THE UNIVERSITY *of* EDINBURGH

Efficient Exploration in Single-Agent and Multi-Agent Deep Reinforcement Learning

Lukas Schäfer

Doctor of Philosophy
Institute of Perception, Action and Behaviour
School of Informatics
The University of Edinburgh
2024

Abstract

This thesis is concerned with *reinforcement learning* (RL) in which decision-making agents learn desirable behaviour by interacting with their environment and receiving feedback in the form of rewards. Learning decision making from interactions in this way is different from most machine learning paradigms in that RL agents need to both learn to collect data that informs their future decisions and learn the desired behaviour that maximises cumulative rewards. These processes are referred to as *exploration* and *exploitation*. In this thesis, we focus on the challenge of exploration for RL and propose novel solutions to guide agents towards efficient data collection and learn to balance exploration and exploitation.

The first part of this thesis is concerned with exploration for single-agent reinforcement learning in sparse-reward environments. In this setting, a single decision-making agent is learning from interactions with its environment and rarely receives (non-zero) rewards from the environment. A common approach to explore in such challenging environments where informative rewards are sparse is to introduce intrinsically computed rewards that incentivise agents to explore. However, by introducing this second optimisation objective, the agent needs to explicitly and carefully balance the exploration objective of intrinsic rewards and the exploitation objective of extrinsic rewards of the environment. We propose *decoupled reinforcement learning* (DeRL) to address this challenge. In DeRL, agents learn separate policies for both exploration and exploitation to account for the different objectives of these policies. The exploration policy is trained with intrinsic rewards and used to gather informative data for the exploitation policy while the exploitation policy is trained on the gathered data to solve the task at hand. We show that DeRL outperforms existing intrinsically motivated exploration approaches in terms of sample efficiency and robustness to hyperparameters that are responsible for the trade-off between exploration and exploitation.

For the subsequent parts of this thesis, we will shift our focus to exploration in the setting of *multi-agent reinforcement learning* (MARL). In MARL, multiple decision-making agents concurrently learn from interactions with their shared environment. In this thesis, we are concerned with environments that require agents to cooperate, i.e. agents need to learn to coordinate their actions to achieve their goals. This additional consideration in contrast to single-agent RL further complicates the learning process and exploration which now needs to account for

the interactions between agents. The first contribution of this thesis to MARL is a comprehensive benchmark of ten algorithms across a set of 25 cooperative common-reward environments. As part of this study, we open-source EPyMARL, a codebase for MARL that extends the previously existing PyMARL codebase with more algorithms, support for more environments, and configurability. Following the analysis of this benchmark, we identify remaining challenges in MARL to efficiently train agents to cooperate in environments where informative feedback is sparse. Motivated by this challenge, we propose *shared experience actor-critic* (SEAC). SEAC leverages symmetry present in many multi-agent environments to share experiences across agents and learn from the collective experience of all agents using an actor-critic algorithm. In empirical experiments, we establish that experience sharing can significantly improve the efficiency of learning and help agents to learn skills simultaneously. However, the benefits of experience sharing are less pronounced for value-based algorithms where agents do not learn explicit policies. To guide exploration for value-based MARL algorithms, we propose *ensemble value functions for multi-agent exploration* (EMAX). EMAX trains an ensemble of value functions for each agent and steers the exploration of agents towards states and actions that might require cooperation between multiple agents. By doing so, agents learn to coordinate their actions more efficiency, and we show that EMAX as an extension of three common value-based MARL algorithms can significantly improve the sample efficiency and stability of training.

Lastly, this thesis presents a case study in which we discuss the application of MARL to warehouse logistics automation. The chapter is the result of an industry collaboration with Dematic GmbH for which we formalise warehouse logistics problems, and propose a twofold solution to the scalability challenge of this setting. Our approach leverages a hierarchical decomposition of the multi-agent learning architecture and masks out actions that are deemed ineffective. Together, these techniques simplify the learning objective of individual agents, allowing MARL agents to learn more efficiently and scale to larger warehouse instances with more agents and locations while outperforming industry-standard heuristics and standard MARL algorithms.

Lay summary

Autonomous systems become increasingly prominent in various applications, including in autonomous driving, robotics, and industrial automation. For many of these systems, we would like autonomous decision-making entities to be able to learn behaviour from interactions with their environment and potentially other decision-making entities within their environment. This thesis is concerned with the machine learning paradigm of reinforcement learning that aims to train such decision-making agents by trial-and-error. To be able to learn from their interactions, these decision-making agents need to try many possible behaviours to learn which ones are desirable and which ones are not. This process of trying out different behaviours is referred to as exploration. In contrast, the process of exploiting learned knowledge to exhibit the desirable behaviour as best as possible is referred to as exploitation. In this thesis, we focus on the challenge of exploration in reinforcement learning and propose novel solutions to guide agents towards efficient data collection and learn to balance exploration and exploitation.

We first propose a novel approach to exploration in single-agent reinforcement learning that decouples the exploration and exploitation objectives of the agent. Our approach allows agents to learn separate strategies for exploration, i.e. to gather data that informs their learning, and exploitation which corresponds to learning a strategy to solve the respective decision-making task. We show that our approach can improve the efficiency of learning and the robustness of the agent.

We then shift our focus to multi-agent reinforcement learning, where multiple decision-making agents concurrently learn. These agents need to cooperate to solve challenging tasks. For example, one might envision a problem that requires multiple autonomous robots to jointly pick-up a heavy object, which is too heavy for each robot to carry it by themselves. We propose two novel approaches to exploration in multi-agent reinforcement learning that makes learning more efficient. The first approach leverages the similarity of data gathered by each agent. Following this similarity of data across agents, we show that agents can learn more efficiently by sharing their individual experiences with each other. The second approach relies on agents maintaining multiple predictions of future outcomes following their strategies. We can use these multiple predictions to determine in which situations there might be a possibility for agents to cooperate with each

other. We leverage this information to guide agents towards such situations, leading to agents to trying to cooperate more often, and as a consequence help them to learn how to cooperate more efficiently.

Lastly, we look at the real-world problem of automating warehouse logistics in which robots might navigate through a warehouse to pick-up and deliver items with the objective of completing orders. We formally define these problems and develop simulators that allow us to train decision-making agents to solve these tasks. To make learning more efficient, we simplify the task for each agent by first assigning each agent a part of the warehouse to navigate to before letting each agent individually decide which task to complete within their assigned part of the warehouse. Through this simplification, agents need to consider only a part of the many possible actions they could take within the warehouse, which simplifies their learning process. We show that our proposed approach improves the efficiency of learning and scalability to large warehouse instances compared to standard learning approaches and heuristic industry-standard solutions.

Acknowledgements

This thesis would not have been possible without the help and support of many people whom I am deeply grateful to.

First and foremost, I would like to thank my supervisor Stefano Albrecht. Already during my master's studies, I experienced Stefano as a reliable and supportive supervisor who guided me through the PhD application process. Throughout my PhD, this image of Stefano got increasingly reinforced. Whenever I was looking for advice or last-minute feedback towards submission deadlines (or thesis chapters), Stefano made sure to find time and help out. Beyond his reliability, I greatly appreciate how Stefano always encouraged me to carve my own path, even if this meant that I would temporarily leave to pursue research internships elsewhere. By being both reliable and supportive whenever needed, and giving myself the freedom to follow my own ideas, Stefano struck the right balance that helped me to grow as a researcher and individual during the past years. Lastly, I would like to thank Stefano for providing me and Filippos with the unique opportunity to write a textbook together. Throughout the entire journey, Stefano treated us as equal partners and trusted in our judgement and ability to communicate our understanding of the field. I consider myself very fortunate to have been a PhD student under Stefano and am deeply grateful for all the support and opportunities he has given me.

I would also like to thank everyone at the Autonomous Agents Research Group in Edinburgh for creating a fun and productive environment to learn and work in. Everyone has contributed to making my time in Edinburgh an experience that I will always look back on fondly. In particular, I would like to thank Filippos, Georgios, Arrasy, and Samuel for being great friends and colleagues throughout my PhD. Our discussions constantly sparked new inspiration, motivation, and encouragement without which this journey would have not been the same. I also thank my office mates in Edinburgh, especially Asif, Eric, Nick, Ibrahim, Resul, Adarsh, Mahesh, and Elliot, for all our shared conversations, lunch breaks, and much needed distractions.

I am deeply grateful to Tom Spink without whom I would have not embarked on this journey. As my personal tutor during my master's studies, Tom calmly and patiently responded to my endless flood of anxious questions about the process of doing a PhD and how to apply to PhD programs. His advice has been invaluable,

and most importantly, he reassured me to be confident in myself and take the leap to apply. For his reassurance, patience, and advice at a time when I felt anxious and overwhelmed, I am deeply grateful.

I am fortunate to have had the opportunity to collaborate with many great researchers during internships from whom I have learned a lot. I would like to thank Aleks who has been a great mentor, collaborator, and friend during and after my internship with Dematic. I greatly enjoyed our close collaboration and many discussions on even seemingly small implementation details and decisions, and I am grateful that he always made sure to find time for discussions despite being on opposite sides of the planet. I thank all my collaborators and friends at Huawei who contributed to an intense but productive summer. Special thanks goes to David and Jun for their mentorship and perspective, as well as to Alexandre, Matthieu, Juliusz, and Lisa for our discussions on research and life who have made the experience more collaborative and enjoyable. I would also like to thank the entire game intelligence team at Microsoft Research in Cambridge and my collaborators at XEmTech. They showed me how kind, supportive, and productive a work environment can be, all while teaching me about many technical, research, and soft skills. I am particularly grateful to Sam for his unwavering support and mentorship during and after my internship whose insightfulness is only exceeded by his kindness. I would also like to express special thanks to Sergio, Dave, Anssi, and Raluca for teaching me how to be a better engineer, to Yuhan, Anssi, and Dave for making sure that we somehow got all results together in time for a submission, to Logan and Andrea for letting me find my own path within the project, and to my fellow interns Tarun, Gunshi, Eloi, and Adam for the many conversations and discussions we had together that inspired, and sparked ideas.

I would also like to thank Prof. Subramanian Ramamoorthy and Prof. Karl Tuyls for examining this thesis. I am grateful for their thoughtful feedback and suggestions that have helped to improve the quality of this thesis, and for making the daunting process of the viva feel less intimidating and even enjoyable. I would also like to thank the Institute of Perception, Action and Behaviour for travel funding that allowed me to attend conferences and workshops during my PhD, and the organisers of the Heidelberg Laureate Forum who provided me with the unique opportunity to meet inspiring laureates and ambitious researchers. I am grateful for the mentorship of Marc Lanctot whose guidance on research and career advice have been invaluable.

On a personal note, I would like to thank my parents and brothers for their unconditional love and support. They sparked my curiosity, always encouraged me to ask questions, showed me the beauty that lies in understanding the world, and supported me to pursue my dreams and ambitions, all while keeping me grounded. I am deeply grateful for everything they have done for me. Lastly, I would like to thank Chentian for being a constant source of joy, love, and support throughout my PhD. She showed me how to be a more patient, kind, and considerate person, all while being endlessly patient and supportive of me. I am incredibly grateful for all the memories we have shared together and look forward to many more to come.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Lukas Schäfer)*

Contents

List of Figures

List of Tables

Acronyms

A2C advantage actor-critic. 20

AGV automated guided vehicle. 130

BPush boulder-push environment. 113

COMA counterfactual multi-agent policy gradients. 31

CTA closest task assignment; heuristic for warehouse order picking. 139

CTDE centralised training with decentralised execution. 26

CVaR conditional value-at-risk. 114

DDPG deep deterministic policy gradient. 31

DDQN double deep Q-networks. 19

DeA2C decoupled advantage actor-critic. 43

Dec-POMDP decentralised partially observable Markov decision process. 23

DeDQN decoupled deep Q-networks. 43

DePPO decoupled proximal policy optimisation. 43

DeRL decoupled reinforcement learning. 36, 40

DQN deep Q-networks. 18

EMAX ensemble value functions for multi-agent exploration. 101, 103

FM follow me; heuristic for warehouse order picking. 138

GTP Goods-to-person warehouse paradigm. 130

IA2C independent advantage actor-critic. 26, 86

ICM intrinsic curiosity module. 38

IDQN independent deep Q-networks. 26

IGM individual-global-max. 27

IL independent learning. 25

IPPO independent proximal policy optimisation. 26

IQM interquartile mean. 113

IS importance sampling. 41, 83

LBF level-based foraging. 58, 63

MAA2C multi-agent advantage actor-critic. 31

MADDPG multi-agent deep deterministic policy gradient. 31

MAPPO multi-agent proximal policy optimisation. 31

MARL multi-agent reinforcement learning. 4, 23

MDP Markov decision process. 13

MPE multi-agent particle environment. 60

PDM pick, don't move; heuristic for warehouse order picking. 139

POMDP partially observable Markov decision process. 14

POSG partially observable stochastic game. 22

PPO proximal policy optimisation. 21

PTG Person-to-goods warehouse paradigm. 130

RIDE rewarding impact-driven exploration. 39

RL reinforcement learning. 1, 15

RND random network distillation. 39

RNN recurrent neural network. 19

RWARE multi-robot warehouse environment. 58, 64

SEAC shared experience actor-critic. 80, 82

SEDQN shared experience deep Q-networks. 94

SMAC StarCraft multi-agent challenge. 62

SNAC shared network actor-critic; refers to IA2C with parameter sharing. 86

TA-RWARE task assignment multi-robot warehouse environment. 132

UCB upper confidence bound. 101

VDN value decomposition networks. 28

Chapter 1

Introduction

Reinforcement learning (RL) (Sutton and Barto, 2018) as a sub-field of machine learning is concerned with training decision-making agents by trial-and-error. Agents repeatedly interact with their (simulated or real) environment and receive numerical feedback after each action that indicates how well the agent is doing. Using this feedback, the agent can learn the desired behaviour that maximises such feedback. RL has seen impressive results in various domains, including but not limited to playing complex video games (Berner et al., 2019; Vinyals et al., 2019; Wurman et al., 2022), board games (Silver et al., 2016; Bakhtin et al., 2022; Pérolat et al., 2022), and learning control policies for robotics (Akkaya et al., 2019; Wu et al., 2023), warehouse logistics automation (Shetty et al., 2020; Krnjaic et al., 2024), and autonomous driving (Kendall et al., 2019; Zhou et al., 2021). Similar to other fields of machine learning, training RL agents in complex tasks requires large amounts of data and computational resources. However, RL is unique in its collection and usage of training data. Many fields of machine learning, including the majority of works in computer vision (e.g., Krizhevsky et al., 2012; He et al., 2016; Dosovitskiy et al., 2021), natural language processing (e.g., Devlin et al., 2018; Brown et al., 2020; Touvron et al., 2023; Team et al., 2023b), and recommender systems (e.g., Covington et al., 2016; He et al., 2017; Kang and McAuley, 2018), are trained on previously generated datasets that often leverage human-generated content such as text written by humans. These approaches in combination with immense computational resources and technical advancements, led to a surge in machine learning applications such as computer vision models that can detect objects in images with high accuracy, recommendation systems that can suggest products, films, or music one might be interested in based on

user data, and most recently large language models that power chat-based AI assistants like ChatGPT (Brown et al., 2020).

In contrast to this approach, training RL agents typically requires the agent to interact with its environment to collect its own data. We note that the subfield of offline reinforcement learning (Levine et al., 2020) represents a deviation from this paradigm. Offline RL is concerned with training RL agents from datasets of previously collected behavioural demonstration data and, thus, does not require nor allow the agent to gather its own data. In this thesis, we focus on the more common "online" RL setting in which the agent needs to interact with its environment to gather training data. This seemingly subtle difference of requiring agents to collect their own data leads to major challenges in RL. First, the learning agent needs to solve two interconnected problems in this setting: (1) they need to explore, i.e. to traverse the problem space of an environment, to collect information on what behaviour might be promising, and (2) they need to exploit this knowledge to solve the task given their current understanding of the environment. The challenge of balancing these interconnected processes is also known as the *exploration-exploitation trade-off*. Second, it is highly non-trivial to identify how the agent should behave in order to explore the environment effectively, i.e. to gather information that is useful for solving the task. Different exploration strategies can lead to vastly different data so the efficacy of exploration strategies depends on the task at hand. Third, RL is most commonly concerned with *sequential* decision making, i.e. the agent needs to make a series of decisions over time to solve the task. This sequential nature of the problem further complicates exploration as the agent needs to consider the long-term consequences of its actions. For example, an agent might need to solve a maze by reaching a goal location that is hidden behind a locked door. To reach the goal and receive a reward, the agent needs to first explore the environment to find and collect the key. However, the agent can only infer the importance of obtaining the key after successfully using the key to open the door and reaching the goal. If the agent collects the key but does not manage to reach the goal, it receives no feedback that the key could be useful. These long-term consequences of actions without immediate feedback make it challenging for the agent to learn accurate values and how to explore effectively.

The exploration process by which the agent gathers data strongly impacts the learning process. To learn efficiently, the agent needs to systematically explore

the environment to gather data that, either, directly informs the agent about the efficacy of its actions, or helps the agent to learn more about the environment and make better decisions in the future. Therefore, the process of exploration is closely tied to the concept of *sample efficiency* in RL. Sample efficiency in RL corresponds to how efficient the learning process is with respect to the data acquired during learning. This can be measured by the number of samples or environment interactions required by the agent to solve a task or to reach a particular level of performance. In some tasks, being more sample efficient might enable the agent to learn a solution in fewer interactions with the environment, which can be crucial in real-world settings, such as robotics, where data collection can be costly and time-consuming. In other tasks, being more sample efficient might enable the agent to reach a higher level of performance within the same number of interactions with the environment. We emphasise that sample efficiency does not necessarily imply that the agent needs less time to learn a solution, but rather that the agent makes more effective use of the data it collects, or collects data more effectively, to learn a solution. It is possible for an agent to be more sample efficient but needing more time to learn a solution due to the learning process requiring more computation per sample.

As a key challenge of RL, exploration has received significant attention in the research community. A plethora of approaches have been proposed, many of which follow one of three key ideas to encourage exploration: add random noise to the agent's policy (Watkins and Dayan, 1992; Silver et al., 2014; Mnih et al., 2015; Lillicrap et al., 2016) or encourage highly stochastic policies (Ziebart et al., 2008; Mnih et al., 2016; Haarnoja et al., 2018), maintain optimistic estimates of outcomes of previously unvisited states or actions (Auer et al., 2002; Osband et al., 2016a; Ciosek et al., 2019; Rashid et al., 2020a; Lee et al., 2021), and intrinsically incentivise exploration in parts of the environment which the agent has rarely visited or does not yet understand, akin to the concept of curiosity (Oudeyer and Kaplan, 2008; Barto, 2013). In particular the last type of intrinsically motivated exploration methods have been found to be highly effective in environments in which informative feedback in the form of (non-zero) rewards is sparse (Bellemare et al., 2016; Ostrovski et al., 2017; Taïga et al., 2020; Pathak et al., 2017; Burda et al., 2019b). However, the addition of intrinsic motivation to the learning process necessitates explicit balancing of the intrinsic and extrinsic rewards to ensure that the agent does not become overly focused on exploration and neglect

the task at hand. The first contribution of this thesis is in answering this open research question:

1. How can intrinsically motivated reinforcement learning agents effectively balance exploration and exploitation in the single-agent setting?

After addressing this research question, we will shift our attention to *multi-agent reinforcement learning* (MARL) (Albrecht et al., 2024). This field is concerned with concurrently training multiple agents that interact with a shared environment and each other, either to cooperate, compete, or a mixture of both. Exploration in this setting is particularly challenging since agents need to adapt their behaviour and exploration strategies to the behaviour of other agents. For example, in a task where agents need to coordinate to pick up a heavy object, which neither of the agents can pick up by themselves, the agents need to learn to coordinate their actions to successfully solve the task. Formally speaking, the action space of all agents together grows exponentially with the number of agents, leading to a large space of actions that needs to be explored. Previous research frequently extends single-agent exploration strategies to the multi-agent setting, for example by integrating existing single-agent intrinsic reward formulations in MARL (Iqbal and Sha, 2019; Schäfer, 2019) or by incentivising agents to explore the multi-agent specific coordination between agents (Wang et al., 2020b; Zheng et al., 2021). Other research coordinates exploration of multiple agents by providing agents with a common goal or strategy (Mahajan et al., 2019; Liu et al., 2021b) or by identifying and starting exploration from interesting initial states (Ryu et al., 2022). However, little research has been conducted on how to effectively leverage the different perspectives and information of multiple agents to improve exploration, and how to guide the exploration of agents towards states and actions that have the potential for cooperation. In this thesis, we contribute to filling this gap by answering the following research questions and proposing novel algorithms to address them:

2. How can multiple learning agents share experiences with each other in order to explore and learn more efficiently?

3. How can multiple learning agents identify states and actions with the potential for cooperation and guide their exploration towards these states and actions?

Lastly, this thesis will consider the real-world problem of warehouse logistics (Zong et al., 2021; Yan et al., 2022) in which exploration and learning effi-

ciency are a pivotal concern due to the scale of warehouses and the cost involved in the training of such systems. We consider this setting as a case study on how MARL can be used to efficiently train decision-making agents to coordinate and solve real-world problems.

1.1 Scope and Limitations

To calibrate the reader's expectations about the scope and limitations of this thesis, we briefly discuss the focus of this thesis, and explicitly state any aspects that are not considered in this work. We note that these alternative approaches and research directions are important in their own right, and are not considered in this thesis to maintain a clear focus on the research questions and contributions outlined above.

Throughout this thesis, we focus on reinforcement learning as a fundamental approach to learn decision-making policies for sequential decision making problems. We note that there are alternative approaches to decision making that we do not consider in this thesis which commonly make different assumptions about the problem. For example, imitation learning approaches learn decision making from provided demonstrations of behaviour and, as the same suggests, learn to imitate such demonstrated behaviour. For planning approaches, it is commonly assumed that the agent has access to a (possibly approximate) model of the environment that allows to simulate the consequences of actions and plan ahead. In contrast to these approaches, reinforcement learning learns decision making from interactions with the environment and, in its most general form, does not require a model of the environment. This type of reinforcement learning algorithms is also known as model-free reinforcement learning algorithms, and stands in contrast to model-based reinforcement learning algorithms that learn a model of the environment as part of their learning process. Such a model might allow model-based reinforcement learning algorithms to plan ahead, akin to planning approaches, or learn a decision-making policy in the abstract latent space of their learned model. In this thesis, we focus on model-free reinforcement learning algorithms, in particular deep reinforcement learning algorithms, that represent their decision making policies using deep neural networks.

The research presented in this thesis is guided by the goal of making RL more sample efficient and the aforementioned research questions. To enable sample

efficient RL, agents need to gather data in a systematic way. Therefore, the goal of sample efficiently directly relates to the algorithmic challenges of exploration in RL. We note that the challenge of exploration gives rise to related aspects that are motivated by objectives that are not sample efficiency. For example, there are important safety implications for exploration, particularly in real-world systems such as robotics, where exploratory actions might cause damage to the physical agent or its environment. In this thesis, we focus on RL application in simulated environments and algorithmic innovations to improve the sample efficiency of RL agents where safety is not a primary concern.

Within the setting of multi-agent reinforcement learning, which is integral to this thesis, we focus on cooperative and mixed-objective problems that require agents to coordinate their actions. In particular, we do not consider competitive tasks, such as zero-sum games like chess and Go, in this thesis. These settings introduce alternative challenges to cooperative tasks that are not discussed in this thesis. We also do not consider specific aspects of multi-agent coordination, such as how to learn communication between agents or how to model the behaviour of other agents in the environment.

Lastly, we emphasise that improved sample efficiency through novel exploration strategies, as proposed in this thesis, does not represent the only approach to make RL more efficient. Orthogonal to the exploration challenge, RL can become more efficient by improving its generalisation capabilities. If an agent can be trained once across a suit of tasks and learn a policy that effectively solves all these tasks, it might require fewer total training samples than training individual agents for each task. Such generalisation can be achieved through transfer learning, in which an agent might learn a policy in one or multiple tasks and then learn to transfer this policy to a new task. Alternatively, generalisation can be achieved through meta-learning or multi-task learning, in which agents are directly trained across a distribution of tasks to learn generalisable policies. We note that these approaches are orthogonal to the exploration challenge in that they could be combined with novel exploration strategies to further improve the efficiency of RL.

1.2 Thesis Outline and Contributions

The structure of this thesis and its eight chapters is outlined in Figure 1.1.

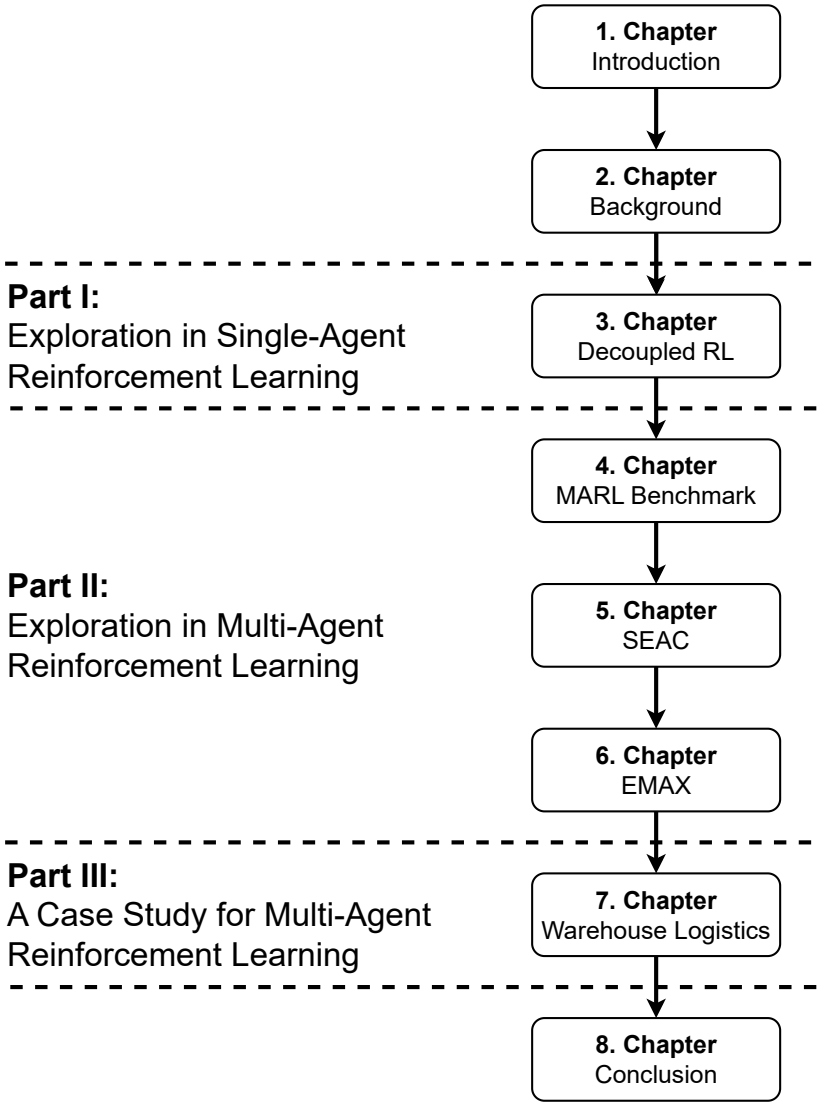


Figure 1.1: Outline of the thesis and its structure.

Chapter 2 provides the preliminary background on reinforcement learning, deep reinforcement learning, and multi-agent reinforcement learning, including the definitions and formalisms of problem settings as well as common algorithms. Subsequent chapters will refer to these definitions and further build upon introduced algorithms.

Chapter 3 discusses the challenge of exploration in single-agent reinforcement learning with a focus on intrinsically motivated exploration. We identify research questions that arise from the challenge of balancing intrinsically motivated exploration and exploitation, and propose a novel algorithm that decouples these

challenges by learning separate policies for exploration and exploitation. We show that our approach of decoupled reinforcement learning (DeRL) can lead to improved sample efficiency and robustness to varying hyperparameters that determine the balancing of intrinsic and extrinsic rewards.

In Chapter 4, we transition to the multi-agent setting and present a comprehensive empirical evaluation of nine MARL algorithms across 25 cooperative tasks. Following the analysis of this benchmark, we identify remaining challenges salient in deep MARL to efficiently train agents to cooperate in tasks where informative feedback is sparse.

Motivated by this identified challenge, Chapter 5 introduces the novel shared experience actor-critic (SEAC) algorithm for exploration in MARL. SEAC leverages the fact that many multi-agent environments require agents to learn similar but not identical behaviour and proposes to share experiences across agents. The chapter describes the algorithm and studies its efficacy in a variety of cooperative and mixed cooperative-competitive tasks.

Chapter 6 introduces the novel framework of ensemble value functions for multi-agent exploration (EMAX) to extend existing value-based algorithms for multi-agent tasks. The approach proposes a novel exploration strategy that identifies and leads exploration towards states and actions with the potential for cooperation, and computes more robust target values that stabilise training. The chapter describes the approach, evaluates the novel framework as an extension of three commonly-used MARL algorithms, and demonstrates the resulting improvements in both sample efficiency and stability of training.

Chapter 7 discusses the problem of warehouse logistics and how MARL can be used to efficiently train decision-making agents to coordinate in such a setting. The chapter is the result of an industry internship and ongoing research collaboration with Dematic GmbH, a global company focusing on warehouse automation and robotics. We formalise two settings of warehouse logistics problems, introduce simulated environments for both settings, and propose a hierarchical decomposition and masking of ineffective actions to improve the efficiency of training. The proposed approach is evaluated in both simulated settings and demonstrates improved sample efficiency and scalability to larger warehouse instances compared to industry-standard heuristics and standard MARL baselines.

Finally, Chapter 8 concludes the thesis by summarising the contributions and discussing potential future research directions.

1.3 Publications

Parts of this thesis have been published in several publications which are the result of collaborative research projects. Below, we list all publications which are included in this thesis as well as clearly state the individual contributions of the author (Lukas Schäfer):

- Chapter 3: **Lukas Schäfer**, Filippos Christianos, Josiah P. Hanna, and Stefano V. Albrecht. "Decoupled reinforcement learning to stabilise intrinsically-motivated exploration." In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. 2020.
  Available at arXiv 2107.08966
  Contributions: I led the research project including identifying the challenges of intrinsically motivated exploration, ideation and development of the DeRL algorithm, implementation of the algorithm and experiments, and the writing of the paper. The fellow co-authors contributed to the discussion of the project, and the writing of the paper.

- Chapter 4: Georgios Papoudakis, Filippos Christianos, **Lukas Schäfer**, and Stefano V. Albrecht. "Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks." In *Advances in Neural Information Processing Systems, Track on Datasets and Benchmarks*. 2021.
  Available at arXiv 2006.07869
  Contributions: I contributed to the design of the benchmark, including the selection of algorithms, environments, and evaluation metrics. The joint leading co-authors Papoudakis and Christianos led the implementation of algorithms and the experiments. Analysis of the results and writing of the paper was a joint effort shared across all authors.

- Chapter 5: Filippos Christianos, **Lukas Schäfer**, and Stefano V. Albrecht. "Shared experience actor-critic for multi-agent reinforcement learning." In *Advances in Neural Information Processing Systems*. 2020.
  Available at arXiv 2006.07169
  Contributions: I contributed to the formal derivation of the loss function of the algorithm (which has been generalised as part of this thesis), led the implementation and experimentation of the complementary SEDQN algorithm, and contributed to the writing and analysis of the paper. Christianos

led the research project, including the ideation of the SEAC algorithm as well as its implementation and experiments.

- Chapter 6: **Lukas Schäfer**, Oliver Slumbers, Stephen McAleer, Yali Du, Stefano V. Albrecht, and David Mguni. "Ensemble value functions for efficient exploration in multi-agent reinforcement learning." In *Adaptive and Learning Agents Workshop at the AAMAS conference*. 2023.
  Available at arXiv 2302.03439
  Contributions: I led the research project through ideation, development of the EMAX algorithm, implementation, conducting experiments, and the writing of the paper. Slumbers contributed by assisting in running experiments. All co-authors contributed to the discussion of the project, and providing feedback on the writing. This work has partially been conducted during an internship at Huawei Noah's Ark Lab.

- Chapter 7: Aleksandar Krnjaic, Raul D. Steleac, Jonathan D. Thomas, Georgios Papoudakis, **Lukas Schäfer**, Andrew Wing Keung To, Kuan-Ho Lao, Murat Cubuktepe, Matthew Haley, Peter Börsting, Stefano V. Albrecht "Scalable multi-agent reinforcement learning for warehouse logistics with robotic and human co-workers." In *Proceedings of the 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2024.
  Available at arXiv 2212.11498
  Contributions: I co-led the implementation of the Dematic Person-to-Goods (PTG) simulator as well as the foundational codebase with MARL baselines (IA2C, SNAC, SEAC) together with Krnjaic. I also contributed to the writing of the paper and the formalisation of the problem setting. This work has been conducted during an internship and ongoing collaboration with Dematic GmbH.

Not all research conducted over the past four years fits into the scope of this thesis. Further publications to which I have contributed include:

- Stefano V. Albrecht, Filippos Christianos, and **Lukas Schäfer**. "Multi-agent reinforcement learning: Foundations and modern approaches." In *MIT Press*. Cambridge, MA, USA. 2024.
  Available at marl-book.com

- Trevor McInroe, **Lukas Schäfer**, and Stefano V. Albrecht. "Multi-Horizon Representations with Hierarchical Forward Models for Reinforcement Learning." In *Transactions on Machine Learning Research*. 2023.
  Available at arXiv 2206.11396

- Rujie Zhong, Duohan Zhang, **Lukas Schäfer**, Stefano V. Albrecht, and Josiah P. Hanna. "Robust on-policy sampling for data-efficient policy evaluation in reinforcement learning." In *Advances in Neural Information Processing Systems*. 2022.
  Available at arXiv 2111.14552

- **Lukas Schäfer**. "Task generalisation in multi-agent reinforcement learning." In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. 2022.
  Available at ACM Digital Library

- **Lukas Schäfer**, Filippos Christianos, Amos Storkey, and Stefano V. Albrecht. "Learning task embeddings for teamwork adaptation in multi-agent reinforcement learning." In *Workshop on Generalization in Planning at the NeurIPS conference*. 2023.
  Available at arXiv 2207.02249

- Alain Andres, **Lukas Schäfer**, Esther Villar-Rodriguez, Stefano V. Albrecht, and Javier Del Ser. "Using Offline Data to Speed-up Reinforcement Learning in Procedurally Generated Environments." In *Adaptive and Learning Agents Workshop at the AAMAS conference*. 2023.
  Available at arXiv 2304.09825

- **Lukas Schäfer**, Logan Jones, Anssi Kanervisto, Yuhan Cao, Tabish Rashid, Raluca Georgescu, David Bignell, Siddhartha Sen, Andrea Treviño Gavito, and Sam Devlin. "Visual Encoders for Data-Efficient Imitation Learning in Modern Video Games." In *arXiv preprint*. 2023.
  Available at arXiv 2312.02312

Chapter 2

Problem Settings and Preliminary Algorithms

In this chapter, we introduce necessary formalisms to define sequential decision-making problems before introducing the paradigm of reinforcement learning with common algorithms to solve such problems. Subsequent chapters will build on these algorithms to define novel algorithms for decision-making problems defined under the introduced formalisms.

2.1 Markov Decision Processes

The most foundational formalism for sequential decision-making problems is a *Markov decision process* (MDP) (Bellman, 1957; Howard, 1964). A MDP is defined as a tuple $(S, A, \mathcal{T}, \mathcal{R}, \mu, S_T, \gamma)$. S and A denote the sets of states and actions, respectively, and $\mathcal{T} : S \times A \mapsto \Delta(S)$ represents the transition function defining a probability distribution over the next state given current state and applied action. A scalar reward is given at any transition following the reward function $\mathcal{R} : S \times A \times S \mapsto \mathbb{R}$. μ defines a distribution over initial states, and $S_T \subset S$ defines a set of terminal states.

Under this formalism, a decision-making agent sequentially interacts with the environment which is initialised in a random state $s^0 \sim \mu$. At each time step t, the agent observes the current state $s^t \in S$ and selects an action $a^t \in A$ sampled from its policy π. The policy defines a probability distribution over the action space conditioned on a state, i.e. $\pi : S \mapsto \Delta(A)$. After applying the action a^t, the environment transitions to a next state $s^{t+1} \in S$ sampled from the transition

function $\mathcal{T}(s^{t+1} \mid s^t, a^t)$, and the agent receives a reward $r^t = \mathcal{R}(s^t, a^t, s^{t+1})$. This sequential interaction terminates when the environment reaches a terminal state $s^t \in S_T$, or after a fixed number of time steps. In the following, we denote the terminal time step as T and refer to a single interaction loop from initial state to terminal state as an *episode*. The objective of the decision-making agent is to learn a policy π that maximises the *expected discounted returns*

$$\mathbb{E}_\pi\Big[G^t\Big] = \mathbb{E}_{s^0 \sim \mu, a^t \sim \pi(s^t)}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s^t, a^t, s^{t+1})\right] \tag{2.1}$$

with discount factor $\gamma \in [0, 1)$. We note that episodes can continue indefinitely but in this thesis, we focus on episodic tasks with a finite number of time steps T per episode. In this episodic setting, the discount factor γ can be omitted since the returns are guaranteed to be finite but we will keep it for generality.

2.2 Partially Observable Markov Decision Processes

In some environments, it might not be realistic for the agent to have full information about the state of the environment. These environments can be formulated as a *partially observable Markov decision process* (POMDP) (Kaelbling et al., 1998) which builds on the MDP formalism. A POMDP is defined as a tuple $(S, O, A, \mathcal{T}, \mathcal{O}, \mathcal{R}, \mu, S_T, \gamma)$. S, A, $\mathcal{T}$, $\mathcal{R}$, μ, S_T, and γ are defined identically to MDPs. Additionally, a POMDP defines the observation space O as the set of all possible observations an agent might receive. $\mathcal{O} : S \times A \times \Delta(O)$ is the observation probability function that maps states and actions to a probability distribution of next observations.

At every time step t, the agent receives an observation $o^t \in \mathcal{O}$ and decides on an action a^t to apply. The policy of an agent is now conditioned on the episodic observation history $h^t \in H$, i.e. all observations within the current episode $h^t = (o^0, o^1, \ldots, o^t)$, instead of the full environment state. Given the action, the POMDP transitions into a next state $s^{t+1} \in S$ sampled according to $\mathcal{T}(s^{t+1} \mid s^t, a^t)$, the next observation $o^{t+1} \in O$ is sampled from $\mathcal{O}(o^{t+1} \mid s^t, a^t)$, and the agent receives a reward $r^t = \mathcal{R}(s^t, a^t, s^{t+1})$. Identical to MDPs, the objective in a POMDP is to learn a policy that maximises the expected discounted returns defined for partially observable environments:

$$\mathbb{E}_\pi\Big[G^t\Big] = \mathbb{E}_{s^0 \sim \mu, a^t \sim \pi(h^t)}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s^t, a^t, s^{t+1})\right] \tag{2.2}$$

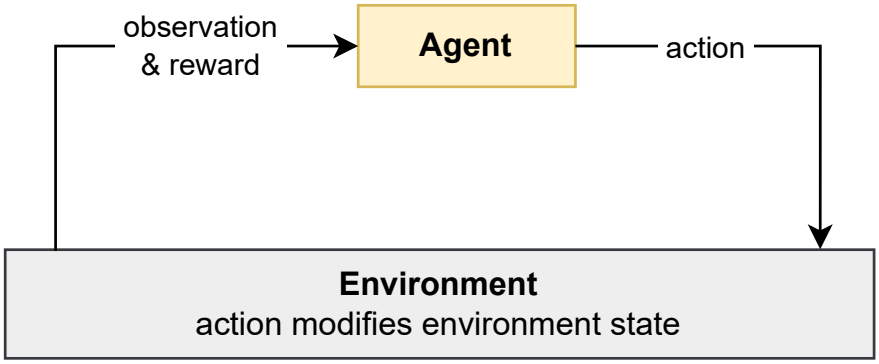


Figure 2.1: Illustration of the interaction loop for single-agent decision-making problems. The agent repeatedly receives observations at each time step and selects actions a to interact with the environment which then transitions into a new state and provides the agent with a reward. From Albrecht et al. (2024).

The interaction loop for such single-agent decision-making problems is illustrated in Figure 2.1.

2.3 Single-Agent Reinforcement Learning

Reinforcement learning (RL) is a general framework of machine learning that aims to learn policies to solve decision-making problems. In this section, we introduce building blocks and common RL algorithms that learn policies to solve MDPs and POMDPs. For generality, we follow the notation for POMDPs and condition policies on the history of observations h^t but note that the same algorithms can be applied to MDPs by conditioning policies on the state s^t. For a comprehensive introduction to RL, we refer to Sutton and Barto (2018) and Albrecht et al. (2024) for deep RL.

2.3.1 Value Functions

Value functions are essential components of RL algorithms. Formally, a value function estimates the expected returns as defined in Equation 2.2. We note that the expectation of expected returns depends on the policy π that is used to select actions. We denote value functions with their underlying policy as a superscript. A *state-value function* that estimates the expected returns of following policy π from state s is denoted with $V^\pi(s)$. For partially observable environments, the

value function is conditioned on the history of observations h^t:

$$V^\pi(h^t) = \mathbb{E}_{a^t \sim \pi(h^t)}\Big[G^t\Big] \tag{2.3}$$

$$= \mathbb{E}_{a^t \sim \pi(h^t), s^{t+1} \sim \mathcal{T}(s^t, a^t)}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s^t, a^t, s^{t+1})\right] \tag{2.4}$$

Similarly, we can define an *action-value function* $Q^\pi(s, a)$ that estimates the expected returns of following policy π from state s after taking action a. For partially observable environments, the action-value function is conditioned on the history of observations h^t instead of the state and can be defined as follows:

$$Q^\pi(h^t, a^t) = \mathbb{E}\Big[G^t\Big] \tag{2.5}$$

$$= \mathbb{E}_{s^{t+1} \sim \mathcal{T}(s^t, a^t)}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s^t, a^t, s^{t+1})\right] \tag{2.6}$$

In the following, we will often use the notation $V(h^t)$ and $Q(h^t, a^t)$ to refer to the value functions without explicitly denoting the conditioned policy when the policy is clear from the context.

Based on these definitions of state-value and action-value functions, we can define the optimal value functions $V^*(h^t)$ and $Q^*(h^t, a^t)$ which estimate the expected returns of following the optimal policy π^* from state s or after taking action a. The optimal value functions are defined as:

$$V^*(h^t) = \max_\pi V^\pi(h^t) \tag{2.7}$$

$$Q^*(h^t, a^t) = \max_\pi Q^\pi(h^t, a^t) \tag{2.8}$$

Following the Bellman equations from dynamic programming (Bellman, 1966), we can define rules that allow iterative updates of value functions. This process is the foundation of many RL algorithms since it allows to iteratively improve and learn value functions that converge to the true value functions as defined above under certain conditions (e.g., Melo, 2001).

2.3.2 Q-Learning and Deep Q-Networks

Q-learning (Watkins and Dayan, 1992) is one of the most commonly built upon RL algorithms. Q-learning directly learns the optimal action-value function $Q^*(h, a)$ to estimate the expected returns under the optimal policy with observation history h after taking action a. The algorithm starts with randomly or zero-initialised

value estimates for all history-action pairs and then iteratively updates the action-value function:

$$Q(h^t, a^t) \leftarrow Q(h^t, a^t) + \alpha \left(r^t + \gamma \max_{a \in A} Q(h^{t+1}, a) - Q(h^t, a^t) \right) \tag{2.9}$$

where $\alpha > 0$ is the learning rate that determines the magnitude of each update and represents a trade-off. For larger values of α, value estimates might get closer to convergence with fewer updates, but convergence might be less stable. In contrast, for small values of α convergence becomes more stable but might require more updates. The iterative update rule is based on the Bellman optimality equation for action-value functions and converges under certain conditions to the optimal action-value function $Q^*(h, a)$ (Melo, 2001). Based on the learned action-value function, the agent can derive the policy by selecting the action with the highest value estimate for the current observation history, i.e. $\pi(h^t) = \arg\max_a Q(h^t, a)$. For simplicity, we slightly abuse notation here by defining a deterministic policy based on the action-value function instead of defining the policy as a probability distribution over actions. To encourage exploration and ensure that many state-action pairs are visited, Q-learning typically uses an ϵ-greedy policy to interact with the environment to collect experiences, i.e. with probability ϵ the agent selects an action uniformly at random and with probability $1-\epsilon$ the greedy action with highest action-value estimate is followed. Since the learned policy and the policy that is used to collect experiences are different, Q-learning is considered a off-policy algorithm.

The Q-learning algorithm represents the learned action-value function as a large table with states or observation histories representing the rows and actions representing the columns. Each cell within the table then contains the estimated value for the particular state-action pair. We refer to RL algorithms with such tabular value functions as *tabular RL algorithms*. Tabular RL algorithms come with convergence guarantees (under certain conditions) but are limited in their ability to solve complex problems with large state or action spaces. One challenge of a tabular representation of the value function is its linear growth with the state and action space, so it becomes infeasible to store such a table for complex problems. Furthermore, tabular value functions update each value estimate in isolation. After encountering a state and action, the value estimate for this exact state-action pair is updated and no other value is changed. This can be helpful for formal guarantees but also implies that every possible state and action needs

to be visited multiple times to learn accurate value estimates which is infeasible in most problems.

To overcome these limitations, it is desirable to learn value functions using general techniques of function approximation. Most recently, neural networks and the field of deep learning (Rosenblatt, 1958; Fukushima, 1980; Rumelhart et al., 1986; Goodfellow et al., 2016) emerged as particularly effective and general-purpose function approximators. *Deep Q-networks* (DQN) (Mnih et al., 2015) is an approach that extends the tabular Q-learning algorithm with deep neural networks to represent an approximate value function. DQN represents the learned action-value function with a neural network that receives the state or observation history h as input and outputs the estimated value for each possible action. Due to the dependence on a finite action space, the tabular Q-learning algorithm and DQN are designed for discrete (finite) action spaces and are not applicable to continuous action spaces (without modifications). We denote the learned parameters of the value function with θ. To optimise the value function parameters θ, we minimise the following loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{(h^t, a^t, r^t, h^{t+1}) \sim \mathcal{D}} \left[\left(r^t + \gamma \max_{a \in A} Q(h^{t+1}, a; \bar{\theta}) - Q(h^t, a^t; \theta) \right)^2 \right] \tag{2.10}$$

with $\bar{\theta}$ denoting the parameters of a target network. The target network is of identical structure to the main value function network and is periodically updated by copying the parameters θ of the main value function. The loss is derived from the temporal difference error of the Q-learning update rule. The target network is used to stabilise training by providing a fixed target value for the temporal difference error for a period of time, thereby reducing the moving target problem of RL. Besides the neural network value function and target network, the DQN algorithm extends Q-learning by using a replay buffer of experiences $\mathcal{D}$ (Lin, 1992b). This buffer stores tuples of experiences (h^t, a^t, r^t, h^{t+1}) collected by interacting with the environment using an ϵ-greedy policy. For each update, the expected loss in Equation 2.10 is approximated by sampling a batch of experiences from the replay buffer and computing the mean loss over the batch. The parameters θ are then updated by minimising the loss using gradient descent. In practice, any gradient-based optimisation algorithm can be used to update the parameters θ to minimise the given loss. Most commonly, the Adam optimiser (Kingma and Ba, 2015) is used.

We note that the original DQN algorithm was proposed for fully observable

environments. To extend DQN to POMDPs, the observation history h is used as input to the neural network instead of the state s. To condition the value function on the observation history in an efficient manner, Hausknecht and Stone (2015) propose to use recurrent neural networks (RNNs) (Rumelhart et al., 1986) to process the observation history. RNNs are a class of neural networks that can sequentially process sequences of inputs by maintaining an internal state of the previous sequence. Commonly used architectures of RNNs include gated recurrent units (GRU) (Cho et al., 2014) and long short-term memory cells (LSTM) (Hochreiter and Schmidhuber, 1997).

Further extensions of the DQN algorithm have been proposed to improve the learning stability and sample efficiency of the algorithm. Double DQN (DDQN) (van Hasselt et al., 2016) observes that the computation of DQN target values as the maximum action-value estimate in the next state leads to overestimation bias. To reduce the overestimation of target values, DDQN identifies the greedy action in the next state using the main value function and then computes the target value for this action with the target network. For discussion on further extensions, we refer to the Rainbow algorithm (Hessel et al., 2018) which combines several extensions of DQN.

2.3.3 Policy Gradient Algorithms

We refer to Q-learning and similar derived algorithms as *value-based* RL algorithms since they learn value functions to estimate the expected returns and derive their policy from the value function. In contrast, *policy gradient* algorithms directly learn a policy π with parameters ϕ that is optimised to maximise the expected returns.

Most policy gradient algorithms derive their optimisation objectives from the policy gradient theorem (Sutton and Barto, 2018) which formulates an expression for the gradient of the expected returns under a policy with respect to the parameters of the policy:

$$\nabla_\phi J(\phi) = \mathbb{E}_{s^0 \sim \mu, a^t \sim \pi(h^t;\phi)} \Big[G^t \nabla_\phi \log \pi(a^t \mid h^t; \phi) \Big] \tag{2.11}$$

By iteratively following this gradient, the parameters of the policy ϕ can be optimised such that the policy maximises the expected returns. This theorem is applicable to any parameterised and differentiable policy. All following policy

gradient algorithms derive their optimisation objectives from this theorem by approximating the expectation with samples from the environment and policy, and by approximating the expected returns in various ways. We note that, in contrast to the previously introduced Q-learning algorithm, all policy gradient algorithms derived by this theorem are on-policy algorithms. This means that the policy that is used to collect experiences is the same policy that is optimised during training, and is required since the expectation of the policy gradient theorem Equation 2.11 is conditioned on the experiences being collected by the optimised policy π.

The REINFORCE algorithm (Williams, 1992) instantiates the policy gradient by computing Monte Carlo estimates of the returns. This approach is simple but has high variance in the gradient estimates due to each trajectory resulting from the stochasticity of the initial state distribution, policy, and transition function. To reduce variance of gradient estimates, *actor-critic* algorithms estimate expected returns by using a parameterised value function $V(h;\theta)$ with parameters θ. The value function of actor-critic algorithms is often referred to as the *critic*, and the policy is denoted as the *actor*. A commonly used actor-critic algorithm is the *advantage actor-critic* (A2C) algorithm (Mnih et al., 2016). A2C uses the advantage estimate

$$A(h^t, a^t) = Q(h^t, a^t) - V(h^t) \tag{2.12}$$

$$= r^t + \gamma V(h^{t+1}) - V(h^t) \tag{2.13}$$

to obtain action-conditioned value estimates with lower variance. To optimise the policy parameters ϕ and the value function parameters θ, A2C minimises the following loss functions for the policy:

$$\mathcal{L}(\phi) = \mathbb{E}_{s^0\sim\mu, a^t\sim\pi(h^t;\phi)}\Big[-\log\pi(a^t \mid h^t;\phi)\left(r^t + \gamma V(h^{t+1};\theta) - V(h^t;\theta)\right)\Big] \tag{2.14}$$

and the value function:

$$\mathcal{L}(\theta) = \mathbb{E}_{s^0\sim\mu, a^t\sim\pi(h^t;\phi)}\Big[\left(r^t + \gamma V(h^{t+1};\theta) - V(h^t;\theta)\right)^2\Big] \tag{2.15}$$

Both V and π are neural networks with parameters θ and ϕ, respectively, and are optimised by iteratively minimising the loss functions above. In practise, training can be further stabilised by sampling multiple trajectories in parallel to obtain batches of experiences to average over, computing n-step return estimates instead of 1-step returns (as defined above), and by using entropy regularisation

to encourage exploration (Mnih et al., 2016). For entropy regularisation, the policy loss is extended with a term of the policy entropy:

$$H(\pi(a^t \mid h^t; \phi)) = -\mathbb{E}_{a^t \sim \pi(h^t;\phi)}\Big[\log \pi(a^t \mid h^t; \phi)\Big] \tag{2.16}$$

with a hyperparameter to control the weight of the entropy term in the loss function. The entropy term encourages the policy to explore and deincentivises early convergence to suboptimal policies.

