chenghe Wang, ZongZhang Zhang, Yang Yu, and Chongjie Zhang. Multi-agent incentive communication via decentralized teammate modeling. In *AAAI*, volume 36, pages 9466–9474, Jun. 2022. 99

[170] Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021. 99

[171] Zhenghao Peng, Quanyi Li, Ka Ming Hui, Chunxiao Liu, and Bolei Zhou. Learning to simulate self-driven particles system with coordinated policy optimization. In *NeurIPS*, volume 34, pages 10784–10797, 2021. 99

[172] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018. 99

[173] Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning. In *ICLR*, 2019.

[174] Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P Adams, and Sergey Levine. Why generalization in rl is difficult: Epistemic pomdps and implicit partial observability. In *NeurIPS*, volume 34, 2021.

[175] Clare Lyle, Mark Rowland, Will Dabney, Marta Kwiatkowska, and Yarin Gal. Learning dynamics and generalization in reinforcement learning. *arXiv preprint arXiv:2206.02126*, 2022. 99

[176] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *ICML*, pages 1480–1490, 2017. 99

[177] Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *ICLR*, 2020. 99

[178] Daochen Zha, Wenye Ma, Lei Yuan, Xia Hu, and Ji Liu. Rank the episodes: A simple approach for exploration in procedurally-generated environments. In *ICLR*, 2020. 99

[179] Nicolas Carion, Nicolas Usunier, Gabriel Synnaeve, and Alessandro Lazaric. A structured prediction approach for generalization in cooperative multi-agent reinforcement learning. In *NeurIPS*, volume 32, 2019. 99

[180] Anuj Mahajan, Mikayel Samvelyan, Tarun Gupta, Benjamin Ellis, Mingfei Sun, Tim Rocktäschel, and Shimon Whiteson. Generalization in cooperative multi-agent systems. *arXiv preprint arXiv:2202.00104*, 2022. 99

[181] Kevin R McKee, Joel Z Leibo, Charlie Beattie, and Richard Everett. Quantifying the effects of environment and population diversity in multi-agent reinforcement learning. In *AAMAS*, volume 36, pages 1–16. Springer, 2022. 99

[182] Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993. 99

[183] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *ICML*, pages 501–510, 2018. 99

[184] DJ Strouse, Kevin McKee, Matt Botvinick, Edward Hughes, and Richard Everett. Collaborating with humans without human data. In *NeurIPS*, volume 34, pages 14502–14515, 2021. 99

[185] Andrei Lupu, Brandon Cui, Hengyuan Hu, and Jakob Foerster. Trajectory diversity for zero-shot coordination. In *ICML*, pages 7204–7213, 2021. 99, 100

[186] Zhenggang Tang, Chao Yu, Boyuan Chen, Huazhe Xu, Xiaolong Wang, Fei Fang, Simon Du, Yu Wang, and Yi Wu. Discovering diverse multi-agent strategic behavior via reward randomization. *arXiv preprint arXiv:2103.04564*, 2021. 99, 100

[187] Peter Stone and Sarit Kraus. To teach or not to teach?: decision making under uncertainty in ad hoc teams. In *AAMAS*, pages 117–124, 2010. 100, 102

[188] Pengjie Gu, Mengchen Zhao, Jianye Hao, and Bo An. Online ad hoc teamwork under partial observability. In *ICLR*, 2021. 100, 102

[189] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *ICML*, pages 805–813, 2015. 100, 104, 112, 113

[190] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 104, 113

[191] Daniel Hernandez, Kevin Denamganaï, Yuan Gao, Peter York, Sam Devlin, Spyridon Samothrakis, and James Alfred Walker. A generalized framework for self-play training. In *2019 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2019. 100

[192] Maruan Al-Shedivat, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *ICLR*, 2018. 100

[193] Dong Ki Kim, Miao Liu, Matthew D Riemer, Chuangchuang Sun, Marwa Abdulhai, Golnaz Habibi, Sebastian Lopez-Cot, Gerald Tesauro, and Jonathan How. A policy gradient algorithm for learning to learn in multiagent reinforcement learning. In *ICML*, pages 5541–5550, 2021. 100, 101

[194] George W Brown. Iterative solution of games by fictitious play. *Act. Anal. Prod Allocation*, 13(1):374, 1951. 104, 113

[195] Robert Sugden. Rights, co-operation and welfare. In *The Economics of Rights, Co-operation and Welfare*, pages 170–182. Springer, 2005. 107

[196] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *NeurIPS Datasets and Benchmarks Track (Round 1)*, 2021. 112, 115, 117

[197] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937, 2016. 112

[198] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *EMNLP*, 2014. 112, 113

[199] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging AI applications. In *OSDI*, pages 561–577, 2018. 115, 117

[200] Johannes Heinrich. *Reinforcement learning from self-play in imperfect-information games*. PhD thesis, UCL (University College London), 2017. 115

[201] Tyna Eloundou, Sam Manning, Pamela Mishkin, and Daniel Rock. Gpts are gpts: An early look at the labor market impact potential of large language models, 2023. 128

[202] OpenAI. GPT-4 technical report, 2023. 128

[203] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *cdn.openai.com*, 2018. 128

[204] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[205] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[206] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *NeurIPS*, volume 33, pages 1877–1901, 2020.

[207] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, volume 35, pages 27730–27744, 2022.

[208] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023. 128