Proximal policy optimisation (PPO) (Schulman et al., 2017) extends the A2C algorithm to further improve the stability and sample efficiency. The algorithm is based on the idea of *trust region policy optimisation* (Schulman et al., 2015) that constrains the policy update to prevent the policy from diverging too far from the current policy. PPO achieves this constraint by clipping the policy loss in Equation 2.14 to a region around the current policy with importance sampling weights. The clipped policy loss is defined as:

$$\mathcal{L}(\phi) = \mathbb{E}_{s^0 \sim \mu, a^t \sim \pi(h^t;\phi)}\left[-\min\left(\begin{array}{l}\rho(h^t, a^t)A(h^t, a^t; \theta), \\ \text{clip}\left(\rho(h^t, a^t), 1-\epsilon, 1+\epsilon\right) A(h^t, a^t; \theta)\end{array}\right)\right] \tag{2.17}$$

with ϵ being a hyperparameter to control the size of the trust region, $A(h^t, a^t; \theta)$ being the advantage estimate computed from a state value function (Equation 2.12), and ρ denoting the importance sampling (IS) weights:

$$\rho(h, a) = \frac{\pi(a \mid h; \phi)}{\pi_\beta(a \mid h)} \tag{2.18}$$

given by the ratio of the current policy and the policy that was used to collect experiences, denoted with π_β.

The clipped policy loss ensures that the policy update does not deviate too far from the current policy and that experience is re-weighted to be on-policy using importance sampling weights. PPO further extends the algorithm by using multiple epochs of policy updates on the same batch of experiences to improve sample efficiency. The critic in PPO is trained using the same loss as in A2C (Equation 2.15).

2.4 Partially Observable Stochastic Games

So far in this chapter, we have introduced formalisms for single-agent decision-making problems under the MDP and POMDP formalisms and have seen common

RL algorithms to solve these problems. In the following, we extend these formalisms to multi-agent decision-making problems before introducing multi-agent reinforcement learning algorithms to solve these problems.

Stochastic games (Shapley, 1953), also known as Markov games (e.g. (Littman, 1994)), are extensions of the formalism of MDPs for multi-agent problems. In particular, we consider *partially observable stochastic games* (POSG) (Hansen et al., 2004) for N agents as an extension of POMDPs. A POSG is given by the tuple $(I, S, \mathbf{O}, \mathbf{A}, \mathcal{T}, \mathcal{O}, \{\mathcal{R}_i\}_{i \in I}, \mu, S_T, \gamma)$. Agents are indexed by $i \in I = \{1, \ldots, N\}$. S denotes the state space of the environment and $\mathbf{A} = A_1 \times \ldots \times A_N$ denotes the joint action space of all agents. Each agent has access to its local observations $o_i \in O_i$. The joint observation space is denoted $\mathbf{O} = O_i \times \ldots \times O_N$. $\mathcal{T} : S \times \mathbf{A} \mapsto \Delta(S)$ denotes the transition function of the environment and defines a distribution of successor states given the current state and the applied joint action. The observation transition function $\mathcal{O} : S \times \mathbf{A} \times \Delta(\mathbf{O})$ defines a distribution of next joint observations received by agents given the current state and joint action of all agents. $\mathcal{R}_i : S \times \mathbf{A} \times S \mapsto \mathbb{R}$ denotes the reward function for each agent i, and similar to previous formalisms μ defines a distribution over initial states, $S_T \subset S$ defines a set of terminal states, and $\gamma \in [0, 1)$ denotes the discount factor.

Each agent learns a policy $\pi_i(a_i^t \mid h_i^t)$ conditioned on its history of observations until time step t. Note that the history of agent i until time step t only includes the observations of agent i, i.e. $h_i^t = (o_i^0, o_i^1, \ldots, o_i^t)$. The objective of a POSG is for all agents to learn a joint policy $\pi = (\pi_1, \ldots, \pi_N)$ such that the expected discounted returns of each agent are maximised with respect to the policies of all other agents. The discounted returns for agent i can be written as

$$\mathbb{E}_\pi\left[G_i^t\right] = \mathbb{E}_{s^0 \sim \mu, a_i^t \sim \pi_i(h_i^t), \mathbf{a}_{-i} \sim \pi_{-i}(\mathbf{h}_{-i}^t)}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_i(s^t, \mathbf{a}^t, s^{t+1})\right] \tag{2.19}$$

where $\gamma \in [0; 1)$ denotes the discount factor, $\mathbf{a}^t = (a_1^t, \ldots, a_N^t)$ denotes the joint action, and subscript $-i$ denotes all agents but agent i, i.e. $\pi_{-i} = \pi \setminus \{\pi_i\}$ and $\mathbf{a}_{-i} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N)$. Formally, the objective is to learn a joint policy π that satisfies the following condition:

$$\forall_i : \pi_i \in \arg\max_{\pi_i'} \mathbb{E}_{a_i^t \sim \pi_i'(h_i^t), \mathbf{a}_{-i}^t \sim \pi_{-i}(\mathbf{h}_{-i}^t)}[G_i] \tag{2.20}$$

The interaction loop for multi-agent decision-making problems modelled as a POSG is illustrated in Figure 2.2.

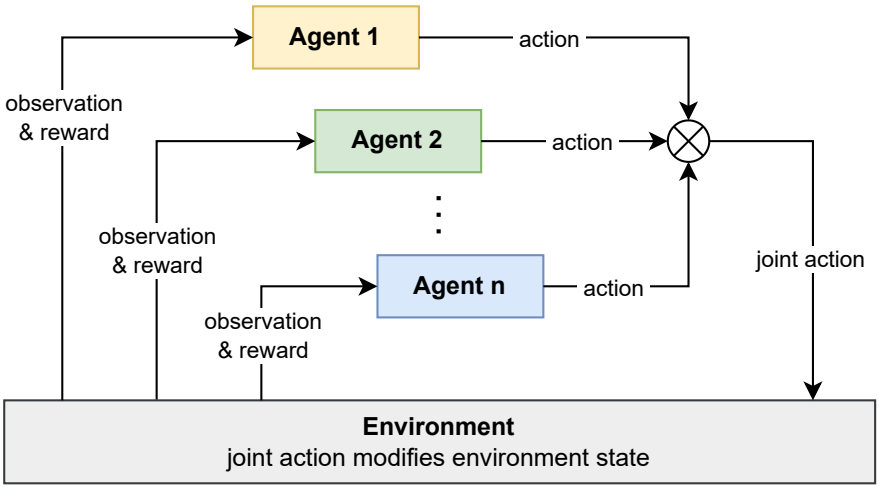


Figure 2.2: Illustration of the interaction loop for multi-agent decision-making problems. The agents repeatedly receives observations at each time step and selects actions a to interact with the environment. Based on the joint action $\mathbf{a}$, the environment transitions into a new state and provides each agent with a reward. From Albrecht et al. (2024).

2.5 Decentralised Partially Observable Markov Decision Processes

Stochastic games allow different agents to optimise for different objectives, given by their reward functions. In the special case of *common reward* tasks, all agents optimise for the same reward function, i.e. $\forall i, j \in I : \mathcal{R}_i = \mathcal{R}_j$. Environments that fulfil this property are also commonly referred to as fully cooperative and can be defined under the formalism of *decentralised partially observable Markov decision processed* (Dec-POMDP) (Bernstein et al., 2002; Oliehoek and Amato, 2016). A Dec-POMDP can be formalised almost identical to a POSG with the only difference being that it only has a single common reward function denoted with $\mathcal{R}$ without any subscript.

2.6 Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning (MARL) is concerned with learning policies in multi-agent systems. In such systems, as formalised by a POSG defined above, multiple agents concurrently learn policies for sequential decision making within the environment. In this section, we will provide the necessary background for

the following chapters. For a comprehensive introduction to MARL, we refer to Albrecht et al. (2024).

One concept that underlies most MARL algorithms and all algorithms that will be discussed in this thesis is *decentralised execution*. Under this paradigm we assume that all agents need to be able to execute their policies, i.e. to select actions, in a decentralised manner. This is particularly important in partially observable environments where each agent may only condition their policy on the history of their observations and may not assume access to the observations or actions of other agents or the full state of the environment. In contrast to single-agent RL where we only learn a single policy, this has the major distinction that different agents might receive different inputs and observe different information about the environment. MARL can be seen as a decomposition of the large joint action space into smaller individual action spaces across multiple learning agents. This allows MARL to learn in tasks where the joint action space is too large for single-agent RL techniques, which quickly become infeasible. Imagine a warehouse in which 12 robotic workers have to be controlled. Each of these robots chooses one of 6 actions at each time step. Applying single-agent RL to this problem over the joint action space of all workers would have to reason over $6^{12} = 2,176,782,336$ actions which is infeasible. In contrast in MARL with decentralised execution, each agent can learn a policy to control a single robotic worker and only needs to take decisions about the comparably small action space with just 6 actions for that particular worker.

Similar to observations, histories and actions of agents, we will denote policies and value functions with a subscript for the agent index to distinguish the functions of individual agents in MARL. For example, the policy, state-value and action-value functions of agent i will be denoted as π_i, V_i and Q_i, respectively. In the following, we will sometimes omit the subscript for the agent index for policies and value functions when the parameterisation of the functions already indicates the corresponding agent. For example, we will write $\pi(h_i; \phi_i)$ and $V(h_i; \theta_i)$ to denote the policy and state-value function of agent i with parameters ϕ_i and θ_i, respectively, without denoting the subscript of the function directly.

In the following, we introduce several commonly used MARL algorithms across three categories: independent learning, value decomposition, and multi-agent actor-critic algorithms with centralised critics. Table 2.1 provides an overview of the algorithms that will be introduced and lists several of their properties. We

Table 2.1: Overview of MARL algorithms and their properties spanning over three categories: independent learning (IL), multi-agent actor-critic algorithms with centralised critics (MAAC), and value decomposition algorithms (VD) under the centralised training decentralised execution (CTDE) paradigm.

	Category	Centralised training	Off-/On-policy	Policy gradient
IDQN	IL	✗	Off	✗
IA2C	IL	✗	On	✓
IPPO	IL	✗	On	✓
MADDPG	MAAC	✓	Off	✓
COMA	MAAC	✓	On	✓
MAA2C	MAAC	✓	On	✓
MAPPO	MAAC	✓	On	✓
VDN	VD	✓	Off	✗
QMIX	VD	✓	Off	✗

note that the table is not exhaustive and only provides a selection of common algorithms that we will build upon in the following chapters.

2.6.1 Independent Learning

One of the simplest forms of MARL is *independent learning* (IL) (Tan, 1993). In this paradigm, each agent is trained independently from other agents using only its local information. In this way, IL can be considered a reduction of the multi-agent decision-making problem to multiple single-agent decision-making problems. Each agent learns a policy to maximise its own expected returns without directly considering the policies of other agents. However, we note that this does not mean that IL is unable to learn policies that can coordinate with each other. The agents are still concurrently acting in the same environment and, therefore, their experiences depend on the policies of other agents. With enough samples, agents can learn to react to the policies of other agents and effectively coordinate. One can consider the IL setup as each agent treating the policies of other agents as part of the environment, making the transition function of the environment more complicated, stochastic, and highly non-stationary since the policies of agents consistently change. This non-stationarity is generally present in MARL but par-

ticularly challenging in IL since agents do not actively consider the constantly changing policies of other agents unlike in more complicated MARL algorithms we will see later. Despite its simplicity and challenges, prior work found that IL can be surprisingly competitive with more complicated MARL approaches (Papoudakis et al., 2021; de Witt et al., 2020).

Generally, any single-agent RL algorithm can be applied independently for MARL. This immediately gives rise to a large number of possible IL algorithms, including independent DQN (IDQN), independent A2C (IA2C), and independent PPO (IPPO) based on the single-agent algorithms we have introduced in Section 2.3.

2.6.2 Centralised Training Decentralised Execution

While independent learning treats both training and execution of agents as decentralised processes, *centralised training with decentralised execution* (CTDE) (e.g., Oliehoek et al., 2008; Lowe et al., 2017; Rashid et al., 2020b) is a paradigm that considers the training of all agents as a centralised process. In CTDE, agents may be trained with access to the joint observations or actions of all agents as long as the policies of agents can still be executed in a decentralised manner, i.e. the policies of agents are only conditioned on local information of the individual agents. This allows agents to leverage more information during training without compromising decentralised execution.

Below, we will introduce several deep MARL algorithms under the CTDE paradigm that we will build upon in the later chapters. The first family of algorithms will build on top of IDQN and value-based RL algorithms (Section 2.3.2). The second family of algorithms will build on top of policy gradient and actor-critic algorithms (Section 2.3.3).

2.6.2.1 Value Decomposition

Value-based RL algorithms such as DQN learn an action-value function to estimate the expected returns under the optimal policy and derive a greedy policy from this value function. In the context of MARL algorithms under the CTDE paradigm, it might be desirable to learn a value function that is conditioned on additional information such as the observations or actions of other agents. Such value functions might benefit training by leveraging more information but would

prevent the decentralised execution of policies. Additionally, learning such a value function can be computationally infeasible due to the exponential growth of the joint action space with the number of agents. *Value decomposition* algorithms try to overcome these limitations by decomposing a centralised value function, conditioned on the state of the environment and the joint actions of all agents, into individual utility functions for each agent. These utility functions are only conditioned on the observation history and individual actions of each agent and enable decentralised execution. This idea is inspired by the concept of coordination graphs which represent the interactions of agents in multi-agent systems as a graph and have a long history in multi-agent systems and MARL research (e.g., Guestrin et al., 2001, 2002; Kok and Vlassis, 2005; Oliehoek, 2010; Oliehoek et al., 2012, 2013; van der Pol, 2016; Böhmer et al., 2020).

Due to the assumption of decomposing a single centralised value function for all agents, value decomposition algorithms assume that the agents optimise for a common reward function, i.e. the environment can be formalised as a Dec-POMDP (Section 2.5). The decomposed centralised action-value function in such a setting is defined as

$$Q(s, \mathbf{a}; \theta) = \mathbb{E}_{s^0 \sim \mu, \mathbf{a}^t \sim \pi(\mathbf{h}^t)} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s^t, \mathbf{a}^t, s^{t+1}) \right] \tag{2.21}$$

with $\mathcal{R}$ denoting the common reward function of all agents. We note that we never learn such a centralised value function directly but instead train individual utility functions for each agent that are jointly optimised to approximate the centralised value function. The utility function of agent i is written as $Q(h_i, a_i; \theta_i)$ and is conditioned on the observation history and action of agent i only. Value decomposition algorithms and the following formalism is closely related to the notion of collective intelligence (COINs) studied in multi-agent systems. This literature asks the question how to design agents that can learn to coordinate towards a global objective without centralised control or communication (Wolpert and Tumer, 1999), and various utility functions have been proposed to incentivise agents to coordinate in this setting (Wolpert et al., 2000; Wolpert and Tumer, 2002).

Many recent value decomposition methods define a decomposition that satisfies the *individual-global-max* (IGM) property (Rashid et al., 2020b; Son et al., 2019). The IGM property ensures that (1) agents choosing their actions by greedily following their individual utility functions leads to joint actions that also max-

imise the decomposed centralised action value function, and (2) the joint action that maximises the decomposed centralised action value function is composed of the individual greedy actions with respect to the individual utility functions of all agents. In this way, the IGM property enables effective training and action selection of value decomposition methods. The IGM property was first defined by Son et al. (2019) but previously Rashid et al. (2020b) introduced the term consistency following an almost identical definition. However, the definition found in this thesis follows Albrecht et al. (2024) who consider the possibility that there could be multiple greedy actions with respect to value functions.

In the following, we denote the sets of greedy actions with respect to the decomposed centralised action-value function and the individual utility function of agent i, respectively, as follows:

$$A^*(s;\theta) = \arg\max_{\mathbf{a}\in\mathbf{A}} Q(s,\mathbf{a};\theta) \tag{2.22}$$

$$A_i^*(h_i;\theta_i) = \arg\max_{a_i\in A_i} Q(h_i,a_i;\theta_i) \tag{2.23}$$

with $Q(s,\mathbf{a};\theta)$ and $Q(h_i,a_i;\theta_i)$ denoting the centralised action-value function and individual utility function of agent i, respectively. The IGM property is satisifed if and only if the following holds for all states $s \in S$ and corresponding individual histories $h_i \in H_i$ of agent i:

$$\forall \mathbf{a} = (a_1,\ldots,a_N) \in \mathbf{A} : \mathbf{a} \in A^*(s,\theta) \iff \forall i \in I : a_i \in A_i^*(h_i;\theta_i) \tag{2.24}$$

Besides providing a feasible decomposition of the centralised action-value function, the individual utility functions of agents can also be useful to identify the contributions of individual agents to the received common rewards. In this way, value decomposition methods can address the *multi-agent credit assignment* problem of common-reward settings in MARL (Du et al., 2019; Rashid et al., 2020b). The multi-agent credit assignment problem refers to the challenge of attributing the received common rewards to the actions of individual agents in a multi-agent system.

Various value decomposition algorithms have been proposed based on the IGM property. The simplest algorithm is *value decomposition networks* (VDN) (Sunehag et al., 2018) which assumes a linear decomposition of the centralised value function:

$$Q(s,\mathbf{a};\theta) = \sum_{i\in I} Q(h_i,a_i;\theta_i) \tag{2.25}$$

We refer to Albrecht et al. (2024) for a proof that the VDN decomposition satisfies the IGM property. VDN jointly optimises the individual utility functions of all agents by minimising the following loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{h}^{t+1}) \sim \mathcal{D}} \left[\left(r^t + \gamma \sum_{i \in I} \max_{a_i \in A_i} Q(h_i^{t+1}, a_i; \bar{\theta}_i) - \sum_{i \in I} Q(h_i^t, a_i^t; \theta_i) \right)^2 \right] \tag{2.26}$$

with $\bar{\theta}_i$ denoting the parameters of the periodically updated target network of agent i. Similar to DQN, the loss is optimised from tuples of experiences containing the joint observations and actions of all agents which are stored in and sampled from an experience replay buffer.

Another algorithm that builds on the idea of value decomposition is *QMIX* (Rashid et al., 2020b). QMIX formulates a more expressive decomposition of the centralised action-value function based on the assumption that the centralised action-value function should be strictly monotonic with respect to the individual utility functions of all agents:

$$\forall i \in I, \forall \mathbf{a} \in \mathbf{A} : \frac{\partial Q(s, \mathbf{a}; \theta)}{\partial Q(h_i, a_i; \theta_i)} > 0 \tag{2.27}$$

This assumption is a sufficient condition to ensure the IGM property. We refer to Albrecht et al. (2024) for a proof that the QMIX monotonicity assumption is sufficient to satisfy the IGM property. To implement and ensure this monotonicity assumption in practise, QMIX learns a monotonic mixing network f_{mix} with parameters θ_{mix} that takes the individual utility estimates as inputs and computes a monotonic aggregation to obtain an estimate of the centralised action-value function. The learned decomposition can then be written as follows:

$$Q(s, \mathbf{a}, \theta) = f_{\text{mix}} \left(Q(h_1, a_1; \theta_1), \ldots, Q(h_N, a_N; \theta_N); \theta_{\text{mix}} \right) \tag{2.28}$$

We note that the mixing function is guaranteed to be strictly monotonic with respect to the individual utilities as long as the network has only positive weights. To ensure that the weights of the mixing network are always positive, the parameters θ_{mix} are obtained as the output of a separate hypernetwork f_{hyper} with parameters θ_{hyper} that receives the state of the environment as input and outputs the parameters of the mixing network. In environments, where the state is not available during training, we use the joint observation as a proxy for the state. To ensure positive weights, the absolute value function is used as the activation function to the outputs corresponding to the weights of the mixing network. The

parameters of the hypernetwork θ_{hyper} as well as the parameters of the individual utility functions $\{\theta_i\}_{i \in I}$ are jointly optimised by minimising the following loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{(\mathbf{h}^t, s^t, \mathbf{a}^t, r^t, \mathbf{h}^{t+1}, s^{t+1}) \sim \mathcal{D}} \Big[\Big(r^t + \gamma y_{\text{tot}} - f_{\text{mix}} \Big(Q(h_1^t, a_1^t; \theta_1), \dots, Q(h_N^t, a_N^t; \theta_N); \theta_{\text{mix}} \Big) \Big)^2 \Big] \quad (2.29)$$

with target value

$$y_{\text{tot}} = f_{\text{mix}} \Big(\max_{a_1 \in A_1} Q(h_1^{t+1}, a_1; \bar{\theta}_1), \dots, \max_{a_N \in A_N} Q(h_N^{t+1}, a_N; \bar{\theta}_N); \bar{\theta}_{\text{mix}} \Big) \quad (2.30)$$

with $\bar{\theta}_i$ denoting the parameters of the target utility network of agent i and $\bar{\theta}_{\text{mix}}$ denoting the parameters of the target mixing network. All target networks are updated periodically by copying the parameters of the corresponding main network. As for VDN, batches of experiences are sampled from an experience replay buffer to compute the loss in Equation 2.29.

2.6.2.2 Centralised Critics for Actor-Critic Algorithms

In Section 2.3.3 we introduced actor-critic algorithms that simultaneously learn a policy and value function, also referred to as the actor and critic, respectively. We note that the critic in these algorithms is only used during training to obtain estimates for expected returns needed to compute the policy gradients. During execution for action selection, the critic is not used and we directly query the parameterised policy to obtain actions. Following this observation, we can extend actor-critic algorithms for the CTDE paradigm by conditioning the critic on additional centralised information, such as the joint observations or the state of the environment. This allows the critic to leverage more information during training without compromising decentralised execution.

Such *centralised critics* can be used with any actor-critic algorithm in MARL. For example, we can extend the previously introduced A2C algorithm with centralised critics by conditioning the critic of each agent on the joint observation history of all agents. The loss for the critic of agent i can be written as:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{s^0 \sim \mu, a_i^t \sim \pi(h_i^t; \phi_i)} \Big[\Big(r_i^t + \gamma V(\mathbf{h}^{t+1}; \theta_i) - V(\mathbf{h}^t; \theta_i) \Big)^2 \Big] \quad (2.31)$$

and the corresponding actor loss then becomes:

$$\mathcal{L}(\phi_i) = \mathbb{E}_{s^0 \sim \mu, a_i^t \sim \pi(h_i^t; \phi_i)} \Big[-\log \pi(a_i^t \mid h_i^t; \phi_i) \Big(r^t + \gamma V(\mathbf{h}^{t+1}; \theta_i) - V(\mathbf{h}^t; \theta_i) \Big) \Big] \quad (2.32)$$

Note that the policy of agent i is still only conditioned on the individual observation history of agent i, but the critic is conditioned on the observation histories of all agents. Besides the difference in inputs to the critic, this algorithm is identical to the independent A2C algorithm. We refer to this algorithm with centralised critics as *multi-agent A2C* (MAA2C). Likewise, we can extend the PPO algorithm with centralised critics to obtain *multi-agent PPO* (MAPPO). These algorithms are commonly used in MARL research and have been found to perform well in many different benchmarks (Papoudakis et al., 2021; Yu et al., 2022).

Counterfactual multi-agent policy gradient (COMA) (Foerster et al., 2018) was one of the first MARL algorithms to leverage centralised critics and is focused on the common reward setting. As discussed in Section 2.6.2.1, the multi-agent credit assignment problem is a major challenge whenever training agents from a common reward, requiring agents to identify their contributions to the received common reward signal. To address this challenge, COMA derives an advantage estimation that computes a counterfactual baseline based on the idea of *difference rewards* (Wolpert and Tumer, 2002; Tumer and Agogino, 2007) using a centralised action-value function:

$$\mathcal{L}(\phi_i) = \mathbb{E}_{s^0 \sim \mu, a_i^t \sim \pi(h_i^t;\phi_i)} \Big[-\log \pi(a_i^t \mid h_i^t;\phi_i) \Big(Q(s^t, \mathbf{a}^t;\theta) - \sum_{a_i' \in A_i} \pi(a_i' \mid h_i^t;\phi_i) Q(s^t, (a_i';\mathbf{a}_{-i}^t);\theta) \Big) \Big] \quad (2.33)$$

with θ denoting the parameters of the centralised action-value function shared across all agents, $\mathbf{a}_{-i}^t$ denoting the joint action of all agents except agent i, and $(a_i';\mathbf{a}_{-i}^t)$ denotating the joint action with action a_i' of agent i and actions $\mathbf{a}_{-i}^t$ for all other agents. Through its advantage formulation, COMA compares how much better the applied action a_i^t of agent i within $\mathbf{a}^t$ was compared to applying any action sampled from its policy $\pi(\cdot \mid h_i^t;\phi_i)$. In this way, the counterfactual baseline of COMA allows it to identify the contribution of the action of agent i to the received common reward. Since all agents optimise for the same common reward signal, a single centralised action-value function can be shared across all agents. This critic is trained using the TD-lambda algorithm (Sutton, 1988).

Another algorithm that builds on the idea of centralised critics is *multi-agent deep deterministic policy gradient* (MADDPG) (Lowe et al., 2017). This algorithm extends the deep deterministic policy gradient (DDPG) algorithm (Lillicrap et al., 2016) from single agent RL to the multi-agent setting. The DDPG algo-

rithm was originally designed for continuous control, i.e. agents choose actions within a continuous range rather than choosing from a discrete set of actions, since it assumes differentiability of actions with respect to the parameters of the actor. However, DDPG and MADDPG can and have been applied to environments with discrete action spaces using the gumbel softmax reparameterisation trick (Jang et al., 2017; Maddison et al., 2017). Motivated by the problem of continuous control, MADDPG trains a deterministic policy μ_i with parameters ϕ_i for each agent i which given an observation history outputs a continous action. We denote deterministic policies with μ and stochastic policies with π. Additionally, a centralised action-value function is trained for each agent which is conditioned on the joint observation history and joint actions of all agents. The critic is optimised to minimise the following mean squared error loss:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(\mathbf{h}^t, \mathbf{a}^t, r_i^t, \mathbf{h}^{t+1}) \sim \mathcal{D}} \Big[\Big(r_i^t + \gamma Q(\mathbf{h}^{t+1}, \mathbf{a}^{t+1}; \bar{\theta}_i)|_{a_i^{t+1} = \mu_i(h_i^{t+1}; \bar{\phi}_i)} - Q(\mathbf{h}^t, \mathbf{a}^t; \theta_i) \Big)^2 \Big] \tag{2.34}$$

with $\bar{\theta}_i$ denoting the parameters of the periodically updated target critic network of agent i and $\bar{\phi}_i$ denoting the parameters of the target actor network of agent i. The actor is trained to maximise the expected returns by minimising the deterministic policy gradient loss:

$$\mathcal{L}(\phi_i) = \mathbb{E}_{\mathbf{h}^t, \mathbf{a}^t \sim \mathcal{D}} \Big[-Q(\mathbf{h}^t, (\mu(h_i^t; \phi_i); \mathbf{a}_{-i}^t); \theta_i) \Big] \tag{2.35}$$

We note that, unlike other discussed policy gradient algorithms, DDPG and MADDPG are off-policy algorithms and can learn from experiences sampled from an experience replay buffer like DQN.

Chapter 3

Decoupled Reinforcement Learning to Stabilise Intrinsically Motivated Exploration

Publication

This chapter is based on and adapted from the following publication:

Lukas Schäfer, Filippos Christianos, Josiah P. Hanna, and Stefano V. Albrecht. "Decoupled reinforcement learning to stabilise intrinsically-motivated exploration." In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. 2020.

In this chapter, we will discuss the challenge of exploration in the context of single-agent reinforcement learning, in which a single agent is exploring its environment to gather information with the aim of learning an optimal policy. We follow the problem setting of a MDP defined in Section 2.1.

Exploration is one of the essential challenges in reinforcement learning. However, many RL algorithms still use simple randomised methods, e.g. applying ϵ-greedy policies (Watkins, 1989) or adding random noise to continuous actions (Lillicrap et al., 2016), to explore without any consideration for which experiences are valuable to explore or not. Such random exploration techniques can be insufficient to solve hard exploration problems that require agents to explore large state spaces without frequent or even any rewards from the environment. One category of exploration techniques that has been successfully applied in such sparse-reward

settings is *intrinsically motivated exploration* (Oudeyer and Kaplan, 2008, 2009; Barto, 2013). For the remainder of this chapter, which focuses on single-agent RL, we denote the intrinsic rewards of the agent with r_i. This should not be confused with the (extrinsic) rewards of agent i in a multi-agent system!

For intrinsically motivated exploration, the agent itself computes intrinsic rewards r_i that incentivise the exploration of novel or underexplored parts of the environment. But how are parts of the state space identified to be "novel" or "underexplored"? Many approaches in this family of algorithms predict transitions of the environment (e.g., Schmidhuber, 1991; Pathak et al., 2017; Raileanu and Rocktäschel, 2020; Burda et al., 2019b) or compute (pseudo-)counts of states (e.g., Strehl and Littman, 2008; Tang et al., 2017; Ostrovski et al., 2017). If the agent can accurately predict transitions or has frequently visited states in a part of the environment then further exploration might not be necessary in this part of the states space. Likewise, if the agent is unable to predict transitions or has rarely encountered states in a part of the environment then further exploration of this part of the environment might be beneficial for learning a policy.

To train a policy with these intrinsic rewards, approaches typically maximise the expected returns over a combined reward signal of the intrinsic rewards r_i and the extrinsic rewards of the environment r_e with some weighting factor λ:

$$r = r_e + \lambda r_i \tag{3.1}$$

This combined objective directly formulates the exploration-exploitation trade-off in RL. The agent is optimised to maximise the sum of extrinsic rewards r_e from the environment, which indicate how well the agent is able to exploit its current knowledge to solve the task, and the intrinsic rewards r_i, which indicate the agent's ability to effectively explore the environment. In this context, the weighting factor λ determines the balance between exploration and exploitation. A high λ incentivises the agent to put more emphasis on exploration, whereas a low λ encourages the agent to exploit more.

However, it appears clear that no single policy can be optimal for both exploration and exploitation in most tasks. A more exploratory policy will likely aim to encounter a large part if not all parts of the state space while an exploitative policy will likely only need to encounter a small subset of the state space to optimally solve the task. This indicates a potential conflict between the two objectives of exploration and exploitation that leads to several challenges with

intrinsically motivated exploration techniques:

1. **Intrinsic rewards lead to non-stationary rewards.** Intrinsic rewards indicate the need for more exploration and, thus, naturally diminish throughout training as the agent has already explored larger parts of the environment. Such decay is intentional and necessary for the agent to eventually only optimise for extrinsic rewards to solve the task, but renders the combined reward signal to be non-stationary. This violates the Markov assumption and can cause the learning progress to be unstable.

2. **Intrinsically motivated exploration is sensitive to the scale λ.** As indicated above, λ is used to balance extrinsic and intrinsic rewards in the combined reward signal. If λ is too large, the intrinsic rewards might dominate and prevent the agent from learning to solve the task. On the other hand, we show that small intrinsic rewards have no sufficient impact and do not incentivise exploration as intended (as we will see later in Figure 3.3).

3. **Intrinsically motivated exploration is sensitive to the rate of decay**. While intrinsic rewards naturally decay throughout training, the rate at which they decay can typically be defined with hyperparameters of the particular approach. Intrinsically motivated RL can be highly sensitive to this rate of decay since slowly decaying intrinsic rewards might disrupt training and prevent agents from learning to solve the task whereas quickly vanishing intrinsic rewards have insufficient impact on exploration (as we will see later in Figure 3.4).

These challenges lead to a significant dependency of intrinsic rewards on hyperparameters. Additionally, determining these hyperparameters for scale and rate of decay is specific to a task due to its dependence on the scale of extrinsic rewards and required exploration in the respective task. Current approaches usually address the difficulties caused by sensitivity to hyperparameters using a large hyperparameter search to find effective parameterisation of a method. However, such a search can be considered an exploration by itself, and can introduce bias in reported results that only show the performance of the best-identified hyperparameters. We argue that this bias is particularly harmful in approaches focusing on exploration as best-identified hyperparameters may have performed well in a hard exploration task because they biased the exploration towards particular parts of the state space that was required to solve the task. However, this

effectively shifts the desirable exploration from the proposed method to the hyperparameter search and, thus, the reported best-identified performance results may not be indicative of the quality of the exploration method. All these properties make the practical application of such methods difficult (Taïga et al., 2020) and motivate the need for more robust approaches.

Motivated by these challenges, success in off-policy RL (Zhong et al., 2021; Fujimoto et al., 2018; Haarnoja et al., 2018; Silver et al., 2014; Degris et al., 2012), and the observation that several of these challenges arise due to the optimisation of a single policy for both exploration and exploitation, we propose to decouple the RL training into two separate policies. We train an *exploration policy* π_β with the combined signal of extrinsic and intrinsic rewards. This policy is trained with the combined reward signal to collect exploratory experiences with a bias towards experiences that indicate solutions to the task of the environment. Simultaneously, we train an *exploitation policy* π_e using only extrinsic rewards on the data collected by the exploration policy. We refer to this approach as *decoupled reinforcement learning* (DeRL)[1]. Using such decoupling addresses the aforementioned challenges of intrinsically motivated exploration. The exploration policy is optimised using the combined objective of extrinsic and intrinsic rewards as in typical intrinsically motivated RL. This policy is not needed to learn to solve the task but only serves to generate data for the training of the exploitation policy. The exploitation policy is only trained with extrinsic rewards and is, thereby, decoupled from the challenges of training with intrinsic rewards and optimised to be an effective policy in the given environment. Our experiments show that DeRL leverages the benefits of intrinsically motivated exploration while stabilising its inherent sensitivity to scale and rate of decay of intrinsic rewards.

We implement and evaluate two versions of DeRL built upon on-policy actor-critic and off-policy Q-learning with five types of intrinsic rewards in two learning environments that focus on exploration. We analyse the sensitivity of DeRL and RL baselines to the scale and the rate of decay of intrinsic rewards to verify the general dependency of these methods on the hyperparameters of intrinsic rewards and show that DeRL is more robust to varying hyperparameters. Additionally, the exploitation policy of several DeRL algorithms is able to converge to higher evaluation returns using up to $\sim$40% fewer interactions and reaches higher returns

[1]We provide an implementation of DeRL and the Hallway environment at `https://github.com/uoe-agents/derl`

in some tasks compared to intrinsically motivated RL baselines. Such improved robustness and sample efficiency can justify the additional cost of training a second policy. However, we also observe that DeRL still suffers from variability in the off-policy optimisation of the exploitation policy π_e in several tasks. We hypothesise that distribution shift caused by the divergence of π_e and π_β leads to these instabilities, and show that regularisers can be applied to restrict divergence of both policies (Wu et al., 2019) and effectively reduce deviations in returns of exploration and exploitation policies.

3.1 Intrinsically Motivated Exploration

A plethora of approaches have been proposed to replicate the concept of curiosity for exploration in RL. Oudeyer and Kaplan (2008), Oudeyer and Kaplan (2009), and Barto (2013) provide an overview over multiple such approaches for efficient exploration in RL and robotics. The underlying idea behind these approaches is fairly simple: agents should be incentivised to explore novel or poorly understood parts of the environment. Therefore, an agent learns an intrinsic reward function which incentivises the agent to explore. During training, the agent optimises the combined reward signal of extrinsic rewards of the environment and intrinsic rewards. Over time, the agent should become less "curious" with completed exploration and exploitation will gradually take over. These exploration bonuses are a form of *reward shaping*, trying to compute additional rewards to facilitate training (Devlin and Kudenko, 2012). There are two major categories of intrinsic rewards for exploration: (1) count-based intrinsic rewards and (2) prediction-based intrinsic rewards. Below, we will introduce several commonly used representatives of both of these categories.

3.1.1 Count-based Intrinsic Rewards

Count-based intrinsic rewards count occurrences of states or state-action pairs. Using these counts, agents can then compute an intrinsic reward that is inverse proportional to the count of state visitations denoted with $N(s^t)$:

$$r_i^t = \frac{1}{\sqrt{N(s^t)}} \tag{3.2}$$

Thereby, agents are incentivised to visit states within the environment that

have previously been less frequently encountered. Likewise, agents are discouraged from visiting frequently encountered states that are deemed less valuable for exploration. While this approach is easily applicable in small and discrete state spaces, pseudo-counts have to be computed for large or continuous state spaces where encountering any state multiple times is rare. Previous work compute pseudo-counts using density models that predict visitations of states (Bellemare et al., 2016; Ostrovski et al., 2017) or locality-sensitive (Andoni and Indyk, 2008) hash functions (Tang et al., 2017) like the SimHash (Charikar, 2002) function.

3.1.2 Prediction-based Intrinsic Rewards

Alternatively, agents can predict transitions in the environment to define intrinsic rewards. Most prominently, agents predict the next state given the current state and applied action, and define the intrinsic reward as the discrepancy between the predicted next state and the actually encountered next state.

Schmidhuber (1991) proposed such a prediction-based intrinsic reward for exploration and identified one of its major shortcomings: stochasticity within the environment can lead to unpredictable dynamics, leading to intrinsic rewards remaining high at the face of stochasticity. This problem is commonly known as the "noisy TV problem" (Burda et al., 2019a), named after an example of this problem in which an agent is trained with a prediction-based intrinsic reward in an environment with a random noise TV. The TV consistently shows random noise images that change at every time step, so the agent would be unable to predict the next frame. As a result, the agent consistently receives high intrinsic rewards when walking in front of the TV and is incentivised to keep doing so.

Intrinsic curiosity module (ICM) Pathak et al. (2017) propose to learn state representations using an inverse-dynamics objective and predict the next state representation using a forward prediction model. The state representation network takes in a state s and computes a lower dimensional representation $z(s)$. This state representation network is jointly trained with an inverse dynamics model that predicts the applied action a^t from two consecutive state representations $z(s^t)$ and $z(s^{t+1})$. By training the state representations using this objective, the representations are incentivised to only encode information that can be affected by the agent's actions. In addition, ICM trains a forward prediction network that takes the representation of the current state $z(s^t)$ and the current action a^t

and tries to predict the representation of the next state $\hat{z}(s^{t+1})$. This network is trained using a mean-squared error of the predicted representation of the next state and the ground truth representation of the next state. This prediction error also defines the intrinsic rewards:

$$r_i^t = \left(\hat{z}(s^{t+1}) - z(s^{t+1})\right)^2 \tag{3.3}$$

Random network distillation (RND) Burda et al. (2019b) propose a simplified prediction-based intrinsic reward for exploration based on two observations: (1) training and computation of complex models of dynamics is expensive and (2) static random feature representation is surprisingly effective for observation predictions (Burda et al., 2019a). RND randomly initialises two networks with identical architectures that take states as input and compute a lower dimensional representation. We refer to one of the networks as the target network and the other as the main network. The parameters of the target network $\overline{\psi}$ are frozen throughout training, and the main network parameters ψ are optimised to mimic the representations of the target network by minimising the mean squared error between the representations of both networks. As for ICM, the intrinsic reward is defined as this error between the representations:

$$r_i^t = \left(z(s^t;\psi) - z(s^t;\overline{\psi})\right)^2 \tag{3.4}$$

Rewarding impact-driven exploration (RIDE) Raileanu and Rocktäschel (2020) propose to reward the agent for applying actions which lead to significant change in the environment. Such change is defined as the difference between representations of consecutive states, where the representation function z is trained using an inverse dynamics model identical to ICM (Pathak et al., 2017). In order to avoid the agent going back and forth between a group of states, an episodic state-count N_{ep} is added to the objective. This episodic count is reset after every episode and, thus, incentivises the agent to visit many states within each episode. The intrinsic reward is then defined as a combination of the mean squared error between the representations of consecutive states and the episodic count:

$$r_i^t = \frac{\left(z(s^{t+1}) - z(s^t)\right)^2}{\sqrt{N_{ep}(s^{t+1})}} \tag{3.5}$$

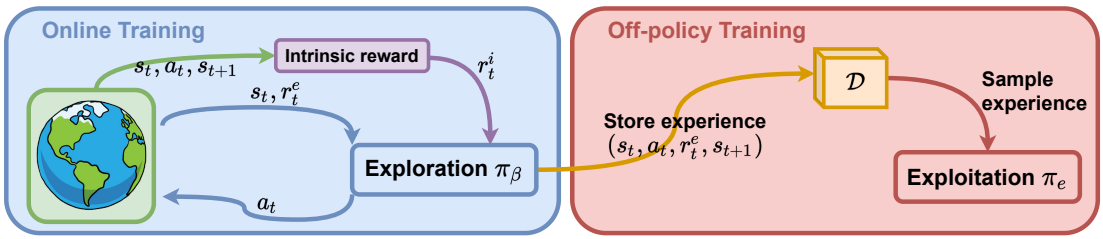


Figure 3.1: Decoupled reinforcement learning (DeRL) training loop. On the left in blue, the exploration policy π_β is trained from online interactions with the environment to maximise the cumulative sum of intrinsic and extrinsic rewards. On the right in red, the exploitation policy π_e is trained only from off-policy experiences collected by the exploration policy π_β to maximise the cumulative sum of extrinsic rewards.

3.2 Decoupled Reinforcement Learning

In this chapter, we propose *decoupled reinforcement learning* (DeRL) in which we train two separate policies for exploration and exploitation to improve sample efficiency and reduce sensitivity to hyperparameters of intrinsic rewards. We train an exploration policy π_β with the intent to explore the environment. Using the data collected by the exploration policy, we train a separate exploitation policy π_e, as visualised in Figure 3.1. Separating exploration and exploitation in this way enables training of the exploration policy with intrinsic rewards without modifying the training objective of the exploitation policy.

Formally, an agent trains an exploration policy π_β to maximise the sum of intrinsic and extrinsic rewards,

$$\pi_\beta \in \arg\max_\pi \mathbb{E}_{s^0\sim\mu, a^t\sim\pi(s^t)}\left[\sum_{t=0}^{\infty}\gamma^t\left(r_e^t + \lambda r_i^t\right)\right] \tag{3.6}$$

$$= \arg\max_\pi \mathbb{E}_{s^0\sim\mu, a^t\sim\pi(s^t)}\left[G_{e+i}^t\right] \tag{3.7}$$

with G_{e+i}^t denoting the discounted returns computed using the combination of extrinsic and intrinsic rewards with scaling factor λ and discount factor $\gamma \in [0, 1)$. During training of π_β, experience samples $(s^t, a^t, r_e^t, s^{t+1})$ with extrinsic rewards are collected in an experience storage $\mathcal{D}$.

In addition to this typical intrinsically motivated RL, we train a separate exploitation policy π_e to maximise only expected cumulative extrinsic rewards

using experience accumulated in $\mathcal{D}$ with G_e^t denoting discounted extrinsic returns.

$$\pi_e \in \arg\max_{\pi} \mathbb{E}_{(s^t,a^t,r_e^t,s^{t+1})\sim\mathcal{D}}\left[\sum_{t=0}^{\infty}\gamma^t r_e^t\right] \tag{3.8}$$

$$= \arg\max_{\pi} \mathbb{E}_{(s^t,a^t,r_e^t,s^{t+1})\sim\mathcal{D}}\left[G_e^t\right] \tag{3.9}$$

Both exploration policy π_β and exploitation policy π_e can be trained using any RL algorithm given the defined objectives. We optimise the exploration policy π_β as RL with intrinsic rewards, whereas we train π_e on experience from $\mathcal{D}$ that is generated from π_β's interaction in the environment. Note that $\mathcal{D}$ only contains extrinsic rewards and is off-policy data for the optimisation of π_e as it was generated by following π_β. Therefore, training the exploitation policy using experience generated by the exploration policy requires off-policy RL. Off-policy RL is concerned with the optimisation of a policy using experience generated within the environment by following a separate behaviour policy. Below, we propose two methods to apply such decoupled RL using an actor-critic and Q-learning framework.

3.2.1 Decoupled Actor-Critic

In order to use traditionally on-policy RL such as the majority of policy gradient algorithms (Section 2.3.3) to train π_e using $\mathcal{D}$, we can use off-policy correction to account for differences in trajectory distributions of both π_e and π_β.

One technique for off-policy correction is *importance sampling* (IS). In the following, we train π_e using an on-policy actor-critic RL algorithm with state value function V, parameterised by θ, and policy π_e, parameterised by ϕ. We optimise the latter by minimising the actor loss given by:

$$\mathcal{L}(\phi) = \mathbb{E}_{(s^t,a^t,r_e^t,s^{t+1})\sim\mathcal{D}}\Big[-\rho(a^t \mid s^t)\log\pi_e(a^t \mid s^t;\phi)\ A_e(s^t)\Big] \tag{3.10}$$

with bootstrapped advantage estimates $A_e(s^t)$ and IS weights $\rho(a^t \mid s^t)$:

$$A_e(s^t) = \Big(r_e^t + \gamma V(s^{t+1};\theta) - V(s^t;\theta)\Big) \tag{3.11}$$

$$\rho(a^t \mid s^t) = \frac{\pi_e(a^t \mid s^t;\phi)}{\pi_\beta(a^t \mid s^t)} \tag{3.12}$$

Similarly, the value loss for π_e using IS weights can be defined as follows:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s^t,a^t,r_e^t,s^{t+1})\sim\mathcal{D}}\Big[\rho(a^t \mid s^t)\ \Big(r_e^t + \gamma V(s^{t+1};\theta) - V(s^t;\theta)\Big)^2\Big] \tag{3.13}$$

Algorithm 1 Decoupled Actor-Critic

Initialise: parameters ϕ of π_e, θ of critic V, and π_β
Initialise: empty exploration data $\mathcal{D} \leftarrow \emptyset$
for each episode **do**
  Obtain initial state $s^0 \sim \mu$
  for each step $t = 0, \cdots, T$ **do**
    Select action $a^t \sim \pi_\beta(s^t)$
    $s^{t+1}, r_e^t \leftarrow$ apply action a^t
    Update π_β using RL on combined rewards (Equation 3.7)
    $\mathcal{D} \leftarrow \mathcal{D} \cup (s^t, a^t, r_e^t, s^{t+1})$
    Update ϕ by minimising Equation 3.10 with $\mathcal{D}$
    Update θ by minimising Equation 3.13 with $\mathcal{D}$
    $\mathcal{D} \leftarrow \emptyset$
  end for
end for

We note that IS weights ρ can lead to high variance gradients with exploding IS weights whenever $\pi_e(a^t \mid s^t; \phi) \gg \pi_\beta(a^t \mid s^t)$ and vanishing weights for $\pi_e(a^t \mid s^t; \phi) \ll \pi_\beta(a^t \mid s^t)$ (Christianos et al., 2020). In particular very large IS weights can cause significant instability of training. Various techniques have been proposed to address such exploding weights, including clipping of importance weights (e.g., Munos et al., 2016; Espeholt et al., 2018). In our experiments, we consider the application of clipped IS weights but find unclipped IS weights to be sufficient in most experiments. The pseudocode for decoupled actor-critic optimisation of π_e can be found in Algorithm 1.

3.2.2 Decoupled Deep Q-Networks

Instead of optimising π_e using actor-critic algorithms with off-policy corrections, we can also apply off-policy algorithms such as Q-learning without the need for any correction. We consider optimising π_e using deep Q-networks (DQN) (Section 2.3.2). For DQN optimisation, the following loss is minimised:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s^t, a^t, r_e^t, s^{t+1}) \sim \mathcal{D}} \left[\left(r_e^t + \gamma \max_{a'} Q(s_{t+1}, a'; \bar{\theta}) - Q(s^t, a^t; \theta) \right)^2 \right] \tag{3.14}$$

with $\bar{\theta}$ denoting the parameters of the periodically updated target network.

Algorithm 2 Decoupled Deep Q-Networks

Initialise: parameters θ of value function Q, and π_β
Initialise: empty exploration data $\mathcal{D} \leftarrow \emptyset$
for each episode **do**
  Obtain initial state $s^0 \sim \mu$
  for each step $t = 0, \cdots, T$ **do**
    Select action $a^t \sim \pi_\beta(s^t)$
    $s^{t+1}, r_e^t \leftarrow$ apply action a^t
    Update π_β using RL on combined rewards (Equation 3.7)
    $\mathcal{D} \leftarrow \mathcal{D} \cup (s^t, a^t, r_e^t, s^{t+1})$
    Update θ by minimising Equation 3.14 with $\mathcal{D}$
  end for
end for

Pseudocode for decoupled deep Q-networks of π_e can be found in Algorithm 2. Note that $\mathcal{D}$ is only used for a single update in decoupled actor-critic, whereas in decoupled deep Q-networks $\mathcal{D}$ represents a replay buffer (Lin, 1992a) that is continually filled with experience.

3.3 Evaluation Details

We evaluate DeRL in two learning environments with a variety of RL algorithms and intrinsic rewards.

3.3.1 Algorithms

Baselines As baselines, we consider two on-policy RL algorithms: advantage actor-critic (A2C) and proximal policy optimisation (PPO) (Section 2.3.3). Both algorithms are trained using the combined reward (Equation 3.1) with weighting factor λ and varying intrinsic reward definitions.

DeRL For our decoupled RL optimisation, we consistently train π_β using A2C as we found it to be more robust than PPO. For the optimisation of π_e, we consider A2C and PPO for decoupled actor-critic and decoupled deep Q-networks based on DQN. We refer to these algorithms as DeA2C, DePPO and DeDQN. As intrinsic rewards, we use Count and ICM to train π_β.

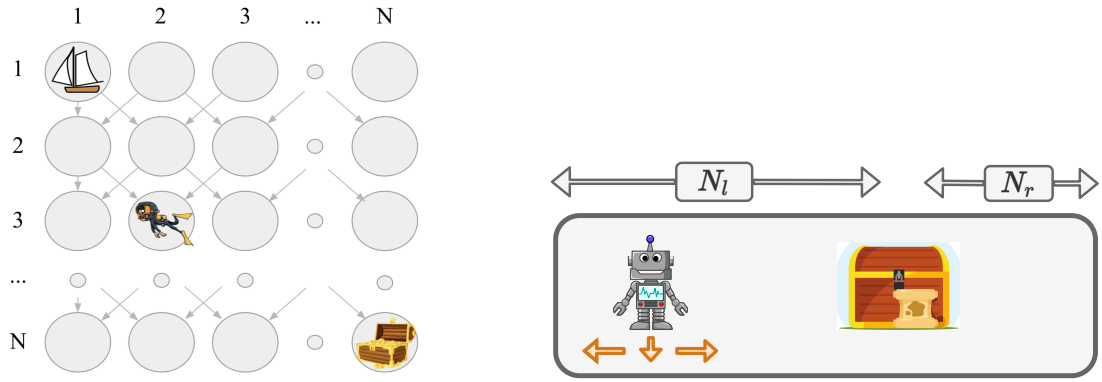
Intrinsic rewards We consider a total of five intrinsic rewards for exploration in our evaluation. For count-based intrinsic rewards, we consider *Count* that directly stores and increments state occurrences in a table as well as *Hash-Count* that first groups states using the SimHash function (Tang et al., 2017). For prediction-based intrinsic rewards, we conduct experiments with *ICM* (Pathak et al., 2017), *RND* (Burda et al., 2019b), and *RIDE* (Raileanu and Rocktäschel, 2020). For more details on these intrinsic reward definitions, see Section 3.1.

3.3.2 Environments

We evaluate in eleven different tasks of two environments, DeepSea and Hallway. Below, we describe both environments in more detail.

DeepSea DeepSea, visualised in Figure 3.2a, is an environment proposed as part of the behaviour suite (Bsuite) for RL (Osband et al., 2020). The environment targets the challenge of exploration and represents a $N \times N$ grid where the agent starts in the top left and has to reach a goal in the bottom right location. At each time step, the agent moves one row down and can choose one out of two actions. For each row, both actions are randomly assigned to left and right movement. The agent observes the current location as a 2D one-hot encoding and receives a small negative reward of $\frac{-0.01}{N}$ for moving right and 0 reward for moving left. Additionally, the agent receives a reward of $+1$ for reaching the goal and the episode ends after N time steps. The difficulty of the exploration in DeepSea can be adjusted using N: the larger N, the harder it becomes for the agent to reach the goal location for optimal returns of 0.99. We evaluate all algorithms in the DeepSea task for $N \in \{10, 14, 20, 24, 30\}$.

Hallway As part of this work, we propose the Hallway environment as a new environment where exploration and exploitation are misaligned, visualised in Figure 3.2b. In DeepSea, agents receive reward by reaching states at the end of the environment, so intrinsic rewards for exploration strongly align with extrinsic rewards from the environment. We hypothesise that tasks in which intrinsic and extrinsic rewards are not well aligned require carefully balanced exploration through intrinsic rewards. Motivated by this hypothesis, we design the Hallway environment in which an agent is located in a hallway starting on the left. A goal can be reached by moving N_l cells to the right. In contrast to DeepSea, the



(a) DeepSea (b) Hallway

Figure 3.2: Visualisation of the DeepSea and Hallway environments.

goal is not necessarily located at the right end of the hallway, but there might be further N_r empty cells to the right of the goal location. At each time step, the agent can choose between three actions: move left, stay or move right. The agent receives a reward of $+1$ for reaching the goal for the first time and every time it stays at the goal location for 10 steps. Therefore, the agent needs to learn to move to the goal and stay there for the remaining time steps of the episode to collect further reward. Episodes end after $2N_l$ steps and small negative reward of -0.01 is assigned for moving right or stay. Hallway tasks, in particular with $N_r > 0$, require exploration through intrinsic rewards to be carefully balanced because staying at the goal for optimal returns and exploration are not aligned. We evaluate all algorithms in the Hallway environment with $N_l \in \{10, 20, 30\}$ and N_r either being 0 or equal to N_l.

3.3.3 Implementation Details

For all algorithms, we compute n-step return estimates (Sutton and Barto, 2018) to reduce the bias of value estimates in all algorithms. On-policy training uses four synchronous environments and an additional entropy regularisation term in the policy loss (Mnih et al., 2016). For DeDQN, we compute DDQN targets (Section 2.3.2). For details on the conducted hyperparameter search and hyperparameters used across experiments, see Appendix A.1.

We train all algorithms for 100,000 episodes and evaluate every 1,000 episodes for a total of 100 evaluations by applying the evaluation policy in the respective task for 8 episodes. We report averaged evaluation returns and stratified boot-

strap 95% confidence intervals (Agarwal et al., 2021) across five random seeds. Optimal returns are indicated as a dashed horizontal line. We use a weighting factor of $\lambda = 1$ for the combined reward signal unless stated otherwise.

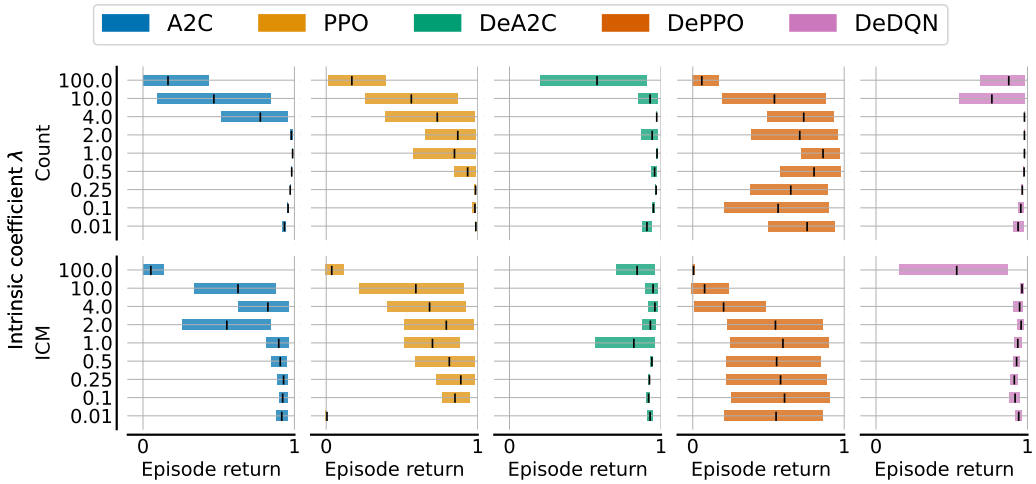
3.4 Evaluation Results

In this section, we discuss the evaluation results of DeRL in comparison to intrinsically motivated RL baselines to investigate the following three hypotheses:

1. Intrinsically motivated RL is sensitive to varying scale λ and rate of decay of intrinsic rewards,
2. DeRL is more robust than intrinsically motivated RL baselines to varying scale and rate of decay, and
3. DeRL leads to similar or improved returns and sample efficiency compared to intrinsically motivated RL baselines.

3.4.1 Hyperparameter Sensitivity

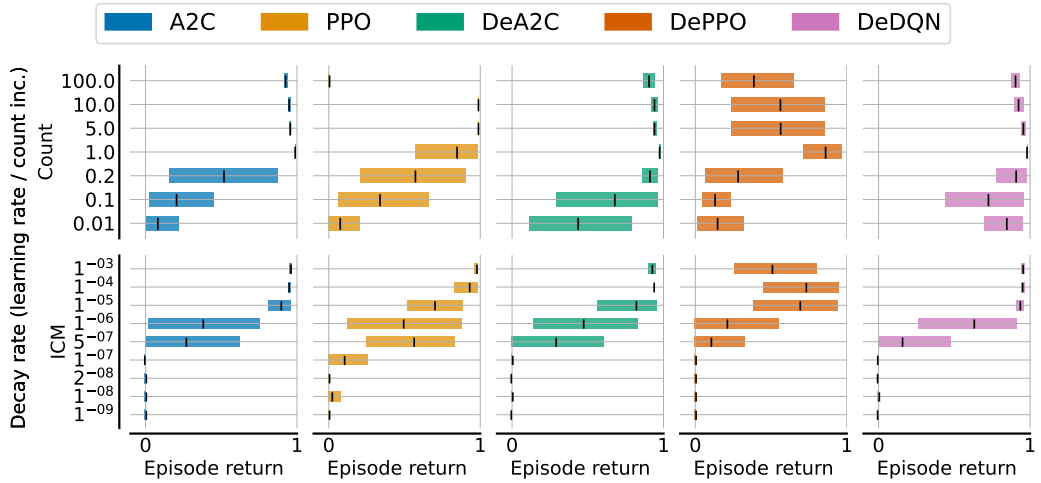
To investigate our first two hypotheses, we train all baselines and DeRL algorithms on the combined reward in DeepSea $N = 10$ and Hallway $N_l = N_r = 10$ with varying λ and rates of decay. Figures 3.3 and 3.4 show the evaluation returns of all baselines and DeRL algorithms with Count and ICM for varying λ and rates of decay. A sensitivity analysis for all remaining intrinsic rewards can be found in Appendix A.3. The evaluation results clearly show that intrinsically motivated RL training is indeed highly sensitive to scale and decay rate of intrinsic rewards, not learning at all or reaching significantly lower evaluation returns for many hyperparameter values. In particular in the Hallway environment, where exploration and extrinsic rewards are misaligned, all algorithms exhibit significant dependency on carefully tuned scale and rate of decay. This confirms our first hypothesis. We further confirm our second hypothesis that DeRL algorithms are more robust to varying scale and decay rate of intrinsic rewards, reaching higher returns across a wider range of hyperparameters.



(a) DeepSea 10

(b) Hallway $N_l = N_r = 10$

Figure 3.3: Average evaluation returns in DeepSea 10 and Hallway $N_l = N_r = 10$ with varying scaling factors λ. Shading indicates 95% confidence intervals. A method that is insensitive to hyperparameters will have final average episodic return concentrated to the right for all hyperparameter values.



(a) DeepSea 10

(b) Hallway $N_l = N_r = 10$

Figure 3.4: Average evaluation returns in DeepSea 10 and Hallway $N_l = N_r = 10$ with varying rates of decay. Shading indicates 95% confidence intervals.

Scale of intrinsic rewards To analyse the sensitivity of intrinsically motivated RL to the scale of intrinsic rewards, Figure 3.3 shows average evaluation returns for varying values of λ for baselines and DeRL algorithms. In DeepSea $N = 10$, DeA2C and DeDQN exhibit improved robustness by reaching close to optimal returns for almost all values of λ. In contrast, DePPO and the baselines are found to be more sensitive in particular to large values of λ. In Hallway, all algorithms exhibit larger variance for varying λ compared to DeepSea with no significant learning being observed for large or small values of λ, with DeA2C and DePPO demonstrating slightly more robustness. These results indicate the sensitivity to values of λ. Even small deviations can make the difference between learning and not learning at all.

Table 3.1: Average evaluation returns and a single standard deviation in all DeepSea and Hallway tasks over 100,000 episodes. The highest achieved returns in each task are highlighted in bold together with all returns within a single standard deviation. For DeRL algorithms, evaluations are executed using the exploitation policy.

Alg	DeepSea 10	DeepSea 14	DeepSea 20	DeepSea 24	DeepSea 30	Hallway 10-0	Hallway 10-10	Hallway 20-0	Hallway 20-20	Hallway 30-0	Hallway 30-30
A2C	**0.93 ± 0.22**	0.00	0.00	0.00	0.00	0.67 ± 0.05	0.49 ± 0.09	0.42 ± 0.02	0.50 ± 0.03	0.28 ± 0.08	0.42 ± 0.08
A2C Count	**0.98 ± 0.07**	**0.94 ± 0.16**	**0.74 ± 0.10**	0.11 ± 0.15	−0.01	**0.85 ± 0.01**	**0.85 ± 0.02**	0.61 ± 0.03	**0.55 ± 0.06**	−0.33 ± 0.15	−0.06 ± 0.07
A2C Hash-Count	**0.98 ± 0.07**	**0.96 ± 0.15**	0.39 ± 0.14	**0.53 ± 0.12**	−0.01	**0.85 ± 0.01**	**0.85 ± 0.03**	0.56 ± 0.03	**0.55 ± 0.06**	−0.34 ± 0.15	−0.13 ± 0.11
A2C ICM	0.87 ± 0.20	0.69 ± 0.31	0.54 ± 0.23	**0.46 ± 0.30**	**0.08 ± 0.12**	0.62 ± 0.17	0.57 ± 0.17	0.27 ± 0.12	**0.78 ± 0.27**	**1.16 ± 0.47**	**0.64 ± 0.38**
A2C RND	0.06 ± 0.01	0.19 ± 0.02	−0.01	−0.01	−0.01	−0.12 ± 0.02	−0.07	−0.20 ± 0.01	−0.24	−0.24 ± 0.01	−0.12
A2C RIDE	0.00	0.00	0.00	0.00	0.00	**0.85 ± 0.04**	**0.85 ± 0.02**	**0.70**	**0.62**	0.37 ± 0.04	0.28 ± 0.08
PPO	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
PPO Count	0.84 ± 0.10	0.70 ± 0.17	0.46 ± 0.19	0.17 ± 0.18	**0.20 ± 0.15**	0.00 ± 0.01	0.00	0.00 ± 0.01	0.00 ± 0.01	0.00	0.00 ± 0.01
PPO Hash-Count	0.86 ± 0.08	0.77 ± 0.13	0.34 ± 0.14	0.28 ± 0.20	**0.12 ± 0.13**	0.39 ± 0.11	0.10 ± 0.07	0.00	0.00	0.00	0.00
PPO ICM	0.84 ± 0.17	0.28 ± 0.17	0.00 ± 0.03	0.12 ± 0.17	0.00 ± 0.03	0.05 ± 0.15	0.11 ± 0.15	0.02 ± 0.16	0.08 ± 0.19	−0.04 ± 0.08	−0.02 ± 0.14
PPO RND	0.26 ± 0.12	0.15 ± 0.08	−0.01	0.00	−0.01	−0.04 ± 0.04	−0.04 ± 0.11	−0.21 ± 0.06	−0.17 ± 0.09	−0.27 ± 0.10	−0.27 ± 0.11
PPO RIDE	0.73 ± 0.08	0.00	0.00 ± 0.02	−0.01	−0.01	−0.10 ± 0.03	0.02 ± 0.08	−0.21 ± 0.03	−0.08 ± 0.08	−0.32 ± 0.04	−0.29 ± 0.08
DeA2C Count	**0.98 ± 0.10**	0.65 ± 0.23	0.42 ± 0.16	0.07 ± 0.10	**0.09 ± 0.08**	**0.84 ± 0.07**	**0.84 ± 0.09**	0.42 ± 0.02	**0.70 ± 0.01**	0.55	0.22 ± 0.02
DeA2C ICM	0.86 ± 0.19	0.52 ± 0.28	0.27 ± 0.24	0.08 ± 0.14	**0.05 ± 0.11**	0.77 ± 0.18	0.80 ± 0.17	0.44 ± 0.15	**0.53 ± 0.20**	0.52 ± 0.34	**0.97 ± 0.51**
DePPO Count	0.61 ± 0.20	**0.92 ± 0.18**	−0.01 ± 0.01	**0.63 ± 0.27**	−0.01	0.73 ± 0.10	0.80 ± 0.08	0.56 ± 0.01	**0.55 ± 0.04**	−0.20 ± 0.17	−0.06 ± 0.07
DePPO ICM	0.61 ± 0.18	0.37 ± 0.17	0.00 ± 0.01	−0.01	0.00	0.82 ± 0.11	0.81 ± 0.11	0.64 ± 0.16	**0.57 ± 0.07**	−0.01 ± 0.25	0.26 ± 0.06
DeDQN Count	**0.98 ± 0.09**	**0.95 ± 0.17**	0.40 ± 0.08	**0.53 ± 0.27**	**0.10 ± 0.10**	−0.13 ± 0.04	−0.15 ± 0.04	−0.05 ± 0.05	−0.12 ± 0.08	−0.17 ± 0.07	−0.10 ± 0.06
DeDQN ICM	**0.94 ± 0.20**	0.59 ± 0.40	0.16 ± 0.12	0.24 ± 0.25	**0.05 ± 0.12**	−0.09 ± 0.09	0.02 ± 0.16	−0.11 ± 0.09	−0.19 ± 0.08	−0.26 ± 0.08	−0.19 ± 0.08

Decay of intrinsic rewards We also investigate the sensitivity of intrinsically motivated baselines and DeRL algorithms to the rate of decay of intrinsic rewards. For count-based and prediction-based intrinsic rewards, we use the increment of the state count $N(s)$ and the learning rate, respectively, to determine and vary the rate of decay. Figure 3.4 shows average evaluation returns of baselines and DeRL with varying rates of decay in both DeepSea $N = 10$ and Hallway $N_l = N_r = 10$. A2C is shown to be more robust to varying rates of decay in both environments compared to PPO. DePPO demonstrates larger sensitivity compared to A2C, but DeA2C and DeDQN are again shown to be the most robust algorithms, especially with Count intrinsic rewards, exhibiting high evaluation returns for most considered values in DeepSea $N = 10$. Similar to λ sensitivity, we observe very significant dependency on the rate of decay in the Hallway task with DeA2C exhibiting improved robustness to varying values.

3.4.2 Evaluation Returns

Lastly, we report evaluation returns of all algorithms across all DeepSea and Hallway tasks in Table 3.1. Average returns and standard deviations are computed across all 100 evaluations after being averaged across five seeds to indicate achieved returns as well as sample efficiency. Additionally, we present normalised returns with 95% confidence intervals across both environments in Figure 3.5.

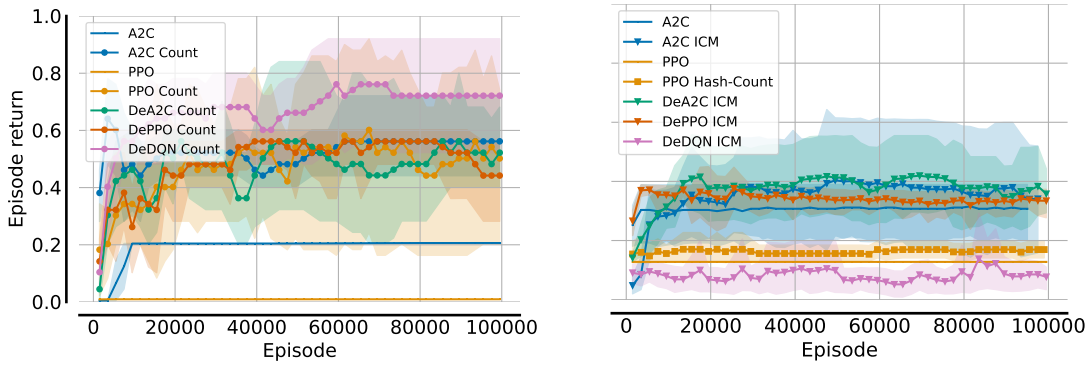


Figure 3.5: Normalised evaluation returns for DeepSea (left) and Hallway (right). Returns for each task are normalised to be within [0, 1] before averaged returns and 95% confidence intervals are computed across all tasks and five random seeds.

Tables with maximum achieved evaluation returns and learning curves for each individual task can be found in Appendix A.2.

In DeepSea, DeDQN performs best out of all algorithms (Figure 3.5). DeDQN converges to returns comparable to or higher than the best performing baselines exhibiting highest average evaluation returns in all tasks but DeepSea 20. DeA2C and DePPO demonstrate similar returns and sample efficiency in some of these tasks (see Figures A.1a and A.1b). In DeepSea 24 (Figure A.1d) and harder Hallway tasks with $N_l = 20, N_r = 0$ and $N_l = N_r = 30$ (Figures A.3b and A.3f), the exploitation policies of DeA2C and DePPO converge to the highest returns and are shown to be more sample efficient reaching high returns after up to 40% fewer episodes of training compared to the best performing baselines. Generally, we can see that DeA2C learns the optimal policy in the majority of Hallway tasks for some of the five executed runs, but fails to converge to such behaviour consistently. Instead, the majority of baselines and some DeRL runs learn to reach the goal but move back and forth between the goal and its left neighboured cell. Presumably, consistently staying at the goal is rarely discovered due to the small negative reward of staying at a cell.

However, we also observe some failure cases for DeRL algorithms. DeDQN achieves low returns in the Hallway environment compared to both on-policy DeA2C and DePPO. Also, significant variance can be observed for baselines and DeRL algorithms in harder DeepSea and most Hallway tasks. Off-policy optimisation is theoretically independent of the policy generating training samples, and in DeA2C and DePPO IS weight correction is applied to correct for the off-policy

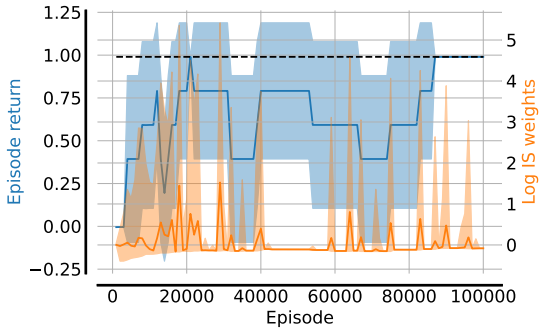


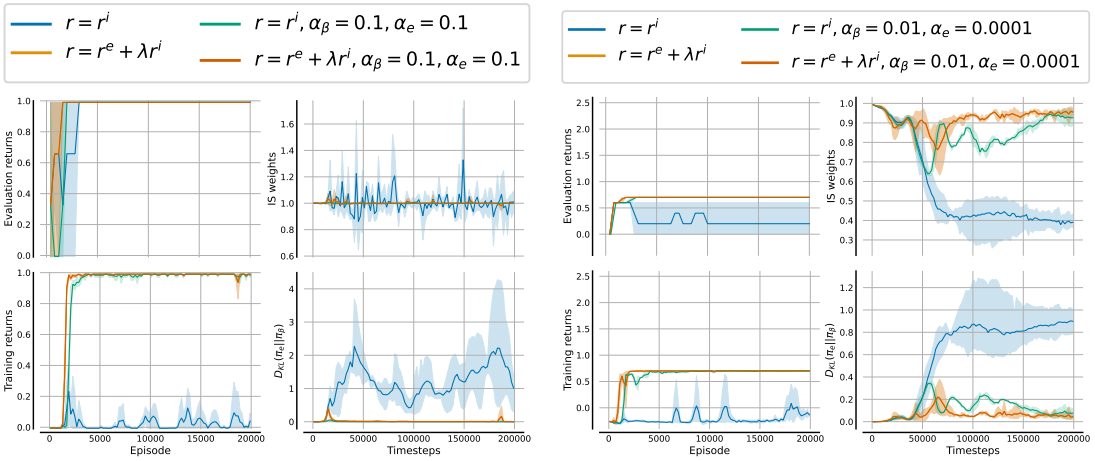
Figure 3.6: Evaluation returns and log importance sampling weights for DeA2C Count in DeepSea $N = 14$, showing the instability of returns and distribution shift between the exploration policy π_β and the exploitation policy π_e, as indicated by the spikes in importance sampling weights.

training data. However, we believe distribution shift (Fujimoto et al., 2018) is causing inconsistent returns when optimising the exploitation policy from data generated by π_β. Figure 3.6 visualises unstable IS weights for DeA2C in the DeepSea task with $N = 14$ averaged over five seeds. These appear to correlate with some of the noticeable drops in returns throughout training, indicating the negative impact of divergence of exploration and exploitation policies on RL training of π_e. Even when applying Retrace(λ) (Munos et al., 2016) to clip IS weights, similar results are observed.

3.4.3 Exploration using only Intrinsic Rewards

Prior work on intrinsic rewards for exploration investigated the effectiveness of training using only intrinsic rewards and no extrinsic rewards from the environment (Pathak et al., 2017; Burda et al., 2019b; Raileanu and Rocktäschel, 2020; Flet-Berliac et al., 2021). Motivated by these experiments, we also investigate the possibility of optimising the exploration policy π_β using only intrinsic rewards. Such optimisation would likely lead to increased robustness to hyperparameters of intrinsic rewards as they would not be combined with extrinsic rewards of the environment. However, we also find that the optimisation of π_β without extrinsic rewards causes further divergence of π_β and π_e. It should be noted that the evaluation policy is still trained using extrinsic rewards.

We conduct experiments in the DeepSea 10 and Hallway $N_l = N_r = 20$ tasks training DeA2C with Count intrinsic rewards for 20,000 episodes. Results are



(a) DeepSea $N = 10$ (b) Hallway $N_l = N_r = 20$

Figure 3.7: Evaluation and training returns, IS weights and KL-divergence of exploration and exploitation policy for training of DeA2C with Count in (a) DeepSea 10 and (b) Hallway $N_l = N_r = 20$ with intrinsic and extrinsic rewards or only intrinsic rewards as training signal for π_β, and with KL-divergence constraints with coefficients α_β and α_e. Evaluation and training returns are achieved using the exploitation and exploration policies, respectively. Shading indicates 95% confidence intervals.

averaged across three seeds and we directly compare optimising π_β using either the sum of extrinsic and intrinsic rewards (orange) or only using intrinsic rewards (blue) in Figure 3.7. First, we find that the exploration policy is unable to learn to solve the task, as shown by the training returns of π_β (bottom left) when only being trained with intrinsic rewards (blue). Second, we find that training the exploration policy with only intrinsic rewards does lead to increased divergence of both policies seen in IS weights (top right) and the KL divergence (bottom right). Training of the exploitation policy appears to suffer from such differences in the more challenging Hallway task, but in DeepSea the exploitation policy was successfully trained to solve the task (top left) despite the exploration policy never reaching high returns. Our results show the feasibility of training π_β using only intrinsic rewards but also the challenge of increased distribution shift.

3.4.4 Divergence Constraints

In order to address distribution shift caused by diverging exploration and exploitation policies, we investigate the application of divergence constraints proposed in the literature of offline RL (Levine et al., 2020). These auxiliary objectives are

introduced to the optimisation and enforce π_e and π_β to not diverge significantly by introducing a term $\alpha D(\pi_e, \pi_\beta)$ to the optimisation loss. This term is based on a distance measure D between the distribution of policies π_β and π_e and some weighting hyperparameter α. A common distance measure is the Kullback-Leibler (KL) divergence, which has been applied in offline RL (Jaques et al., 2019) to restrict divergence of policies, and can be written as follows:

$$D_{\mathrm{KL}}(\pi_e(s^t), \pi_\beta(s^t)) = \mathbb{E}_{a^t \sim \pi_e(s^t)}\Big[\log \pi_e(a \mid s^t) - \log \pi_\beta(a \mid s^t)\Big] \tag{3.15}$$

Wu et al. (2019) discuss further distances measures for policies but found most metrics to perform comparably.

In DeRL, divergence constraints can be directly applied to the optimisation of either the exploration π_β or exploitation policy π_e, i.e. can choose to keep π_β close to π_e and likewise can enforce π_e to stay close to π_β. We consider either of these directions as well as a combination of both constraints.

We evaluate the application of KL constraints as regularisers in the policy loss of the exploration policy, $\alpha_\beta D_{\mathrm{KL}}(\pi_\beta, \pi_e)$, and exploitation policy, $\alpha_e D_{\mathrm{KL}}(\pi_e, \pi_\beta)$, with varying weights α_β and α_e, respectively. These constraints are applied in both settings introduced in Section 3.4.3 with π_β being optimised using only intrinsic (green) or intrinsic and extrinsic rewards (red) with results shown in Figure 3.7 for selected constraint coefficients. We find KL divergence constraints successfully address distribution shift and thereby keep both policies close to each other, even if π_β is only trained using intrinsic rewards. Such minimised divergence also leads to reduced variability of returns in both tasks. These results indicate the feasibility of training π_β using only intrinsic rewards and the effectiveness of divergence constraints to minimise distribution shift. We further evaluate the sensitivity of DeA2C with KL divergence constraints but do not find such regularisation to significantly improve robustness. For figures showing distribution shift, training and evaluation returns for a range of KL constraint coefficients, α_β and α_e as well the conducted sensitivity analysis, see Appendix A.4.

3.5 Related Work

Decoupled policies Concurrently to our work, Whitney et al. (2021); Liu et al. (2021a,b) proposed alternative decoupling approaches. For the meta RL setting, Liu et al. (2021a) proposed to train separate exploration and exploitation poli-

cies guided by task-specific information for fast adaptation in novel tasks. For multi-agent RL, Liu et al. (2021b) trained separate exploration and exploitation policies using off-policy RL to focus coordinated exploration across multiple agents towards underexplored parts within the state space. However, a mixture of both policies is applied to explore whereas our work fully decouples both policies and their training. Furthermore, both of these approaches consider the meta-learning and multi-agent settings, respectively, and do not address the challenge of single-agent exploration we focus on.

Independently from our work, Whitney et al. (2021) proposed to train an exploration policy using only intrinsic rewards and train a task policy (corresponding to our exploitation policy π_e) using off-policy soft Double-DQN (van Hasselt et al., 2016). They apply a factored policy of both the task and exploration policies with optimisations focused on fast adaptation of the exploration policy. In contrast, we fully decouple both trained policies and find that training of the task policy using on-policy actor-critic algorithms with off-policy correction leads to higher returns and less sensitivity to hyperparameters in several tasks. Furthermore, we evaluate DeRL with several intrinsic rewards whereas Whitney et al. (2021) use a single count-based intrinsic reward.

Offline RL Offline RL is a related field to our work, that aims to learn a policy from a fixed dataset of transitions without further interaction with the environment. The training of the exploitation policy in DeRL can be considered to be offline RL, as it is trained using data generated by the exploration policy without ever interacting with the environment itself. However in contrast to our work, offline RL often assumes that the behaviour policy π_β used to generate the data is unknown. In contrast, in DeRL we do have access to both the exploitation policy π_e we train to solve the task as well as the behaviour policy in the form of the exploration policy π_β. This access to both the exploitation and behaviour policy allows us to directly optimise the policy used to gather data with this optimisation in mind. Furthermore, we can directly minimise the distribution shift caused by the exploration and exploitation policies, as studied in Section 3.4.4, which is only possible because we have access to and optimise both of these policies. However, similar to offline RL (Fujimoto et al., 2018; Levine et al., 2020), distribution shift remains a major challenge in our approach and can cause instabilities during the training of the exploitation policy.

Extensions of DeRL Further research has built on our DeRL work and extended it to make it more applicable to more complex exploration tasks and settings. Most notably, Chen et al. (2022) proposed a novel approach that extends DeRL with a constrained optimisation formulation. By formulating the optimisation of both policies as a constrained optimisation problem, they are able to compute the weighting hyperparameter λ in a principled way to optimally trade-off between exploration and exploitation, and alternatively optimise λ and the trained policies. They further find that collecting experiences from the exploration policy and exploitation policy, instead of collecting all experiences with the exploration policy as in our approach, is necessary for stable optimisation in more complex problems. Mark et al. (2023) recently leveraged decoupled policy optimisation for the offline-to-online setting in which a policy is trained offline and then fine-tuned online. They find that decoupling exploration and exploitation can lead to more robust and efficient fine-tuning of the offline-trained policy.

3.6 Conclusion

In this chapter, we proposed Decoupled RL (DeRL) which decouples exploration and exploitation into two separate policies. DeRL optimises the exploration policy with additional intrinsic rewards to incentivise exploration and trains the exploitation policy using only extrinsic rewards from data generated by the exploration policy. Based on this general framework, we formulate decoupled actor-critic and decoupled deep Q-networks and evaluated in two sparse-reward environments. Our results demonstrate that intrinsically motivated RL is highly dependent on careful hyperparameter tuning of intrinsic rewards, indicating the need for more robust solutions. We show that decoupling exploration and exploitation is possible and does lead to significant benefits in robustness to varying scale and rate of decay of intrinsic rewards. Furthermore, we identify distribution shift as a challenge in separating the RL optimisation into two policies with separate optimisation objectives and investigate the application of divergence constraints to minimise such divergence of both policies. Our results demonstrate the effectiveness of divergence constraint regularisation and indicate improved sample efficiency of DeRL in some tasks by reaching high returns in fewer interactions in the environment. Lastly, we demonstrated the feasibility of training the exploration policy using only intrinsic rewards, in particular alongside divergence

constraints to limit distribution shift. Overall, DeRL is a promising approach that demonstrates that exploration and exploitation are interconnected but fundamentally different problems and that decoupled optimisation for both objectives can lead to more robust and efficient learning.

Chapter 4

Benchmarking Cooperative Multi-Agent Reinforcement Learning Algorithms

Publication

This chapter is based on and adapted from the following publication:

Georgios Papoudakis, Filippos Christianos, **Lukas Schäfer**, and Stefano V. Albrecht. "Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks." In *Advances in Neural Information Processing Systems, Track on Datasets and Benchmarks*. 2021.

After discussing the challenge of exploration in the context of single-agent reinforcement learning in Chapter 3, we will now turn our attention to the setting of multi-agent reinforcement learning (MARL). First, we will present a benchmark study that we conducted to compare the performance of commonly used MARL algorithms across a diverse set of cooperative multi-agent tasks in this chapter. The results and analysis of this study will then inform the subsequent chapters in which we address identified limitations of existing algorithms. In this chapter, we focus on the common-reward setting under the Dec-POMDP formalism (Section 2.5).

As outlined in Section 2.6, MARL algorithms use RL techniques to co-train a set of agents in a multi-agent system. Recent years have seen a plethora of new MARL algorithms that integrate ideas from deep learning in MARL (Hernandez-

Leal et al., 2019). However, comparison of MARL algorithms is difficult due to a lack of established benchmark tasks, evaluation protocols, and metrics. While several comparative studies exist for single-agent RL (Duan et al., 2016; Henderson et al., 2018; Wang et al., 2019b), we are unaware of such comparative studies for recent deep MARL algorithms. We believe that a thorough benchmark of existing MARL algorithms would be important for the field to understand the relative strengths and limitations of algorithms, and inform future research.

With this motivation, we conduct an empirical comparison of nine MARL algorithms (listed in Table 2.1) across three classes of MARL algorithms: independent learning (Section 2.6.1), which applies single-agent RL algorithms for each agent without consideration of the multi-agent structure; multi-agent policy gradient algorithms with centralised critics (Section 2.6.2.2); and value decomposition (Section 2.6.2.1) algorithms. All algorithms are evaluated in a total of 25 different cooperative multi-agent tasks within two matrix games and four multi-agent environments.

For fair comparisons, we optimise the hyperparameters of all algorithm for each environment using a grid-search. We report identified best hyperparameters in Appendix B.5 for future research, and we report the maximum and average evaluation returns during training. MARL algorithms commonly share parameters of networks across agents to reduce computational cost and improve sample efficiency (Christianos et al., 2021). Due to the impact of parameter sharing on the learning process, we evaluate all algorithms in all tasks once with and once without parameter sharing. In addition to reporting detailed benchmark results, we analyse the results, and discuss insights regarding the effectiveness of different learning approaches under certain environment properties. Overall, this work has three main contributions:

1. We provide a **comprehensive benchmark of nine state-of-the-art MARL algorithms across 25 cooperative learning tasks**. We report maximum and average evaluation returns, report best-performing hyperparameters for each algorithm in each environment, and provide insights into the relative strengths and limitations of each algorithm.

2. We **open-source two new multi-agent environments** for future research: level-based foraging (LBF) and multi-robot warehouse (RWARE).[1]

[1]The environments are available at `https://github.com/uoe-agents/lb-foraging` and

3. We **open-source the Extended PyMARL (EPyMARL) codebase**,[2] an extension of the PyMARL codebase (Samvelyan et al., 2019), that includes implementations of five additional policy gradient algorithms (IA2C, IPPO, MADDPG, MAA2C, and MAPPO), allows for more flexible configuration of implementation details, and adds compatibility with more environments. Since its release, EPyMARL has received notable attention in the MARL research community with several projects leveraging the codebase for their research (e.g., Torbati et al., 2023; Leroy et al., 2023).

4.1 Multi-Agent Environments

We evaluate the algorithms in two finitely repeated matrix games and four multi-agent environments within which we define a total of 25 different learning tasks. We treat all tasks as common reward problems, i.e. all agents receive identical rewards at all time steps. For the level-based foraging and multi-robot warehouse environments, which are typically defined as general-sum problems, we treat them as common reward problems for this work. In this case, all agents receive rewards for successful collection of items and deliveries of requested shelves, respectively, irrespective of which agents contributed to such collections and deliveries. The considered tasks range over various properties including the degree of observability (whether agents can see the full environment state or only parts of it), reward density (receiving frequent/dense vs infrequent/sparse non-zero rewards), and the number of agents involved. Table 4.1 lists environments with properties, and we give more detailed descriptions below. Each of the following environments addresses a specific challenge of MARL.

4.1.1 Repeated Matrix Games

We consider two cooperative matrix games proposed by Claus and Boutilier (1998): the *climbing* and *penalty* game. The common-payoff matrices of the climbing and penalty game, respectively, are:

$$\begin{bmatrix} 0 & 6 & 5 \\ -30 & 7 & 0 \\ 11 & -30 & 0 \end{bmatrix} \qquad \begin{bmatrix} k & 0 & 10 \\ 0 & 2 & 0 \\ 10 & 0 & k \end{bmatrix}$$

https://github.com/uoe-agents/robotic-warehouse.

[2] https://github.com/uoe-agents/epymarl

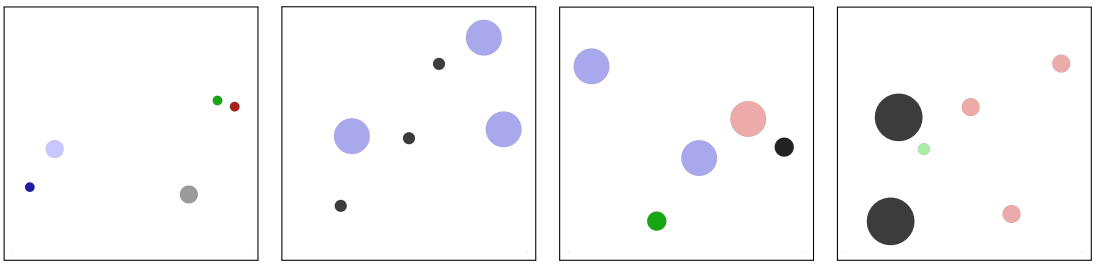
Table 4.1: Overview of environments and properties.

	Observability	Rew. Sparsity	Agents	Main Difficulty
Matrix Games	Full	Dense	2	Sub-optimal equilibria
MPE	Partial / Full	Dense	2-3	Coordination
SMAC	Partial	Dense	2-10	Partial observability, many agents
LBF	Partial / Full	Sparse	2-4	Coordination
RWARE	Partial	Sparse	2-4	Sparse reward, partial observability

where $k \leq 0$ is a penalty term. We evaluate in the penalty game for $k \in \{-100, -75, -50, -25, 0\}$. The difficulty of this game strongly correlates with k: the smaller k, the harder it becomes to identify the optimal policy due to the growing risk of penalty k. Both games are applied as repeated matrix games with an episode length of 25 and agents are given constant observations at each time step. These matrix games are challenging due to the existence of local minima in the form of sub-optimal Nash equilibria (Nash, 1951). Slight deviations from optimal policies by one of the agents can result in significant penalties, so agents might get stuck in risk-free (deviations from any agent does not significantly impede payoff) local optima.

4.1.2 Multi-Agent Particle Environment

The multi-agent particle environments (MPE) (Mordatch and Abbeel, 2018; Lowe et al., 2017) consists of several two-dimensional navigation tasks. We investigate four tasks that emphasise coordination: speaker-listener, spread, adversary, and predator-prey. Agent observations consist of feature vectors including relative agent and landmark locations. Unless specified otherwise, agents choose between five discrete actions consisting of movement in each cardinal direction and to do nothing. All tasks but speaker-listener are fully observable. MPE tasks serve as a benchmark for agent coordination due to the diverse and challenging coordination required by agents. Adversary and predator-prey are originally competitive tasks. To convert them into fully cooperative tasks, we control the adversary and prey, respectively, with a pre-trained policy obtained by training all agents with the



(a) (b) (c) (d)

Figure 4.1: Illustration of MPE tasks (a) speaker-listener, (b) spread, (c) adversary and (d) predator-prey.

MADDPG algorithm for 25,000 episodes. Below, we describe each task in more detail.

Speaker-listener In this task (Section 4.1.2), one static speaker agent has to communicate a goal landmark to a listening agent capable of moving. There are a total of three landmarks in the environment and both agents are rewarded with the negative Euclidean distance of the listener agent towards the goal landmark. The speaker agent only observes the colour of the goal landmark. Meanwhile, the listener agent observes its velocity, relative position to each landmark and the communication of the speaker agent as its observation. The speaker agent chooses between three possible actions to communicate to the listener agent, with which the agent needs to learn to encode the goal landmark, and the listener agent chooses among the usual movement actions.

Spread In this task (Section 4.1.2), three agents are trained to move to three landmarks while avoiding collisions with each other. All agents observe their velocity, position, relative position to all other agents and landmarks. Agents are rewarded with the sum of negative minimum distances from each landmark to any agent and an additional penalty for collisions of agents with each other.

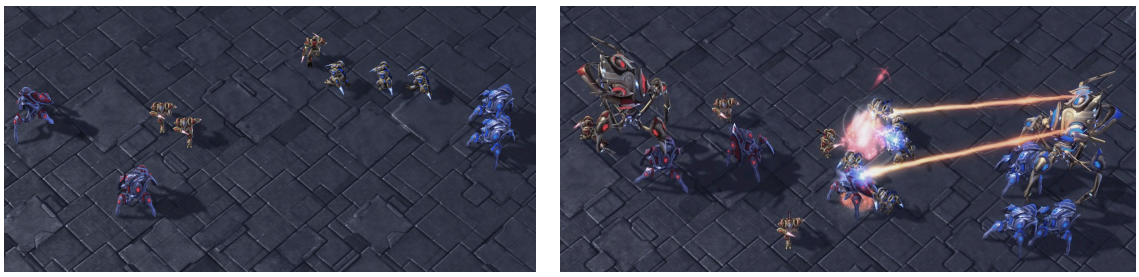
Adversary In this task (Section 4.1.2), two cooperating agents compete with a third adversary agent. There are two landmarks out of which one is randomly selected to be the goal landmark. Cooperative agents observe their relative position to the goal, relative position to all other agents, and relative positions to all landmarks. The adversary agent also observes all relative positions to agents and

landmarks but receives no information about which landmark is the goal. All agents are rewarded with the negative minimum distance to the goal, and the cooperative agents are additionally rewarded for the distance of the adversary agent to the goal landmark. Therefore, the cooperative agents have to move to both landmarks to avoid the adversary from identifying which landmark is the goal and reaching it as well. In our evaluation, we use a pre-trained policy to control the adversary agent and only train the two cooperative agents as a common-reward task.

Predator-prey In this task (Section 4.1.2), three cooperating predators hunt a forth agent controlling a faster prey in an environment with two landmarks as obstacles. All agents observe their own velocity and position as well as relative positions to all other landmarks and agents. Predator agents also observe the velocity of the prey. The agent controlling the prey is punished for any collisions with predators as well as for leaving the observable environment area (to prevent it from simply running away without needing to learn to evade). Predator agents are collectively rewarded for collisions with the prey. Similar to the Adversary scenario, we use a pre-trained policy for the prey and only train the predator agents as a common-reward task.

4.1.3 StarCraft Multi-Agent Challenge

The StarCraft multi-agent challenge (SMAC) (Samvelyan et al., 2019) is a set of fully cooperative, partially observable multi-agent tasks. The environment implements a variety of combat scenarios based on the popular real-time strategy game StarCraft II. Each task is defined by a map and a set of units within each team. The units of one team is controlled by MARL agents, with each agent controlling the actions of a single unit, and the opponent team of units being controlled by a fixed built-in AI of the StarCraft II game. Tasks vary in the number and types of units controlled by the agents as well as the map. Agents observe the health and other statistics of units within a fixed visibility radius, and for actions can choose to move in cardinal directions, select enemies to attack, or allies to heal (depending on the unit controlled). Agents receive common rewards including positive rewards for defeating enemy units and causing damage, and negative rewards for losing their own units and health. The primary challenge of SMAC tasks is in the partial observability and the large number of heterogeneous



(a) (b)

Figure 4.2: Examples of SMAC tasks with various team configurations and unit types.

agents in most tasks. We consider five tasks in SMAC, each with a different team configuration and unit types. We provide a brief description of each task below:

SMAC 2s_vs_1sc In this scenario, agents control two stalker units and defeat the enemy team consisting of a single, game-controlled spine crawler.

SMAC 3s5z In this symmetric scenario, each team controls three stalkers and five zerglings for a total of eight agents.

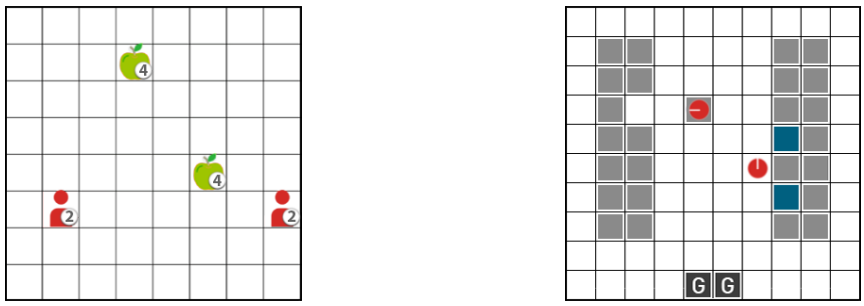
SMAC MMM2 In this symmetric scenario, each team controls seven marines, two marauders, and one medivac unit. The medivac unit assists other team members by healing them instead of inflicting damage to the enemy team.

SMAC corridor In this asymmetric scenario, agents control six zealots fighting an enemy team of 24 zerglings controlled by the game. This task requires agents to make effective use of terrain features to win.

SMAC 3s_vs_5z Finally, in this scenario a team of three stalkers is controlled by agents to fight against a team of five game-controlled zerglings.

4.1.4 Level-Based Foraging

In level-based foraging (LBF) (Albrecht and Ramamoorthy, 2013; Albrecht and Stone, 2017), agents must collect food items that are randomly scattered in a grid-world. Both agents and items are assigned levels at the beginning of each episode. By default, agents observe the level and position of all agents and food items



(a) Level-based foraging (LBF) (b) Multi-robot warehouse (RWARE)

Figure 4.3: Illustrations of the open-sourced multi-agent environments. From Christianos et al. (2020).

but partially observable variants of tasks exist that obfuscate the information of agents and items outside of a defined visibility threshold. Agents choose between six discrete actions: one no-op action, an action to attempt to pick up adjacent items, and one movement action for each of the four cardinal directions. Agents only receive a non-zero reward for successfully picking up food items so rewards are sparse. Agents are only successful in picking up items in neigbouring cells if the sum of the levels of all agents being next to the item and selecting the pick-up action is greater or equal to the level of item. Therefore, agents need to cooperate to pick up items with a higher level. The magnitude of rewards is normalised based on the level of food items such that the optimal returns within each episode, achieved for picking-up all food items, is equal to one.

LBF allows for many different tasks to be configured, including partial observability or a highly cooperative task where agents and item levels are assigned such that all agents need to cooperate to pick up the items. We define seven distinct tasks with a variable world size, number of agents, observability, and cooperation settings indicating whether all agents are required to load a food item or not.

4.1.5 Multi-Robot Warehouse

The multi-robot warehouse environment (RWARE) represents a partially observable environment with sparse rewards. RWARE simulates a grid-world warehouse in which agents (robots) must locate and deliver requested shelves to workstations and return them after delivery. Agents observe information about any agents and shelves in their immediate surrounding, and choose between five discrete actions: no-op, move forward, rotate left, rotate right, and load/unload a shelf. Agents

can not move forward if the cell is already occupied by another agent, and while carrying a shelf agents are only able to move to cells that do not contain shelves. To load a shelf, agents have to move to the same cell as the shelf and choose the load action. Similarly, agents are only able to unload a currently loaded shelf in a location that has currently no shelf but initially did store a shelf. Agents are only rewarded for delivering requested shelves by carrying them to the delivery locations and the bottom of the warehouse. After delivering a requested shelf, a new currently unrequested shelf is sampled uniformly at random and added to the list of requested shelves. In this way, the number of currently requested shelves remains constant.

We define three tasks that vary in size of the grid-world and number of agents. For all considered evaluation tasks, the number of requested shelves is equal to the number of agents at all times. RWARE is a challenging exploration task since agents need to follow a long sequence of actions to receive any rewards, making rewards in RWARE tasks highly sparse. Additionally, for agents to successfully deliver multiple requested shelves, after each delivery of a requested shelf they need to find a empty location in the warehouse to unload the currently carried shelf. They receive no reward for such unloading but need to unload their previously delivered shelf to be able to collect and deliver a new shelf. Furthermore, observations are sparse and high-dimensional compared to the other environments.

4.2 Evaluation

4.2.1 Evaluation Protocol

Off-policy MARL algorithms are typically more sample efficient compared to on-policy algorithms due to their ability to reuse previously collected samples to update their policies and value functions. However, they often also require more wall-clock time to train for the same number of time steps due to the frequency of updates. In our experiments, we collect data from a single environment for off-policy MARL algorithms and update once after an episode has been completed using a batch of 32 episodes sampled from the replay buffer. In contrast for on-policy MARL algorithms, we collect experiences from ten environments in parallel and update once with the just collected experience.

To account for this difference in update frequency per environment interactions and allow for fair comparisons, we train on-policy algorithms for a factor of ten times more samples than off-policy algorithms which results in all algorithms being trained for an identical number of total updates. In matrix games, we train on-policy algorithms for 2.5 million time steps and off-policy algorithms for 250 thousand time steps, in MPE and LBF we train on-policy algorithms for 20 million time steps and off-policy algorithms for two million time steps, while in SMAC and RWARE, we train on-policy and off-policy algorithms for 40 and four million time steps, respectively. Throughout training, we evaluate the current policies at regular intervals for a total of 41 evaluations. For each evaluation, we run 100 episodes and record the average achieved return.

As evaluation metrics, we identify the evaluation in which each algorithm achieves the highest average evaluation returns across five random seeds and report the respective evaluation return. This indicates the maximum returns achieved by the algorithm throughout training. To account for the sample efficiency, we also report the average returns achieved throughout all evaluations during training. Lastly, we show learning curves of the average returns and the 95% confidence intervals across five seeds for each algorithm and task.

4.2.2 Parameter Sharing

In deep MARL algorithms, it is common practise to share parameters of the networks across agents (e.g., Rashid et al., 2020b; Foerster et al., 2018; Mahajan et al., 2019). This reduces the number of parameters that need to be learned and often improves learning efficiency, but can significantly change the learning dynamics of the algorithm. To better understand the impact of parameter sharing on MARL algorithms, we train and evaluate all considered MARL algorithms in all tasks once with and once without parameter sharing. When sharing parameters across agents, the shared network receives a one-hot vector input that encodes the identify of the agent as an additional input. This input allows the network to learn specialised functions for each agent. The loss with parameter sharing is averaged over all agents and a single update of the shared parameters is performed for all agents. In the case of varying observation dimensions of multiple agents, inputs of agents with smaller input dimensions are zero-padded to ensure identical input dimensionality. Similarly if agents have varying numbers

of actions, invalid actions are masked out.

4.2.3 Hyperparameter Optimisation

To ensure fair comparisons, we conduct a hyperparameter search for each algorithm in one task of each environment. For each environment, we select one representative task and optimise the hyperparameters of all algorithms in this task. In the MPE environment, we perform the hyperparameter optimisation in the speaker-listener task, in the SMAC environment in the "3s5z" task, in the LBF environment in the "15x15-3p-5f" task, and in the RWARE environment in the "Tiny 4p" task. We train each combination of hyperparameters using three different seeds and compare the maximum evaluation returns. The best performing combination on each task is used for all tasks in the respective environment for the final experiments. To consider the effect of parameter sharing on the training, we also separately search for hyperparameters for each algorithm once with and once without parameter sharing. We report the best identified hyperparameters of each algorithm and environment in Appendix B.6. For IDQN, we compute target values using double deep Q-networks (van Hasselt et al., 2016; van Hasselt, 2010).

4.2.4 Computational Requirements

All experiments presented in this chapter were executed purely on CPUs. The experiments were executed in compute clusters that consist of several nodes. The main types of CPU models that were used for this work are Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz and AMD EPYC 7502 32-Core processors. All but the SMAC experiments were executed using a single CPU core. All SMAC experiments were executed using 5 CPU cores. The total number of CPU hours that were spent for executing the experiments in this chapter (excluding the hyperparameter search) are 138,916, corresponding to almost 16 years of CPU time.

4.2.5 Extended PyMARL

Implementation details in reinforcement learning significantly affect the returns that each algorithm achieves (e.g., Andrychowicz et al., 2021; Fu et al., 2022b).

To enable consistent evaluation of MARL algorithms, we open-source the Extended PyMARL (EPyMARL) codebase. EPyMARL is an extension of the PyMARL codebase (Samvelyan et al., 2019) that already provides implementations for IDQN, COMA, VDN and QMIX and is focused on the SMAC environment. We increase the scope of the codebase to include five additional policy gradients algorithms: IA2C, IPPO, MADDPG, MAA2C and MAPPO. Additionally, the original PyMARL codebase implementation assumes that agents share parameters and that all the agents' observation have the same shape. Since parameter sharing can act as an information bottleneck in environments with heterogeneous agents (Christianos et al., 2021), EPyMARL allows training MARL algorithms without parameter sharing, training agents with observations of varying dimensionality, and tuning several implementation details such as reward standardisation, entropy regularisation, and the use of recurrent or fully-connected networks. Lastly, EPyMARL integrates support for a diverse set of environments, including all environments used in the evaluation of this work. Since its release, EPyMARL has received wide attention in the research community, and has been built upon in several works (e.g., Leroy et al., 2023; Torbati et al., 2023).

As part of this thesis, we further extended EPyMARL to include support for training agents in general-sum environments with different reward functions. We also integrated support to log training data to the Weights and Biases platform, and updated the codebase alongside several environments (including LBF and RWARE) to be compatible with the maintained Gymnasium library in replacement of the deprecated OpenAI Gym library.

4.3 Results

In this section we compile the results across all environments and algorithms. Figure 4.4 presents the normalised evaluation returns in all environments, except matrix games. We normalise the returns of all algorithms in each task in the $[0, 1]$ range as follows:

$$\text{norm}(u_a^t) = \frac{u_a^t - \min_{a'}(u_{a'}^t)}{\max_{a'}(u_{a'}^t) - \min_{a'}(u_{a'}^t)} \tag{4.1}$$

where u_a^t is the episodic return of algorithm a in task t.

Table 4.3 and Table 4.2 present the maximum and average returns for the nine algorithms in all 25 tasks with parameter sharing, respectively. Maximum and av-

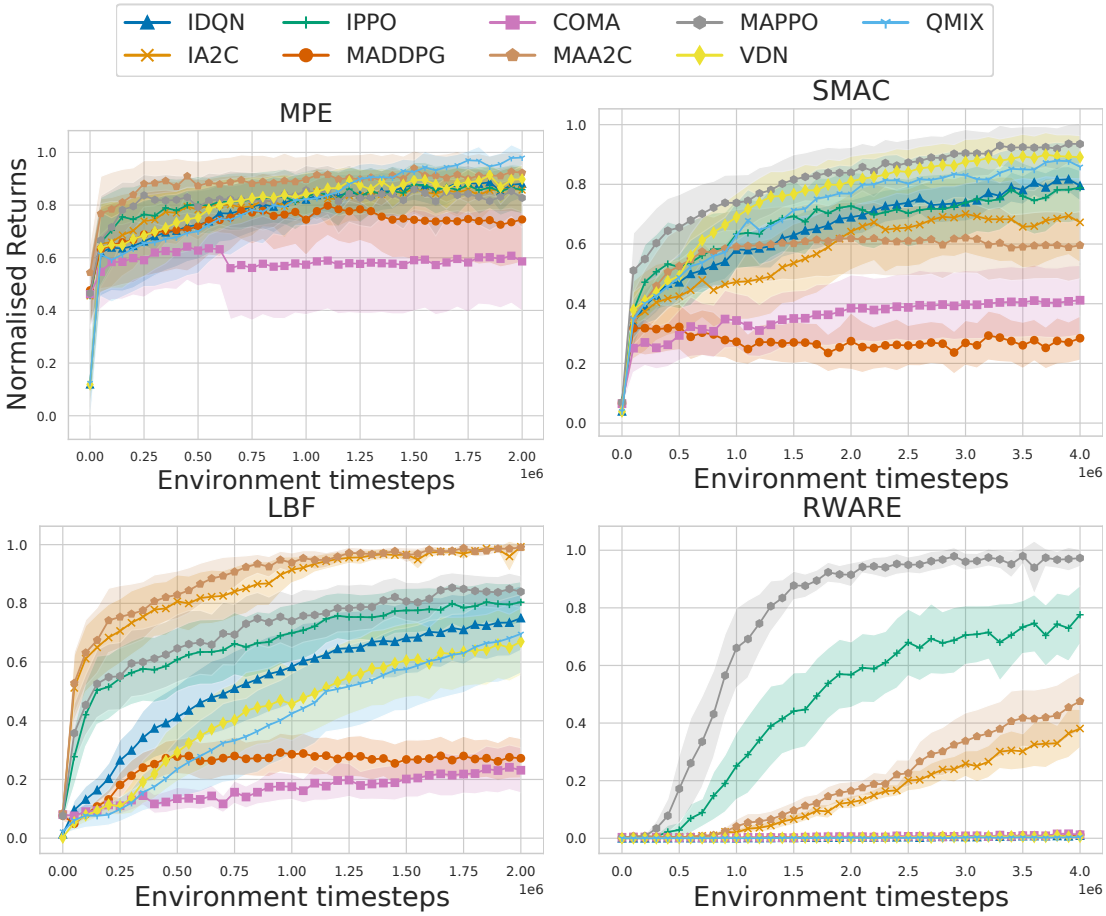


Figure 4.4: Normalised evaluation returns averaged over the tasks in the all environments except matrix games. Shadowed part represents the 95% confidence interval.

erage returns for all algorithms and tasks without parameter sharing can be found in Table 4.5 and Table 4.4, respectively. For tables reporting maximum returns, we highlight the highest performance in bold. Additionally, we performed two-sided t-tests with a significance threshold of 0.05 between the highest performing algorithm and each other algorithm in each task. If an algorithm's performance was *not* statistically significantly different from the best algorithm, the respective value is annotated with an asterisk (i.e. bold or asterisks in the table show the best performing algorithms per task). Learning curves with evaluation returns of all algorithms in all 25 tasks are shown in Figure B.2. Following common practice to report win-rates of algorithms as a percentage in SMAC tasks, we additionally report final win-rates achieved by all algorithms in all SMAC tasks in Appendix B.3. However, we found evaluation returns to be a more informative metric since it is what all algorithms are optimised to maximise. We further note that higher returns in SMAC do not always correspond to higher win-rates which

Table 4.2: Average returns and 95% confidence interval over five seeds for all nine algorithms with parameter sharing in all 25 tasks.

	Tasks \Algs.	IDQN	IA2C	IPPO	MADDPG	COMA	MAA2C	MAPPO	VDN	QMIX
Matrix Games	Climbing	134.65 ± 0.63	169.34 ± 1.09	170.70 ± 1.77	156.45 ± 8.09	177.31 ± 49.52	167.89 ± 4.36	170.76 ± 1.79	125.50 ± 0.54	125.50 ± 0.54
	Penalty k=0	245.64 ± 1.70	244.27 ± 1.13	247.44 ± 0.59	246.39 ± 0.45	245.63 ± 0.64	244.78 ± 0.87	247.69 ± 0.09	239.80 ± 2.93	243.76 ± 2.36
	Penalty k=-25	44.65 ± 2.67	48.29 ± 0.30	48.44 ± 0.21	48.59 ± 0.05	48.33 ± 0.20	48.45 ± 0.12	48.46 ± 0.22	43.32 ± 5.98	45.99 ± 3.27
	Penalty k=-50	39.70 ± 5.15	46.56 ± 0.57	46.79 ± 0.48	47.22 ± 0.17	46.61 ± 0.44	46.81 ± 0.28	46.82 ± 0.49	39.70 ± 9.63	42.28 ± 6.31
	Penalty k=-75	34.75 ± 7.62	44.82 ± 0.84	45.15 ± 0.75	45.83 ± 0.28	44.88 ± 0.68	45.16 ± 0.43	45.19 ± 0.76	34.75 ± 14.26	38.56 ± 9.34
	Penalty k=-100	29.80 ± 10.10	43.08 ± 1.13	43.50 ± 1.02	44.42 ± 0.36	43.15 ± 0.93	43.51 ± 0.59	43.55 ± 1.04	29.80 ± 18.89	34.85 ± 12.37
MPE	Speaker-Listener	−27.64 ± 3.90	−17.61 ± 2.99	−17.42 ± 3.23	−18.46 ± 0.68	−38.20 ± 6.29	−15.17 ± 0.44	−15.01 ± 0.64	−27.41 ± 3.11	−21.29 ± 2.79
	Spread	−155.81 ± 1.50	−152.72 ± 0.96	−149.89 ± 2.91	−157.10 ± 2.30	−245.22 ± 84.46	−144.73 ± 4.09	−149.26 ± 0.94	−148.57 ± 1.67	−154.70 ± 4.90
	Adversary	7.58 ± 0.14	10.18 ± 0.05	10.21 ± 0.16	7.80 ± 1.43	6.12 ± 0.35	10.11 ± 0.14	9.61 ± 0.07	7.64 ± 0.21	8.11 ± 0.37
	Tag	13.70 ± 1.97	12.43 ± 1.05	13.60 ± 2.95	6.65 ± 3.90	5.11 ± 0.58	11.93 ± 2.09	13.78 ± 4.40	15.24 ± 1.59	15.00 ± 2.73
SMAC	2s_vs_1sc	14.76 ± 0.45	19.74 ± 0.02	19.44 ± 0.29	10.15 ± 1.32	9.04 ± 0.83	17.89 ± 0.85	19.67 ± 0.09	16.11 ± 0.23	15.98 ± 0.77
	3s5z	14.09 ± 0.28	14.84 ± 1.29	11.80 ± 1.51	8.60 ± 2.35	15.51 ± 0.98	18.82 ± 0.14	19.09 ± 0.38	17.85 ± 0.25	18.36 ± 0.07
	corridor	10.91 ± 0.82	13.14 ± 1.24	14.60 ± 3.43	5.15 ± 0.25	7.00 ± 0.15	7.89 ± 0.28	13.20 ± 2.98	11.14 ± 1.66	11.67 ± 1.88
	MMM2	10.11 ± 0.32	7.31 ± 1.89	9.97 ± 1.33	3.42 ± 0.05	6.50 ± 0.17	9.07 ± 1.35	15.39 ± 0.16	15.93 ± 0.23	15.63 ± 0.32
	3s_vs_5z	17.35 ± 0.23	4.32 ± 0.04	13.38 ± 4.36	5.34 ± 0.47	1.15 ± 1.35	6.17 ± 0.39	13.09 ± 2.63	14.72 ± 4.01	9.68 ± 1.87
LBF	8x8-2p-2f-c	0.75 ± 0.04	0.97	0.94 ± 0.02	0.32 ± 0.02	0.32 ± 0.12	0.97	0.95 ± 0.01	0.64 ± 0.09	0.39 ± 0.10
	8x8-2p-2f-2s-c	0.86 ± 0.01	0.97	0.50 ± 0.01	0.54 ± 0.05	0.24 ± 0.08	0.97	0.77 ± 0.02	0.83 ± 0.01	0.77 ± 0.03
	10x10-3p-3f	0.54 ± 0.02	0.95 ± 0.01	0.90 ± 0.02	0.20 ± 0.06	0.15 ± 0.05	0.95 ± 0.01	0.91 ± 0.01	0.40 ± 0.05	0.32 ± 0.07
	10x10-3p-3f-2s	0.69 ± 0.02	0.84 ± 0.01	0.62 ± 0.01	0.27 ± 0.02	0.23 ± 0.06	0.85 ± 0.02	0.66 ± 0.01	0.64 ± 0.02	0.67 ± 0.01
	15x15-3p-5f	0.09 ± 0.02	0.61 ± 0.06	0.41 ± 0.09	0.08	0.06 ± 0.03	0.59 ± 0.09	0.43 ± 0.09	0.08 ± 0.01	0.04 ± 0.01
	15x15-4p-3f	0.24 ± 0.05	0.89 ± 0.03	0.82 ± 0.06	0.13 ± 0.01	0.12 ± 0.03	0.92 ± 0.01	0.79 ± 0.03	0.16 ± 0.03	0.08 ± 0.01
	15x15-4p-5f	0.15 ± 0.03	0.59 ± 0.06	0.40 ± 0.13	0.13 ± 0.01	0.07 ± 0.02	0.73 ± 0.02	0.39 ± 0.14	0.15 ± 0.02	0.09 ± 0.02
RWARE	Tiny 2p	0.04 ± 0.03	2.91 ± 0.45	12.63 ± 1.38	0.11 ± 0.07	0.13 ± 0.05	3.20 ± 0.41	15.42 ± 1.20	0.03 ± 0.01	0.03 ± 0.03
	Tiny 4p	0.33 ± 0.13	10.30 ± 0.93	22.68 ± 7.40	0.28 ± 0.03	0.39 ± 0.06	14.39 ± 4.01	40.17 ± 1.42	0.29 ± 0.13	0.10 ± 0.09
	Small 4p	0.03 ± 0.04	2.45 ± 0.18	9.19 ± 2.36	0.06 ± 0.02	0.08 ± 0.01	3.48 ± 0.42	18.12 ± 1.11	0.02 ± 0.03	0.01 ± 0.01

can make the interpretation of win-rate metrics more difficult.

4.3.1 Independent Learning

We find that IL algorithms perform adequately in all tasks despite their simplicity. However, performance of IL is limited in partially observable SMAC and RWARE tasks, compared to their CTDE counterparts, due to IL algorithms' inability to reason over joint information of agents.

IDQN IDQN performs significantly worse than the other IL algorithms in the partially observable speaker-listener task and in all RWARE tasks. IDQN is particularly effective in all but three LBF tasks, where relatively larger grid-worlds are used. IDQN achieves the best performance among all algorithms in the "3s_vs_5z" task, while it performs competitively in the rest of the SMAC tasks.

IA2C: The stochastic policy of IA2C appears to be particularly effective on all

Table 4.3: Maximum returns and 95% confidence interval over five seeds for all nine algorithms with parameter sharing in all 25 tasks. The highest value in each task is presented in bold. Asterisks denote the algorithms that are not significantly different from the best performing algorithm in each task.

	Tasks \Algs.	IDQN	IA2C	IPPO	MADDPG	COMA	MAA2C	MAPPO	VDN	QMIX
Matrix Games	Climbing	**195.00 ± 67.82**	175.00	175.00	170.00 ± 10.00	185.00 ± 48.99	175.00	175.00	175.00 ± 54.77	175.00 ± 54.77
	Penalty k=0	**250.00**	**250.00**	**250.00**	249.98 ± 0.04	**250.00**	**250.00**	**250.00**	**250.00**	**250.00**
	Penalty k=-25	**50.00**	**50.00**	**50.00**	49.97 ± 0.02	**50.00**	**50.00**	**50.00**	**50.00**	**50.00**
	Penalty k=-50	**50.00**	**50.00**	**50.00**	49.98 ± 0.02	**50.00**	**50.00**	**50.00**	**50.00**	**50.00**
	Penalty k=-75	**50.00**	**50.00**	**50.00**	49.97 ± 0.02	**50.00**	**50.00**	**50.00**	**50.00**	**50.00**
	Penalty k=-100	**50.00**	**50.00**	**50.00**	49.97 ± 0.03	**50.00**	**50.00**	**50.00**	**50.00**	**50.00**
MPE	Speaker-listener	−18.36 ± 4.67	−12.60 ± 3.62*	−13.10 ± 3.50	−13.56 ± 1.73	−30.40 ± 5.18	−10.71 ± 0.38*	**−10.68 ± 0.30**	−15.95 ± 2.48	−11.56 ± 0.53
	Spread	−132.63 ± 2.22	−134.43 ± 1.15	−133.86 ± 3.67	−141.70 ± 1.74	−204.31 ± 6.30	−129.90 ± 1.63*	−133.54 ± 3.08	−131.03 ± 1.85	**−126.62 ± 2.96**
	Adversary	9.38 ± 0.91	12.12 ± 0.44*	**12.17 ± 0.32**	8.97 ± 0.89	8.05 ± 0.89	12.06 ± 0.45*	11.30 ± 0.38	9.28 ± 0.90	9.67 ± 0.66
	Predator-prey	22.18 ± 2.83	17.44 ± 1.31	19.44 ± 2.94	12.50 ± 6.30	8.72 ± 4.42	19.95 ± 7.15*	18.52 ± 5.64	24.50 ± 2.19	**31.18 ± 3.81**
SMAC	2s_vs_1sc	16.72 ± 0.38	20.24	20.24 ± 0.01	13.14 ± 2.01	11.04 ± 7.21	20.20 ± 0.05*	**20.25**	18.04 ± 0.33	19.01 ± 0.40
	3s5z	16.44 ± 0.15	18.56 ± 1.31*	13.36 ± 2.08	12.04 ± 0.82	18.90 ± 1.01*	19.95 ± 0.05*	**20.39 ± 1.14**	19.57 ± 0.20*	19.66 ± 0.14*
	corridor	15.72 ± 1.77	**18.59 ± 0.62**	17.97 ± 3.44*	5.85 ± 0.58	7.75 ± 0.19	8.97 ± 0.29	17.14 ± 4.39*	15.25 ± 4.18*	16.45 ± 3.54*
	MMM2	13.69 ± 1.02	10.70 ± 2.77	11.37 ± 1.15	3.96 ± 0.32	6.95 ± 0.27	10.37 ± 1.95	17.78 ± 0.44	**18.49 ± 0.31**	18.40 ± 0.24*
	3s_vs_5z	**21.15 ± 0.41**	4.42 ± 0.02	19.36 ± 6.15*	5.99 ± 0.58	3.23 ± 0.05	6.68 ± 0.55	18.17 ± 4.17*	19.03 ± 5.77*	16.04 ± 2.87
LBF	8x8-2p-2f-c	**1.00**	**1.00**	**1.00**	0.46 ± 0.02	0.61 ± 0.30	**1.00**	**1.00**	**1.00**	0.96 ± 0.07*
	8x8-2p-2f-2s-c	**1.00**	**1.00**	0.78 ± 0.05	0.70 ± 0.04	0.45 ± 0.15	**1.00**	0.85 ± 0.06	**1.00**	**1.00**
	10x10-3p-3f	0.93 ± 0.02	**1.00**	0.98 ± 0.01	0.24 ± 0.04	0.19 ± 0.06	**1.00**	0.99 ± 0.01	0.84 ± 0.08	0.84 ± 0.08
	10x10-3p-3f-2s	0.86 ± 0.01	0.94 ± 0.03*	0.70 ± 0.03	0.41 ± 0.03	0.29 ± 0.12	**0.96 ± 0.02**	0.72 ± 0.03	0.90 ± 0.03	0.90 ± 0.01
	15x15-3p-5f	0.17 ± 0.08	**0.89 ± 0.04**	0.77 ± 0.08	0.10 ± 0.02	0.08 ± 0.04	0.87 ± 0.06*	0.77 ± 0.02	0.15 ± 0.02	0.09 ± 0.04
	15x15-4p-3f	0.54 ± 0.18	0.99 ± 0.01*	0.98 ± 0.01	0.17 ± 0.03	0.17 ± 0.04	**1.00**	0.96 ± 0.02	0.38 ± 0.13	0.15 ± 0.06
	15x15-4p-5f	0.22 ± 0.04	0.93 ± 0.03*	0.67 ± 0.22	0.12 ± 0.06	0.12 ± 0.06	**0.95 ± 0.01**	0.70 ± 0.25*	0.30 ± 0.04	0.25 ± 0.09
RWARE	Tiny 4p	0.72 ± 0.37	26.34 ± 4.60	31.82 ± 10.71	0.54 ± 0.10	1.16 ± 0.15	32.50 ± 9.79	**49.42 ± 1.22**	0.80 ± 0.28	0.30 ± 0.19
	Small 4p	0.14 ± 0.28	6.54 ± 1.15	19.78 ± 3.12	0.18 ± 0.12	0.16 ± 0.16	10.30 ± 1.48	**27.00 ± 1.80**	0.18 ± 0.27	0.06 ± 0.08
	Tiny 2p	0.28 ± 0.38	8.18 ± 1.25	20.22 ± 1.76*	0.44 ± 0.34	0.48 ± 0.34	8.38 ± 2.59	**21.16 ± 1.50**	0.12 ± 0.07	0.14 ± 0.19

environments except in a few SMAC tasks. In the majority of tasks, it performs similarly to IPPO with the exception of RWARE and some SMAC tasks. However, it achieves higher returns than IDQN in all but two SMAC tasks. Despite its simplicity, IA2C performs competitively compared to all CTDE algorithms, and significantly outperforms COMA and MADDPG in the majority of the tasks.

IPPO: IPPO in general performs competitively in all tasks across the different environments. On average (Figure 4.4) it achieves higher returns than IA2C in MPE, SMAC and RWARE tasks, but lower returns in the LBF tasks. IPPO also outperforms MAA2C in the partially observable RWARE tasks, but in general it performs worse compared to its centralised MAPPO version.

4.3.2 Centralised Training Decentralised Execution

Centralised training aims to learn powerful critics over joint observations and actions to enable reasoning over a larger information space. We find that learning such critics is valuable in tasks that require significant coordination under partial observability, such as the MPE speaker-listener and harder SMAC tasks. In contrast, IL is competitive compared to CTDE algorithms in fully-observable

Table 4.4: Average returns and 95% confidence interval over five seeds for all nine algorithms without parameter sharing in all 25 tasks.

	Tasks \Algs.	IDQN	IA2C	IPPO	MADDPG	COMA	MAA2C	MAPPO	VDN	QMIX
Matrix Games	Climbing	130.20 ± 4.37	164.72 ± 0.69	171.68 ± 0.41	150.05 ± 3.75	187.50 ± 41.19	169.43 ± 1.01	171.62 ± 0.39	132.52 ± 3.20	132.43 ± 3.37
	Penalty k=0	246.73 ± 1.39	243.65 ± 0.93	247.72 ± 0.04	247.88 ± 0.05	247.10 ± 0.38	246.61 ± 0.52	247.73 ± 0.03	247.03 ± 1.85	247.03 ± 0.99
	Penalty k=-25	49.70 ± 0.24	46.95 ± 0.16	88.05 ± 79.01	86.14 ± 75.19	48.32 ± 0.10	125.90 ± 95.14	88.06 ± 79.03	50.00 ± 0.99	50.00 ± 0.99
	Penalty k=-50	49.70 ± 0.24	44.31 ± 0.34	46.99 ± 0.23	47.12 ± 0.07	46.54 ± 0.20	46.56 ± 0.33	46.99 ± 0.24	50.00 ± 0.99	50.00 ± 0.99
	Penalty k=-76	49.70 ± 0.24	41.64 ± 0.50	45.44 ± 0.36	45.74 ± 0.12	44.77 ± 0.31	44.88 ± 0.47	45.44 ± 0.37	50.00 ± 0.99	50.00 ± 0.99
	Penalty k=-100	49.70 ± 0.24	38.94 ± 0.66	43.89 ± 0.48	44.33 ± 0.16	42.99 ± 0.41	43.21 ± 0.62	43.90 ± 0.52	50.00 ± 0.99	50.00 ± 0.99
MPE	Speaker-Listener	−30.49 ± 4.67	−23.33 ± 2.67	−22.78 ± 3.10	−17.79 ± 0.97	−33.88 ± 3.38	−19.48 ± 2.92	−20.51 ± 2.85	−29.49 ± 2.36	−20.31 ± 1.84
	Spread	−160.10 ± 1.59	−141.31 ± 3.86	−142.86 ± 4.49	−149.53 ± 2.41	−184.72 ± 2.99	−139.54 ± 4.78	−139.20 ± 4.58	−158.60 ± 2.06	−157.04 ± 1.38
	Adversary	7.82 ± 0.19	9.18 ± 1.52	9.46 ± 1.02	7.24 ± 2.13	7.28 ± 0.76	9.43 ± 1.93	10.20 ± 0.24	8.06 ± 0.27	8.81 ± 0.52
	Tag	12.59 ± 1.60	9.59 ± 4.30	11.90 ± 3.31	1.91 ± 2.09	14.28 ± 5.43	11.79 ± 5.40	10.90 ± 5.47	10.71 ± 0.69	14.27 ± 2.81
SMAC	2s_vs_1sc	13.37 ± 0.35	19.69 ± 0.05	8.59 ± 1.90	7.91 ± 0.16	15.32 ± 3.21	16.22 ± 3.75	19.68 ± 0.08	15.12 ± 0.46	15.66 ± 0.82
	3s5z	14.23 ± 0.84	12.65 ± 1.00	12.26 ± 0.98	7.65 ± 0.54	17.13 ± 1.11	17.94 ± 0.28	19.03 ± 0.19	17.06 ± 0.28	15.46 ± 0.59
	MMM2	8.27 ± 0.64	6.98 ± 1.72	6.23 ± 0.88	3.11 ± 0.12	3.82 ± 0.36	9.85 ± 0.19	9.41 ± 0.19	12.72 ± 0.56	8.05 ± 2.05
	corridor	8.38 ± 0.82	9.81 ± 1.98	7.80 ± 0.68	4.99 ± 0.17	7.85 ± 0.46	7.78 ± 0.73	8.18 ± 0.43	9.25 ± 0.85	10.76 ± 1.79
	3s_vs_5z	9.15 ± 0.46	4.26 ± 0.02	4.20 ± 0.22	3.53 ± 0.39	3.20 ± 0.89	4.91 ± 0.41	4.55 ± 0.02	10.85 ± 1.61	10.09 ± 1.10
LBF	8x8-2p-2f-c	0.95 ± 0.02	0.96 ± 0.01	0.91 ± 0.02	0.40 ± 0.05	0.63 ± 0.13	0.97	0.93 ± 0.02	0.93	0.90
	8x8-2p-2f-2s-c	0.97	0.96	0.50	0.52 ± 0.02	0.63 ± 0.08	0.97	0.79 ± 0.03	0.96	0.96
	10x10-3p-3f	0.77 ± 0.05	0.92 ± 0.01	0.85 ± 0.01	0.15	0.24 ± 0.03	0.92 ± 0.01	0.85 ± 0.02	0.80 ± 0.03	0.81 ± 0.04
	10x10-3p-3f-2s	0.78 ± 0.02	0.76 ± 0.01	0.61 ± 0.01	0.22 ± 0.03	0.23 ± 0.05	0.80	0.62	0.86 ± 0.02	0.86 ± 0.02
	15x15-3p-5f	0.11 ± 0.02	0.39 ± 0.09	0.33 ± 0.13	0.07	0.06	0.40 ± 0.03	0.24 ± 0.09	0.18 ± 0.01	0.08 ± 0.03
	15x15-4p-3f	0.29 ± 0.04	0.81 ± 0.01	0.60 ± 0.01	0.10 ± 0.01	0.16 ± 0.03	0.74 ± 0.03	0.65 ± 0.06	0.55 ± 0.06	0.12 ± 0.04
	15x15-4p-5f	0.17 ± 0.03	0.49 ± 0.10	0.40 ± 0.11	0.11	0.09 ± 0.01	0.44 ± 0.03	0.25 ± 0.04	0.25 ± 0.01	0.11 ± 0.02
RWARE	Tiny 2p	0.01 ± 0.01	2.02 ± 0.68	5.36 ± 3.05	0.07 ± 0.05	0.37 ± 0.04	2.23 ± 0.45	10.39 ± 3.04	0.01 ± 0.01	0.01 ± 0.01
	Tiny 4p	0.13 ± 0.04	6.10 ± 1.38	9.24 ± 4.20	0.25 ± 0.04	0.50 ± 0.12	10.73 ± 1.31	25.14 ± 1.44	0.10 ± 0.02	0.05 ± 0.02
	Small 4p	0.01	1.02 ± 0.10	3.64 ± 0.38	0.03 ± 0.02	0.07 ± 0.02	1.27 ± 0.09	7.06 ± 0.63	0.01 ± 0.01	0.01 ± 0.01

tasks of MPE and LBF. Our results also indicate that in most RWARE tasks, MAA2C and MAPPO significantly improve the achieved returns compared to their IL (IA2C and IPPO) versions. However, training state-action value functions appears challenging in RWARE tasks with sparse rewards, leading to very low performance of the remaining CTDE algorithms (MADDPG, COMA, VDN, and QMIX).

Centralised Multi-Agent Policy Gradient Centralised policy gradient methods vary significantly in performance.

MADDPG: MADDPG performs worse than all the other algorithms except COMA, in the majority of the tasks. It only performs competitively in some MPE tasks. It also exhibits very low returns in discrete grid-world environments LBF and RWARE. We believe that these results are a direct consequence of the biased categorical reparameterisation using gumbel-softmax (Jang et al., 2017). Recent

Table 4.5: Maximum returns and 95% confidence interval over five seeds for all nine algorithms without parameter sharing in all 25 tasks.

	Tasks \Algs.	IDQN	IA2C	IPPO	MADDPG	COMA	MAA2C	MAPPO	VDN	QMIX
Matrix Games	Climbing	150.00	175.00	175.00	170.00 ± 10.00	195.00 ± 40.00	175.00	175.00	150.00	155.00 ± 10.00
	Penalty k=0	250.00	250.00	250.00	249.94 ± 0.08	250.00	250.00	250.00	250.00	250.00
	Penalty k=-25	50.00	50.00	90.00 ± 80.00	89.99 ± 80.01	50.00	130.00 ± 97.98	90.00 ± 80.00	50.00±100.00	50.00±100.00
	Penalty k=-50	50.00	50.00	50.00	49.98 ± 0.01	50.00	50.00	50.00	50.00±100.00	50.00±100.00
	Penalty k=-75	50.00	50.00	50.00	49.98 ± 0.01	50.00	50.00	50.00	50.00±100.00	50.00±100.00
	Penalty k=-100	50.00	50.00	50.00	49.98 ± 0.01	50.00	50.00	50.00	50.00±100.00	50.00±100.00
MPE	Speaker-Listener	−18.61 ± 5.65	−17.08 ± 3.45	−15.56 ± 4.40*	−12.73 ± 0.73*	−26.50 ± 0.50	−13.66 ± 3.67*	−14.35 ± 3.56*	−15.47 ± 1.26	**−11.59 ± 0.67**
	Spread	−141.87 ± 1.68	−131.74 ± 4.33*	−132.46 ± 3.54*	−136.73 ± 0.83	−169.04 ± 2.72	−130.88 ± 2.44*	**−128.64 ± 2.83**	−142.13 ± 1.86	−130.97 ± 2.51*
	Adversary	9.09 ± 0.52	10.80 ± 1.97*	11.17 ± 0.85*	8.81 ± 0.61	9.18 ± 0.43	10.88 ± 2.43*	**12.04 ± 0.53**	9.34 ± 0.57	11.32 ± 0.78*
	Tag	19.18 ± 2.30	16.04 ± 8.08*	18.46 ± 5.19*	2.82 ± 3.56	19.14 ± 7.50*	26.50 ± 1.42*	17.96 ± 8.82*	18.44 ± 2.51	**26.88 ± 5.61**
SMAC	2s_vs_1sc	15.73 ± 1.08	20.23 ± 0.01	20.15 ± 0.10*	10.35 ± 2.20	18.48 ± 3.28*	19.88 ± 0.38*	**20.25 ± 0.01**	17.22 ± 0.90	18.83 ± 0.47
	3s5z	16.85 ± 1.43	14.44 ± 2.08	14.77 ± 2.51	15.05 ± 0.81	19.55 ± 0.45*	19.38 ± 0.30	**19.77 ± 0.11**	19.08 ± 0.29	18.40 ± 0.70
	MMM2	10.86 ± 1.04	8.38 ± 2.90	7.35 ± 0.48	4.92 ± 0.10	4.98 ± 0.42	10.79 ± 0.34	10.02 ± 0.19	**16.20 ± 0.44**	12.27 ± 2.28
	corridor	11.06 ± 1.36	13.11 ± 4.27*	10.29 ± 4.60	6.57 ± 0.53	8.34 ± 0.81	9.26 ± 3.08	8.51 ± 0.76	11.42 ± 2.18*	**15.12 ± 3.42**
	3s_vs_5z	11.80 ± 1.05*	4.41 ± 0.02	4.26 ± 0.24	6.21 ± 1.97	4.13 ± 0.05	5.39 ± 0.74	4.62 ± 0.11	14.42 ± 2.22*	**15.13 ± 3.86**
LBF	8x8-2p-2f-c	**1.00**	**1.00**	**1.00**	0.43 ± 0.02	0.95 ± 0.07*	**1.00**	**1.00**	0.99 ± 0.01*	0.57 ± 0.15
	8x8-2p-2f-2s-c	0.99 ± 0.01*	**1.00**	0.83 ± 0.03	0.66 ± 0.04	0.92 ± 0.02	**1.00**	0.92 ± 0.05	0.99 ± 0.01*	0.98 ± 0.01
	10x10-3p-3f	0.54 ± 0.12	**1.00 ± 0.01**	0.96 ± 0.01	0.20 ± 0.03	0.31 ± 0.08	0.99 ± 0.01*	0.98 ± 0.01	0.39 ± 0.06	0.22 ± 0.03
	10x10-3p-3f-2s	0.77 ± 0.06	0.89 ± 0.02	0.72 ± 0.02	0.29 ± 0.02	0.27 ± 0.09	**0.96 ± 0.01**	0.70 ± 0.04	0.76 ± 0.04	0.78 ± 0.05
	15x15-3p-5f	0.11 ± 0.02	**0.74 ± 0.12**	0.62 ± 0.12*	0.10 ± 0.01	0.09 ± 0.02	0.72 ± 0.05*	0.49 ± 0.19*	0.11 ± 0.01	0.06 ± 0.01
	15x15-4p-3f	0.16 ± 0.03	**0.99**	0.90 ± 0.01	0.17 ± 0.01	0.21 ± 0.07	0.94 ± 0.05*	0.89 ± 0.02	0.13 ± 0.02	0.10 ± 0.01
	15x15-4p-5f	0.17	**0.77 ± 0.10**	0.69 ± 0.05*	0.15 ± 0.01	0.14 ± 0.03	0.76 ± 0.04*	0.45 ± 0.17	0.14 ± 0.01	0.08 ± 0.01
RWARE	Tiny 2p	0.06 ± 0.08	5.56 ± 1.68	8.70 ± 4.04	0.24 ± 0.21	1.46 ± 0.34	6.16 ± 1.94	**17.48 ± 4.52**	0.06 ± 0.08	0.06 ± 0.08
	Tiny 4p	0.44 ± 0.10	18.02 ± 5.87	14.10 ± 5.20	0.44 ± 0.24	1.48 ± 0.48	31.56 ± 4.29	**38.74 ± 2.99**	0.34 ± 0.17	0.16 ± 0.15
	Small 4p	0.04 ± 0.05	3.10 ± 0.68	5.78 ± 0.42	0.16 ± 0.12	0.16 ± 0.10	5.00 ± 0.68	**13.78 ± 1.63**	0.04 ± 0.05	0.06 ± 0.08

work confirmed this hypothesis and showed that alternative reparameterisation techniques can improve the performance of the MADDPG algorithm by reducing its bias (Tilbury et al., 2023).

COMA: In general, COMA exhibits one of the lowest performances in most tasks and only performs competitively in one SMAC task. We found that COMA suffers very high variance in the computation of the counterfactual advantage. In the speaker-listener task, it fails to find the sub-optimal local minima solution that correspond to returns around to -17. Additionally, it does not exhibit any learning in the RWARE tasks in contrast to other on-policy algorithms.

MAA2C: MAA2C in general performs competitively in the majority of the tasks, except a couple of SMAC tasks. Compared to MAPPO, MAA2C achieves slightly higher returns in the MPE and the LBF tasks, but in most cases significantly lower returns in the SMAC and RWARE tasks.

MAPPO: MAPPO achieves high returns in the vast majority of tasks and only performs slightly worse than other algorithms in some MPE and LBF tasks.

Its main advantage is the combination of on-policy optimisation with its surrogate objective that significantly improves the sample efficiency compared to MAA2C. Its benefits can be observed in RWARE tasks where its returns exceed the returns of all other algorithms, even if not always by statistically significant margins.

Value Decomposition Value decomposition is an effective approach in most environments. In the majority of tasks across all environments except RWARE, VDN and QMIX outperform or at least match the highest returns of any other algorithm. This suggests that VDN and QMIX share the major advantages of centralised training also seen in MAA2C and MAPPO. In RWARE, VDN and QMIX do not exhibit any learning, similar to IDQN, COMA and MADDPG, indicating that value decomposition methods require sufficiently dense rewards to successfully learn to decompose the value function into the individual agents.

VDN: While VDN and QMIX perform similarly in most environments, the difference in performance is most noticeable in some MPE tasks. It appears VDN's assumption of linear value function decomposition is mostly violated in this environment. In contrast, VDN and QMIX perform similarly in most SMAC tasks and across all LBF tasks, where it seems that the global utility can be sufficiently accurately represented by a linear decomposition into agent utilities.

QMIX: Across almost all tasks, QMIX achieves consistently high returns, but does not necessarily achieve the highest returns among all algorithms. Its value function decomposition allows QMIX to achieve slightly higher returns in some of the more complicated tasks where the linear value decomposition of VDN in appears insufficient.

4.3.3 Parameter Sharing

Figure 4.5 presents the normalised maximum returns averaged over the nine algorithms and tasks with and without parameter sharing in all environments. We observe that in all environments except the matrix games, parameter sharing improves the returns over no parameter sharing. While the average values presented in Figure 4.5 do not seem statistically significant, by looking closer in Tables 4.3 and 4.5 we observe that in several cases of algorithm-task pairs the improvement due to parameter sharing seems significant. Such improvements can be observed for most algorithms in MPE tasks, especially in speaker-listener and predator-prey. For SMAC, we observe that parameter sharing improves the returns in

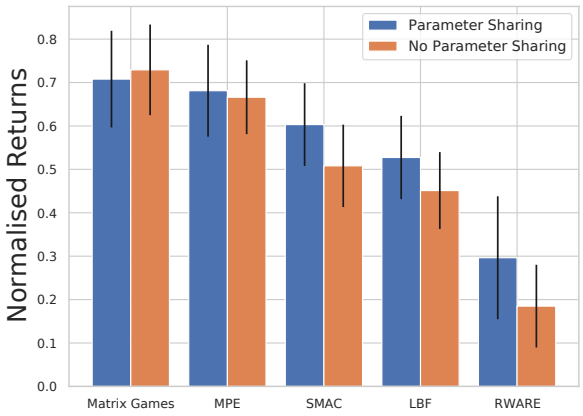


Figure 4.5: Normalised maximum returns averaged over all algorithms with/without parameter sharing (with standard error).

harder tasks. Similar observations can be made for LBF and RWARE. In these environments, the return improvement of parameter sharing appears to correlate with the sparsity of rewards. For tasks with larger grid-worlds or fewer agents, where the reward is more sparse, parameter sharing leads to large increases in returns compared to simpler tasks. These results indicate that parameter sharing constitutes a trade-off between improving sample efficiency and limiting the expressiveness of the individual agents' policies. By sharing parameters, the trajectories of all agents are used to train the same network which improves sample efficiency. However, using the same network for all agents also limits their ability to learn specialised policies that might be beneficial in some tasks. Overall, the results suggest that parameter sharing is beneficial in many environments, likely indicating that these environments require agents to learn similar or identical policies, or that without parameter sharing the sample efficiency is too low to learn effective policies within the training budget.

4.4 Analysis

Independent learning can be effective in multi-agent systems. Why and when? It is often stated that IL is inferior to centralised training methods due to the environment becoming non-stationary from the perspective of any individual agent. This is true in many cases and particularly crucial when IL is paired with off-policy training from an experience replay buffer, as pointed out by Lowe et al.

(2017). In our experiments, IDQN trains agents independently using such a replay buffer and is thereby limited in its performance in tasks that require extensive coordination among the agents. There, agents depend on the information about other agents and their current behaviour to choose well-coordinated actions and hence learning such policies from a replay buffer (where other agents differ in their behaviour) appears infeasible. However, this is not a general concern in multi-agent environments. In smaller SMAC tasks and most LBF tasks, each agent can independently learn a policy that achieves relatively high returns by utilising only its local observation history, and without requiring extensive coordination with other agents. For example in LBF, agents "only" need to learn to move to and pick up food. Of course, they will have to coordinate such behaviour with other agents, but naively going to food (especially when others are also close) and attempting to pick it up can be a viable local optima policy, and hard to improve upon. Whenever more complicated coordination is required, such as simultaneously picking up an item with higher level, exploring and learning those joint actions becomes difficult. IA2C and IPPO on the other hand learn policies from on-policy trajectories, so the behaviour of other agents observed within the trajectories represents the current policies of other agents. It appears that such on-policy training can improve the stability of the learning process despite the remaining non-stationarity of the policies of other agents. Recent work further supports this hypothesis by showing that off-policy DRL algorithms can be more effectively and stably trained from on-policy trajectories compared to training from off-policy trajectories sampled from a replay buffer (Gallici et al., 2024).

Centralised information can improve coordination under partial-observability. We note that the availability of joint observations and actions over all agents serves as a powerful training signal to optimise individual policies whenever the full state is not available to individual agents. Comparing the performance in Table 4.3 of IA2C and MAA2C, two almost identical algorithms aside from their critics, we notice that MAA2C achieves equal or higher returns in the majority of the tasks. This difference is particularly significant in tasks where agent observations lack important information about other agents or parts of the environment outside of their receptive field due to partial observability. This can be observed in RWARE tasks with 4 agents that requires extensive coordination so that agents are not stuck in the narrow passages. However, in RWARE Tiny

2p task, the performance of IA2C and MAA2C is similar as only two agents rarely get stuck in the narrow passages. Finally, IA2C and MAA2C have access to the same information in fully-observable tasks, such as most LBF and MPE tasks, leading to similar returns. A similar pattern can be observed for IPPO and MAPPO. However, we also observe that centralised training algorithms such as COMA, MADDPG, VDN, and QMIX are unable to learn effective behaviour in the partially observable RWARE. We hypothesise that training larger networks over the joint observation- and action-space, as required for these algorithms, demands sufficiently dense rewards. However, rewards are sparse in RWARE and observations are comparably large which makes learning centralised action-value functions difficult.

Value decomposition – VDN vs QMIX. Lastly, we discuss the differences observed in value decomposition applied by VDN and QMIX. Such decomposition offers significant improvements in comparison to the otherwise similar IDQN algorithm across most tasks. As detailed in Section 2.6.2.1, VDN and QMIX differ in their decomposition of the centralised action-value function with VDN assuming a linear decomposition while QMIX uses a more expressive monotonic decomposition. With the additional flexibility of its decomposition, QMIX aims to improve learnability, i.e. it simplifies the learning objective for each agent to maximise, while ensuring the global objective is maximised by all agents (Agogino and Tumer, 2008). Such flexibility appears to mostly benefit convergence in harder MPE tasks, such as speaker-listener and predator-prey, but comes at additional expense seen in environments like LBF, where the decomposition did not have to be complex. It appears that the dependency of rewards with respect to complicated interactions between agents in MPE tasks benefits from the more complex decomposition of QMIX. Finally, VDN and QMIX perform significantly worse than the policy gradient methods (except COMA) in the sparse-reward RWARE tasks. This does not come as a surprise, since the utility of the agents is rarely greater than 0, which makes it hard to successfully learn the individual utilities.

4.5 Conclusion

We evaluated nine MARL algorithms in a total of 25 cooperative learning tasks, including combinations of partial and full observability, sparse and dense rewards,

and a number of agents ranging from two to ten. We compared algorithm performance in terms of maximum and average returns. Additionally, we further analysed the effectiveness of independent learning, centralised training, and value decomposition and identified types of environments in which each strategy is expected to perform well. We furthermore identified that many MARL algorithms, in particular value-based MARL algorithms, are comparably ineffective in tasks with highly sparse rewards such as RWARE. We also found that parameter sharing can improve the performance of MARL algorithms in most tasks. To provide additional value to the MARL research community, we created EPyMARL as an open-source codebase for consistent evaluation of MARL algorithms in cooperative tasks, and provided LBF and RWARE as two new open-source multi-agent environments with sparse rewards. Our work is limited to cooperative environments and commonly-used MARL algorithms. Competitive environments as well as solutions to a variety of MARL challenges such as exploration, communication, and opponent modelling require additional studies in the future.

Chapter 5

Experience Sharing for Multi-Agent Reinforcement Learning

Publication

This chapter is based on and adapted from the following publication:

Filippos Christianos, **Lukas Schäfer**, and Stefano V. Albrecht. "Shared experience actor-critic for multi-agent reinforcement learning." In *Advances in Neural Information Processing Systems*. 2020.

As we have seen in the previous chapter, many MARL algorithms still struggle to learn efficient policies in environments with sparse rewards such as multi-robot warehouse (RWARE) and level-based foraging (LBF). In this chapter, we focus on this challenge of MARL in environments with sparse rewards, and study it under the more general POSG formalism (Section 2.4). Motivated by the observation that many environments require agents to learn similar but not identical policies, we propose a new algorithm for efficient MARL exploration based on the idea of agents *sharing experience* with each other.

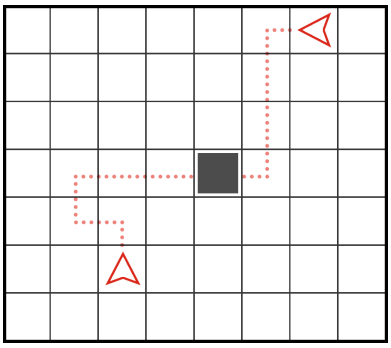


Figure 5.1: Two randomly-placed agents (triangles) must simultaneously arrive at the goal (square).

Consider the simple multi-agent game shown in Figure 5.1 in which two agents must simultaneously arrive at a goal. This game presents a difficult exploration

problem, requiring the agents to wander for a long period before stumbling upon a reward. When the agents finally succeed, the idea of sharing experience is appealing: both agents can learn how to approach the goal from two different directions after a successful episode by leveraging their collective experience. Such experience sharing facilitates a steady progression of all learning agents, meaning that agents improve at approximately equal rates as opposed to diverging in their learning progress. We show in our experiments that this approach of experience sharing can lead to significantly faster learning and higher final returns.

We demonstrate this idea in a novel actor-critic MARL algorithm, called *shared experience actor-critic* (SEAC).[1] SEAC operates similarly to independent learning (Tan, 1993) but updates the actor and critic parameters of an agent by combining gradients computed on the agent's experience with weighted gradients computed on other agents' experiences. We evaluate SEAC in ten sparse-reward multi-agent tasks across four environments and find that it learns substantially faster (up to 70% fewer required training steps) and achieves higher final returns compared to several baselines, including: independent learning without experience sharing; using data from all agents to train a single shared policy; and MADDPG (Lowe et al., 2017), QMIX (Rashid et al., 2020b), and ROMA (Wang et al., 2020a). Lastly, we show that SEAC only neglectably increases the computation cost of training, increasing training time per time steps by less than 3% across all environments compared to independent learning, despite significantly improving sample efficiency.

5.1 Shared Experience Actor-Critic

Our goal is to enable more efficient learning by sharing experience among agents. To facilitate experience sharing, we assume environments in which the local policy gradients of agents provide useful learning directions for all agents. Intuitively, this means that agents can learn from the experiences of other agents without necessarily having identical reward functions. We formalise this assumption in Section 5.2 and provide examples of environments that fulfil this property in Section 5.3.

In each episode, each agent generates one on-policy trajectory. Usually, when training agents with an on-policy RL algorithm, each agent is training its networks

[1]We provide open-source implementations of SEAC in `www.github.com/uoe-agents/seac`.

using only its own observations, actions, and rewards. Here, we propose to also use such trajectories of other agents while considering that it is *off-policy* data, i.e. the trajectories are generated by agents executing different policies than the one optimised. One possibility to use such off-policy samples is to correct the difference in data distribution using importance sampling weights. The policy loss for a single-agent advantage actor-critic algorithm (Section 2.3.3) using such off-policy experience from a behavioural policy can be written as:

$$\mathcal{L}(\phi) = \mathbb{E}_{a^t \sim \beta(h^t)} \left[-\frac{\pi(a^t \mid h^t; \phi)}{\beta(a^t \mid h^t)} \log \pi(a^t \mid h^t; \phi) \left(r^t + \gamma V(h^{t+1}; \theta) - V(h^t; \theta) \right) \right] \tag{5.1}$$

with β denoting the behavioural policy that collected the experience, π with parameters ϕ denoting the target policy we update, and V denoting a state value function with parameters θ.

We now combine the on-policy advantage actor-critic policy loss for a independent learning multi-agent algorithm with the off-policy policy loss using the experience of all the other agents. We show the loss for agent i and denote the indices of all other agents with $k \in I \setminus \{i\}$:

$$\begin{aligned} \mathcal{L}(\phi_i) = \mathbb{E}_{\substack{a_i^t \sim \pi(h_i^t; \phi_i) \\ a_k^t \sim \pi(h_k^t; \phi_k)}} \Big[& -\log \pi(a_i^t \mid h_i^t; \phi_i) \left(r_i^t + \gamma V(h_i^{t+1}; \theta_i) - V(h_i^t; \theta_i) \right) \\ & -\lambda \sum_{k \in I \setminus \{i\}} \frac{\pi(a_k^t \mid h_k^t; \phi_i)}{\pi(a_k^t \mid h_k^t; \phi_k)} \log \pi(a_k^t \mid h_k^t; \phi_i) \left(r_k^t + \gamma V(h_k^{t+1}; \theta_i) - V(h_k^t; \theta_i) \right) \Big] \end{aligned} \tag{5.2}$$

Using this loss function, each agent is trained on both on-policy data sampled from its own policy and the off-policy data collected by all other agents at each training step. The value loss can leverage the experience of other agents in a similar fashion:

$$\begin{aligned} \mathcal{L}(\theta_i) = \mathbb{E}_{\substack{a_i^t \sim \pi(h_i^t; \phi_i) \\ a_k^t \sim \pi(h_k^t; \phi_k)}} \Big[& \left(r_i^t + \gamma V(h_i^{t+1}; \theta_i) - V(h_i^t; \theta_i) \right)^2 \\ & +\lambda \sum_{k \in I \setminus \{i\}} \frac{\pi(a_k^t \mid h_k^t; \phi_i)}{\pi(a_k^t \mid h_k^t; \phi_k)} \left(r_k^t + \gamma V(h_k^{t+1}; \theta_i) - V(h_k^t; \theta_i) \right)^2 \Big] \end{aligned} \tag{5.3}$$

The hyperparameter λ in Equation 5.2 and Equation 5.3 weights the experience of other agents, i.e. for small values of λ, the agent learns mostly from its own experience, while for large values of λ, the agent learns mostly from the experience of other agents. We show in our experiments that SEAC is comparably insensitive to values of λ (Section 5.3.2) and use $\lambda = 1$ in our experiments. We refer to

Algorithm 3 Shared Experience Actor-Critic Training

Initialise: parameters ϕ_i and θ_i of the policy and value function of agent i for each agent $i \in I$
for each episode **do**
  Obtain initial state $s^0 \sim \mu$ and joint observation $\mathbf{o}^0$
  for each step $t = 0, \cdots, T$ **do**
    for each agent $i \in I$ **do**
      Select action $a_i^t \sim \pi(h_i^t; \theta_i)$
    end for
    Apply joint action $\mathbf{a}^t = (a_1^t, \ldots, a_N^t)$
    Receive next state $s^{t+1} \sim \mathcal{T}(s^t, \mathbf{a}^t)$, rewards $r_i^t = \mathcal{R}_i(s^t, \mathbf{a}^t, s^{t+1})$ for each agent i, and joint observation $\mathbf{o}^{t+1} \sim \mathcal{O}(s^t, \mathbf{a}^t)$
    for each agent $i \in I$ **do**
      Update ϕ_i by minimising $\mathcal{L}(\phi_i)$ (Equation 5.2)
      Update θ_i by minimising $\mathcal{L}(\theta_i)$ (Equation 5.3)
    end for
  end for
end for

the resulting algorithm as *shared experience actor-critic* (SEAC) and provide pseudocode in Algorithm 3.

5.2 Formal Derivation

Within the POSG formalism, we assume that all agents have identical observation and action spaces, i.e. $O_1 = \ldots = O_N$ and $A_1 = \ldots = A_N$, which we will denote with O and A thereafter. We note that this is a common assumption made in the practical implementations of MARL algorithms.

Furthermore, for agents to be able to sensibly share experiences with each other following SEAC, we assume that the environment is symmetrical. Intuitively, we consider an environment symmetrical if agents can swap places and receive the same next observations and rewards after applying an action as the other agent they swapped places with would have received when applying the same action. We note that the agents within the SEAC algorithm are unable to identify from their experiences whether an environment is symmetrical or not,

so it is within the responsibility of the algorithm designer to verify that this assumption is uphold. Below, we will formalise the assumptions underlying this symmetry for the simplified case of a fully observable two-agent stochastic game where agents are able to observe the full state of the environment.[2]

Assumption 1 (Symmetry Assumption)**.** *We assume there exists a function* $f : S \mapsto S$ *such that*

$$\forall s, s' \in S : \forall (a_1, a_2) \in \mathbf{A} : \mathcal{R}_1(f(s), (a_2, a_1), f(s')) = \mathcal{R}_2(s, (a_1, a_2), s') \quad (5.4)$$

$$\textit{and } \forall s, s' \in S : \forall (a_1, a_2) \in \mathbf{A} : \mathcal{T}(f(s') \mid f(s), (a_2, a_1)) = \mathcal{T}(s' \mid s, (a_1, a_2)) \quad (5.5)$$

Intuitively, given a state s, $f(s)$ swaps the agents: in $f(s)$ agent 1 is wherever agent 2 was in s and vice versa.

In the following, we abbreviate the notation for policy and value functions of agents for clarity and denote the policy of agent 1 and 2 with π_1 and π_2, respectively, instead of writing out the parameterisation with $\pi(\cdot; \phi_1)$ and $\pi(\cdot; \phi_2)$. Similarly, we will denote the value functions of agent 1 and 2 with V_1 and V_2.

To account for the differences in data distributions of experiences when following different policies, we compute importance sampling (IS) weights that can be defined for any function g over actions as follows:

$$\mathbb{E}_{a \sim \pi_1(a|s)}[g(a)] = \mathbb{E}_{a \sim \pi_2(a|s')}\left[\frac{\pi_1(a \mid s)}{\pi_2(a \mid s')} g(a)\right] \quad (5.6)$$

During data collection, SEAC agents follow their respective policies to collect data (Algorithm 3), i.e. agent 1 follows π_1 and agent 2 follows π_2. We will now derive the off-policy part of the SEAC policy and critic losses for training the policy and value function of agent 1, using experience collected from agent 2. Note, we only derive the losses for agent 1 in the case of a stochastic game with two agents. The derivation for the loss of agent 2 using experience of agent 1 can be done analogously by substituting agent indices. Furthermore, we only derive the off-policy terms of both losses. For the on-policy terms of both losses, we refer to the previously introduced A2C algorithm (Mnih et al., 2016) which SEAC

[2]We note that the derivations provided in this section differ from the original derivations provided within Christianos et al. (2020). We modified the starting condition of the proofs and now start from the situation where all actions are sampled on-policy instead of from the off-policy policy gradient algorithm (Degris et al., 2012) as done in the original work. This change allowed us to remove the reward independence assumption which is quite restrictive and unnecessary for the following proofs. Overall these changes result in a more general derivation of the SEAC algorithm.

builds on in the independent learning setting (Section 2.6.1). Lastly, we derive the policy loss for expected return estimates rather than advantage estimates as used in SEAC. The SEAC loss is obtained from the derived policy loss by subtracting a baseline value given by the value estimate of the current state. This baseline is shown to not affect the gradients in expectation and can therefore be omitted in this derivation (Sutton and Barto, 2018).

The off-policy term of the SEAC policy loss for agent 1 using the experience of agent 2 can be written as follows:

Proposition 1 (Off-policy Policy Loss)**.**

$$\mathcal{L}(\phi_1) = \mathbb{E}_{\substack{a_1^t \sim \pi_1(s^t) \\ a_2^t \sim \pi_2(s^t)}} \left[\frac{\pi_1(a_2^t \mid f(s^t))}{\pi_2(a_2^t \mid s^t)} \log \pi_1(a_2^t \mid f(s^t)) \left(r_2^t + \gamma V_1(f(s^{t+1})) \right) \right] \tag{5.7}$$

For the following proof, we will start with the simplified case where both agents 1 and 2 follow the policy π_1, i.e. actions a_1^t and a_2^t are both sampled from π_1, a_1^t by applying π_1 in the original state s^t and a_2^t by applying π_1 in the state $f(s^t)$ where agent 1 is put in the place of agent 2. In this case, it is clear that all the experience is on-policy for policy π_1 and, thus, can be used in the usual policy gradient loss to update the parameters of the policy ϕ_1. From this point, we will derive the off-policy SEAC loss as defined in Equation 5.7 by applying our symmetry assumption to "swap" the reward of agent 1 with the reward of agent 2, and then "swapping" agent policies such that a_2^t is sampled from policy π_2 with IS weights to account for the distribution shift:

Proof.

$$\mathcal{L}(\phi_1) = \mathbb{E}_{\substack{a_1^t \sim \pi_1(s^t) \\ a_2^t \sim \pi_1(f(s^t))}} \left[\log \pi_1(a_2^t \mid f(s^t)) \left(\mathcal{R}_1(f(s^t), (a_2^t, a_1^t), f(s^{t+1})) + \gamma V_1(f(s^{t+1})) \right) \right] \tag{5.8}$$

$$\stackrel{A}{=} \mathbb{E}_{\substack{a_1^t \sim \pi_1(s^t) \\ a_2^t \sim \pi_1(f(s^t))}} \left[\log \pi_1(a_2^t \mid f(s^t)) \left(\mathcal{R}_2(s^t, (a_1^t, a_2^t), s^{t+1}) + \gamma V_1(f(s^{t+1})) \right) \right] \tag{5.9}$$

$$= \mathbb{E}_{\substack{a_1^t \sim \pi_1(s^t) \\ a_2^t \sim \pi_1(f(s^t))}} \left[\log \pi_1(a_2^t \mid f(s^t)) \left(r_2^t + \gamma V_1(f(s^{t+1})) \right) \right] \tag{5.10}$$

$$\stackrel{IS}{=} \mathbb{E}_{\substack{a_1^t \sim \pi_1(s^t) \\ a_2^t \sim \pi_2(s^t)}} \left[\frac{\pi_1(a_2^t \mid f(s^t))}{\pi_2(a_2^t \mid s^t))} \log \pi_1(a_2^t \mid f(s^t)) \left(r_2^t + \gamma V_1(f(s^{t+1})) \right) \right] \tag{5.11}$$

□

For notational clarity, we omit the next states sampled from $\mathcal{T}$ under the expectation:

$$s^{t+1} \sim \mathcal{T}(s^t, (a_1^t, a_2^t)) \text{ and } f(s^{t+1}) \sim \mathcal{T}(f(s^t), (a_2^t, a_1^t)) \tag{5.12}$$

In a similar manner, we can derive the off-policy term of the SEAC value loss which can be written as follows:

Proposition 2 (Off-policy Value Loss)**.**

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\substack{a_1^t \sim \pi_1(s^t) \\ a_2^t \sim \pi_2(s^t)}} \left[\frac{\pi_1(a_2^t \mid f(s^t))}{\pi_2(a_2^t \mid s^t))} \left(r_2^t + \gamma V_1(f(s^{t+1})) - V_1(f(s^t)) \right)^2 \right] \tag{5.13}$$

Proof.

$$\mathcal{L}(\theta_1) = \mathbb{E}_{\substack{a_1^t \sim \pi_1(s^t) \\ a_2^t \sim \pi_1(f(s^t))}} \left[\left(\mathcal{R}_1(f(s^t), (a_2^t, a_1^t), f(s^{t+1})) + \gamma V_1(f(s^{t+1})) - V_1(f(s^t)) \right)^2 \right] \tag{5.14}$$

$$\stackrel{A}{=} \mathbb{E}_{\substack{a_1^t \sim \pi_1(s^t) \\ a_2^t \sim \pi_1(f(s^t))}} \left[\left(\mathcal{R}_2(s^t, (a_1^t, a_2^t), s^{t+1}) + \gamma V_1(f(s^{t+1})) - V_1(f(s^t)) \right)^2 \right] \tag{5.15}$$

$$= \mathbb{E}_{\substack{a_1^t \sim \pi_1(s^t) \\ a_2^t \sim \pi_1(f(s^t))}} \left[\left(r_2^t + \gamma V_1(f(s^{t+1})) - V_1(f(s^t)) \right)^2 \right] \tag{5.16}$$

$$\stackrel{IS}{=} \mathbb{E}_{\substack{a_1^t \sim \pi_1(s^t) \\ a_2^t \sim \pi_2(s^t)}} \left[\frac{\pi_1(a_2^t \mid f(s^t))}{\pi_2(a_2^t \mid s^t))} \left(r_2^t + \gamma V_1(f(s^{t+1})) - V_1(f(s^t)) \right)^2 \right] \tag{5.17}$$

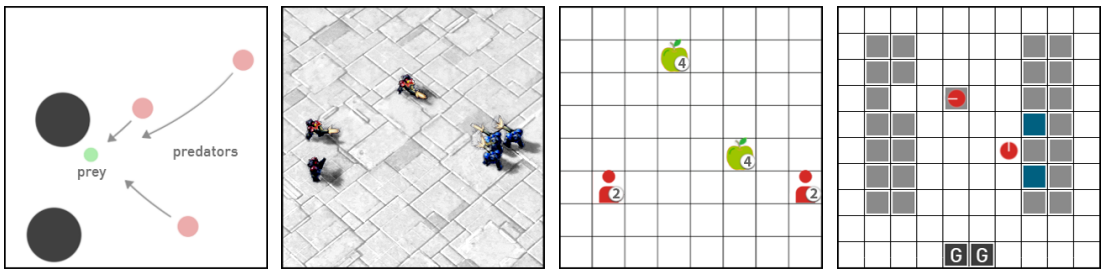
□

This concludes the proofs for the off-policy terms of the SEAC policy and value losses for agent 1 in a two-agent fully observable stochastic game.

5.3 Experiments

We conduct experiments in ten multi-agent tasks across four environments, visualised in Figure 5.2, with a focus on sparse-reward problems. We compare SEAC to two baselines as well as three MARL algorithms which have been shown to perform well in multi-agent tasks: MADDPG (Lowe et al., 2017), QMIX (Rashid et al., 2020b) and ROMA (Wang et al., 2020a).

Environments We evaluate in four tasks of both the multi-robot warehouse and level-based foraging environments introduced in Section 4.1. However, we note that in this chapter we train agents in these environments using individual



(a) Predator Prey (b) SMAC (c) LBF (d) RWARE

Figure 5.2: Environments used in our evaluation: (a) predator prey, (b) SMAC 3m with sparse rewards, (c) four LBF tasks, and (d) four RWARE tasks. Controlled agents are coloured red.

rewards for all agents instead of using common rewards. Additionally, we evaluate in a sparse reward variant of a SMAC task (Section 4.1) where agents only receive terminal rewards of -1, 0, or $+1$ for a defeat, draw, or victory, respectively. Lastly, we include a sparse reward variant of the MPE predator prey task (Section 4.1), where predators receive a reward of 1 for catching the prey and 0 otherwise.

We compare SEAC to two main baselines across all experiments.

Independent Advantage Actor-Critic (IA2C) The first baseline is applying the A2C algorithm in an independent way for MARL (Section 2.6.1). This baseline differs from SEAC in that it each agent updates its policy and value function using only its own experience, i.e. SEAC with $\lambda = 0$ would be identical to IA2C. Comparison to this baseline allows us to investigate the impact of using the experience of other agents in SEAC.

Shared Network Actor-Critic (SNAC) For the second baseline, we train a single shared policy and value function for all agents using the experience of all agents. During evaluation, each agent gets a copy of the policy and independently follows it. During training, each agent independently computes the policy and value loss of A2C (Mnih et al., 2016) using its experience. Then, the sum of policy and value loss gradients across all agents are used to optimise the parameters of the shared policy and value function. For this baseline, no importance sampling is needed since all trajectories are on-policy. This baseline is similar to SEAC in that the shared parameters are trained using the collective experience of all agents which can significantly improve sample efficiency as already shown in Sec-

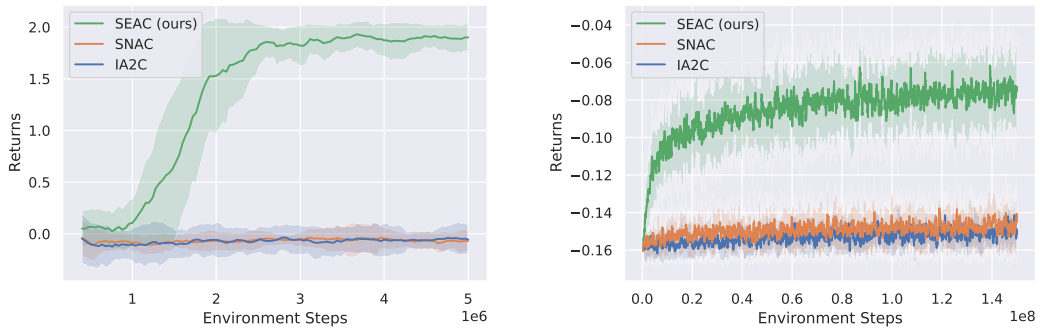
tion 4.3.3. In SEAC, agents similarly train their policy and value functions using the experience of all agents. Comparing SEAC against this baseline allows us to identify whether agents using experience sharing are learning identical policies, in which case SEAC and SNAC should perform identical, or whether SEAC agents are able to learn different policies despite being trained on the same collective experience.

In a subset of the evaluation tasks, we further compare to three more sophisticated MARL algorithms: MADDPG (Lowe et al., 2017), QMIX (Rashid et al., 2020b) and ROMA (Wang et al., 2020a). We use the implementations provided by the authors and tuned the learning rate, exploration rate, and batch sizes for each evaluation environment for these algorithms. We note that during evaluation, QMIX and ROMA use greedy policies ($\epsilon = 0$), while MADDPG, SEAC, IA2C, and SNAC apply stochastic policies.

Algorithm Details For IA2C, SNAC, and SEAC, we build on the A2C algorithm using 5-step returns and sampling experiences across four synchronous environments in parallel (Mnih et al., 2016). We also add a entropy regularisation term to the policy loss as outlined in Section 2.3.3. For SEAC, we compute the entropy regularisation term for the policy loss of agent i using only the policy and on-policy experiences of agent i. Due to computational cost of experiments, we conducted a small hyperparameter search for IA2C in RWARE; all further experiments, including for SNAC and SEAC, use the same hyperparameters listed in Table 5.1. All results presented are averaged across five seeds, with the standard deviation plotted as a shaded area. For calculation of evaluation returns reported in Table 5.2, the final policies per seed were evaluated for 100 episodes.

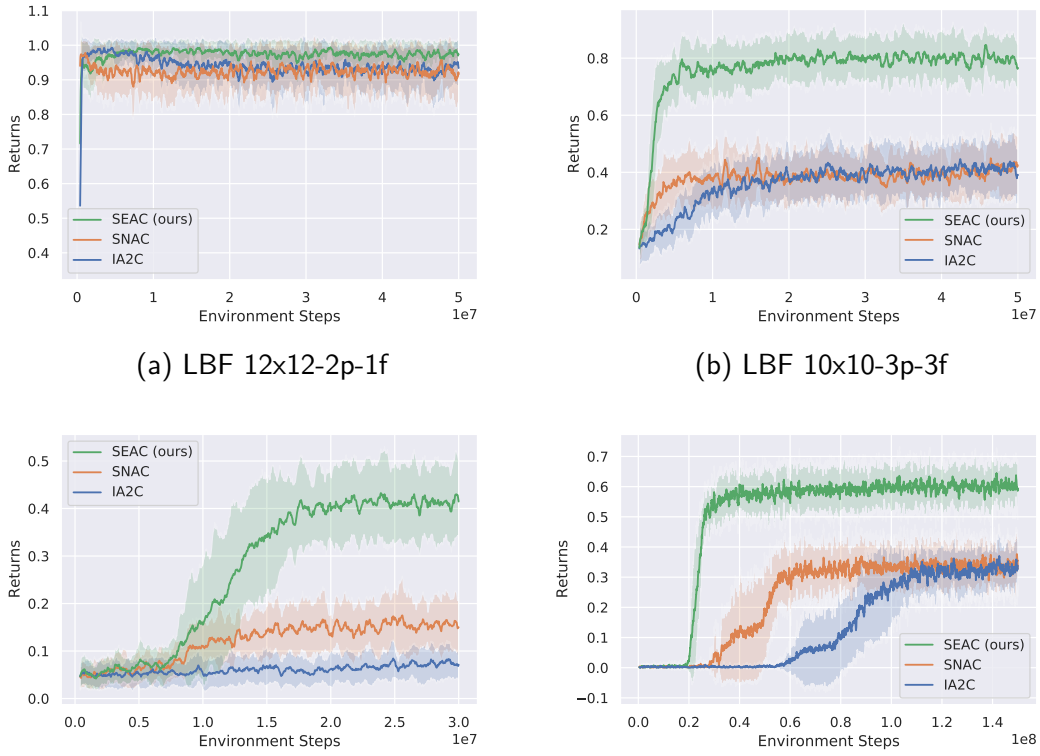
Table 5.1: Hyperparameters used for SEAC, IA2C and SNAC.

Hyperparameter	Value
learning rate	$3e^{-4}$
network size	64×64
γ	0.99
entropy coef	0.01
grad clip	0.5
parallel processes	4
n-steps	5
SEAC λ	1.0



(a) PP, sparse rewards (b) SMAC with three marines, sparse rewards

Figure 5.3: Training returns in sparse-reward variations of PP and SMAC-3m.



(a) LBF 12x12-2p-1f (b) LBF 10x10-3p-3f

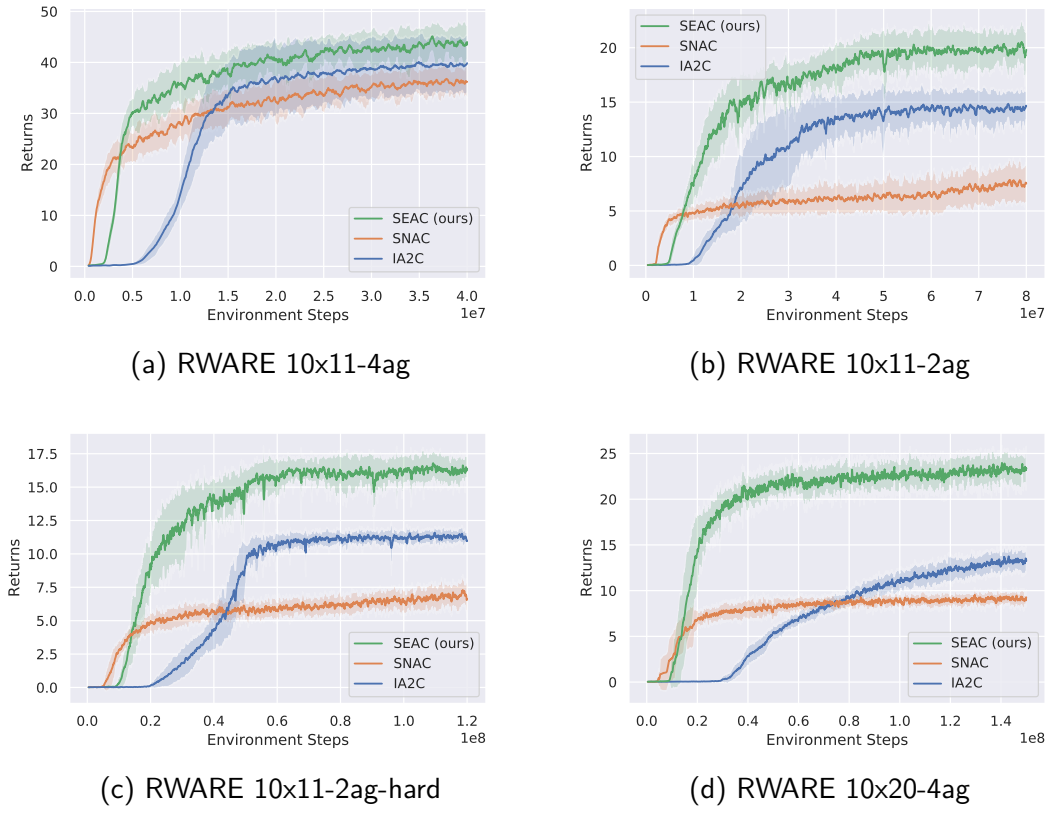
(c) LBF: 15x15-3p-4f (d) LBF: 8x8-2p-2f-coop

Figure 5.4: Training returns in four LBF tasks.

5.3.1 Results

Figures 5.3 to 5.5 show the training curves of SEAC, SNAC and IA2C for all tested environments. For RWARE and LBF, tasks are sorted from easiest to hardest.

In the sparse PP task (Figure 5.3a) only SEAC learns to solve the task with



(a) RWARE 10x11-4ag

(b) RWARE 10x11-2ag

(c) RWARE 10x11-2ag-hard

(d) RWARE 10x20-4ag

Figure 5.5: Training returns in four RWARE tasks.

consistent learning across seeds while IA2C and SNAC are unable to learn to catch the prey.

In SMAC with sparse rewards (Figure 5.3b), SEAC outperforms both baselines. However, with mean returns close to zero, the agents have not learned to win the battles but rather to run away from the enemy. This is not surprising since our experiments (Table 5.2) show that even state-of-the-art methods designed for these environments (e.g. QMIX) do not successfully solve this sparsely rewarded task.

For LBF (Figure 5.4), no significant difference can be observed between SEAC and the two baseline methods IA2C and SNAC for the easiest variant (Figure 5.4a). However, as the tasks become more challenging and rewards become sparser the improvement becomes apparent. For increased number of agents, foods and gridsize (Figures 5.4b to 5.4d), IA2C and SNAC converge to significantly lower average returns than SEAC. In the largest grid (Figure 5.4c), IA2C does not show any signs of learning due to the sparsity of the rewards whereas SEAC learns to collect some of the food. We also observe that SEAC tends to

Table 5.2: Final mean evaluation returns across five random seeds with standard deviation on a selection of tasks. Highest means per task (within one standard deviation) are shown in bold.

	IA2C	SNAC	SEAC (ours)	QMIX	MADDPG	ROMA
PP (sparse)	−0.04 ± 0.13	−0.04 ± 0.01	**1.93 ± 0.13**	0.05 ± 0.07	**2.04 ± 0.08**	0.04 ± 0.07
SMAC-3m (sparse)	−0.13 ± 0.01	−0.14 ± 0.02	**−0.03 ± 0.03**	**0.00**	**−0.01 ± 0.01**	**0.00**
LBF-15x15-3p-4f	0.13 ± 0.04	0.18 ± 0.08	**0.43 ± 0.09**	0.03 ± 0.01	0.01 ± 0.02	0.03 ± 0.02
LBF-8x8-2p-2f-coop	0.37 ± 0.10	0.38 ± 0.10	**0.64 ± 0.08**	**0.79 ± 0.31**	0.01 ± 0.02	0.01 ± 0.02
RWARE-10x20-4ag	13.75 ± 1.26	9.53 ± 0.83	**23.96 ± 1.92**	0.00	0.00	0.00
RWARE-10x11-4ag	**40.10 ± 5.60**	36.79 ± 2.36	**45.11 ± 2.90**	0.00	0.00	0.01 ± 0.01

converge to its final policy in fewer time steps than IA2C.

In RWARE (Figure 5.5), results are similar to LBF. Again, the two baseline methods IA2C and SNAC converge to lower average returns than SEAC for harder tasks. In the hardest task (Figure 5.5d), SEAC converges to final mean returns ∼70% and ∼160% higher than IA2C and SNAC, respectively, and again converges in fewer steps than IA2C.

In Table 5.2 we also present the final evaluation returns of three state-of-the-art MARL algorithms, QMIX, MADDPG, and ROMA, in a selection of tasks. These algorithms show no signs of learning in most of these tasks. The only exceptions are MADDPG matching the returns of SEAC in the sparse PP task and QMIX performing comparably to SEAC in the cooperative LBF task. QMIX and ROMA assume tasks to be fully-cooperative, i.e. all agents receive the same reward signal. Hence, in order to apply the two algorithms, we modified non-cooperative environments to return the sum of all individual agent returns as the shared reward. While common rewards could make learning harder, IA2C can also learn successfully in RWARE tasks under the common reward setting as seen in Chapter 4.

5.3.2 Analysis

Similar patterns can be seen for the different algorithms across all tested environments. It is not surprising that IA2C requires considerably more environment samples to converge, given that the algorithm is less efficient in using them; IA2C agents only train on their own experience. This is further evident when noticing that in RWARE (Figures 5.5a to 5.5d) the learning curve of SEAC starts moving upwards in roughly $1/N$ the time steps compared to IA2C, where N refers to the

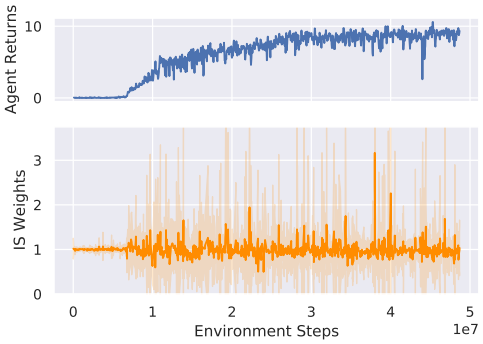


Figure 5.6: Importance weights of one SEAC agent in RWARE 10x11-2ag-hard.

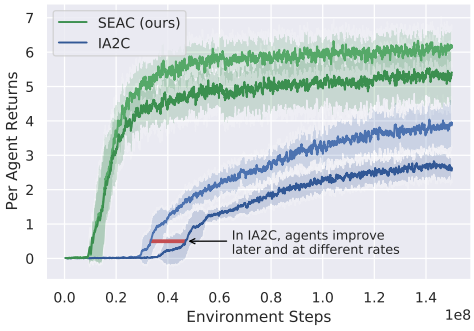
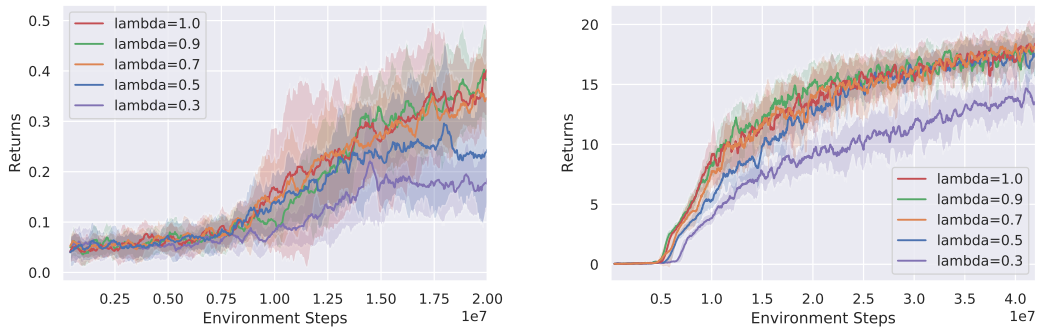


Figure 5.7: Best vs. worst performing agents on RWARE 10x20-4ag.

number of agents. Also, it is not surprising that SNAC does not achieve as high returns after convergence: sharing a single policy across all agents impedes their ability to coordinate or develop distinct behaviours that lead to higher returns.

We conjecture that SEAC converges to higher final returns due to agents learning simultaneously when sharing experiences, combined with the flexibility to develop differences in policies to improve coordination. We observe that SEAC is able to learn similarly quickly to SNAC because the combined local gradients provide a very strong learning direction. However, while SNAC levels off at some point due to the use of identical policies, which limit the agents' ability to coordinate, SEAC can continue to explore and improve because agents are able to develop different policies to further improve coordination. Figure 5.6 shows that encountered importance weights during SEAC optimisation are centred around 1, with most weights staying in the range $[0.5; 1.5]$. This indicates that the agents indeed learn similar but not identical policies. The divergence of the policies is attributed to the random initialisation of networks, along with the agent-centred entropy factor. The range of the importance weights also shows that, in our case, importance sampling does not introduce significant instability in the training. The latter is essential for learning since importance weighting for off-policy RL is known to suffer from significant instability and high variance through diverging policies (e.g., Sutton and Barto, 2018; Precup, 2000).

In contrast, we observe that IA2C starts to improve at a much later stage than SEAC because agents need to explore for longer, and when they start improving it is often the case that one agent improves first while the other agents catch up



(a) LBF 15x15-3p-4f (b) RWARE 10x20-2ag

Figure 5.8: Training returns of SEAC with different values of λ in a LBF and RWARE task.

later, which can severely impede learning. Figure 5.7 shows that agents using IA2C end up learning at different rates, and the slowest one ends up with the lowest final returns. In learning tasks that require coordination, an agent being ahead of others in its training can impede overall training performance.

We find examples of agents learning at different rates being harmful to overall training in all our tested environments. In RWARE, an agent that learns to fulfil requests can make the learning more difficult for others by delivering all requests on its own. Agents with slightly less successful exploration have a harder time learning a rewarding policy when the task they need to perform is constantly done by others. In LBF, agents can choose to cooperate to gather highly rewarding food or focus on food that can be foraged independently. The latter is happening increasingly often when an agent is ahead in the learning curve as others are still aimlessly wandering in the environment. In the PP task, the predators must approach the prey simultaneously, but this cannot be the case when one predator does not know how to do so. In the SMAC-3m task, a single agent cannot be successful if its team members do not contribute to the fight. The agent would incorrectly learn that fighting is not viable and therefore prefer to run from the enemy, which however is not an optimal strategy.

We also investigate the sensitivity of the SEAC algorithm to the choice of λ by conducting a sensitivity analysis in one LBF and one RWARE task in which the gap in sample efficiency between SEAC and IA2C was particularly large. Figure 5.8 shows that SEAC achieves similar evaluation returns in both tasks for a wide range of values for λ. These results indicate that SEAC is not highly sensitive

Table 5.3: Average training time of IA2C and SEAC across all evaluation tasks, measured by mean process time (mins:secs) required for 100,000 time steps of training.

	IA2C	SEAC	Increase
Foraging-10x10-3p-3f	2:00	2:04	3.86%
Foraging-12x12-2p-1f	1:22	1:24	2.94%
Foraging-15x15-3p-4f	2:01	2:06	3.90%
Foraging-8x8-2p-2f-coop	1:21	1:24	3.78%
rware-tiny-2ag	1:41	1:43	1.65%
rware-tiny-2ag-hard	2:05	2:09	2.97%
rware-tiny-4ag	2:49	2:53	2.25%
rware-small-4ag	2:50	2:55	2.44%
Predator Prey	2:44	2:49	3.39%
SMAC (3m)	6:23	6:25	0.38%

to values of the hyperparameter λ. We further observe that the performance of SEAC decreases as λ moves from 1 to 0, eventually converging to IA2C for $\lambda = 0$.

Lastly, we consider the computational cost introduced by SEAC. Table 5.3 contains process time required for running IA2C and SEAC. Timings were measured on a 6^{th} Gen Intel i7 @ 4.6 Ghz running Python 3.7 and PyTorch 1.4. The average time for running and training on 100,000 environment iterations is displayed. Only process time (the time the program was active in the CPU) was measured, rounded to seconds. As we can see, experience sharing with SEAC increased running time by less than 3% across all environments compared to IA2C. We find that often the environment is the bottleneck for training speeds rather than the algorithm optimisation and as such, more complex and slower simulators, such as SMAC, show a lower percentage difference between algorithms.

5.4 Shared Experience Deep Q-Networks

To investigate the generality of experience sharing and its efficacy in MARL, we apply the idea of experience sharing to off-policy MARL algorithms at the example of independent Q-learning (Tan, 1993) based on DQN (Mnih et al., 2015)

Algorithm 4 Shared Experience Deep Q-Networks

Initialise: parameters θ_i and $\bar{\theta}_i = \theta_i$ of the value function and target network of agent i for each agent $i \in I$

Initialise: empty shared replay buffer $\mathcal{D} \leftarrow \emptyset$

for each episode **do**

Obtain initial state $s^0 \sim \mu$ and joint observation $\mathbf{o}^0$

for each step $t = 0, \cdots, T$ **do**

for each agent $i \in I$ **do**

Select action $a_i^t \sim \epsilon$-greedy$(Q(h_i^t, \cdot; \theta_i))$

end for

Apply joint action $\mathbf{a}^t = (a_1^t, \ldots, a_N^t)$

Receive next state $s^{t+1} \sim \mathcal{T}(s^t, \mathbf{a}^t)$, rewards $r_i^t = \mathcal{R}_i(s^t, \mathbf{a}^t, s^{t+1})$ for each agent i, and joint observation $\mathbf{o}^{t+1} \sim \mathcal{O}(s^t, \mathbf{a}^t)$

for each agent $i \in I$ **do**

Store experience tuple $(h_i^t, a_i^t, r_i^t, h_i^{t+1})$ in $\mathcal{D}$

end for

Sample batch of experiences from $\mathcal{D}$

for each agent $i \in I$ **do**

Update θ_i by minimising the average loss $\mathcal{L}(\theta_i)$ (Equation 5.18) computed over the sampled batch

end for

end for

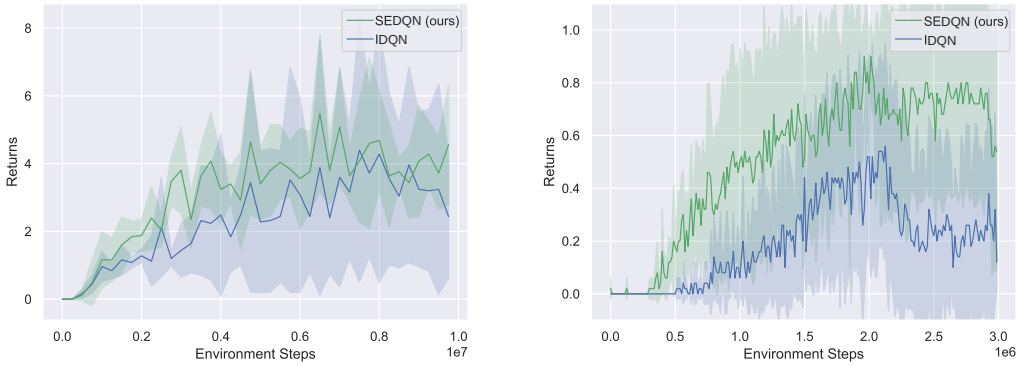
end for

(introduced in Sections 2.3.2 and 2.6.1). The loss for IDQN can be written as:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(h_i^t, a_i^t, r_i^t, h_i^{t+1}) \sim \mathcal{D}_i} \left[\left(r_i^t + \gamma \max_{a \in A_i} Q(h_i^{t+1}, a_i; \bar{\theta}_i) - Q(h_i^t, a_i^t; \theta_i) \right)^2 \right] \quad (5.18)$$

with θ and $\bar{\theta}$ denoting the parameters of the Q-network and the target network, respectively. For each update, a batch of experience tuples $(h_i^t, a_i^t, r_i^t, h_i^{t+1})$, consisting of observation history, applied action, received reward, and next observation history of agent i, are sampled from a replay buffer $\mathcal{D}_i$. In IDQN, each agent i maintains its own replay buffer $\mathcal{D}_i$ containing its individual experience tuples.

To extend IDQN with experience sharing, agents can compute the same loss as given in Equation 5.18 but sample experience tuples from the replay buffer of any agent, i.e. agents sample experiences from both their own replay buffer as



(a) RWARE 10x20-2ag (b) LBF 8x8-2p-2f-coop

Figure 5.9: Average total returns of SEDQN and IDQN in one RWARE and LBF task.

well as the replay buffer of all other agents. We refer to this novel algorithm as *shared experience deep Q-networks* (SEDQN). In contrast to SEAC, no off-policy correction using importance sampling weights is required in SEDQN since DQN is naturally a off-policy algorithm and, thus, does not assume that the loss is computed on experiences generated by following the updated policy. SEDQN optimises the same loss as IDQN (Equation 5.18) but samples experiences from a shared replay buffer $\mathcal{D} = \bigcup_{i \in I} \mathcal{D}_i$ containing the individual experiences of all agents. In practise, we maintain a single replay buffer for all agents that contains the experiences of all agents. In contrast to SEAC and IA2C, SEDQN and IDQN are naturally optimised using exactly the same number of samples as determined by the batch size. We provide pseudocode for SEDQN in Algorithm 4.

We evaluate SEDQN and IDQN in one RWARE and one LBF task with sparse rewards. Figure 5.9 shows the evaluation returns of SEDQN and IDQN in both tasks over five seeds. Overall, we find that the experience sharing of SEDQN improves performance compared to IDQN, but we do not observe similar improvements as observed for SEAC compared to IA2C. In the RWARE task, sharing experience appears to reduce variance considerably despite not impacting average returns significantly. On the other hand, on the LBF task average returns increased significantly by sharing experience and at its best evaluation even exceeded average returns achieved by SEAC. However, variance of off-policy SEDQN and IDQN is found to be significantly larger compared to on-policy SEAC, IA2C and SNAC.

We hypothesise that the distribution shift of the off-policy experiences of

different agents introduces instabilities in training (van Hasselt et al., 2018; Fujimoto et al., 2018) and that off-policy optimisation might suffer more from the non-stationarity of the policies of other agents (Section 4.4). Furthermore, while SEAC benefited from leveraging more experience samples for each update compared to IA2C, no such benefit is leveraged in SEDQN. We leave further investigation into the applicability of experience sharing in off-policy MARL algorithms for future work, but consider these experiments as a preliminary indication that experience sharing can provide value for both on- and off-policy algorithms.

Gerstgrasser et al. (2024) built on top of SEDQN and introduced a novel method that modifies SEDQN by only sharing a subset of experiences between agents. They find that sharing all experiences across agents, as done in our experiments, can harm training and cause instability. Instead, they propose to identify particularly valuable experience samples based on the TD-error, following the intuition of prioritised experience replay (Schaul et al., 2016) which prioritises experiences from the replay buffer proportional to their TD-error. They show that sharing experiences in such a selective manner can significantly improve upon the performance of IDQN and SEDQN.

5.5 Related Work

Centralised Training with Decentralised Execution Many recent MARL algorithms leverage the paradigm of centralised training with decentralised execution (CTDE) (Section 2.6.2) where data of all agents is used during training but each agent learns decentralised (locally-executable) policies. Due to its use of data of all agents during training, SEAC can be considered a CTDE algorithm. However, SEAC is notably distinct from many CTDE algorithms like value decomposition algorithms (Section 2.6.2.1) and actor-critic algorithms with centralised critics like MADDPG and COMA (Section 2.6.2.2) in that these algorithms only reinforce an agent's own tried actions. In contrast, SEAC uses shared experience to reinforce good actions tried by any agent. Furthermore, SEAC does so without learning larger critic networks conditioned on centralised information such as the joint observation history, full environment state, or joint action. Our experiments show that CTDE algorithms like MADDPG, QMIX, and ROMA were unable to learn effective policies in sparse-reward environments while SEAC learned successfully in most cases.

Agents Teaching Agents Other related approaches propose to leverage expertise of existing *teacher* agents to improve the efficiency of training new *learner* agent (Da Silva et al., 2020). Such teaching can be regarded as a form of transfer learning (Pan and Yang, 2009) among RL agents. The *teacher* would either implicitly or explicitly be asked to evaluate the behaviour of the *learner* and send instructions to the other agent. Contrary to SEAC, most such approaches do focus on single-agent RL (Clouse, 1996; Fachantidis et al., 2019). However, even in such teaching approaches for multi-agent systems (Da Silva et al., 2017, 2018) experience is shared in the form of knowledge exchange following a teacher-learner protocol. Instead, SEAC shares agent trajectories for learning and does not rely on the exchange of explicit queries or instructions, introducing minimal additional cost.

Learning from Demonstrations Training agents from trajectories (Schaal, 1997) of other agents (Zimmer et al., 2014) or humans (Taylor et al., 2011) is a common case of teaching agents. Demonstration data can be used to derive a policy (Ho and Ermon, 2016) which might be further refined using typical RL training (Gao et al., 2018) or to shape the rewards biasing towards previously seen expert demonstrations (Brys et al., 2015). These approaches leverage expert trajectories to speed up or simplify learning for single-agent problems. In contrast, SEAC makes use of trajectories from other agents which are generated by concurrently learning agents in a multi-agent system. As such, we aim to speed up and synchronise training in MARL whereas learning from demonstrations focuses on using previously generated data for application in domains like robotics where generating experience samples is expensive.

Distributed Reinforcement Learning Sharing experience among agents is related to recent work in distributed RL that aim to effectively use large-scale computing resources for RL. Asynchronous methods such as A3C (Mnih et al., 2016) execute multiple actors in parallel to generate trajectories more efficiently and break data correlations. Similarly, IMPALA (Espeholt et al., 2018) and SEED RL (Espeholt et al., 2020) are off-policy actor-critic algorithms to distribute data collection across many actors with optimisation being executed on a single learner. Network parameters, observations or actions are exchanged after each episode or time step, respectively, and off-policy correction is applied. These approaches

are similar to SEAC in their use of multiple policies to gather data that can be leveraged to make training more efficient but distinct in their setting and goal. SEAC aims to leverage the experience of multiple policies in a multi-agent setting to improve sample efficiency and synchronisation between agents in MARL. In contrast, all these approaches only share experience of multiple actors to speed up the learning of a single RL agent and by breaking correlations in the data.

Population-play Population-based training is another line of research aiming to improve exploration and coordination in MARL by training a population of diverse sets of agents (Lanctot et al., 2017; Wang et al., 2019a; Leibo et al., 2019; Jaderberg et al., 2019; Long et al., 2020). Leibo et al. (2019) note the overall benefits on exploration when sets of agents are dynamically evolved and mixed. In their work, some agents share policy networks and are trained alongside evolving sets of agents. Similarly to Leibo et al. (2019), we observe benefits on exploration due to agents influencing each other's trajectories. However, SEAC is different from such population-play as it only trains a single set of distinct policies for all agents, thereby avoiding the significant computational cost involved in training multiple sets of agents.

5.6 Conclusion

This paper introduced SEAC, a novel multi-agent actor-critic algorithm in which agents learn from the experience of others. In our experiments, SEAC outperformed independent learning, shared policy training, and state-of-the-art MARL algorithms in ten sparse-reward learning tasks across four environments, demonstrating improved sample efficiency and final returns. We discussed a theme commonly found in MARL environments: agents learning at different rates impedes exploration, leading to sub-optimal policies. SEAC overcomes this issue by combining the local gradients to concurrently learn similar policies for all agents without restricting the policies of all agents to be identical. Lastly, sharing experience is appealing especially due to its simplicity. SEAC is simple to implement, introduces almost no additional computational cost, and no additional hyperparameter tuning is required but can significantly improve learning efficiency. Therefore, its use should be considered in all environments that satisfy the underlying symmetry assumption.

Chapter 6

Ensemble Value Functions for Multi-Agent Exploration

Publication

This chapter is based on and adapted from the following publication:

Lukas Schäfer, Oliver Slumbers, Stephen McAleer, Yali Du, Stefano V. Albrecht, and David Mguni. "Ensemble value functions for efficient exploration in multi-agent reinforcement learning." In *Adaptive and Learning Agents Workshop at the AAMAS conference*. 2023.

In the previous chapter, we proposed a novel approach for exploration in MARL that shares experience across agents to improve the efficiency of MARL training, in particular for multi-agent policy gradient algorithms in sparse-reward tasks. In this chapter,[1] we focus on exploration for value-based MARL algorithms that often still rely on random exploration processes, such as ϵ-greedy (e.g., Sunehag et al., 2018; Rashid et al., 2020b). We argue that such random exploration can lead to inefficient exploration of the joint action space and, thereby, can result in poor performance in challenging cooperative exploration tasks, as previously seen in Chapter 4.

To understand how we can design better exploration strategies to overcome this inefficiency, we first outline two important properties that an effective exploration strategy in MARL should have. Similar to single-agent RL, an effective

[1]The presented research was partly conducted during an internship at Huawei Noah's Ark Lab and has since been extended.

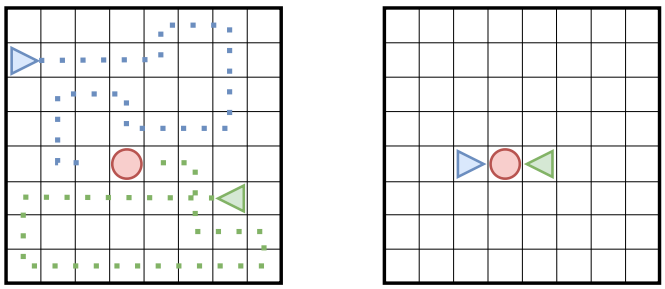


Figure 6.1: Motivational example: Two agents (triangles) can independently explore their movement (left), but rely on joint cooperation to pick up the heavy circular goal object (right). To overcome the inefficiency of random exploration in discovering such cooperation, we leverage uncertainty across ensembles of value functions to guide the multi-agent exploration towards state-action pairs that might require cooperation.

exploration strategy in MARL might incentivise agents to explore many states and actions, for example by encouraging exploration of states and actions that have infrequently been visited. This is important for agents to learn accurate value estimates and policies that can solve a task. Additionally and specific to MARL, an exploration strategy should also incentivise agents to explore their possible interactions with other agents. Due to the large joint action space of all agents, it is challenging to do so exhaustively, but we can design exploration strategies that incentivise agents to focus on such interactions in states where they matter. We believe such exploration strategies are particularly promising in tasks that require agents to cooperate with each other and cannot be solved by any agent individually.

Consider the example, visualised in Figure 6.1, in which two agents have to navigate a grid-world to jointly pick up a heavy object. To learn to pick up the goal object, agents need to cooperate by selecting the pick-up action in a state where both agents are next to the object. However, such cooperation is highly unlikely when following a random exploration policy. In contrast, if agents are not yet next to the object, there is no potential for cooperation across agents and, thus, there is no need to exhaustively explore the joint action space. Instead, agents can independently explore their movement in such states, so random exploration might be sufficient in these cases. This example illustrates that random exploration can be inefficient in exploring cooperation between agents, and shows the need for exploration strategies that identify and focus on exploring actions of all agents in states that require cooperation or, more generally, agent interactions.

Following this intuition, we propose *ensemble value functions for multi-agent exploration* (EMAX), a general framework to seamlessly extend any value-based MARL algorithms by training ensembles of value functions for each agent. To incentivise agents to focus their exploration on state-action pairs that may require cooperation across multiple agents, EMAX follows an upper-confidence bound (UCB) policy (Auer, 2002) over the average and disagreement of value estimates across the ensemble. Originally proposed for multi-armed bandit problems, UCB follows the principle of optimism in the face of uncertainty and selects greedy actions with respect to value estimates in addition to an uncertainty term that decays as actions (and states) are selected. In the beginning of training, the uncertainty term is large and, thus, a UCB policy would try many different actions but as actions have been tried many times the uncertainty term decays and the UCB policy would converge towards a greedy policy with respect to its value estimates. In the case of EMAX, the exploration strategy prioritises the exploration of actions that appear promising (as measured by high average value estimates) and that are infrequently visited or have the potential for cooperation between agents (as measured by high disagreement in value estimates). To better understand the effects of such an exploration strategy, we discuss multiple effects that can cause disagreement in value estimates across the ensemble:

1. Value estimates might have not converged yet, indicating that the states and actions have not yet been sufficiently explored and, thus, more exploration in this part of the environment is needed. This epistemic uncertainty is commonly followed in prior work on single-agent exploration (e.g., Auer, 2002; Osband et al., 2016a; Lee et al., 2021; Liang et al., 2022).

2. The transitions of the environment are stochastic, leading to variability in the returns. Following such aleatoric uncertainty for exploration can be valuable when learning accurate value estimates in these states is particularly challenging, as long as the exploration strategy does not overly focus on such stochasticity (Burda et al., 2019a).

3. In MARL, the variability in returns can also be due to the stochasticity of the policies of other agents and indicate the potential for agent interactions such as cooperation. To illustrate why this might be the case, we return to our motivational example, in which both agents only receive rewards for successfully picking up the object. In a state where both agents are next

to the goal object (Figure 6.1, right), agents may receive varying rewards when choosing the pick-up action since their reward depends on whether the other agent also selects the pick-up action. Only if both agents select the pick-up action do they succeed and receive a positive reward, whereas if any agent does not select the pick-up action they do not receive such a reward. As we can see, the reward an agent receives when choosing the pick-up action in such a state depends on the action of the other agent. This dependency can lead to variability in rewards (and consequently returns) whenever the other agent follows a stochastic policy that sometimes leads to successful and sometimes to unsuccessful cooperation. In contrast, in a state where agents are not next to the goal object (Figure 6.1, left), agents will always receive the same reward of zero no matter their actions and actions of the other agent. This lack of variability in rewards in such states indicates that there is no dependency on the actions of the other agent and, thus, no potential for cooperation with the other agent.

We empirically validate that the variability in returns across the ensemble can indeed indicate the potential for cooperation between agents, and that EMAX with its exploration policy guides agents to explore states and actions that require cooperation more often that random exploration. By following the EMAX exploration policy, agents are more likely to choose actions with potential for cooperation in the current state, and more effectively navigate towards states with such potential for cooperation (Section 6.4.2). We also provide an ablation of the exploration policy of EMAX in which we replace it with a random exploration policy. We show that agents trained with this ablation achieve lower returns and learn less efficiently, indicating that the exploration policy is crucial for the success of EMAX (Section 6.4.5).

In addition to its exploration strategy, EMAX leverages the ensemble to compute target values as the average value estimate across the ensemble instead of using target networks. These target values eliminate the need for additional target networks, and have been shown to exhibit lower variance, thereby stabilising the optimisation of agents (Liang et al., 2022). Lastly, actions are selected using a majority vote across the greedy actions of all value functions in the ensemble during evaluation. This evaluation policy reduces the likelihood of suboptimal decision making and, thus, improves the robustness of evaluation performance (Osband et al., 2016a). We empirically validate that the EMAX target computation

reduces the variance of gradients throughout training (Section 6.4.1), and show that the EMAX evaluation policy improves the robustness of the evaluation performance (Section 6.4.3).

First, we extend independent DQN (Mnih et al., 2015; Tan, 1993) with EMAX and evaluate it in 11 multi-agent tasks with sparse rewards. These environments are mixed cooperative-competitive, i.e. agents receive individual rewards under the POSG formalism (Section 2.4) but must cooperate with other agents in some states. In this setting, EMAX improves the final evaluation returns of IDQN by 185% across all tasks (Section 6.3). Afterwards, we focus on the cooperative MARL setting and extend value decomposition methods of VDN (Sunehag et al., 2018) and QMIX (Rashid et al., 2020b) with EMAX, and conduct an extensive evaluation of EMAX in common-reward multi-agent tasks. Overall, we evaluate EMAX on top of three value-based MARL algorithms in 21 common-reward tasks across four diverse multi-agent environments, and find that EMAX improves sample efficiency and final achieved returns across all tasks over all three extended algorithms (IDQN, VDN, QMIX) by 60%, 47%, and 538%, respectively (Section 6.3). Lastly, we show that comparably small ensembles with five value functions are sufficient to benefit from the advantages of EMAX, discuss the computational cost of ensemble models (Section 6.4.4), and provide ablations to understand the impact of the key ideas of EMAX (Section 6.4.5).

6.1 Ensemble Value Functions for Multi-Agent Exploration

In this section, we present *ensemble value functions for multi-agent exploration* (EMAX), a general framework that trains an ensemble of value functions for each agent in value-based MARL. Formally, each agent i trains an ensemble of K value functions $\{Q_i^k\}_{k=1}^K$ with Q_i^k being parameterised by θ_i^k. Each value function is conditioned on agent i's local observation-action history. EMAX leverages these ensembles of value functions to guide the exploration of agents and stabilise their optimisation. First, we introduce all components of IDQN with EMAX before integrating value decomposition methods such as VDN and QMIX into EMAX. Figure 6.2 illustrates the exploration policy and target computation of IDQN-EMAX, and pseudocode for IDQN training with EMAX and EMAX evaluation

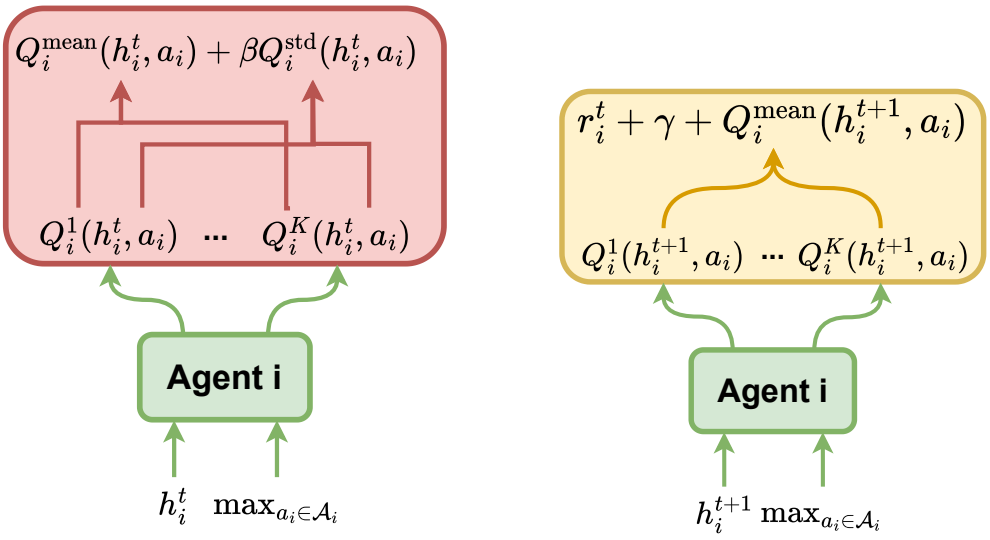


Figure 6.2: Illustration of IDQN-EMAX with (left) the UCB exploration strategy for agent i, and (right) the target computation. The value functions of individual agents are highlighted in green, the exploration policy computation in red, and target aggregation in orange.

is shown in Algorithm 5 and Algorithm 6, respectively. Illustration of EMAX value estimation and target computation with value decomposition is shown in Figure 6.3, and pseudocode for QMIX with EMAX is shown in Algorithm 7. The pseudocode for VDN with EMAX is analogous to the pseudocode for QMIX with EMAX without the mixing network and computing the loss as defined in Equation 6.7. As we will show in Section 6.4.4, we find that comparably small ensembles with five value functions ($K = 5$) are sufficient to benefit from the advantages of EMAX.

Exploration policy In multi-agent problems that require agents to cooperate to achieve high returns, agents should focus their exploration on states and actions that require cooperation. To guide agents to explore such states and actions, EMAX agents follow a UCB policy akin to prior work (Lee et al., 2021; Liang et al., 2022) using the average and standard deviation of value estimates across the ensemble:

$$\pi_i^{\text{expl}}(h_i^t; \theta_i) \in \arg\max_{a_i \in A_i} Q_i^{\text{mean}}(h_i^t, a_i; \theta_i) + \beta Q_i^{\text{std}}(h_i^t, a_i; \theta_i) \tag{6.1}$$

with the average and standard deviation across the ensemble of value functions of agent i defined as:

$$Q_i^{\text{mean}}(h_i^t, a_i^t; \theta_i) = \frac{1}{K} \sum_{k=1}^{K} Q_i^k(h_i^t, a_i^t; \theta_i^k) \tag{6.2}$$

$$Q_i^{\text{std}}(h_i^t, a_i^t; \theta_i) = \sqrt{\frac{\sum_{k=1}^{K} \left(Q_i^k(h_i^t, a_i^t; \theta_i^k) - Q_i^{\text{mean}}(h_i^t, a_i^t; \theta_i) \right)^2}{K}} \tag{6.3}$$

with $\theta_i = \{\theta_i^k\}_{k=1}^K$ denoting the parameters of the ensemble of value functions of agent i, and $\beta > 0$ denoting an uncertainty weighting hyperparameter chosen in consideration of the scale of rewards and the amount of exploration required for a task. This policy guides agents to explore actions that are promising (as measured by the mean value estimate) and are likely to require more exploration to reduce epistemic uncertainty or might need cooperation of multiple agents (as measured by the standard deviation of value estimates). Similar strategies have been applied in single-agent RL to guide exploration towards infrequently visited states (e.g., Auer, 2002; Lee et al., 2021; Liang et al., 2022). In this chapter, we argue that in multi-agent tasks the disagreement of value estimates across the ensemble can also indicate whether state-action pairs require agent interactions such as cooperation.

To see why, consider states in which multiple agents have to cooperate, i.e. multiple agents need to select specific actions, to receive a large reward. If any agent deviates from the required joint action, the agents receive no reward. In such states, received rewards for a given action of agent i will vary significantly whenever other agents follow stochastic policies, since the reward depends on the actions of other agents. In contrast, in states where agent i receives identical rewards independent of the action selection of the other agents, no such variability of rewards is experienced. Due to this variability of rewards (or lack thereof), value estimates across the ensemble will quickly converge in states that require no or limited cooperation, and will exhibit high disagreement in states that require cooperation. Therefore, the EMAX exploration policy focuses the exploration of agents on state-action pairs that might require cooperation in contrast to common random exploration for value-based MARL such as ϵ-greedy policies. We empirically demonstrate these benefits in Sections 6.4.2 and 6.4.5.

We consider this exploration policy an inductive bias towards exploring interactions between agents and believe that such policies are particularly beneficial in tasks that require cooperation across agents. In such tasks, any deviation of re-

turn estimates as a consequence of other agents' actions likely indicates desirable potential for cooperation. We note that the disagreement diminishes throughout training as the learned value functions of agents converge and agents learn to reliably cooperate. Once agents always succeed at cooperating in a state with the potential for cooperation, returns will no longer be variable, and the disagreement of value estimates will reduce. Furthermore, we note that the disagreement of value estimates incentivises not just the exploration of agent interactions in the current state but also guide agents towards states with potential for future interactions, since value functions estimate expected returns that are computed over entire episodes. However, due to discounting in returns with a discount factor of $\gamma < 1$, the exploration strategy will give more importance to exploring states and actions that lead to near-term potential for agent interactions.

Optimisation To extend IDQN with EMAX, we optimise the k-th value function of agent i by minimising the following loss:

$$\mathcal{L}(\theta_i^k) = \mathbb{E}_{(h_i^t, a_i^t, r_i^t, h_i^{t+1}) \sim \mathcal{D}} \left[\left(r_i^t + \gamma \max_{a_i \in A_i} Q_i^{\text{mean}}(h_i^{t+1}, a_i; \theta_i) - Q_i^k(h_i^t, a_i^t; \theta_i^k) \right)^2 \right] \quad (6.4)$$

Computing target values as the average across all value estimates of the ensemble (Liang et al., 2022) reduces the computational and memory cost of training ensemble networks by eliminating the need for target networks and, as we empirically show in Section 6.4.1, reduces the variability of gradients. Such reduced variability of gradients improves the stability of training. Furthermore, we argue that such improved stability of gradients during the optimisation is particularly valuable in MARL where the exploration and non-stationarity of the policies of other agents can otherwise result in unstable training, as observed in particular for off-policy value-based MARL algorithms that EMAX is designed to extend (Section 4.4).

Evaluation policy When evaluating agents, value-based MARL algorithms typically follow the greedy policy with respect to their value function. With EMAX, agent i selects its action during evaluation using a majority vote across the greedy actions of all models in its ensemble akin to prior work (Osband et al., 2016a):

$$\pi_i^{\text{eval}}(h_i^t; \theta_i) \in \arg\max_{a_i \in A_i} \sum_{k=1}^{K} [1]_{\mathcal{A}_{\text{opt},i}^k}(a_i) \quad (6.5)$$

$$\mathcal{A}_{\text{opt},i}^k = \{ a_i \in A_i \mid a_i \in \arg\max_{a_i'} Q_i^k(h_i^t, a_i'; \theta_i^k) \} \quad (6.6)$$

Algorithm 5 Training IDQN with EMAX

Initialise: $\{\theta_i^k\}_{k=1}^K$ of value functions $\{Q_i^k\}_{k=1}^K$ for each agent $i \in I$
Initialise: empty episodic replay buffer $\mathcal{D} \leftarrow \emptyset$
for each episode **do**
  Obtain initial state $s^0 \sim \mu$ and joint observation $\mathbf{o}^0$
  for each step $t = 0, \cdots, T$ **do**
    for each agent $i \in I$ **do**
      Select action $a_i^t \sim \pi_i^{\text{expl}}(h_i^t; \theta_i)$ (Equation 6.1)
    end for
    Apply joint action $\mathbf{a}^t = (a_1^t, \ldots, a_N^t)$
    Receive next state $s^{t+1} \sim \mathcal{T}(s^t, \mathbf{a}^t)$, rewards $r_i^t = \mathcal{R}_i(s^t, \mathbf{a}^t, s^{t+1})$ for each agent $i \in I$, and joint observation $\mathbf{o}^{t+1} \sim \mathcal{O}(s^t, \mathbf{a}^t)$
  end for
  Sample bootstrap masks $\{m^k\}_{k=1}^K$ from Bernoulli(p)
  Store episode $(s^{0:T}, \mathbf{h}^{0:T}, \mathbf{a}^{0:T}, \mathbf{r}^{0:T}, \{m_k\}_{k=1}^K)$ in $\mathcal{D}$
  for each agent $i \in I$ **do**
    for each model in the ensemble $k = 1, \ldots, K$ **do**
      Sample batch of episodes B from $\mathcal{D}$ with $m_k = 1$
      Update θ_i^k by minimising $\mathcal{L}(\theta_i^k)$ (Equation 6.4) averaged over each time step t in B
    end for
  end for
end for

with indicator function $[1]_{\mathcal{A}^k_{\text{opt},i}}(a) = 1$ for the greedy action(s) of the k-th value function of agent i, $a_i \in \mathcal{A}^k_{\text{opt},i}$, and 0 otherwise. Such a policy decreases the likelihood of taking poor actions because any individual value function preferring a poor action due to errors in value estimates does not impact the action selection as long as the majority of models agree on the optimal action. We empirically demonstrate the benefits of such an evaluation policy in Section 6.4.3.

Ensemble value functions Motivated by previous work in cooperative MARL (e.g., Papoudakis et al., 2021; Christianos et al., 2021; Albrecht et al., 2024) that shares networks across agents to improve sample efficiency and scalability, also demonstrated in Section 4.3.3, and the computational cost of training K value

Algorithm 6 Evaluating with EMAX

Require: Trained ensemble of value functions $\{Q_i^k\}_{k=1}^K$ for each agent $i \in I$

Obtain initial state $s^0 \sim \mu$ and joint observation $\mathbf{o}^0$

for each step $t = 0, \cdots, T$ **do**

for each agent $i \in I$ **do**

Select action $a_i^t \sim \pi_i^{\text{eval}}(h_i^t; \theta_i)$ (Equation 6.6)

end for

Apply joint action $\mathbf{a}^t = (a_1^t, \ldots, a_N^t)$

Receive next state $s^{t+1} \sim \mathcal{T}(s^t, \mathbf{a}^t)$, rewards $r_i^t = \mathcal{R}_i(s^t, \mathbf{a}^t, s^{t+1})$ for each agent $i \in I$, and joint observation $\mathbf{o}^{t+1} \sim \mathcal{O}(s^t, \mathbf{a}^t)$

end for

functions for each agent, we share a single ensemble of value functions across all agents. All aforementioned techniques rely on value functions within the ensemble to be sufficiently diverse, in particular early in training. To ensure such diversity, we implement the K value functions within the ensemble as entirely separate networks with no sharing of parameters across the value functions in the ensemble. To efficiently compute the value estimates of all value functions and all agents in a single forward pass, we vectorise the computation across agents and networks within the ensemble. Beyond separate networks within the ensemble, we employ three techniques from prior work (Osband et al., 2016a; Liang et al., 2022) to incentivise diversity of value functions within the ensemble: (1) Ensemble models are separately and randomly initialised. (2) We sample separate batches of experiences from the replay buffer to train each model in the ensemble.(3) Each model is trained on bootstrapped samples of the entire experience collected.

For the boostrapped sampling process, we follow the methodology of Osband et al. (2016a) to draw bootstrapped samples for each model as subsets of the entire training experiences to train on. More specifically, we draw a Bernoulli mask $\{m_k\}_{k=1}^K$ for each model in the ensemble whenever an episode is added to the episodic replay buffer. This mask determines whether the k-th model within the ensemble is trained on this episode ($m_k = 1$) or not ($m_k = 0$). Each mask is drawn from a Bernoulli distribution with probability p of being 1 and $1 - p$ of being 0, i.e. $m_k \sim \text{Bernoulli}(p)$. For $p = 1$, each episode would be used to train each model in the ensemble so all models in the ensemble would receive the same training data. In contrast for a small p, the training data is likely to be

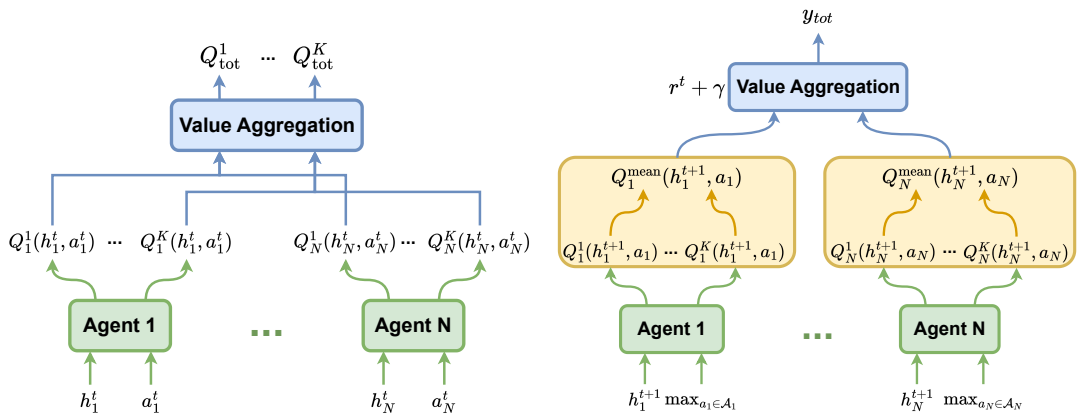


Figure 6.3: Illustration of EMAX with value decomposition algorithms. (left) The value estimation and (right) target computation with value decomposition. The value functions of individual agents are highlighted in green, value decomposition in blue, and target aggregation in orange.

diverse across models in the ensemble but each model would also only be trained on a small subset of the episodes which might sacrifice learning efficiency. In our experiments, we adopt to use $p = 0.9$ but similar to prior work (Osband et al., 2016a) we have not found the choice of p to significantly affect the performance of our algorithm.

Value decomposition So far, we presented EMAX as an extension of IDQN. We now discuss the extension of value decomposition algorithms for common-reward tasks with EMAX. Cooperative multi-agent tasks are often formalised with common rewards (Section 2.5). In this setting, agents might frequently receive rewards despite not having contributed to them. Identifying the contribution of individual agents to the received common reward is known as the multi-agent credit assignment problem (Sunehag et al., 2018; Du et al., 2019; Rashid et al., 2020b). In EMAX, this problem has the additional implication that the exploration policy defined in Equation 6.1 does not distinguish which agents need to cooperate in a particular state. This is problematic and might lead to undesirable exploration. Consider the following example of a task with three agents in which two agents cooperate in a state to receive a large reward but the third agent did not contribute to such reward. However, all agents receive the same large reward and, consequently, the third agent under the EMAX exploration policy might follow this reward that depended on the actions of other agents in the belief that

Algorithm 7 Training QMIX with EMAX

Initialise: parameters $\{\theta_i^k\}_{k=1}^K$ of value functions $\{Q_i^k\}_{k=1}^K$ for each agent $i \in I$
Initialise: parameters θ_{mix} and $\bar{\theta}_{\text{mix}}$ of the main and target mixing networks
Initialise: empty episodic replay buffer $\mathcal{D} \leftarrow \emptyset$
for each episode **do**
  Obtain initial state $s^0 \sim \mu$ and joint observation $\mathbf{o}^0$
  for each step $t = 0, \cdots, T$ **do**
    for each agent $i \in I$ **do**
      Select action $a_i^t \sim \pi_i^{\text{expl}}(h_i^t; \theta_i)$ (Equation 6.1)
    end for
    Apply joint action $\mathbf{a}^t = (a_1^t, \ldots, a_N^t)$
    Receive next state $s^{t+1} \sim \mathcal{T}(s^t, \mathbf{a}^t)$, reward $r^t = \mathcal{R}(s^t, \mathbf{a}^t, s^{t+1})$, and joint observation $\mathbf{o}^{t+1} \sim \mathcal{O}(s^t, \mathbf{a}^t)$
  end for
  Sample bootstrap masks $\{m^k\}_{k=1}^K$ from Bernoulli(p)
  Store episode $(s^{0:T}, h^{0:T}, \mathbf{a}^{0:T}, r^{0:T}, \{m_k\}_{k=1}^K)$ in $\mathcal{D}$
  for each model in the ensemble $k = 1, \ldots, K$ **do**
    Sample batch of episodes B from $\mathcal{D}$ with $m_k = 1$
    Update θ^k and θ_{mix} by minimising $\mathcal{L}(\theta^k)$ (Equation 6.8) averaged over each time step t in B with QMIX targets (Equation 6.9)
  end for
  In a set interval, update target mixing network $\bar{\theta}_{\text{mix}} \leftarrow \theta_{\text{mix}}$
end for

it indicates the potential for cooperation with other agents. To overcome this challenge and to make sure that the EMAX exploration policy guides each agent towards states and actions in which that particular agent's cooperation, rather than any agents' cooperation, is required, it is important that the value estimates of that agent correspond to its individual contribution to the common rewards.

Value decomposition algorithms such as VDN (Sunehag et al., 2018) and QMIX (Rashid et al., 2020b) are specifically designed to address the multi-agent credit assignment problem by learning such individual value functions for each agent that identify their contribution to received common rewards. By integrating these techniques into EMAX, the exploration policy can benefit from the multi-agent credit assignment achieved by the value decomposition, and the op-

timisation of the value functions can be stabilised by computing target estimates across the ensemble as proposed in EMAX. To integrate value decomposition methods into EMAX, each agent trains an ensemble of independent utility functions as proposed above. The total loss for the k-th utility functions of all agents with parameters $\theta^k = \{\theta_i^k\}_{i \in I}$ when using VDN is given by

$$\mathcal{L}(\theta^k) = \mathbb{E}_{(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{h}^{t+1}) \sim \mathcal{D}} \left[\left(r^t + \gamma \sum_{i \in I} \max_{a_i \in A_i} Q_i^{\text{mean}}(h_i^{t+1}, a_i; \theta_i) - \sum_{i \in I} Q_i^k(h_i^t, a_i^t; \theta_i^k) \right)^2 \right] \tag{6.7}$$

and for QMIX is defined as follows:

$$\mathcal{L}(\theta^k) = \mathbb{E}_{(\mathbf{h}^t, s^t, \mathbf{a}^t, r^t, \mathbf{h}^{t+1}, s^{t+1}) \sim \mathcal{D}} \Big[\Big(r^t + \gamma y_{\text{tot}} - f_{\text{mix}} \left(Q_1^k(h_1^t, a_1^t; \theta_1^k), \ldots, Q_N^k(h_N^t, a_N^t; \theta_N^k); \theta_{\text{mix}} \right) \Big)^2 \Big] \tag{6.8}$$

with target value

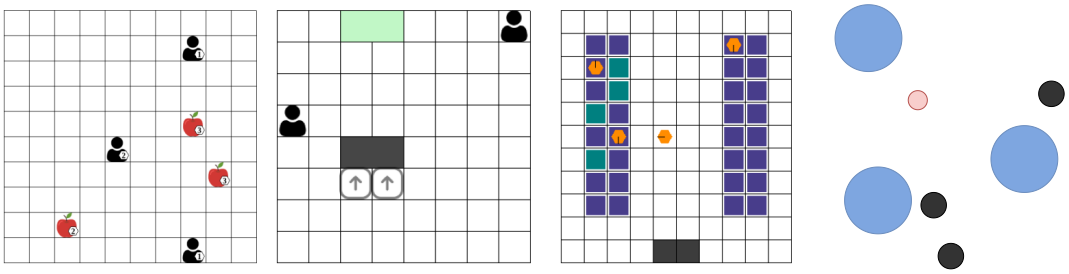
$$y_{\text{tot}} = f_{\text{mix}} \left(\max_{a_1 \in A_1} Q_1^{\text{mean}}(h_1^{t+1}, a_1; \theta_1), \ldots, \max_{a_N \in A_N} Q_N^{\text{mean}}(h_N^{t+1}, a_N; \theta_N); \bar{\theta}_{\text{mix}} \right) \tag{6.9}$$

For QMIX, we use a single mixing network and target mixing network with parameters θ_{mix} and $\bar{\theta}_{\text{mix}}$, respectively, to aggregate the utility estimates for all utility functions in the ensemble. We consider both VDN and QMIX for our experimence since the aggregation of QMIX is able to represent a wider set of centralised value functions, but VDN has been shown to be more sample efficient in tasks that do not seem to require a non-linear aggregation as discussed in Section 4.4.

6.2 Experimental Setup

We conduct two evaluations to assess the efficacy of EMAX in multi-agent environments. First, we focus on mixed-objective environments under the POSG formalism (Section 2.4), in which all agents receive individual rewards. In this setting, we evaluate IDQN with and without EMAX. Second, we evaluate EMAX as an extension of IDQN as well as value decomposition approaches VDN and QMIX in common-reward environments under the Dec-POMDP formalism (Section 2.5).

Mixed-objective evaluation In the mixed-objective evaluation, we compare IDQN with and without EMAX in 11 multi-agent tasks of the level-based foraging



(a) LBF (b) BPUSH (c) RWARE (d) MPE

Figure 6.4: Visualisations of four multi-agent environments: (a) level-based foraging (LBF), (b) boulder-push (BPUSH), (c) multi-robot warehouse (RWARE), and (d) multi-agent particle environment (MPE).

(LBF) (Albrecht and Ramamoorthy, 2013; Papoudakis et al., 2021) and multi-robot warehouse (RWARE) environments (Christianos et al., 2020; Papoudakis et al., 2021). These tasks were selected due to their mixture of cooperation and competition in the setting with individual rewards. In LBF, agents are incentivised to collect all items that they can collect by themselves as fast as possible, to receive their reward by themselves and not share it with other agents. For items of sufficiently high level, agents need to cooperate to pick it up and collect their rewards. In RWARE, agents are rewarded for collecting and delivering requested shelves. Cooperation in RWARE might constitute agents giving way to each other to avoid agents blocking each others' path. In contrast to LBF, the successful cooperation and resolution of such situations does not lead to immediate rewards for any agent but enables further rewards in the episode by allowing agents to deliver more shelves after the resolution. Both environments are visualised in Figure 6.4. For detailed descriptions of both environments, we refer to Section 4.1.

Common-reward evaluation In the common-reward evaluation, we conduct a large evaluation comparing a total of eleven MARL algorithms in 21 tasks across four multi-agent environments. We compare IDQN, VDN, and QMIX as well as their extensions with EMAX, which we will denote IDQN-EMAX, VDN-EMAX, and QMIX-EMAX, respectively, three value-based exploration algorithms in MAVEN (Mahajan et al., 2019), CDS (Li et al., 2021a), and EMC (Zheng et al., 2021), and independent and multi-agent PPO (IPPO and MAPPO) that have been shown to exhibit strong MARL performance (Papoudakis et al., 2021;

Yu et al., 2022). We consider 21 common-reward tasks across four multi-agent environments: eight level-based foraging (LBF) tasks, four boulder-push (BPUSH) tasks (Christianos et al., 2023), six multi-robot warehouse (RWARE) tasks, and three multi-agent particle environment (MPE) tasks (Mordatch and Abbeel, 2018; Lowe et al., 2017). These tasks were selected since they represent a diverse set of cooperative MARL tasks which require agents to cooperate to achieve high rewards. Many of these tasks are further considered challenging exploration tasks due to sparse rewards. All environments are visualised in Figure 6.4. For detailed descriptions of LBF, RWARE, and MPE, we refer to Section 4.1, and we provide a more detailed description of BPUSH below. We hypothesise that EMAX is particularly well suited for this common-reward setting due to its exploration policy focusing on states and actions with the potential for cooperation with other agents.

In the boulder-push environment (BPush) (Christianos et al., 2023), agents need to navigate a grid-world to move a boulder to a target location. Agents observe the location of the boulder, all other agents, and the direction the boulder needs to be pushed in. The action space of all agents consists of the same discrete actions $A = \{\text{move up}, \text{move down}, \text{move left}, \text{move right}\}$. Agents only receive rewards of 0.1 per agent for successfully pushing the boulder forward in its target direction, which requires cooperation of all agents, and a reward of 1 per agent for the boulder reaching its target location. Unsuccessful pushing of the boulder by some but not all agents leads to a penalty reward of -0.01. Episodes terminate after the boulder reached its target location or after at most 50 time steps. BPUSH tasks considered in this chapter vary in the size of the grid-world and the number of agents varying between two and four.

Evaluation metrics We report the mean evaluation returns as well as 95% confidence intervals computed over five runs in all individual tasks across both settings. In common-reward tasks, we report the returns computed over the common rewards, and in mixed-objective tasks we report the sum of all agents' evaluation returns. Following Agarwal et al. (2021), we report aggregated normalised evaluation returns and performance profiles with the interquartile mean (IQM) and 95% confidence intervals computed over all tasks in each setting. The learning curves indicate the sample efficiency of agents, which is largely determined by their ability to effectively explore the environment, and performance profiles allow to

compare the distribution of final evaluation returns indicating the robustness of the final policies learned by each algorithm. We normalise returns between the minimum (0) and maximum (1) achieved returns following Equation 4.1.

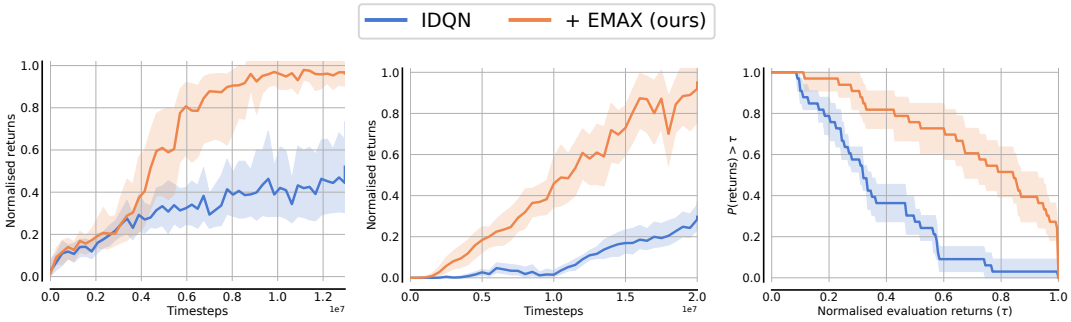
To evaluate the training stability of algorithms, we would like to capture how variable and noisy gradients are throughout training. To measure this variability, we first detrend gradient norms throughout training by deducting each stored gradient norm from its subsequent norm, and compute the conditional value at risk (CVaR) of detrended gradient norms:

$$\text{CVaR}(g') = \mathbb{E}\left[g' \mid g' \geq \text{VaR}_{95\%}(g')\right] \text{ and } g'_t = |\nabla_{t+1}| - |\nabla_t| \tag{6.10}$$

where the value at risk (VaR) corresponds to the value at the 95% quantile of all detrended gradient norm values. This metric corresponds to the short-term risk across time suggested by Chan et al. (2020) and indicates the stability of gradients throughout training. A larger CVaR value indicates more variability in gradients which can indicate unstable training, while a smaller CVaR value indicates less variability in gradients and more stable training.

Implementation details We provide details on conducted hyperparameter optimisation and used hyperparameters across all evaluations in Appendix C.1. Furthermore, we note that across all experiments agents share network parameters with each other due to its established benefits for sample efficiency (as shown in Section 4.3.3). To allow for agent specialisation, shared networks receive the agent identity as additional input in the form of a one-hot vector. EMAX uses ensembles with $K = 5$ value functions unless stated otherwise.

Computational resources All experiments were conducted on (1) desktop computers with two Nvidia RTX 2080 Ti GPUs, Intel i9-9900X @ 3.50GHz CPU, 62GB RAM, running Ubuntu 20.04, (2) two server machines with four Nvidia V100 GPUs, Intel Xeon Platinum 8160 @ 2.10GHz CPU, 503GB RAM, running CentOS Linux 7 OS, and (3) one server machine with Nvidia RTX A4500 GPUs, an AMD EPYC 7763 CPU with 64 cores at up to 3.6GHz, 1TB RAM, running Ubuntu 22.04. The speedtest for varying ensemble sizes reported in Table 6.1 has been conducted on the desktop computer.



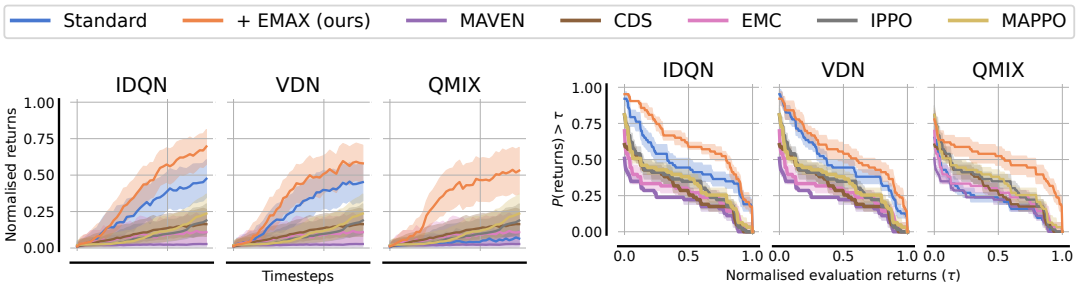
(a) LBF – Normalised (b) RWARE – Normalised (c) Performance profile

Figure 6.5: Normalised evaluation returns throughout training of IDQN with and without EMAX across mixed-objective (a) LBF and (b) RWARE tasks, and (c) performance profiles across all 11 mixed-objective tasks. Lines and shading represent the interquartile mean and 95% confidence intervals of evaluation returns, respectively.

6.3 Evaluation Results

First, we evaluate IDQN with and without EMAX in 11 mixed-objective tasks of the LBF and RWARE environments. Figure 6.5 shows the learning curves with normalised evaluation returns throughout training across all mixed-objective tasks in the respective environments, and the performance profile at the end of training. Across all 11 tasks, EMAX improves final evaluation returns of IDQN by 189%, with 105% improvement in LBF and 274% improvement in RWARE. From the performance profile, we also see that EMAX significantly improves the robustness of IDQN across all tasks. We provide learning curves for each individual task in Appendix C.2.

Following the same evaluation protocol, we then evaluate EMAX on top of IDQN, VDN, and QMIX across 21 common-reward tasks. Figure 6.6 visualises the learning curve and performance profile of evaluation returns of all algorithms. Similar to our evaluation in mixed-objective tasks, EMAX substantially improves final evaluation returns of IDQN, VDN, and QMIX in common-reward tasks, shown in Figure 6.6a, by 60%, 47%, and 538%, leading to higher final returns compared to their baselines in 18, 16, and 20 out of 21 tasks, respectively. These results arise from EMAX improving the sample efficiency and learning stability of the extended algorithms, as we will show in Section 6.4. Additionally, QMIX-EMAX is able to learn effective policies in several hard exploration tasks where QMIX fails to achieve any reward. From the performance profile in Figure 6.6b



(a) Normalised evaluation returns

(b) Performance profile

Figure 6.6: (a) Normalised evaluation returns throughout training and (b) performance profile of all algorithms aggregated across all 21 tasks. EMAX (orange) significantly improves the sample efficiency and final achieved returns of all algorithms. Lines and shading represent the interquartile mean and 95% confidence intervals of evaluation returns, respectively, aggregated over five runs for every task, for a total of 105 runs per algorithm.

we also see the distribution of evaluation returns at the end of training across all algorithms and tasks, indicating the improved robustness with EMAX across all tasks. We provide normalised evaluation returns for each environment in Figure 6.7, as well as learning curves in all individual tasks in Appendix C.3.

In LBF, EMAX significantly improves the performance of QMIX whereas minor improvements can be seen for IDQN and VDN in the common-reward setting, and significant gains are observed in the mixed-objective setting. Inspecting learning curves in individual tasks (Appendix C.3.1) shows that QMIX, MAVEN, CDS and EMC fail to achieve any rewards in several LBF tasks with particularly sparse rewards. We hypothesise that these algorithms suffer from the large dimensionality of the joint observation as input to the mixing network which is inefficient to train with the sparse learning signal of these tasks. The uncertainty-guided exploration of EMAX seems to alleviate these inefficiencies.

A similar trend can be observed in BPUSH where, most notably, VDN-EMAX and QMIX-EMAX learn to solve a BPUSH task with four agents in which no baseline demonstrates any positive rewards (see Figure C.3d). To successfully solve this task, all agents need to cooperate with any miscoordination leading to negative rewards. CDS performs best among all baselines but still worse than VDN and QMIX with EMAX, and IPPO and MAPPO fail to learn in all tasks. BPUSH and LBF are both challenging exploration tasks with sparse rewards and

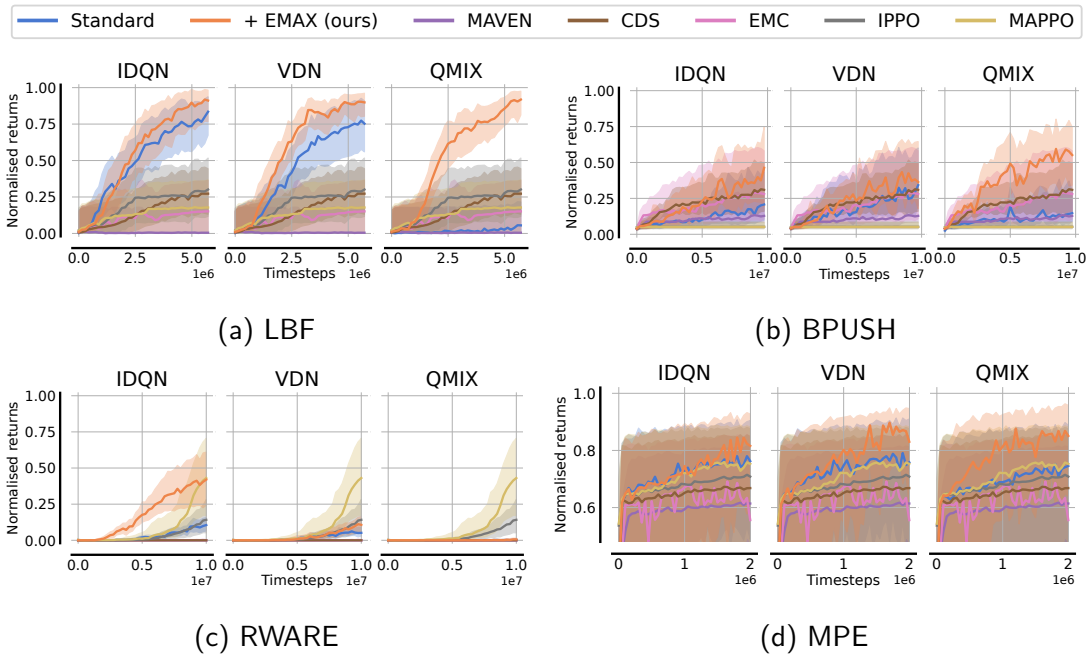


Figure 6.7: Interquartile mean and 95% confidence intervals of normalised evaluation returns for all algorithms in each environment.

many states where agents clearly need to cooperate to achieve high returns, e.g. by moving the boulder in BPUSH or collecting high-level items in LBF. EMAX significantly improves the sample efficiency and final performance in these tasks, indicating its ability to guide the multi-agent exploration towards states and actions with the potential for cooperation.

In RWARE, consistent with prior work (Papoudakis et al., 2021), independent learning value-based algorithms outperform centralised value decomposition methods due to highly sparse rewards in the common-reward setting. IDQN-EMAX outperforms all baselines across four RWARE tasks with larger warehouses, and IDQN-EMAX and VDN-EMAX both significantly improve upon their extended baselines in all RWARE tasks, achieving 330% and 252% higher final evaluation returns, respectively, whereas QMIX with and without EMAX fail to learn. Lastly, IPPO and MAPPO perform well in RWARE with MAPPO reaching highest evaluation returns in two smaller RWARE tasks. However, in tasks with larger warehouses and, thus, more required exploration, IDQN-EMAX outperforms all other algorithms. It is worth highlighting that no value-based algorithm achieved non-zero rewards in this environment within four million time steps of training in prior evaluations (Papoudakis et al., 2021). To the best of

our knowledge, IDQN-EMAX is the first algorithm that outperforms actor-critic methods like IPPO and MAPPO in RWARE tasks. We further note that RWARE, in contrast to LBF and BPUSH, contains many states in which agents need to cooperate by giving way to other agents. Without doing so, agents block each others' paths and, thereby, limit the evaluation returns the team can achieve. However, successful cooperation to give way does not directly lead to a reward signal since agents only receive rewards for delivering requested shelves. Due to efficacy of EMAX in RWARE tasks, we hypothesise that EMAX appears to be able to successfully guide agents towards such cooperative states, even if the reward for successful cooperation are only received in future time steps. Similar benefits are also observed for IDQN with EMAX in mixed-objective RWARE tasks in which EMAX improves final evaluation returns by 274%.

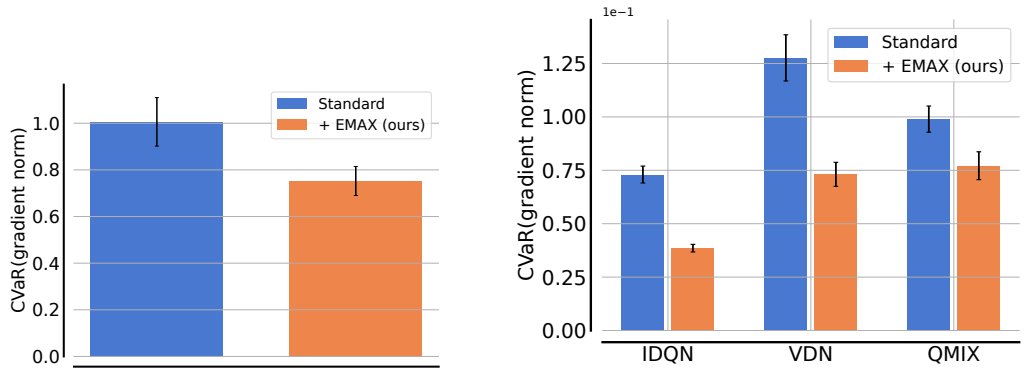
Lastly, we evaluate in three common-reward tasks of the MPE environment. In contrast to other environments, MPE features continuous observations and dense rewards. Furthermore, the adversary and predator-prey tasks contain stochastic transitions due to the adversarial agent being controlled by a pre-trained policy. In all three MPE tasks, we see improvements in sample efficiency and final performance for algorithms with EMAX compared to extended algorithms, even if the improvements are less severe than in the other environments that feature sparse rewards. In particular in the MPE spread task, EMAX significantly improves the performance of all extended algorithms by significant margins. It is also worth highlighting that despite the stochasticity of the environment transitions in the predator-prey and adversary tasks, EMAX still improves the performance of the baselines. This indicates that EMAX is able to effectively guide the exploration even in environments with stochastic transitions.

6.4 Analysis and Ablations

In this section, we further investigate the efficacy of all components of EMAX to study our hypotheses from Section 6.1:

1. EMAX targets reduce the variability of gradients during training.

2. The EMAX exploration policy leads to more exploration of states and actions with the potential for cooperation.



(a) Mixed-objective tasks (b) Common-reward tasks

Figure 6.8: Average and standard error of the conditional value at risk (CVaR) of detrended consecutive gradient norms of (a) IDQN with and without EMAX across 11 mixed-objective and (b) IDQN, VDN, and QMIX with and without EMAX across 21 common-reward tasks.

3. The EMAX evaluation policy reduces the likelihood of selecting sub-optimal actions.

After answering these questions, we provide further ablations of these techniques, show that comparably small ensembles of value functions are sufficient to achieve the benefits of EMAX, and that baselines even with networks of comparable size to EMAX do not achieve the same benefits.

6.4.1 Training Stability

In Section 6.1, we stated that the computation of EMAX target values reduces the variability of gradients during training, and, thus, improves stability of the optimisation (as previously observed in single-agent RL (Liang et al., 2022)). To demonstrate this stabilising effect, Figure 6.8 visualises the average and standard error of the stability of gradients measured by the conditional value at risk (CVaR) of gradient norms computed for IDQN with and without EMAX across 11 mixed-objective tasks (Figure 6.8a) as well as for IDQN, VDN, QMIX with and without EMAX across all 21 common-reward tasks (Figure 6.8b). We observe that the target computation of EMAX significantly reduces the CVaR of gradient norms for all algorithms in both settings, indicating more stable optimisation, thus, confirming our hypothesis. The difference for QMIX in the common-reward setting is less pronounced as for other algorithms since the base algorithm fails

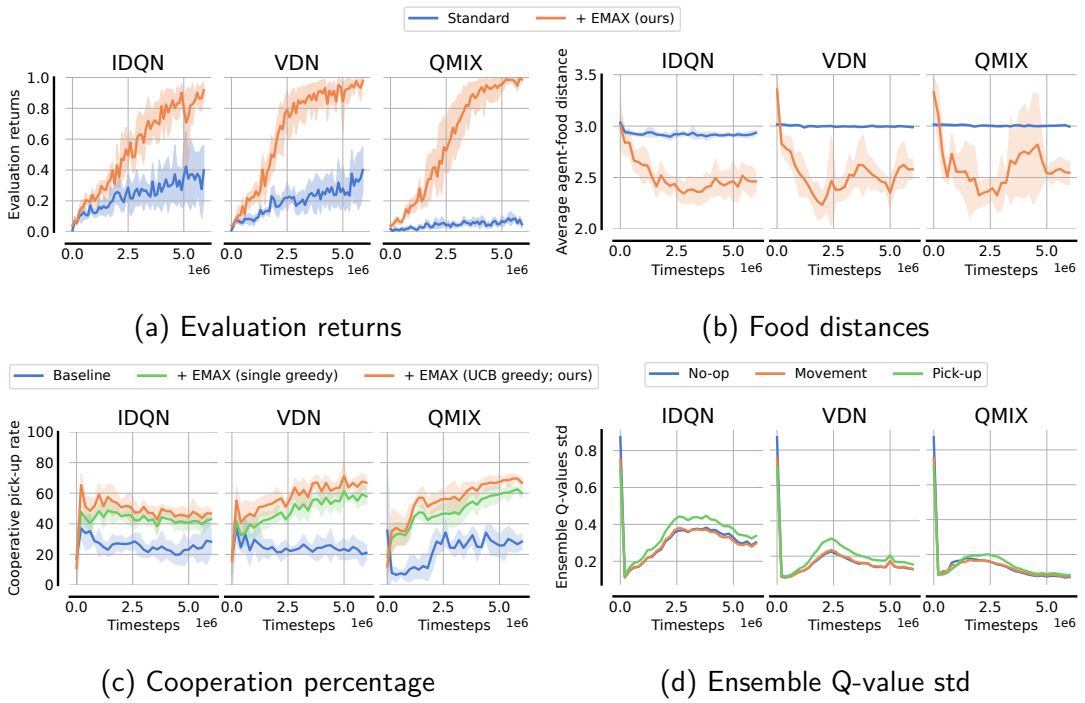


Figure 6.9: (a) Mean and 95% bootstrapped confidence intervals of evaluation returns, (b) mean and standard deviation of average food distances across rollouts, (c) mean and standard deviation of percentages of agents selecting the pick-up action and in states with a chance for cooperation in the LBF 10x10-3p-5f task, and (d) the standard deviation of action-value estimates for no-op, movement, and pick-up actions in states with a chance for cooperation in the LBF 10x10-3p-5f task with common rewards.

to learn in several tasks, leading to little training with low gradient variability independent of the target values.

6.4.2 Exploration Policy

To validate our hypothesis that the EMAX exploration policy leads to more exploration of states and actions with the potential for cooperation (Section 6.1), we train IDQN, VDN, and QMIX with and without EMAX in the LBF 10x10-3p-5f task with common rewards where agents need to cooperate to pick up some of the food items. Figure 6.9 shows the evaluation returns throughout training, the average distances of agents to the closest food, and the percentage of agents selecting the pick-up action in states where multiple agents need to cooperate to pick up food. To obtain the average distances to nearby food and cooperation percentages, we rollout the exploration policy of the baseline algorithm and the

EMAX UCB exploration policy in the LBF task for 50 episodes every 200,000 time steps of training, and compute the respective values across rollouts. We emphasise that a lower average distance of agents to the closest food and a higher percentage of selecting the pick-up action in cooperative states indicates that agents seek out states with the potential for cooperation and prefer to apply actions with the chance for cooperation, respectively. Figure 6.9 shows that our hypotheses about the EMAX exploration policy hold in the tested task:

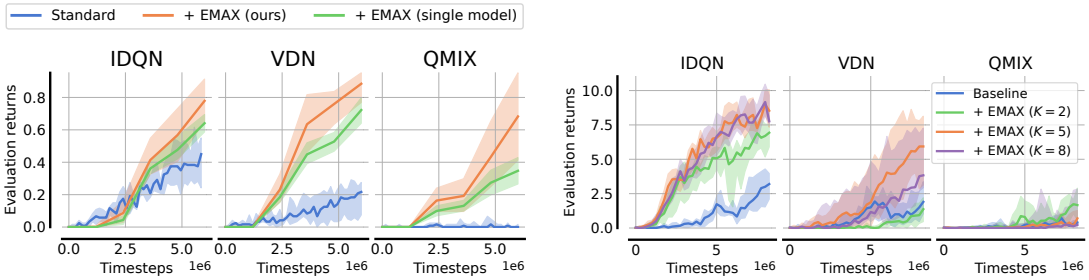
1. Agents following the EMAX exploration policy seek out states with the potential for cooperation more often compared to the baseline following a random exploration policy, as indicated by the lower average distance of EMAX agents to food items compared to the baseline in Figure 6.9b.

2. Agents following the EMAX exploration policy are more likely to select the pick-up action in states with potential for cooperation, as shown in Figure 6.9c.

Together, these effects lead to EMAX agents learning significantly more efficiently and achieving higher evaluation returns compared to the baseline (Figure 6.9a).

To separate the effect of the efficacy of the EMAX training and the exploration policy, we also compare to the percentage of choosing to pick up in cooperative states by greedily following any of the value functions in the ensemble instead of following the UCB policy across the ensemble. While this ablation leads to a significant improvement over the extended algorithms, it still exhibits a lower rate of choosing to cooperate compared to the EMAX exploration policy.

Lastly, Figure 6.9d visualises the standard deviation of action-value estimates across the ensemble for the no-op action, movement actions, and the pick-up action in states with the potential for cooperation between agents. This plot shows that the deviation of value estimates across the ensemble for all types of actions is similar early in training but as training progresses and agents sometimes cooperate successfully and sometimes fail to cooperate, the deviation for the pick-up action with potential for cooperation rises higher than the deviation for other actions in states with the potential for cooperation. Furthermore, alongside Figure 6.9a showing evaluation returns, we can see that once agents successfully cooperate most of the time, the standard deviation for the pick-up action starts to reduce. For QMIX with EMAX, we can see that this reduction ends in the standard deviation of action values for the cooperative pick-up action



(a) Evaluation policy analysis

(b) Ensemble size analysis

Figure 6.10: (a) Evaluation returns for the baselines and EMAX algorithms in the LBF 10x10-4p-4f-coop task with common rewards, and an ablation of the evaluation policy. For the single model ablation, the agents follow the greedy policy with respect to a single value function within their model instead of computing a majority vote across greedy policies. (b) Evaluation returns for varying ensemble sizes $K \in \{2, 5, 8\}$ in the RWARE 11×10 4ag task.

and non-cooperative actions reaching similar levels once close-to-optimal performance is reached since now agents almost always cooperative successfully. This further indicates that, as desired, EMAX incentivises exploration of cooperative actions as long as such cooperation is not reliably achieved yet, but such bias towards cooperative actions diminishes as the policy starts to reliably cooperate successfully.

6.4.3 Evaluation Policy

The evaluation policy of EMAX selects actions by a majority vote across all policies in the ensemble (Equation 6.6). We hypothesised that such a policy is more robust to sub-optimal action selection since any individual policy taking sub-optimal actions does not impact the executed policy as long as the majority of policies agree on the optimal action. Figure 6.10a shows the evaluation returns of IDQN, VDN, and QMIX with and without EMAX in the LBF 10x10-4p-4f-coop task with common rewards. For EMAX, we show the evaluation policy using majority voting (ours) as well as an ablation following the greedy policy with respect to any of the individual value functions within the ensemble (single model). We highlight that no further agents were trained, but we directly extract the individual value functions within the ensemble and evaluate them, so the only difference in the EMAX single policy ablation and ours is the followed policy, not

the underlying value functions. This experiment indicates the improved robustness of our majority voting to select actions leading to higher evaluation returns in a task that frequently requires agents to cooperate.

6.4.4 Ensemble Size

The computational cost of training an ensemble of models scales with the ensemble size K. To illustrate the additional cost, we investigate the training speed of EMAX for varying K and pose the question of how many models are needed in the ensemble to benefit from the improved training stability and exploration. To investigate this question, we evaluate all algorithms with varying K in the RWARE 11x10 task with four agents (Figure 6.10b), in which EMAX led to substantial improvements for IDQN and VDN. In this task, we observe that the benefits of larger ensemble models saturate at $K = 5$. EMAX with $K = 8$ performs identical or worse for all algorithms, and the smaller ensemble $K = 2$ reaches lower returns for IDQN and VDN. These results suggest that a comparably small ensemble with $K = 5$ can significantly improve sample efficiency with EMAX. Additionally, we hypothesise that larger ensemble value functions may require more data to train, thus leading to diminishing benefits for ensembles of many value functions. Table 6.1 shows the average time to train IDQN, VDN, QMIX, and their corresponding EMAX extensions with $K \in \{2, 5, 8\}$ for 10,000 time steps in the LBF 10x10-3p-3f task. These times were averaged across ten runs. We can see that training an ensemble of $K = 5$ value functions, as applied in our evaluation, increases the training time by less than 100%. While this cost is significant, the increase in computational cost is notably less than a linear increase due to parallelisation on modern hardware, and we believe that it is justified in cases where the significant improvements of EMAX in both sample efficiency and stability are of importance.

To further investigate whether the performance benefits of EMAX arise solely due to its larger number of learnable parameters compared to the baselines algorithms, we evaluate all baselines with larger network sizes. We keep the overall architecture of networks identical, so all value function networks consist of one hidden layer projecting the input observations to a hidden size of d^h, followed by a gated recurrent unit (GRU) (Cho et al., 2014) with identical hidden dimensionality, and a final linear layer projecting the hidden output state of the

Table 6.1: Average training time (in seconds) for baselines and EMAX algorithms with varying ensemble sizes K to complete $10{,}000$ time steps of training in the LBF 10x10-3p-3f task. Relative increase to the training time of the baseline algorithm ($K = 1$) is given in parenthesis. Times are averaged across ten runs.

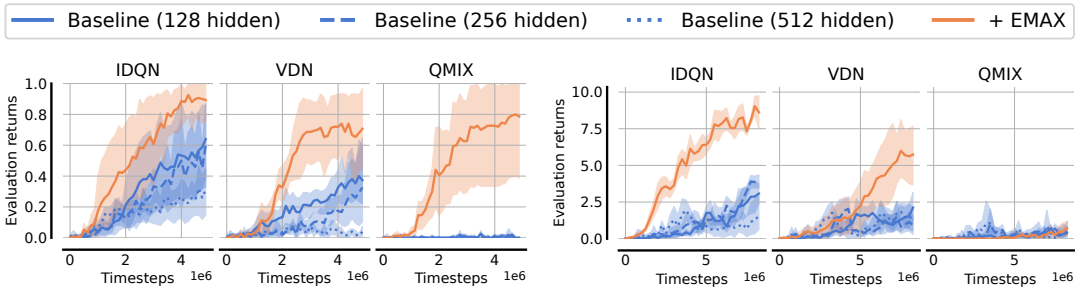
Algorithm	Baseline	$K = 2$	$K = 5$	$K = 8$
IDQN	16.80	21.29 (+27%)	33.04 (+97%)	48.06 (+186%)
VDN	16.92	21.56 (+27%)	33.25 (+97%)	48.16 (+185%)
QMIX	17.70	22.53 (+27%)	33.71 (+90%)	48.66 (+175%)

Table 6.2: Observation dimensionality and the resulting number of parameters within the main value function networks for baseline algorithms with hidden sizes of 128, 256, and 512, as well as for EMAX with $K = 5$ models in the ensemble and hidden size of 128 for one LBF and RWARE task.

Task	$\|o\|$	$\|A_i\|$	Base (128)	Base (256)	Base (512)	EMAX ($K = 5$)
LBF 10x10-4p-3f-coop	25	6	103,174	402,950	1,592,326	515,870
RWARE 11×20 4ag	95	5	112,005	420,613	1,627,653	560,025

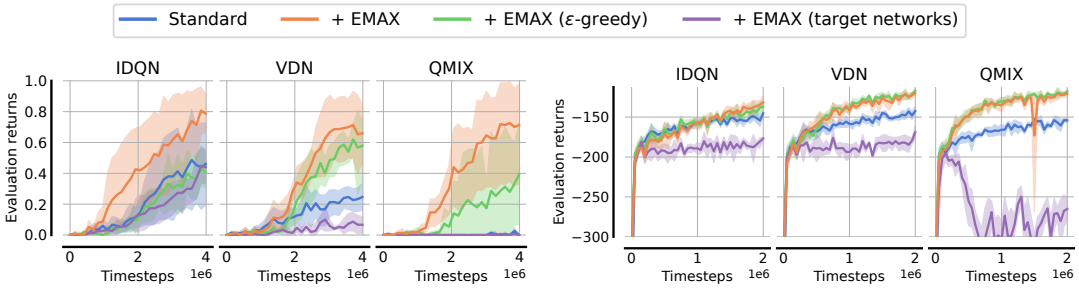
GRU to action-values for each action with the dimensionality of the action space of an individual agent i, i.e. $|A_i|$. We evaluate the baselines with hidden sizes of $d^h \in \{128, 256, 512\}$ and compare their performance to EMAX with $K = 5$ models in the ensemble and hidden size of 128. The number of total parameters resulting from these models for one LBF and one RWARE task are shown in Table 6.2. As we can see, EMAX with $K = 5$ (and hidden size of 128) has exactly five times more parameters in the model compared to the baseline with one model with hidden size of 128. The baseline model with hidden size of 256 is comparable to the model size of EMAX while the baseline model with hidden size of 512 is roughly three times larger than the ensemble of EMAX.

Figure 6.11 shows the evaluation returns of all baseline algorithms for varying model sizes compared to EMAX with $K = 5$ models in the ensemble in one LBF and one RWARE task. As we can see, the baselines are unable to make effective use of larger networks and reach similar evaluation returns to the original baseline with hidden sizes of 128 despite four times and fifteen times more parameters in the model. Despite these larger networks, the baseline algorithms are unable



(a) LBF 10x10-4p-3f-coop (b) RWARE 11×20 4ag

Figure 6.11: Mean and 95% confidence intervals of evaluation returns for all baseline algorithms with default and larger network sizes, and EMAX extensions.



(a) Ablations in LBF 10x10-4p-3f-coop. (b) Ablations in MPE spread.

Figure 6.12: Mean and 95% confidence intervals of evaluation returns for all baselines and EMAX algorithms and two ablations in (a) LBF 10x10-4p-3f-coop, and (b) MPE spread. For the ablations, we replace the UCB exploration policy with ϵ-greedy exploration (green) and the EMAX target computation with standard target networks (purple), respectively.

to reach the performance of EMAX with $K = 5$ models in the ensemble. This suggests that the ensemble in EMAX and its use in the exploration policy, evaluation policy, and target computation is important to make effective use of the increase in parameters and EMAX does not outperform the baselines due to its larger computational budget.

6.4.5 Ablations

We already demonstrated that the EMAX target computation and exploration policy lead to more robust gradients during optimisation and focus exploration on cooperative actions. To discriminate the importance of both of these compo-

nents to the performance of EMAX algorithms, we conduct an ablation study in two tasks: LBF 10x10-4p-3f-coop, and MPE spread. We compare the performance of the full EMAX algorithm to two ablations which (1) substitute the UCB exploration policy with an ϵ-greedy exploration policy, and (2) use target networks to compute target values with each network in the ensemble having its own target network. The results of this ablation study (Figure 6.12) demonstrate the importance of both of these components, leading to notably better or similar performance for all algorithms in both tasks. In particular the EMAX target computation significantly improves performance across all algorithms and tasks. The UCB exploration policy leads to significant improvements in LBF, but only marginal gains in MPE. We hypothesise that agents need to explore less in the MPE spread task due to the dense rewards, and that there are only few states in which agents have to apply a particular action to coordinate.

6.5 Related Work

Uncertainty for exploration in RL Using uncertainty to guide exploration is a well-established idea in RL . One family of algorithms that leverages this idea are randomised value functions (Osband et al., 2019) that build on the idea of Thompson sampling (Thompson, 1933) from the multi-armed bandits literature (Scott, 2010; Chapelle and Li, 2011). Posterior sampling RL extends Thompson sampling by maintaining a distribution of plausible tasks, computes optimal policies for sampled tasks, and continually updates its distribution of tasks from the collected experience (Osband et al., 2013). This approach has extensive theoretical guarantees (Osband and van Roy, 2017) but is difficult to apply to complex tasks (Osband et al., 2016b). This limitation has been addressed in subsequent works (Osband et al., 2016b,a; Janz et al., 2019), most notably in bootstrapped DQN (Osband et al., 2016a) which approximates randomised value functions by training an ensemble of value functions and explores by randomly sampling a value function to greedily follow at the beginning of each episode. SUNRISE (Lee et al., 2021) and MeanQ (Liang et al., 2022) also train an ensemble of value functions but instead of sampling value functions to explore, they follow a UCB policy using the average and standard deviation of value estimates across the ensemble to explore. Moreover, SUNRISE computes a weighting of values loss terms based on the variance of target values, and MeanQ stabilises the optimisation by

computing lower variance target values as the average value estimate across the ensemble (Anschel et al., 2017). Separately, Fu et al. (2022a) extend posterior sampling to model-based RL by learning a probabilistic model of the environment, and Dearden et al. (1998) applied these ideas to tabular Q-learning to learn distributions over Q-values and approximate the value of information of actions. Related to all these ideas, optimistic value estimates in the face of uncertainty can be used to promote exploration for actor-critic (Ciosek et al., 2019) and model-based RL (Sessa et al., 2022). All these approaches leverage uncertainty to guide their exploration, similar to EMAX. However, in contrast to discussed approaches, EMAX focuses on exploration of cooperation in environments with multiple concurrently learning agents.

Multi-agent exploration For the multi-agent setting, Wang et al. (2020b) incentivise agents to interact with each other by intrinsically rewarding them for mutually influencing their transition dynamics or value estimates. Similar intrinsic rewards can be assigned for reaching goal states to train separate exploration policies (Liu et al., 2021b). However, intrinsic rewards for exploration have to be carefully balanced for each task due to the modified optimisation objective (Schäfer et al., 2022; Chen et al., 2022). To address this challenge, LIGS (Mguni et al., 2022) formulate the assignment of intrinsic rewards as a MARL problem and train an agent to determine when and which intrinsic reward should be given to each agents. Experience and parameter sharing have been leveraged to greatly improve sample efficiency for MARL by synchronising agents' learning and make use of more data (Christianos et al., 2020, 2021). REMAX (Ryu et al., 2022) identifies valuable initial states for episodes to guide exploration based on a latent representation of states learned using the interactions of agents in the environment. However, there is little research using distributional and ensemble-based techniques for MARL exploration. Zhou et al. (2020) extend posterior sampling (Osband et al., 2013) for MARL, but are limited to two-player zero-sum extensive games. We aim to close this gap by proposing EMAX, an ensemble-based technique for efficient exploration in cooperative MARL. We further highlight that EMAX is a plug-and-play algorithm that can enhance any value-based MARL algorithm, including most existing MARL exploration techniques described in this paragraph.

6.6 Conclusion

In this chapter, we proposed EMAX, a general framework to extend any value-based MARL algorithms using ensembles of value functions. EMAX leverages the disagreement of value estimates across the ensemble with a UCB policy to guide exploration towards parts of the environment which might require coordination between multiple agents. Additionally, gradients during training are stabilised by computing target values as the average value estimate across the ensemble. Empirical results in 11 mixed-objective and 21 common-reward tasks across four environments demonstrated that EMAX significantly improves sample efficiency, final performance, and training stability as an extension of IDQN and value decomposition algorithms VDN and QMIX. We provided extensive analysis to establish the effects of the EMAX exploration policy and target computation, provided ablations for all components of EMAX, and discussed the computational cost introduced by EMAX with experiments indicating that comparably small ensemble models are sufficient to achieve the demonstrated improvements and that baselines even with comparable network sizes do not achieve the same benefits. We believe that EMAX is a promising approach to improve exploration in cooperative MARL due to its plug-and-play nature and demonstrated efficacy in complex cooperative tasks.

Chapter 7

Warehouse Logistics: A Case Study for Scalable Multi-Agent Reinforcement Learning

Publication

This chapter is based on and adapted from the following publication:

Aleksandar Krnjaic, Raul D. Steleac, Jonathan D. Thomas, Georgios Papoudakis, **Lukas Schäfer**, Andrew Wing Keung To, Kuan-Ho Lao, Murat Cubuktepe, Matthew Haley, Peter Börsting, Stefano V. Albrecht "Scalable multi-agent reinforcement learning for warehouse logistics with robotic and human co-workers." In *Proceedings of the 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2024.

So far in this thesis, we have discussed the challenges of exploration and sample efficiency in single-agent and multi-agent reinforcement learning settings. We have seen that, in particular in the multi-agent setting, reinforcement learning can be quite sample inefficient due to the challenges that arise from the concurrent learning of multiple agents and need for cooperation across agents in many tasks. However, we proposed novel approaches that improve the sample efficiency by sharing experiences with each other (Chapter 5) and focusing exploration on states where interactions between agents matter (Chapter 6). In this chapter, we will look at the setting of warehouse logistics automation as a case study for the complexity of MARL and the considerations required to effectively apply MARL

solutions to real-world problems.[1]

More concretely, in this chapter we will take a look at the problem of *order-picking* within warehouse logistics. Order-picking refers to the process of retrieving items in the warehouse and delivering them to a target location in the warehouse for further handling (Petersen and Schmenner, 1999). An order received by a commercial warehouse operator may comprise of several order-lines that represent specifications of a required item and its required quantity as part of an order. The goal of many order-picking problems can be formulated as the maximisation of the *pick rate*, i.e. the rate at which required items are retrieved in a warehouse to complete orders. Below, we describe two different order-picking paradigms which we will consider in this chapter *person-to-goods* and *goods-to-person*. In both paradigms, for a given set of orders, the objective is to minimise the time for order completion, which is equivalent to maximising the pick rate.

In the person-to-goods (PTG) paradigm, human workers will receive orders and travel around the warehouse with a push cart and pick required items manually. We consider the augmentation of this process with robotic vehicles which we will refer to as automated guided vehicles (AGVs). The general idea of AGV-assisted order-picking has begun to receive attention in the academic literature (Löffler et al., 2022; Žulj et al., 2022; Azadeh et al., 2023). Typically, this involves decoupling a traditional picker's role into order transportation and item picking, where transportation is handled by the AGVs and picking is handled by human or robotic pickers. Augmentation of this paradigm with AGVs requires minimal modification to existing infrastructure and allows for scaling with variation in demand by changing AGV and picker numbers (Azadeh et al., 2023).

In the alternative goods-to-person (GTP) paradigm, large-scale autonomous systems comprised of conveyors, picking robots and transport robots move storage mediums (such as totes, cartons or shelves) containing items to stationary human pickers, who pick and consolidate items out of the storage medium. Automation efforts have generally focused on this GTP paradigm, with numerous examples including the Dematic Multishuttle (Dematic, 2024), Autostore (Dematic, 2024), Quicktron QuickBin (Quicktron, 2024) and Amazon KIVA (Wurman et al., 2008). In comparison to PTG systems, GTP systems have higher throughputs, but require significant capital investment and can be costlier to adjust to varying ware-

[1]The presented research is the result of an industry collaboration and internship at Dematic GmbH, a global company focused on warehouse automation and robotics.

house capacity and consumer demand. For these reasons, adoption is generally limited to larger operations.

Established industry methods for order-picking using heuristic approaches require significant engineering efforts to optimise for innately variable warehouse configurations (Azadeh et al., 2023). Ideally, the derivation of optimal methods for worker control should be an automatic process. Due to the focus on learning decision making and notable prior successes for a number of complex real-world control problems (Bellemare et al., 2020; Li et al., 2021c; Song et al., 2023; Azagirre et al., 2024), RL is a natural choice to automate this process. Since order-picking naturally is a multi-agent problem requiring cooperation between multiple AGVs, robots, and human pickers, we leverage MARL to model the learning of all decision-making entities in the warehouse. Existing heuristic approaches require significant engineering effort and tuning to fit different specifications depending to varying factors such as demand, supply, labour conditions and order profiles. In contrast, MARL is flexible to operate with diverse warehouse and worker specifications such that the same learning process can be applied to learn optimised solutions for varying settings without the need for manual intervention.

In this chapter, we develop a general-purpose and scalable MARL solution for the order-picking problem in warehouses with heterogeneous agents, i.e. robotic and human co-workers. Our approach constructs a multi-layer hierarchy in which a manager agent assigns goals to worker agents (pickers, AGVs), where each goal represents a section of the warehouse (e.g., aisle) in which the worker needs to choose an item to pick. The policies of the manager and worker agents are jointly trained via MARL to maximise a global objective given by the pick rate as defined in Section 7.1. The hierarchical approach effectively reduces the action space of the workers by several magnitudes and facilitates better cooperation through the centrally trained manager agent.

We apply the hierarchical architecture on top of existing MARL algorithms, including independent actor-critic (Mnih et al., 2016; Papoudakis et al., 2021), shared network actor-critic (Gupta et al., 2017; Christianos et al., 2021), and shared experience actor-critic (Christianos et al., 2020), and demonstrate that it significantly improves the sample efficiency and overall pick rate of these algorithms in a diverse set of warehouse configurations. For our experiments, we implement the high-performance Dematic PTG simulator which is capable of representing real-world warehouse operations. Additionally, we introduce an open-

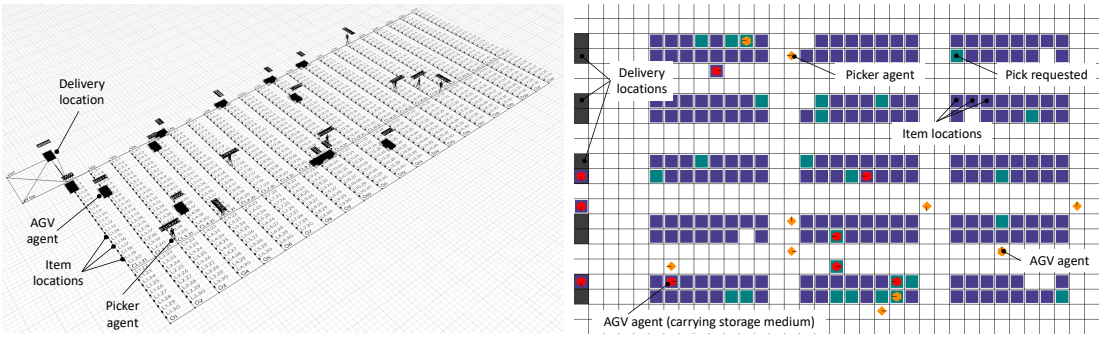


Figure 7.1: Left: Dematic PTG simulator with human pickers and AGVs. Right: TA-RWARE GTP simulator with picking bots (diamond) and AGVs (hexagon).

source adaptation of the RWARE environment (Papoudakis et al., 2021) to represent a GTP warehouse based on the Quicktron QuickBin systems (Quicktron, 2024), named task assignment multi-robot warehouse (TA-RWARE).[2] These two simulation environments are displayed in Figure 7.1. We introduce competitive human-engineered heuristic methods for both picking paradigms as baselines, and show that agents trained via our MARL algorithms achieve superior overall pick rates in both picking paradigms.

7.1 Problem Setting

First, we define and formulate the order-picking problem in the context of warehouse logistics. In the following, we will denote a warehouse as $\mathcal{W} = \{L, Z, W\}$ which is composed of three components:

- L refers to the set of locations within the warehouse, showed in Figure 7.1, and can be further broken down into $L = L_{item} \cup L_{delivery} \cup L_{other}$, where L_{item} refers to the set of locations with items stored inside storage mediums, $L_{delivery}$ refers to locations where completed orders or storage mediums are delivered, and L_{other} refers to other locations (e.g. idle or charging locations).

- Z defines the order distribution, which is dependent on the warehouse's supplier and customer behaviour and is assumed to be known. An order $z = \{(u_0, q_0), \ldots, (u_n, q_n)\}$ is sampled from Z. Each pair (u_k, q_k) represents

[2]The TA-RWARE environment is available open-source at `https://github.com/uoe-agents/task-assignment-robotic-warehouse`.

an order-line, where u represents the item and q the required quantity. Items u are stored inside a storage medium at an item location $l \in L_{item}$.

- $W = V \cup P$ represents the set of workers, where V and P are homogeneous sets of AGVs and pickers, respectively. AGVs $v \in V$ can visit locations $l \in L$, and pickers $p \in P$ can visit locations $l \in L_{item}$.

The order-picking paradigms PTG and GTP differ in the way items are retrieved and delivered:

- In PTG picking, $|L_{delivery}| = 1$. In this paradigm, AGVs are assigned orders sampled from Z with z_v denoting the current order of AGV $v \in V$. A human picker $p \in P$ will pick an order-line (u, q) out of a storage medium at an item location $l \in L_{item}$ and place it into an AGV v. Once the AGV has received all order-lines, the order z_v is completed and the AGV will deliver it to a delivery station $l \in L_{delivery}$.

- In GTP picking, multiple AGVs $v \in V$ will carry separate storage mediums containing items $\{u_0, ..., u_n\}$ which are required in an order z. A picker robot $p \in P$ will move a storage medium from an item location L_{item} containing item u onto an AGV v, and the AGV will take the storage medium to a picking station $l \in L_{delivery}$, where an operator will pick an order-line (u, q) from the AGV. Once all order-lines for order z are picked, the order is completed.

As the evaluation objective, we seek to derive a joint policy π that maximises the average pick rate K in a given warehouse, formally denoted with:

$$\pi \in \arg\max_{\pi} K(\mathcal{W}, \pi) \tag{7.1}$$

The pick rate is measured in completed order-lines per hour. A key desideratum of our solution is to automatically learn optimal policies for any given warehouse configuration and order profile. Specifically, we desire a general-purpose algorithm that can learn to handle variations in multiple dimensions, including the number of total item locations $|L|$, the order distribution Z, and the number of workers $|V| + |P|$. Controlling all workers with a single decision-making entity becomes infeasible due to the joint action space growing exponentially with the number of workers. Hence, we consider MARL approaches in which pickers and AGVs are modeled as individual agents. This multi-agent problem given by a warehouse can be modelled as a POSG with N agents (Section 2.4).

7.2 Warehouse Simulators

We implement two simulators that model the previously described warehouse problems, one for each picking paradigm.

For the action space of agents, we need to consider the tasks of both types of agents. Picker agents need to be able to visit all item locations $l \in L_{item}$ to assist in picking of good. Similarly, AGV agents need to be able to visit all locations $l \in L$ which include all item locations as well as additional locations such as delivery and charging stations. We enable this by defining the action space of pickers and AGVs as $A_p = L_{item}$ and $A_v = L$, respectively. Agents are considered *busy* until they transit to their selected action location. This proposed action space simplifies AGV and picker collaboration by allowing policies to focus on coordinating item location selection between agents while leaving the low-level navigation task to a pre-defined controller. Such a high-level action space has the additional benefit of simplifying the future challenge of sim-to-real for real-world application. However, this modelling decision also results in a large action space that scales with the number of item locations within the warehouse which makes learning in larger warehouses challenging.

7.2.1 Dematic Person-to-Goods Simulator

Our first environment is Dematic's high-performance PTG warehouse simulator that is capable of representing real-world PTG warehouse operations. An example snapshot of a simulated warehouse in our experiments is shown in Figure 7.1 (left). Pickers in this simulator are human workers, and cooperate with AGVs that transport picked items. AGVs travel to multiple storage locations to receive items successively, picked by human pickers. Once all items within an order are collected, the AGV delivers the order to a single delivery location.

The agents are presented with an episodic task consisting of N orders that are randomly distributed within locations L_{item}, terminating when all orders are completed. The observations of agents in the PTG simulator are defined as follows with $\oplus$ denoting the concatenation operator:

$$O_p = \{(l_i^c, l_i^t) \mid i \in I\} \oplus \{z_v \mid v \in V\} \tag{7.2}$$

$$O_v = \{(l_i^c, l_i^t) \mid i \in I\} \oplus z_v \tag{7.3}$$

with O_p and O_v denoting the observation spaces of picker and AGV agents, re-

spectively. In short, pickers and AGVs observe the current and target locations of all agents, denoted $l_i^c \in L$ and $l_i^t \in L$ for agent i. Additionally, pickers observe all orders z_v of AGVs $v \in V$ while AGVs only observe their own order.

Pickers are rewarded +0.1 for picking an item onto an AGV. Similarly, AGVs are rewarded +0.1 for receiving a picked item with an additional reward of +0.1 for delivering the order. To incentivise efficient and fast deliveries, all agents receive a small negative reward of -0.01 per time step.

7.2.2 TA-RWARE Goods-to-Person Simulator

Our second environment is an open-source simulator named TA-RWARE visualised in Figure 7.1 (right). TA-RWARE models the GTP paradigm and extends the previously introduced multi-robot warehouse (RWARE) environment (Section 4.1.5) with heterogeneous agents (AGVs and picker robots) and a high-level action space where agents select target locations as actions. The execution of agents moving towards their desired location is achieved using a fixed path planning algorithm. In this environment, AGVs travel to a single warehouse location to retrieve a storage medium containing items, transferred to the AGV by a picking robot. The storage mediums are then delivered to one of multiple delivery locations which represents a human picking station. Human pickers are not modelled in our simulator since they would be placed outside of the bounds of the system at the delivery locations. As such, when we refer to a picker in the context of this simulator, we are referring to a picking robot that lifts storage mediums (such as totes or boxes) onto the AGV.

Given the shared order profile, all agents observe the current and target locations of every agent, the carrying and loading states of all AGVs, as well as whether carried items are currently requested, and a map indicating which storage locations are empty, contain unrequested and requested shelves:

$$\begin{aligned} O = O_p = O_v = &\{(l_i^c, l_i^t) \mid i \in I\} \\ &\oplus \{(\text{carrying}_v, \text{requested}_v, \text{loading}_v) \mid v \in V\} \\ &\oplus \{\text{occupied}_l, \text{requested}_l\} \mid l \in L_{item}\} \end{aligned} \tag{7.4}$$

Pickers are rewarded +0.1 for loading or unloading a storage medium onto an AGV. AGVs are rewarded +1 for delivering storage mediums, and all agents receive a negative reward of -0.01 per time step.

7.3 Methodology

In this section, we will outline two techniques to address the scalability and exploration challenges that arise due to the large action space in our warehouse simulators: (1) action masking and (2) a hierarchical policy structure for MARL.

7.3.1 Action masking

To reduce the effective action space of each agent and, thus, simplify learning, we mask out actions that are considered sub-optimal by adjusting logits (Huang and Ontañón, 2022). For instance, in PTG, one immediate observation is that whilst fulfilling an order z, AGVs should only move to locations $l \in L_{item}$ within the warehouse that contain requested items in z. Typically, this number is significantly smaller than the total number of item locations, i.e. $|A_v| \leq |z| \ll |L|$. Below, we detail action masking approaches for both studied settings and corresponding simulators that are then applied to varying warehouses within these settings.

For the Dematic PTG simulator (Section 7.2.1), we define order-specific action masks for AGVs that remove item locations that are not part of the current order of a particular AGV from the action space. Pickers have a shared action mask, that enables them to choose the target item location of any AGV to enable them to coordinate with AGVs. Lastly, pickers cannot take actions to choose locations that other pickers have chosen and are in transit to.

For the TA-RWARE GTP simulator (Section 7.2.2), we define a similar action mask for AGVs, where they can only choose locations that are requested storage medium locations or delivery locations. For pickers, we follow the same masking scheme as in the PTG paradigm, where pickers can travel to load and unload from current AGV target locations that are not already serviced by other pickers.

We note that action masking simplifies the decision-making problem by limiting the pool of available actions, but also introduces bias in the generated policies by limiting their expressiveness. For example, under our proposed action masking, picker agents are unable to pre-emptively move to locations to wait for AGVs that might move there in the future. Instead, they have to wait for a AGV to start moving to a location before they are able to choose it as a target. While the proposed action masking aims to minimise bias, we leave their complete exception from our large action space training regime to future work.

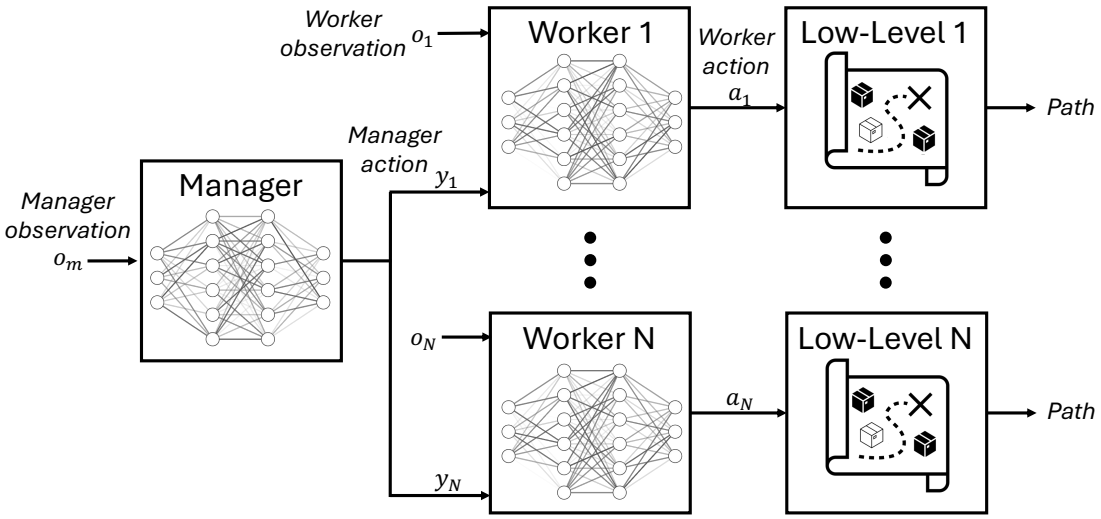


Figure 7.2: 3-layer manager and worker agent hierarchy. A manager agent observes global information about the warehouse state o_m, and assigns a target zone in the warehouse to each worker agent. Worker agents receive local observations o_i and the assigned target zone from the manager y_i, and select an item location from the assigned target zone given by a_i. A low-level controller then navigates the worker to the selected item location.

7.3.2 Hierarchical MARL for Order-Picking

By employing a hierarchical model, we further reduce the complexity of the action space and improve the handling of the differing termination durations for actions. We introduce a 3-layer adaptation of feudal multi-agent hierarchies (FMH) (Ahilan and Dayan, 2019), shown in Figure 7.2, that involves the introduction of a manager agent that produces goals for worker agents to satisfy. In contrast to FMH, manager goals do not affect worker reward functions, but instead the goals partition the worker action spaces, and worker agents do not execute primitive actions in the environment, instead delegating their decisions to lower-level controllers. The manager goals divide the locations within the warehouse into a set of disjoint zones Y, formally $L = \bigcup_{y \in Y} y$. The manager's action space consists of a choice from the set of zones Y for each agent $i \in I$, given by $A_m = Y_{|I|}$. Given assigned zone y_i to worker agent i, its policy π_i selects a new target location $l_i^t \in y_i$ within the assigned zone. This decomposition greatly reduces the effective action space of each agent's policy that is now bounded by $\max_{y \in Y} |y| \ll |L|$. Once the target location of each worker agent is determined, a lower-level controller calculates the shortest path from its current location and execute the necessary

sequence of primitive actions (we use the A* algorithm (Hart et al., 1968) in our experiments).

To be able to decide on effective assignment of worker agents to zones, the manager receives global information about the state of the warehouse. Within the Dematic PTG simulator, the manager, observes the current and target locations of all worker agents as well as the orders of all AGVs:

$$O_m = \{(l_i^c, l_i^t) \mid i \in I\} \oplus \{z_v \mid v \in V\} \tag{7.5}$$

Within the TA-RWARE GTP simulator, the manager observation space is identical to the observation space of worker agents ($O_m = O$) as defined in Equation 7.4.

All worker agents and the manager are jointly trained with MARL. At every time step, the manager receives a reward r_m^t equivalent the sum of rewards received by non-busy workers after each intermediate step $\tau \in [t, t + k_i]$, where k_i represents the number of steps taken by worker i before reaching the goal:

$$r_m^t = \sum_{i \in I} r_i^{t:t+k_i}, \text{ with} \tag{7.6}$$

$$r_i^{t:t+k_i} = \begin{cases} \sum_{\tau=t}^{t+k_i} r_i^\tau, & \text{if } i \text{ received a goal at } t \\ 0, & \text{otherwise} \end{cases} \tag{7.7}$$

7.4 Empirical Evaluation

7.4.1 Algorithms

We compare several MARL baselines, with and without our hierarchical manager agent, with established heuristics commonly used in warehouse logistics solutions. We outline details about all evaluated algorithms below.

Heuristic Solutions Two established industry heuristics used by Dematic for order-picking under the PTG paradigm are *follow me* (FM) and *pick, don't move* (PDM). These heuristics are similar to the strategies described by Azadeh et al. (2023) as No Zoning and Progressive Zoning. For the GTP paradigm, we define a third heuristic which we call *closest task assignment* (CTA).

Follow me (FM): Multiple AGVs are assigned to each picker to form a group and will follow them through the warehouse. Each AGV's order is concatenated and the travelling salesman problem (TSP) solution is generated to determine the

order in which the items will be picked. The TSP path minimises the distance of each group of workers with the constraint that they stay together while orders are not yet completed. FM minimises idle time for pickers, as it ensures that they are always travelling or picking, but can also lead to more travelling of pickers than needed.

Pick, don't move (PDM): Pickers are allocated to zones, for example with each picker being assigned a particular aisle within the warehouse, and meet AGVs that travel into their designated zone to pick items for them. The AGVs travel to all item locations in their current order using a TSP solution. Pickers prioritise service of AGVs by the relative proximity of the AGV and picker to the target locations. PDM minimises travel distance for pickers but can also result in under-utilisation of pickers in case there are few items within current orders in their operating zones.

Closest task assignment (CTA): AGVs travel to single storage locations and deliver storage mediums from those storage locations to a plurality of delivery locations. Storage mediums that need to be picked are assigned to the closest AGV which takes the storage medium to the closest delivery location. Once delivered, the AGV then returns the storage medium to the closest empty shelf location. *Closest* in this context refers to the minimum distance path found by the A^* algorithm (Hart et al., 1968). Pickers stick to allocated zones (similar to PDM), but prioritise AGVs in a first-in-first-out (FIFO) queue according to the order in which AGVs were assigned to pick or drop an item within their zone. Similarly to PDM, CTA minimises travel distance for pickers but may also result in picker under-utilisation.

MARL Solutions Motivated by the performance of MARL algorithms in the RWARE environment (Chapter 4), we focus our evaluation on on-policy actor-critic methods and do not evaluate off-policy value-based methods. We evaluate several MARL algorithms based on independent advantage actor-critic (IA2C) (Mnih et al., 2016; Christianos et al., 2020; Papoudakis et al., 2021) that differ in their network- and data-sharing mechanisms. More specifically, we consider IA2C without parameter sharing, SNAC, and SEAC, as introduced in Chapter 5, as well as their hierarchical counterparts HIA2C, HSNAC, and HSEAC with the manager agent as outlined in Section 7.3.2.

The policy and value network of the manager are given by a multi-headed neu-

Table 7.1: Warehouse configurations for the Dematic PTG simulator (top) and the TA-RWARE GTP simulator (bottom) considered in the evaluation.

		Small	Medium	Large	Disjoint
Dematic PTG	Aisles	2	10	22	12 + 12
	Item Locations $\|L_{item}\|$	200	400	1276	1392
	Partitions $\|Y\|$	4	10	22	24
	Pickers $\|P\|$	4	6	8	4
	AGVs $\|V\|$	8	12	16	16
	Avg. order-lines per order $\mathbb{E}(\|z_v\|)$	5	5	5	2
	Orders $\|Z\|$	80	80	80	80
TA-RWARE GTP	Rack Rows	2		4	
	Rack Columns	5		7	
	Column Length	8		8	
	Column Width	2		2	
	Item Locations $\|L_{item}\|$	160		448	
	Partitions $\|Y\|$	10		28	
	Delivery Locations $\|L_{delivery}\|$	10		14	
	Pickers $\|P\|$	4		7	
	AGVs $\|V\|$	8		14	

ral network comprising of three fully-connected layers consisting of 128 neurons each with ReLU activations, and a policy and value head for each worker agent. Each worker agent is parameterised by a policy and critic network represented by two fully connected layers of 64 neurons with ReLU activations. We use the same algorithm hyperparameter values in all MARL algorithms and experiments. The Adam optimiser (Kingma and Ba, 2015) is used with a learning rate of 0.0003 and epsilon of 0.001, target networks are updated every 100 steps, and generalised advantage estimation (GAE) (Schulman et al., 2016) is used to compute return estimates with $\lambda = 0.96$. The discount factor is set to 0.99 for all agents. In all our experiments, we train agents for 10,000 episodes.

Table 7.2: Performance comparisons between heuristics and MARL algorithms, showing mean ± 95% confidence intervals of pick rate in order-lines per hour.

	Dematic Simulator (PTG)				TA-RWARE (GTP)	
	Small	Medium	Large	Disjoint	Small	Large
FM	901.3 ± 1.9	1,098.1 ± 3.8	1,230.2 ± 5.1	568.4 ± 1.7	–	–
PDM	783.6 ± 2.8	982.2 ± 4.0	1,123.9 ± 4.9	677.4 ± 2.1	–	–
CTA	–	–	–	–	52.7 ± 0.9	67.1 ± 0.8
IA2C	**1,053.0 ± 2.8**	1,206.4 ± 4.2	1,263.9 ± 5.8	733.2 ± 2.7	65.2 ± 0.5	80.4 ± 0.6
SNAC	990.9 ± 2.8	1,142.7 ± 4.3	1,235.0 ± 5.7	688.7 ± 2.7	60.8 ± 0.7	72.1 ± 0.9
SEAC	1,019.7 ± 2.9	1,185.1 ± 5.1	1,262.9 ± 5.7	739.8 ± 2.4	64.8 ± 0.4	82.2 ± 0.5
HIA2C (ours)	1,025.9 ± 4.3	1,232.1 ± 4.8	1,354.2 ± 5.9	794.1 ± 2.7	**66.7 ± 0.3**	**86.0 ± 0.5**
HSNAC (ours)	1,030.8 ± 3.8	1,232.8 ± 5.1	1,363.8 ± 6.0	796.9 ± 2.4	66.0 ± 0.7	85.0 ± 0.5
HSEAC (ours)	1,028.2 ± 3.9	**1,242.1 ± 5.0**	**1,370.9 ± 5.7**	**803.5 ± 2.6**	64.6 ± 0.4	84.8 ± 0.6

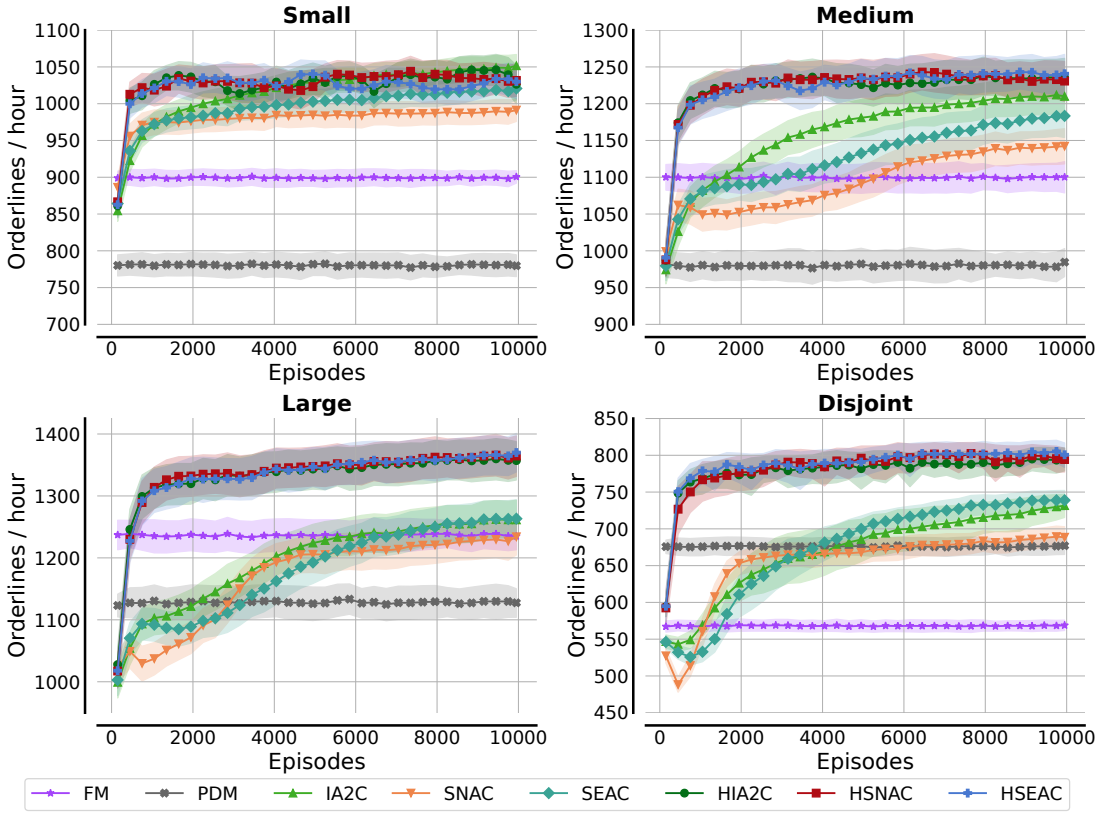
7.4.2 Environment Details

We evaluate the algorithms in four different PTG environment configurations and two different GTP environment configurations based on Dematic customer warehouse profiles. For the PTG warehouse configurations, the Large configuration is shown in Figure 7.1 (Left), with Small and Medium being smaller versions. Disjoint is a warehouse separated into two sub-warehouses joined by a passage. Such warehouse configurations might occur in practice when storing different types of goods with varying storage criteria, such as regular and frozen goods. All details about the configurations of both simulators are detailed in Table 7.1. We use pick rate measured in order-lines per hour as our primary performance measure, indicating the average frequency of picks in each episode.

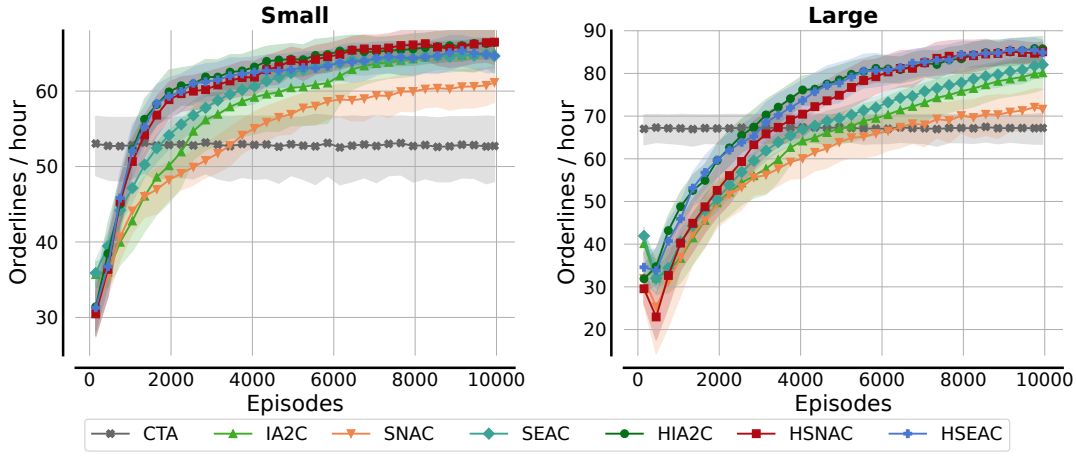
7.4.3 Results

Figure 7.3a shows the pick rate for HIA2C, HSNAC, HSEAC and all baselines across training in PTG. While different warehouse configurations can favour one heuristic or the other (e.g. FM in Large, PDM in Disjoint), we observe that the hierarchical algorithms achieve significantly higher pick rates than these two heuristics independent of the warehouse setting. Comparing the hierarchical versions against the original algorithms (i.e. HIA2C to IA2C, HSNAC to SNAC, HSEAC to SEAC) demonstrates the advantage of the hierarchical architecture, with all hierarchical versions converging significantly faster, especially as the complexity of the warehouse increases.

We perform a similar analysis for the GTP paradigm, shown in Figure 7.3b,



(a) Dematic PTG simulator



(b) TA-RWARE GTP simulator

Figure 7.3: Average pick rate (order-lines per hour) in (a) Dematic PTG simulator and (b) TA-RWARE GTP simulator tasks with heuristics FM/ PDM/ CTA and MARL algorithms IA2C, SNAC, SEAC, HIA2C (ours), HSNAC (ours), HSEAC (ours). Shading indicates the 95% stratified bootstrap confidence intervals (Agarwal et al., 2021) across five seeds. Average window smoothed over 300 episodes.

contrasting the pick rates achieved by the proposed methods against the CTA heuristic and the same list of MARL baselines. All MARL algorithms except SNAC surpass the heuristic by the end of training in the Small GTP warehouse, while this remains true only for the hierarchical methods in the Large GTP warehouse. Similar to our results in Chapter 5, the comparably poor performance of SNAC can be explained with all worker agents sharing identical policies, which can lead to frequent delays due to agent collisions or deadlock situations. In contrast, HSNAC avoids such conflicts with the hierarchical manager effectively coordinating workers by conditioning their policies on assigned target goals to distribute workers across the warehouse, further highlighting the effectiveness of such a hierarchical policy decomposition (see Figure 7.3b, left). We omit the training curve for SNAC in the plot for the Large configuration as it achieves similar low performance as in Small. Finally, analogous to the PTG environment, MARL algorithms with hierarchical models outperform the non-hierarchical baselines in final performance and rate of convergence.

We perform a similar analysis for GTP, shown in Figure 7.3b, contrasting the pick rates achieved by the proposed methods against the CTA heuristic and the same MARL algorithms. All MARL methods surpass the heuristic for both sizes of the warehouse, with SNAC achieving the lowest pick rates by the end of training. The sub-par performance of SNAC can be explained with all worker agents sharing identical policies, which can lead to frequent delays due to agent collisions or deadlock situations among workers, similar to the downsides of parameter sharing discussed in Chapter 5. The advantage of the hierarchical architecture can be observed when comparing HSNAC to SNAC. HSNAC avoids deadlocks through the manager, by conditioning the worker policies on assigned target goals to distribute them across the warehouse. Finally, analogous to the PTG environment, all MARL algorithms with hierarchical models outperform the non-hierarchical counterparts in sample efficiency. This is especially noticeable in the Large configuration, highlighting again the scaling benefits of the hierarchical models.

In Table 7.2, we compare the average pick rates achieved by the algorithms during the final 50 training episodes. In the Small PTG configuration, IA2C achieves on-par pick rates with the hierarchical algorithms, a difference of only 2.2%, while surpassing FM and PDM by 16.8% and 34.4%, respectively. We attribute these results to the relatively low difficulty of the task where the hierarchical approach does not offer substantial benefits. As we scale up warehouse

complexity, the hierarchical algorithms start reaching the highest overall pick rates. In Medium, HSEAC exceeds FM and PDM by 13.1% and 26.5%. In Large, HSEAC exceeds FM and PDM by 11.4% and 22.0%. In Disjoint, HSEAC exceeds FM and PDM by 41.3% and 18.6%. In GTP, the hierarchical models achieve the highest pick rates, HIA2C surpassing the CTA heuristic by 26.6% in the Small configuration and by 28.2% in the Large configuration.

7.5 Related Work

Below we discuss related work that tackles similar order-picking problems and warehouses, or employs similar MARL techniques.

AGV-Assisted Order-Picking Azadeh et al. (2023) model the order-picking problem as a queuing network and explore the impact of different zoning strategies (no zoning and progressive zoning). They then further extend their method by representing it as a MDP and consider dynamic switching based on the order-profile using dynamic programming. Löffler et al. (2022) consider an AGV-assisted picker and provide an exact polynomial time routing algorithm for single-block parallel-aisle warehouses. Žulj et al. (2022) consider a warehouse partitioned into disjoint picking zones, where AGVs meet pickers at handover zones to transport the orders back to the depot. They propose a heuristic for effective order-batching to reduce tardiness. Our work differs from these works as it does not place any restrictions on how workers may cooperate and, to the best of our knowledge, is the first application of MARL to AGV-assisted order-picking.

Multi-Agent Path Finding Reciprocal n-body collision avoidance (van den Berg et al., 2011), or *multi-agent path finding* (MAPF), aims to build systems where teams of agents can traverse the environment to reach individually allocated targets while following optimal trajectories and avoiding collisions. MARL quickly became a promising tool for solving path-finding coordination problems, being adopted to boost scalability (PRIMAL (Sartoretti et al., 2018) and PRIMAL2 (Damani et al., 2020)), enable communication (Li et al., 2020) or facilitate implicit priority learning (Li et al., 2022). *Lifelong-MAPF* (Li et al., 2021b) (LMAPF) extends MAPF, as new target locations are automatically assigned to the agents upon reaching their previous goal location. Li et al. (2021b) utilise

a centralised but bounded planner that minimises re-planning costs accumulated when receiving new targets while also showing an increase in responsiveness and adaptability of the proposed solution. Greshler et al. (2021) introduce cooperative multi-agent path finding, which is applicable within our domain but requires explicit specification of the workers that are required to cooperate and does not allow for optimisation over extended periods of time. While we note the relevance of MAPF and especially LMAPF algorithms for the complete warehouse optimisation problem, we draw a distinction between the path-finding and order-picking settings and highlight their complementary nature. In MAPF, task assignment processes are assumed to be external to the path planning method, in contrast to our setting where task assignment represents the main focus and path-finding is achieved through robust planning methods.

Multi-Agent Pick-up and Delivery Problem Multi-agent pick-up and delivery (MAPD) problems (Ma et al., 2017) consider a set of agents that are sequentially assigned tasks in the form of target pick-up and delivery locations, and then travel to their allocated locations while avoiding collisions with others. The objective is to minimise the time duration required for task completion, which may be further broken down into two sub-problems: multi-agent task assignment (TA) and multi-agent path-finding (MAPF) (Ma et al., 2017; Xu et al., 2022). However, MAPD approaches typically rely on hand-engineered heuristics that assume homogeneity among the agent architectures (Ma et al., 2017), with variance in agent velocities being the general extent for agent diversity (van den Berg et al., 2011). This assumption is a drastic simplification of the complex coordination problem of *order-picking* systems and reduces the cooperation among the agents to collision avoidance, which is solely tackled through the MAPF module. The decoupling of workers within our approach introduces complex interdependencies between the paths of different worker types that significantly complicates the problem. In our setting, *pickers* and *AGVs* need to synchronise and coordinate to meet at certain item locations at matching times in order to complete pick-ups. Furthermore, coordination among agents of the same type is required to minimise situations in which multiple pickers move to the same item location which can cause significant delays and conflicts. In contrast to these hand-crafted heuristic-based solutions, Weyns et al. (2008) proposed an automated decision-making process that models decentralised control for multi-robot warehouse coordination problems using free-

flow trees (Tyrrell, 1993). Alternatively, Claes et al. (2017) leverage distributed planning based on Monte Carlo tree search in similar problems to derive scalable decision-making policies. In contrast, our work leverages multi-agent reinforcement learning to learn decision-making policies completely by trial-and-error in a simulation environment.

Multi-Agent Reinforcement Learning MARL algorithms are designed to train coordinated agent policies for multiple autonomous agents, and have received much attention in recent years with the introduction of deep learning techniques into MARL (Albrecht et al., 2024; Papoudakis et al., 2019). MARL has previously seen application to various warehousing problems, including SEAC to homogeneous GTP systems (Chapter 5), and a deep Q-network variant for sortation control (Kim et al., 2020). For the specific complexities of the order-picking problem, we consider methods at the intersection of MARL and hierarchical RL to enable action space decomposition and temporal abstraction. This combination has been studied by Xiao et al. (2020) who derive MARL algorithms for macro-actions under partial observability, and Ahilan and Dayan (2019) who propose feudal multi-agent hierarchies (FMH) that extends feudal RL (Dayan and Hinton, 1992) to the cooperative MARL domain. We introduce a 3-layer adaptation of FMH and apply it to a partially observable stochastic game with individual agent reward functions.

7.6 Conclusion

In this chapter, we considered the real-world problem of order-picking for automation in warehouses. We formalised two paradigms of the order-picking problem, person-to-goods (PTG) and goods-to-person (GTP), and presented simulators that model both of these paradigms, Dematic PTG and TA-RWARE GTP, the latter of which is made available open-source. Motivated by the problem of large action spaces in these problem settings, we proposed a twofold solution that consists of a hierarchical MARL architecture with a manager agent that assigns individual goals to worker agents inside the warehouse, and an action masking process that filters out sub-optimal actions. The policies of the manager agent and all worker agents are jointly trained using MARL. We integrated our approach into several commonly used actor-critic MARL algorithms that differ in

their mechanisms of sharing data and parameters. Our results indicate that the hierarchical architecture is essential in achieving high performance in these settings, in particular for large warehouses with many possible locations to move to. Besides making learning more efficient and scalable, the hierarchical decomposition also provides a high-level central coordination mechanism, as goals for all agents are selected by a single manager policy. Overall, our results showed that MARL solutions can outperform human-engineered and well-established industry heuristics as well as commonly used MARL baselines in various warehouse configurations across both PTG and GTP paradigms. These results suggest that MARL can derive effective solutions for the order-picking problem and indicate the promise of MARL for warehouse logistics optimisation.

Chapter 8

Conclusion

In this thesis, we have explored the problem of exploration in single-agent and multi-agent reinforcement learning and have proposed novel algorithms to address important challenges in this area. Below, we re-iterate the research questions we posed in Chapter 1 and summarise the contributions of this thesis in answering these questions.

Chapter 3 focused on the setting of intrinsically motivated single-agent reinforcement learning for exploration. Within this setting, the balance of intrinsic rewards for exploration and extrinsic rewards for exploitation is important for the agent to efficiently learn and eventually solve the task. This posed the following research question:

1. *How can intrinsically motivated reinforcement learning agents effectively balance exploration and exploitation in the single-agent setting?*

We discussed a plethora of challenges introduced by naively combining intrinsic and extrinsic rewards. Motivated by these challenges, we proposed the *decoupled reinforcement learning* (DeRL) framework, in which the agent learns two separate policies for exploration and exploitation. This decoupling mechanism led to more sample efficient training and less sensitivity to hyperparameters that determine the balance of exploration and exploitation in several exploration-focused tasks. We believe that the latter result is particularly important for the practical application of reinforcement learning algorithms and exploration techniques, which can often be highly sensitive to hyperparameters. Our research further discovered that distribution shift between decoupled exploration and exploitation policies represents a significant challenge and motivates further research in this area. We also note that while DeRL has been demonstrated to be less sensitive

to hyperparameters, the selection of hyperparameters can still have a significant impact on the performance of the algorithm, and further research is needed to understand the impact of decoupling exploration and exploitation in more complex environments.

Subsequent chapters focused on the setting of multi-agent reinforcement learning. First, Chapter 4 presented a comprehensive benchmark of nine deep MARL algorithms across 25 common-reward environments. To the best of our knowledge, this was the first benchmark that evaluated a wide range of deep MARL algorithms across a diverse set of environments with a consistent evaluation protocol. The reported results shed light on the strengths and weaknesses of different MARL algorithms and identified opportunities for future research. Most notably, the benchmark revealed the inefficiency of existing MARL algorithms in solving problems with highly sparse rewards. In addition to the reported results, we have made the benchmark codebase publicly available to facilitate future research in this area which since has been adopted for MARL research (e.g., Leroy et al., 2023; Torbati et al., 2023).

Building on the insights gained from the benchmark, Chapter 5 studied the problem of exploration and efficient learning in multi-agent environments with sparse rewards and focused on the second research question:

> *2. How can multiple learning agents share experiences with each other in order to explore and learn more efficiently?*

We proposed the *shared experience actor-critic* (SEAC) algorithm that allows agents to share experiences with each other, enabling them to learn different and specialised policies in a more efficient manner. More specifically, SEAC shares experiences of multiple agents with each other and uses simple off-policy correction to account for the differences in data distributions of agents. We compared SEAC to other methods of data sharing, independent learning, and more sophisticated CTDE algorithms and found SEAC to significantly outperform all baselines in both sample efficiency and converged performance despite its simplicity. These results indicated the opportunity that lies in considering the distributed data gathered by all agents for efficient MARL training, and the benefits that can by obtained by enabling agents to learn similar but not identical policies. We also demonstrated that similar experience sharing mechanisms can be applied to off-policy algorithms, which has since been extended in subsequent work. The experience sharing proposed in SEAC relies on the assumption of symmetry of the

transition and reward functions across agents. We observed that this assumption holds across a plethora of multi-agent problems but it remains an open question how to relax this assumption and extend experience sharing to more general multi-agent problems.

Chapter 6 focused on the problem of exploration in value-based MARL algorithms that have previously often resorted to simple exploration strategies such as ϵ-greedy policies. Motivated by the unique challenge in MARL for agents to learn about their interactions with each other, we studied the following research question:

> *3. How can multiple learning agents identify states and actions with the potential for cooperation and guide their exploration towards these states and actions?*

To incentivise agents to explore such states and actions with the potential for interactions across agents, we proposed the *ensemble value functions for multi-agent exploration* (EMAX) framework that can extend existing value-based MARL algorithms with improved exploration towards cooperative states and actions, and more robust training. We demonstrated the benefits of EMAX first as an extension of independent value-based learning in multiple environments with individual rewards for all agents, before extending value decomposition algorithms for cooperative MARL. The results showed that EMAX can significantly improve the sample efficiency of value-based MARL algorithms in a range of mixed-objective and cooperative environments, and provided ablations to understand the individual contributions of the components of the EMAX framework. We note that EMAX and our investigation is limited to value-based MARL algorithms and has been studied in cooperative and mixed-objective environments. It remains an open question how to extend EMAX to more general multi-agent problems and to policy-gradient algorithms.

Lastly, Chapter 7 presented a case study for the application of MARL to warehouse automation. We formulated the problem of warehouse automation as a multi-agent reinforcement learning problem, and described two simulators that have been developed to study MARL in this setting. Motivated by the challenge of scaling MARL solutions to warehouses of realistic scale, we proposed a hierarchical decomposition of the problem and an action masking procedure that can be used to reduce the effective action space agents need to consider. We demonstrated the benefits of this approach in a simulated warehouse environment,

and showed that the proposed algorithm can significantly improve the efficiency of warehouse operations compared to baseline MARL algorithms and industry-standard heuristics. These results demonstrated the promise of MARL for real-world applications, and the potential for future research in this area.

8.1 Directions for Future Work

This thesis provides a foundation for research focused on efficient exploration in single-agent and multi-agent reinforcement learning. We conclude this thesis with several directions for future research that build on the contributions of this thesis in the hope that other researchers will be inspired to further explore these ideas.

DeRL demonstrates that there are notable benefits to considering the different requirements of exploration and exploitation in reinforcement learning. Further research has already built upon this work to consider the application of DeRL to more complex environments (Chen et al., 2022) and extended it with optimisation of dependent hyperparameters using constrained optimisation. They also found that the optimisation can become more stable when training the exploitation policy on both on-policy data of the policy itself as well as off-policy data of the exploration policy. This further supports the discussion in our work that the distribution shift between exploration and exploitation policies can be a significant challenge for decoupling exploration and exploitation. Further ideas from offline RL may be integrated to limit this distribution shift and improve the stability of training the exploitation policy (McInroe et al., 2024). Furthermore, the application of DeRL to MARL could be an interesting to explore, as the decoupling of exploration and exploitation could also lead to more efficient and robust exploration strategies in this setting.

Experience sharing with SEAC relies on an assumption of symmetry of the transition function and reward functions across agents. As demonstrated in Chapter 5, this assumption holds in a variety of multi-agent settings and can enable significant improvements in sample efficiency and converged performance. We presented evaluation results in a variety of mixed-objective and cooperative multi-agent tasks, but have not considered competitive settings in which the symmetry assumption might also hold. For example in various two-player zero-sum games, such as chess, the symmetry assumption might hold and experience sharing may be applied and provide significant benefits. This setting might also further es-

tablish the similarities of our experience sharing paradigm and the concept of self-play commonly applied in two-player zero-sum games. Furthermore, future research could consider to relax SEAC's symmetry assumption to see whether the experience sharing idea can be applicable to more general multi-agent problems. Lastly, this thesis considered the application of experience sharing primarily to independent actor-critic algorithms, and presented preliminary experiments with independent value-based MARL algorithms. It would be interesting to consider the application of experience sharing to further algorithms such as multi-agent policy gradient algorithms with centralised critics to understand the benefits of experience sharing for a broader range of MARL algorithms.

The EMAX framework presented in this thesis provides a novel approach to exploration that provides significant benefits to sample efficiency and optimisation robustness. However, EMAX is currently limited to value-based MARL algorithms and was largely studied in cooperative environments. Future work could consider evaluation in competitive two-player games (Pérolat et al., 2022; McAleer et al., 2023; Sokota et al., 2023) and extend EMAX to multi-agent actor-critic algorithms such as MAPPO and IPPO that have been shown to be effective in cooperative MARL (Papoudakis et al., 2021; Yu et al., 2022). In this case, ensembles of value functions and policies could be trained for each agent, with similar target computation for the critic, potentially advantage estimation across the ensemble, and UCB policies across actors to guide exploration. A further limitation of the current EMAX approach is the considerable computation cost of training ensembles of value functions. Future work could investigate the distribution of EMAX ensemble computation across multiple compute devices to reduce training and inference time. Alternatively, or in addition to such distributed computation, future work could investigate the application of techniques to approximate costly ensembles using a single network through hypernetworks (Dwaracherla et al., 2020) and latent-conditioned models (Shen and How, 2023). These techniques have the potential to significantly reduce the computational cost of EMAX and, thereby, making it more widely accessible.

Orthogonal to the research presented in this thesis, we believe that there is a significant opportunity in the current push towards larger and more generalisable machine learning models for exploration in RL and MARL. First, if we are able to train generalisable decision-making agents that are capable of transferring to novel tasks with limited or no further training required, then training overall may

become significantly more efficient since the same agents can be trained once and applied to many tasks that previously necessitated further training. Second, by leveraging larger and more generalisable models, RL and MARL training can leverage prior knowledge about decision making and already learned tasks to inform exploration. For example, agents that have general understanding of common concepts in the world might have sensible hypotheses about how a novel decision-making problem might work. They might infer that ladders likely enable to agent to climb up, or that a key is likely to open a door. Such prior knowledge is currently absent from the majority of RL agents that are typically trained from scratch in every task, and represents a great opportunity to enable a true step-change in the efficiency of learning how to act. In the past few years, several research directions have emerged that follow this line of thought. Literature on open-ended learning aims to train agents that continually learn and improve their capabilities over time. One paradigm to achieve such continual learning is unsupervised environment design that defines an automated curriculum of diverse tasks that cater to the current capabilities of the agent and, thereby, enables the agent to learn more efficiently and continue to improve (Dennis et al., 2020; Jiang et al., 2021; Parker-Holder et al., 2022). By combining such automated curriculum with large transformer models and large-scale computational resources, Team et al. (2023a) trained an agent that can solve a wide range of challenging tasks and demonstrates an impressive ability to adapt to new tasks from few demonstrations or trials. Another research direction directly leverages large language models and their capability to generalise as agents in complex decision-making tasks with no or limited fine-tuning (Wang et al., 2023; Team et al., 2024).

The rate of progress in the field of RL and MARL in the past decade has been remarkable, and we believe that the field is poised to make further significant strides in the coming years. We hope that the research presented in this thesis will inspire future work and be built upon to obtain more efficient exploration approaches in RL and MARL, and that the field will enable the development of decision-making agents that can operate in and solve real-world problems to benefit society at large.

Appendix A

Decoupled Reinforcement Learning

A.1 Hyperparameter Optimisation

A gridsearch was conducted to identify the best hyperparameter configuration for each algorithm in both environments. In order to keep the search feasible, a hyperparameter search for DeepSea was conducted in DeepSea $N = 20$ and the same identified configuration was applied in all other DeepSea tasks. Similarly, for the Hallway environment a hyperparameter search was conducted in Hallway $N_l = N_r = 10$. Every gridsearch configuration is evaluated using three different random seeds with results being averaged across all seeds.

First, the best hyperparameters for A2C and PPO baselines were identified by training each configuration in the respective training task of both environments with Count intrinsic rewards. We chose to conduct the hyperparameter search with intrinsic rewards as both baselines do perform poorly in most tasks without any intrinsic rewards which make different hyperparameter configurations hardly distinguishable. The Count intrinsic reward was chosen as it does not require any hyperparameter tuning in contrast to other intrinsic reward definitions. For all considered hyperparameter combinations, we conduct training in the same way as described in Section 3.3 with $\lambda = 1$. The best hyperparameter configuration is chosen as the one that leads to highest maximum evaluation returns. If multiple configurations reach the same maximum evaluation returns then the one with the highest average evaluation returns computed over all 100 evaluations throughout training is considered the best. Such latter metric considers achieved returns alongside sample efficiency. Identified hyperparameters for A2C and PPO in both environments can be found in Table A.1 and Table A.2. All considered values

of hyperparameters are listed with the best-identified combination highlighted in bold.

Table A.1: Hyperparameters for A2C baseline.

Hyperparameter	DeepSea	Hallway
Normalise observations	False, **True**	**False**, True
Normalise rewards	False, **True**	**False**, True
Learning rate	$3e^{-4}$, $\mathbf{1e^{-3}}$	$\mathbf{3e^{-4}}$, $1e^{-3}$
Nonlinearity	Tanh, **ReLU**	**Tanh**, ReLU
Maximum gradient norm	**0.5**, 10.0, 40.0	**0.5**, 10.0, 40.0
Entropy loss coefficient	$\mathbf{1e^{-4}}$, $3e^{-4}$, $7e^{-4}$, $1e^{-3}$	$\mathbf{1e^{-4}}$, $3e^{-4}$, $7e^{-4}$, $1e^{-3}$
Actor architecture	**FC (64, 64)**	**FC (64, 64)**
Critic architecture	**FC (64, 64)**	**FC (64, 64)**
N-steps	**5**	**5**
Adam ϵ	**0.001**	**0.001**
Value loss coefficient	**0.5**	**0.5**

Table A.2: Hyperparameters for PPO baseline.

Hyperparameter	DeepSea	Hallway
Normalise observations	**False**, True	**False**, True
Normalise rewards	**False**, True	**False**, True
Learning rate	$3e^{-4}$, $\mathbf{1e^{-3}}$	$\mathbf{3e^{-4}}$, $1e^{-3}$
Nonlinearity	**Tanh**, ReLU	Tanh, **ReLU**
Maximum gradient norm	**0.5**, 10.0, 40.0	**0.5**, 10.0, 40.0
Entropy loss coefficient	$\mathbf{1e^{-4}}$, $3e^{-4}$, $7e^{-4}$, $1e^{-3}$	$1e^{-4}$, $3e^{-4}$, $\mathbf{7e^{-4}}$, $1e^{-3}$
Actor architecture	**FC (64, 64)**	**FC (64, 64)**
Critic architecture	**FC (64, 64)**	**FC (64, 64)**
N-steps	**10**	**10**
Adam ϵ	**0.001**	**0.001**
Value loss coefficient	**0.5**	**0.5**
Number of epochs	**10**	**10**
Number of minibatches	**4**	**4**
Clipping ratio	**0.1**	**0.1**
Clip value loss	**True**	**True**

Following the hyperparameter search of A2C and PPO with Count intrinsic rewards, a search over hyperparameters of all parameterised intrinsic reward def-

initions was conducted. The setup of the hyperparameter search is identical to the one described above and the best identified hyperparameter configuration for the baseline algorithms are used in the gridsearch for intrinsic rewards. Best identified hyperparameters and all considered hyperparameter configurations can be found in Tables A.3 to A.6.

Table A.3: Hyperparameters for Hash-Count.

	Hyperparameter	DeepSea	Hallway
A2C	Hash key dimensionality	**16**, 32, 64, 128	**16**, 32, 64, 128
PPO	Hash key dimensionality	**16**, 32, 64, 128	**16**, 32, 64, 128

Table A.4: Hyperparameters for ICM.

	Hyperparameter	DeepSea	Hallway
General	z architecture	**FC (64, 64)**	**FC (64, 64)**
	$z(s)$ dimensionality	**16**	**16**
	Forward prediction architecture	**FC (64)**	**FC (64)**
	Inverse prediction architecture	**FC (64)**	**FC (64)**
A2C	Learning rate	$1e^{-7}, 5e^{-7}, 1e^{-6}, 5e^{-6}, \mathbf{1e^{-5}}$	$1e^{-7}, 5e^{-7}, \mathbf{1e^{-6}}, 5e^{-6}, 1e^{-5}$
	Forward loss coefficient	$0.5, 1.0, \mathbf{5.0}, 10.0$	$0.5, 1.0, \mathbf{5.0}, 10.0$
	Inverse loss coefficient	$0.5, \mathbf{1.0}, 5.0, 10.0$	$\mathbf{0.5}, 1.0, 5.0, 10.0$
PPO	Learning rate	$1e^{-7}, 5e^{-7}, 1e^{-6}, 5e^{-6}, \mathbf{1e^{-5}}$	$1e^{-7}, 5e^{-7}, 1e^{-6}, 5e^{-6}, \mathbf{1e^{-5}}$
	Forward loss coefficient	$0.5, 1.0, \mathbf{5.0}, 10.0$	$\mathbf{0.5}, 1.0, 5.0, 10.0$
	Inverse loss coefficient	$0.5, \mathbf{1.0}, 5.0, 10.0$	$0.5, 1.0, 5.0, \mathbf{10.0}$

Table A.5: Hyperparameters for RND.

	Hyperparameter	DeepSea	Hallway
General	z architecture	**FC (64, 64)**	**FC (64, 64)**
	$z(s)$ dimensionality	**16**	**16**
A2C	Learning rate	$\mathbf{1e^{-7}}, 5e^{-7}, 1e^{-6}, 5e^{-6}, 1e^{-5}$	$1e^{-7}, 5e^{-7}, 1e^{-6}, 5e^{-6}, \mathbf{1e^{-5}}$
PPO	Learning rate	$\mathbf{1e^{-7}}, 5e^{-7}, 1e^{-6}, 5e^{-6}, 1e^{-5}$	$1e^{-7}, \mathbf{5e^{-7}}, 1e^{-6}, 5e^{-6}, 1e^{-5}$

Based on the aforementioned hyperparameters, a gridsearch for all DeRL algorithms, DeA2C, DePPO and DeDQN, was conducted using the best identified A2C and Count intrinsic rewards to train the exploration policy. Identified configurations are listed in Tables A.7 to A.9. Table A.10 shows hyperparameters for DeRL algorithms with ICM intrinsic rewards used to train the A2C exploration

Table A.6: Hyperparameters for RIDE.

	Hyperparameter	DeepSea	Hallway
General	z architecture	**FC (64, 64)**	**FC (64, 64)**
	$z(s)$ dimensionality	**16**	**16**
	Forward prediction architecture	**FC (64)**	**FC (64)**
	Inverse prediction architecture	**FC (64)**	**FC (64)**
	State count	**Count**	**Count**
A2C	Learning rate	$1e^{-7}, 5e^{-7}, 1e^{-6}, 5e^{-6}, \mathbf{1e^{-5}}$	$1e^{-7}, 5e^{-7}, 1e^{-6}, 5e^{-6}, \mathbf{1e^{-5}}$
	Forward loss coefficient	$\mathbf{0.5}, 1.0, 5.0, 10.0$	$0.5, 1.0, 5.0, \mathbf{10.0}$
	Inverse loss coefficient	$0.5, 1.0, 5.0, \mathbf{10.0}$	$\mathbf{0.5}, 1.0, 5.0, 10.0$
PPO	Learning rate	$1e^{-7}, 5e^{-7}, 1e^{-6}, \mathbf{5e^{-6}}, 1e^{-5}$	$\mathbf{1e^{-7}}, 5e^{-7}, 1e^{-6}, 5e^{-6}, 1e^{-5}$
	Forward loss coefficient	$0.5, 1.0, 5.0, \mathbf{10.0}$	$0.5, \mathbf{1.0}, 5.0, 10.0$
	Inverse loss coefficient	$0.5, \mathbf{1.0}, 5.0, 10.0$	$0.5, \mathbf{1.0}, 5.0, 10.0$

policy. The same general architecture is used for ICM when applied alongside DeRL as reported in Table A.4.

Table A.7: Hyperparameters for DeA2C.

Hyperparameter	DeepSea	Hallway
Importance sampling	**Default IS weights**, Retrace(λ)	Default IS weights, **Retrace(λ)**
Learning rate	$3e^{-4}, \mathbf{1e^{-3}}$	$\mathbf{3e^{-4}}, 1e^{-3}$
Nonlinearity	Tanh, **ReLU**	**Tanh**, ReLU
Maximum gradient norm	**0.5**	**0.5**
Entropy loss coefficient	$0.0, \mathbf{1e^{-6}}, 1e^{-5}, 1e^{-4}$	$0.0, 1e^{-6}, \mathbf{1e^{-5}}, 1e^{-4}$
Actor architecture	**FC (64, 64)**	**FC (64, 64)**
Critic architecture	**FC (64, 64)**	**FC (64, 64)**
N-steps	**5**	**5**
Adam ϵ	**0.001**	**0.001**
Value loss coefficient	**0.5**	**0.5**

Table A.8: Hyperparameters for DePPO.

Hyperparameter	DeepSea	Hallway
Importance sampling	**Default IS weights**	**Default IS weights**
Learning rate	$3e^{-4}$, $\mathbf{1e^{-3}}$	$\mathbf{3e^{-4}}$, $1e^{-3}$
Nonlinearity	Tanh, **ReLU**	Tanh, **ReLU**
Maximum gradient norm	**0.5**	**0.5**
Entropy loss coefficient	$0.0, 1e^{-6}, 1e^{-5}, \mathbf{1e^{-4}}$	$0.0, \mathbf{1e^{-6}}, 1e^{-5}, 1e^{-4}$
Actor architecture	**FC (64, 64)**	**FC (64, 64)**
Critic architecture	**FC (64, 64)**	**FC (64, 64)**
N-steps	**10**	**10**
Adam ϵ	**0.001**	**0.001**
Value loss coefficient	**0.5**	**0.5**
Number of epochs	**10**	**10**
Number of minibatches	**4**	**4**
Clipping ratio	**0.1**	**0.1**
Clip value loss	**True**	**True**

Table A.9: Hyperparameters for DeDQN.

Hyperparameter	DeepSea	Hallway
Learning rate	$1e^{-4}, 3e^{-4}, \mathbf{1e^{-3}}$	$\mathbf{1e^{-4}}, 3e^{-4}, 1e^{-3}$
Soft target update τ	**0.01**, 0.001	0.01, **0.001**
Batch size	128, **256**, 512	128, 256, **512**
N-steps	**5**	**5**
Nonlinearity	**Tanh**, ReLU	Tanh, **ReLU**
Replay buffer	**Default**, Prioritised	**Default**, Prioritised
Replay buffer capacity	**100,000**	**100,000**
Maximum gradient norm	**0.5**	**0.5**
DQN architecture	**FC (64, 64)**	**FC (64, 64)**
Adam ϵ	**0.001**	**0.001**

Table A.10: Hyperparameters for DeRL with ICM.

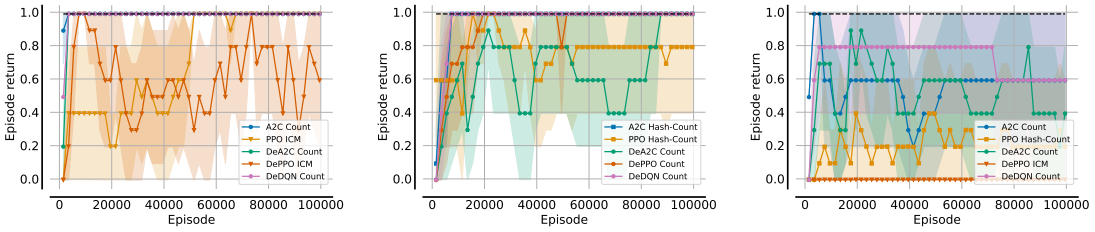
	Hyperparameter	DeepSea	Hallway
DeA2C	Learning rate	$1e^{-7}, 5e^{-7}, 1e^{-6}, 5e^{-6}, \mathbf{1e^{-5}}$	$1e^{-7}, 5e^{-7}, \mathbf{1e^{-6}}, 5e^{-6}, 1e^{-5}$
	Forward loss coefficient	$0.5, 1.0, \mathbf{5.0}, 10.0$	$0.5, 1.0, \mathbf{5.0}, 10.0$
	Inverse loss coefficient	$0.5, \mathbf{1.0}, 5.0, 10.0$	$\mathbf{0.5}, 1.0, 5.0, 10.0$
DePPO	Learning rate	$1e^{-7}, 5e^{-7}, 1e^{-6}, 5e^{-6}, \mathbf{1e^{-5}}$	$1e^{-7}, 5e^{-7}, 1e^{-6}, 5e^{-6}, \mathbf{1e^{-5}}$
	Forward loss coefficient	$0.5, 1.0, \mathbf{5.0}, 10.0$	$\mathbf{0.5}, 1.0, 5.0, 10.0$
	Inverse loss coefficient	$0.5, \mathbf{1.0}, 5.0, 10.0$	$0.5, 1.0, 5.0, \mathbf{10.0}$
DeDQN	Learning rate	$1e^{-7}, 5e^{-7}, 1e^{-6}, 5e^{-6}, \mathbf{1e^{-5}}$	$1e^{-7}, 5e^{-7}, 1e^{-6}, 5e^{-6}, \mathbf{1e^{-5}}$
	Forward loss coefficient	$0.5, 1.0, \mathbf{5.0}, 10.0$	$\mathbf{0.5}, 1.0, 5.0, 10.0$
	Inverse loss coefficient	$0.5, \mathbf{1.0}, 5.0, 10.0$	$0.5, 1.0, 5.0, \mathbf{10.0}$

A.2 Evaluation Results

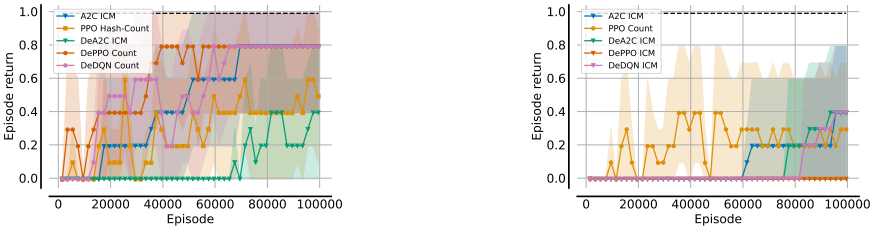
In this section, we provide tables containing maximum evaluation returns for all baselines and DeRL algorithms in every task. As described in Section 3.3, average returns and stratified bootstrap 95% confidence intervals are computed using 5,000 samples across five random seeds for the best identified hyperparameter configuration as reported in Appendix A.1. For maximum evaluation returns, we identify the single evaluation out of all 100 conducted evaluations with the maximum evaluation returns averaged across samples and report its value with the standard deviation across all samples. Within tables, the highest performing algorithms for each task are highlighted in bold together with every algorithm within a single standard deviation of the highest return. Besides these tables, we also provide learning curves for all baselines and DeRL algorithms in every evaluated task. In order to maintain visual clarity, we average two following evaluations (of the total of 100 evaluations throughout training) for a total of 50 plotted evaluation points for each algorithm.

Figure A.1 and Table A.11 show the learning curves over average evaluation returns and a table of maximum evaluation returns for all algorithms in DeepSea tasks for the best intrinsic reward per algorithm. Figure A.3 and Table A.12 show the same for Hallway tasks.

To compare evaluated algorithms and intrinsic rewards across all tasks, we refer to the tables of maximum evaluation returns as well as the learning curves in Figure A.2 for DeepSea, and Figures A.4 and A.5 for Hallway.



(a) DeepSea $N = 10$ (b) DeepSea $N = 14$ (c) DeepSea $N = 20$

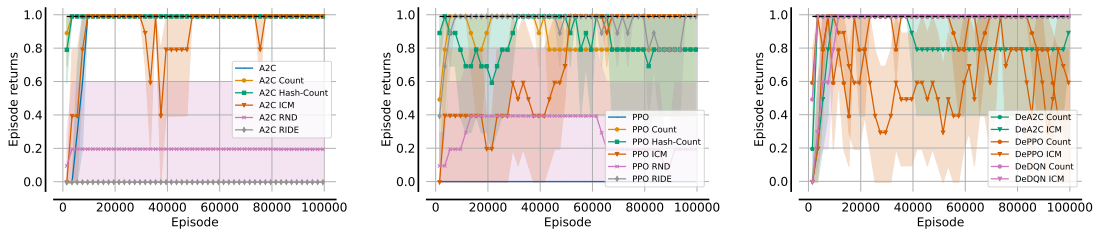


(d) DeepSea $N = 24$ (e) DeepSea $N = 30$

Figure A.1: Average evaluation returns and 95% confidence intervals for A2C, PPO and DeRL with the highest achieving intrinsic reward in all DeepSea tasks.

Table A.11: Maximum evaluation returns in the DeepSea environment with a single standard deviation.

Algorithm \ Task	DeepSea 10	DeepSea 14	DeepSea 20	DeepSea 24	DeepSea 30
A2C	**0.99**	0.00	0.00	0.00	0.00
A2C Count	**0.99**	**0.99**	**0.99**	**0.79 ± 0.40**	−0.01
A2C Hash-Count	**0.99**	**0.99**	0.64 ± 0.48	**0.59 ± 0.49**	0.00
A2C ICM	**0.99**	**0.99**	0.70 ± 0.45	**0.79 ± 0.40**	**0.39 ± 0.49**
A2C RND	0.06 ± 0.24	0.19 ± 0.40	0.00	0.00	0.00
A2C RIDE	0.00	0.00	0.00	0.00	0.00
PPO	0.00	0.00	0.00	0.00	0.00
PPO Count	**0.99**	0.79 ± 0.40	0.73 ± 0.44	**0.59 ± 0.49**	**0.59 ± 0.49**
PPO Hash-Count	**0.99**	**0.99**	0.59 ± 0.49	**0.79 ± 0.40**	**0.59 ± 0.49**
PPO ICM	**0.99**	0.39 ± 0.49	0.31 ± 0.46	**0.79 ± 0.40**	**0.19 ± 0.40**
PPO RND	0.38 ± 0.48	0.19 ± 0.40	0.00	0.00	0.00
PPO RIDE	0.77 ± 0.41	0.00	0.15 ± 0.36	0.00	0.00
DeA2C Count	**0.99**	**0.99**	0.90 ± 0.28	0.19 ± 0.40	**0.16 ± 0.37**
DeA2C ICM	**0.99**	0.82 ± 0.37	0.66 ± 0.47	0.39 ± 0.49	**0.33 ± 0.47**
DePPO Count	**0.99**	**0.99**	0.08 ± 0.28	**0.82 ± 0.37**	0.00
DePPO ICM	**0.99**	0.66 ± 0.47	0.09 ± 0.30	0.00	0.00
DeDQN Count	**0.99**	**0.99**	0.44 ± 0.49	**0.79 ± 0.40**	**0.19 ± 0.40**
DeDQN ICM	**0.99**	**0.99**	0.30 ± 0.46	**0.59 ± 0.49**	**0.44 ± 0.49**



(a) DeepSea $N = 10$ A2C (b) DeepSea $N = 10$ PPO (c) DeepSea $N = 10$ DeRL

(d) DeepSea $N = 14$ A2C (e) DeepSea $N = 14$ PPO (f) DeepSea $N = 14$ DeRL

(g) DeepSea $N = 20$ A2C (h) DeepSea $N = 20$ PPO (i) DeepSea $N = 20$ DeRL

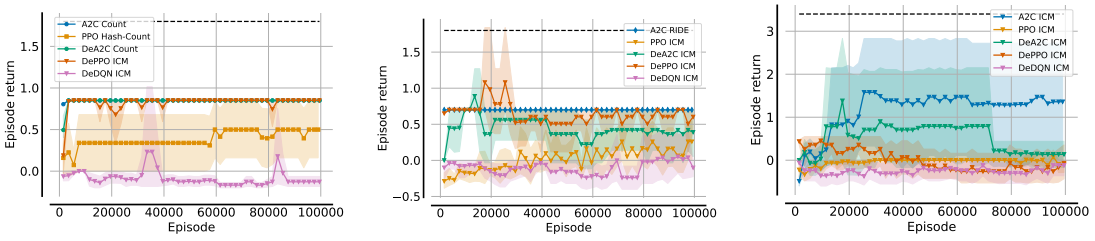
(j) DeepSea $N = 24$ A2C (k) DeepSea $N = 24$ PPO (l) DeepSea $N = 24$ DeRL

(m) DeepSea $N = 30$ A2C (n) DeepSea $N = 30$ PPO (o) DeepSea $N = 30$ DeRL

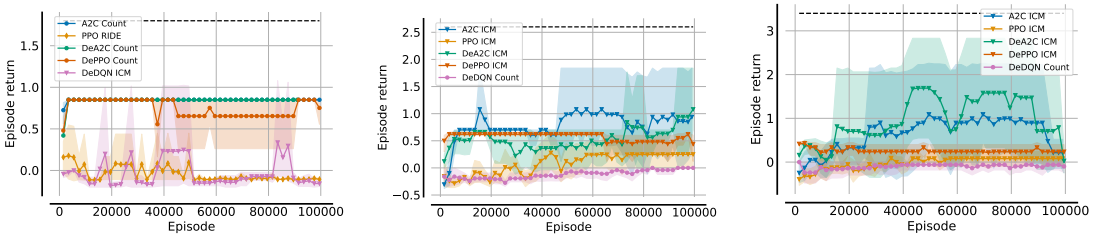
Figure A.2: Average evaluation returns and 95% confidence intervals for A2C (first column), PPO (second column) and DeRL (third column) with all intrinsic rewards for all DeepSea tasks.

Table A.12: Maximum evaluation returns in the Hallway environment with a single standard deviation.

Algorithm \ Task	Hallway 10-0	Hallway 10-10	Hallway 20-0	Hallway 20-20	Hallway 30-0	Hallway 30-30
A2C	0.68 ± 0.34	0.51 ± 0.42	$\mathbf{0.42 \pm 0.34}$	$\mathbf{0.50 \pm 0.25}$	$\mathbf{0.44 \pm 0.22}$	$\mathbf{0.46 \pm 0.07}$
A2C Count	**0.85**	**0.85**	**0.70**	**0.60**	$\mathbf{0.52 \pm 0.06}$	0.14 ± 0.24
A2C Hash-Count	**0.85**	**0.85**	**0.70**	**0.60**	$\mathbf{0.52 \pm 0.06}$	0.22 ± 0.27
A2C ICM	0.68 ± 0.34	0.68 ± 0.34	$\mathbf{0.66 \pm 1.01}$	$\mathbf{1.08 \pm 0.76}$	$\mathbf{1.58 \pm 1.50}$	$\mathbf{1.09 \pm 1.15}$
A2C RND	0.12 ± 0.35	-0.07 ± 0.09	-0.17 ± 0.15	-0.24 ± 0.20	$\mathbf{-0.24 \pm 0.28}$	-0.12 ± 0.24
A2C RIDE	**0.85**	**0.85**	**0.70**	$\mathbf{0.62 \pm 0.04}$	**0.55**	$\mathbf{0.43 \pm 0.06}$
PPO	0.00	0.00	0.00	0.00	0.00	0.00
PPO Count	0.00	0.00	0.00	0.00	0.00	0.00
PPO Hash-Count	0.50 ± 0.41	0.27 ± 0.39	0.00	0.00	0.00	0.00
PPO ICM	0.48 ± 0.39	0.40 ± 0.41	0.28 ± 0.34	$\mathbf{0.39 \pm 0.32}$	$\mathbf{0.08 \pm 0.16}$	0.10 ± 0.20
PPO RND	0.13 ± 0.37	0.47 ± 0.44	0.02 ± 0.31	0.17 ± 0.40	-0.04 ± 0.32	0.00
PPO RIDE	0.10 ± 0.35	0.26 ± 0.44	-0.03 ± 0.37	0.13 ± 0.41	-0.19 ± 0.28	-0.01 ± 0.40
DeA2C Count	**0.85**	**0.85**	**0.60**	**0.70**	**0.55**	$\mathbf{0.43 \pm 0.06}$
DeA2C ICM	**0.85**	**0.85**	$\mathbf{1.08 \pm 0.76}$	$\mathbf{1.08 \pm 0.76}$	$\mathbf{1.38 \pm 1.68}$	$\mathbf{1.69 \pm 1.40}$
DePPO Count	**0.85**	**0.85**	$\mathbf{0.56 \pm 0.28}$	**0.60**	$\mathbf{0.52 \pm 0.06}$	$\mathbf{0.43 \pm 0.06}$
DePPO ICM	**0.85**	**0.85**	$\mathbf{1.08 \pm 0.76}$	$\mathbf{0.62 \pm 0.04}$	**0.55**	$\mathbf{0.43 \pm 0.06}$
DeDQN Count	0.13 ± 0.37	0.11 ± 0.37	0.00	0.00	0.00	0.00
DeDQN ICM	0.27 ± 0.77	0.39 ± 0.82	0.06 ± 0.36	$\mathbf{0.35 \pm 1.12}$	0.00	0.00

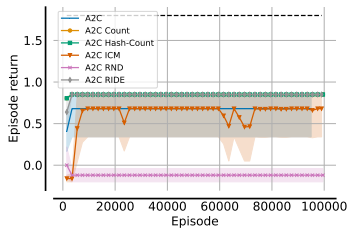


(a) Hallway $N_l = 10, N_r = 0$ (b) Hallway $N_l = 20, N_r = 0$ (c) Hallway $N_l = 30, N_r = 0$

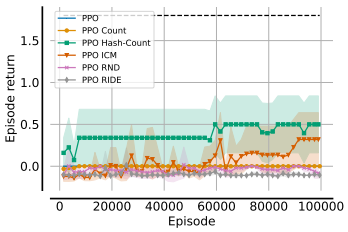


(d) Hallway $N_l = 10, N_r = 10$ (e) Hallway $N_l = 20, N_r = 20$ (f) Hallway $N_l = 30, N_r = 30$

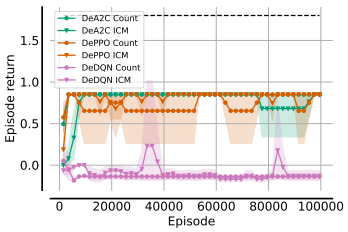
Figure A.3: Average evaluation returns and 95% confidence intervals for A2C, PPO and DeRL with the highest achieving intrinsic reward in all Hallway tasks.



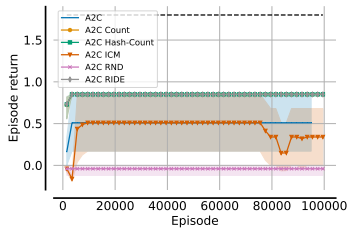
(a) Hallway $N_l = 10, N_r = 0$ A2C



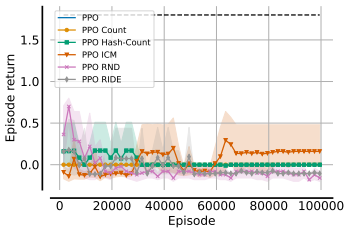
(b) Hallway $N_l = 10, N_r = 0$ PPO



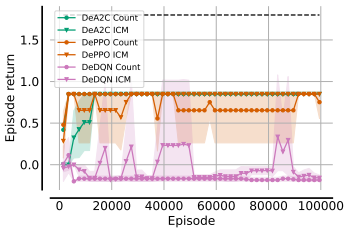
(c) Hallway $N_l = 10, N_r = 0$ DeRL



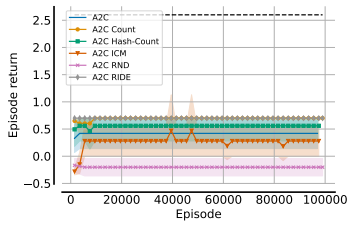
(d) Hallway $N_l = 10, N_r = 10$ A2C



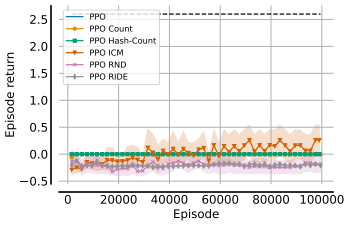
(e) Hallway $N_l = 10, N_r = 10$ PPO



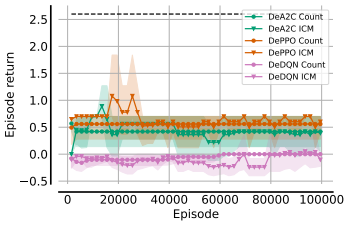
(f) Hallway $N_l = 10, N_r = 10$ DeRL



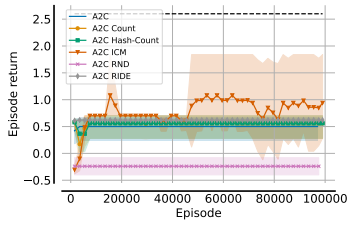
(g) Hallway $N_l = 20, N_r = 0$ A2C



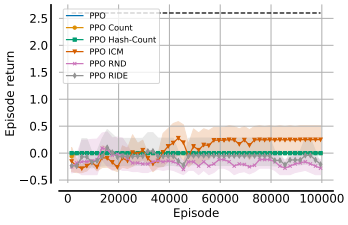
(h) Hallway $N_l = 20, N_r = 0$ PPO



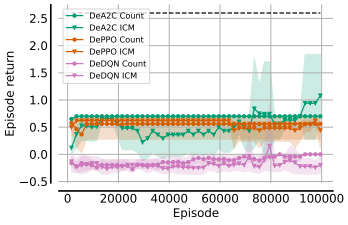
(i) Hallway $N_l = 20, N_r = 0$ DeRL



(j) Hallway $N_l = 20, N_r = 20$ A2C

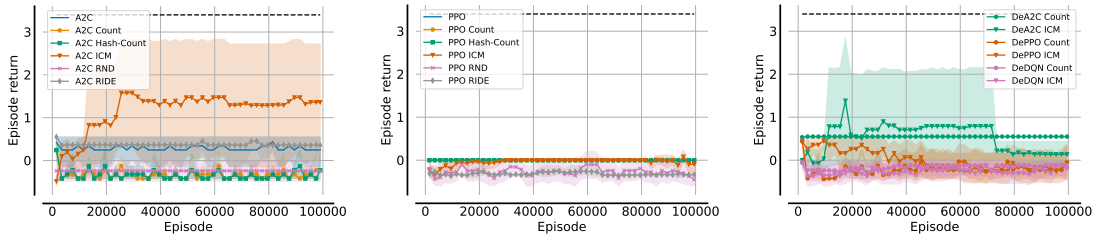


(k) Hallway $N_l = 20, N_r = 20$ PPO

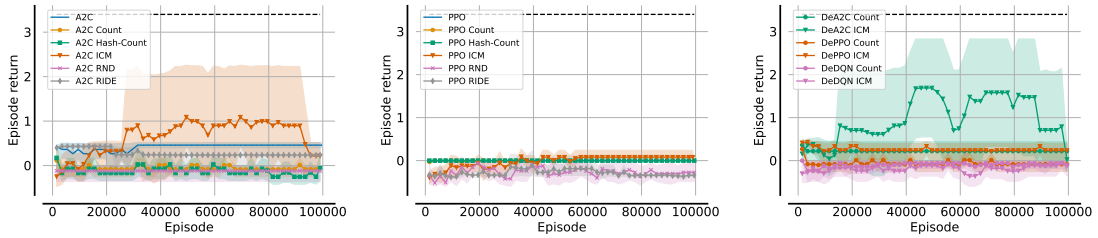


(l) Hallway $N_l = 20, N_r = 20$ DeRL

Figure A.4: Average evaluation returns and 95% confidence intervals for A2C (first column), PPO (second column) and DeRL (third column) with all intrinsic rewards for all Hallway tasks with $N_l \in \{10, 20\}$.



(a) Hallway $N_l = 30, N_r = 0$ A2C (b) Hallway $N_l = 30, N_r = 0$ PPO (c) Hallway $N_l = 30, N_r = 0$ DeRL



(d) Hallway $N_l = 30, N_r = 30$ A2C (e) Hallway $N_l = 30, N_r = 30$ PPO (f) Hallway $N_l = 30, N_r = 30$ DeRL

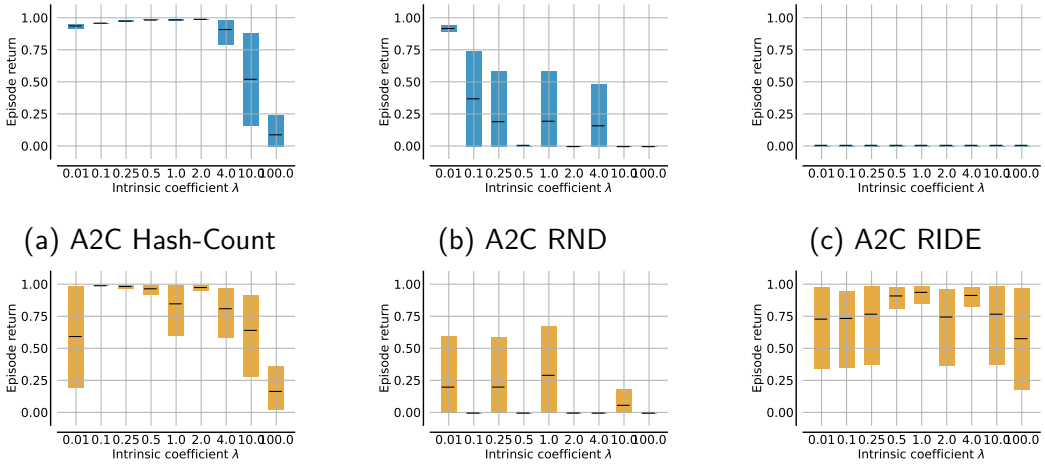
Figure A.5: Average evaluation returns and 95% confidence intervals for A2C (first column), PPO (second column) and DeRL (third column) with all intrinsic rewards for all Hallway tasks with $N_l = 30$.

A.3 Hyperparameter Sensitivity

In this section, we provide further plots and tables containing maximum and average achieved evaluation returns for both sets of hyerparameter sensitivity experiments. As described in Section 3.4.1, we evaluate both baselines and DeRL algorithms with varying intrinsic reward coefficients, λ, and rate of decay of intrinsic rewards. Both experiments were conducted in DeepSea 10 and Hallway $N_l = N_r = 10$. We provide tables showing the maximum evaluation returns and bar plots indicating the varying average evaluation returns for various parameterisation of intrinsic rewards where not yet shown within Section 3.4.1. Within tables, the highest performing configuration for a single algorithm is highlighted in bold within each row with all configurations within a single standard deviation of the highest return.

Figure A.6 and Figure A.7 show average evaluation returns for baselines with Hash-Count, RND, and RIDE intrinsic rewards with varying values of $\lambda \in \{0.01, 0.1, 0.25, 0.5, 1.0, 2.0, 4.0, 10.0, 100.0\}$ in DeepSea 10 and Hallway $N_l = N_r = 10$, respectively. Table A.13 and Table A.14 show the maximum evaluation returns for all baselines and DeRL with all intrinsic rewards and varying λ values in DeepSea 10 and Hallway $N_l = N_r = 10$, respectively.

Similarly, we evaluate all baselines and DeRL algorithms with varying rate of decay of intrinsic rewards in DeepSea and Hallway. For count-based intrinsic rewards, the rate of decay can be determined by the increment of the state count $N(s)$. By default, the count is incremented by 1 for each state occurrence. For this analysis, we consider increments $\{0.01, 0.1, 0.2, 1.0, 5.0, 10.0, 100.0\}$. For deep prediction-based intrinsic rewards, ICM, RND and RIDE, the rate of decay is determined by their learning rates. We consider learning rates $\{1e^{-9}, 1e^{-8}, 2e^{-8}, 1e^{-7}, 5e^{-7}, 1e^{-6}, 1e^{-5}, 1e^{-4}, 1e^{-3}\}$. Figure A.8 and Figure A.9 show average evaluation returns for baselines with Hash-Count, RND, and RIDE intrinsic rewards with varying speeds of decay in DeepSea 10 and Hallway $N_l = N_r = 10$, respectively. Tables A.15 and A.16 show the maximum evaluation returns for all baselines and DeRL with count-based and prediction-based intrinsic rewards, respectively, for varying rates of decay in DeepSea 10. Tables A.17 and A.18 show identical tables for Hallway $N_l = N_r = 10$.



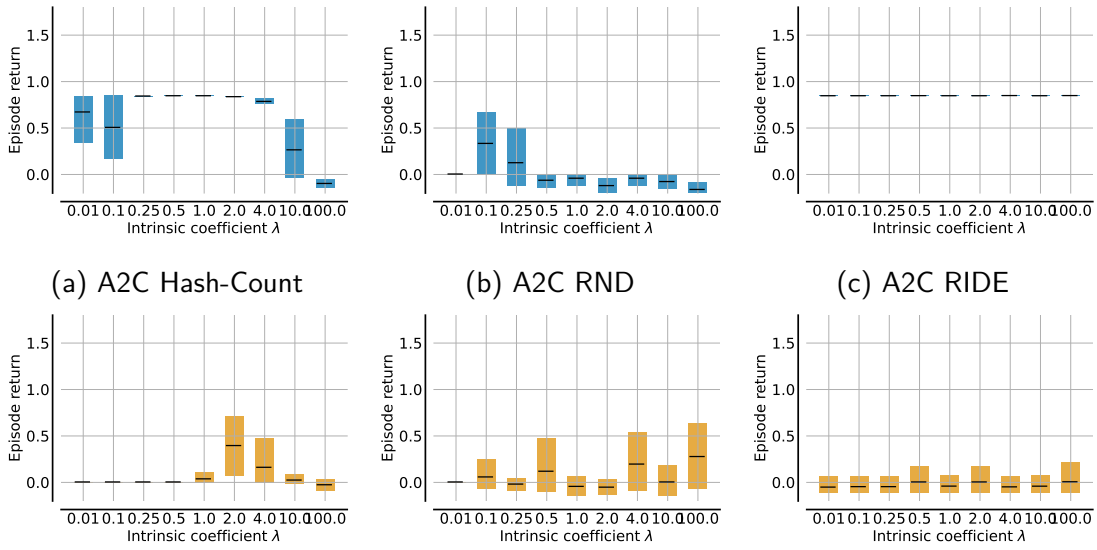
(a) A2C Hash-Count (b) A2C RND (c) A2C RIDE

(d) PPO Hash-Count (e) PPO RND (f) PPO RIDE

Figure A.6: Average evaluation returns and 95% confidence intervals for A2C and PPO with Hash-Count, RND and RIDE intrinsic rewards in DeepSea 10 with $\lambda \in \{0.01, 0.1, 0.25, 0.5, 1.0, 2.0, 4.0, 10.0, 100.0\}$.

Table A.13: Maximum evaluation returns with a single standard deviation in DeepSea 10 for varying λ.

Algorithm \ λ	0.01	0.1	0.25	0.5	1.0	2.0	4.0	10.0	100.0
A2C Count	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	0.79 ± 0.30	0.79 ± 0.30
A2C Hash-Count	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	0.79 ± 0.30	0.39 ± 0.40
A2C ICM	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	0.59 ± 0.40
A2C RND	**0.99**	0.39 ± 0.40	0.19 ± 0.30	0.00	0.20 ± 0.30	0.00	0.19 ± 0.30	0.00	-0.01
A2C RIDE	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
PPO Count	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
PPO Hash-Count	0.59 ± 0.40	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	0.79 ± 0.30
PPO ICM	0.00	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	0.39 ± 0.40
PPO RND	0.20 ± 0.30	0.00	0.39 ± 0.40	0.00	0.39 ± 0.40	0.00	-0.01	0.19 ± 0.30	-0.01
PPO RIDE	0.79 ± 0.30	0.79 ± 0.30	0.79 ± 0.30	**0.99**	**0.99**	0.79 ± 0.30	**0.99**	0.79 ± 0.30	0.59 ± 0.40
DeA2C Count	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
DeA2C ICM	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
DePPO Count	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	0.59 ± 0.40
DePPO ICM	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	0.59 ± 0.40	0.39 ± 0.40	0.19 ± 0.30
DeDQN Count	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
DeDQN ICM	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	0.79 ± 0.30



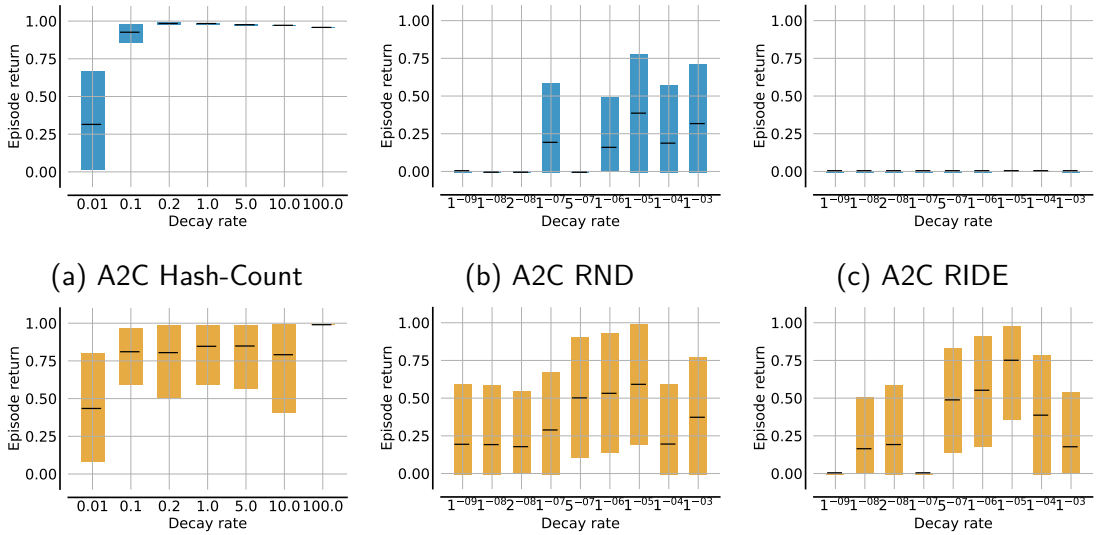
(a) A2C Hash-Count (b) A2C RND (c) A2C RIDE

(d) PPO Hash-Count (e) PPO RND (f) PPO RIDE

Figure A.7: Average evaluation returns with 95% confidence intervals for A2C and PPO with Hash-Count, RND and RIDE intrinsic rewards in Hallway $N_l = N_r = 10$ with $\lambda \in \{0.01, 0.1, 0.25, 0.5, 1.0, 2.0, 4.0, 10.0, 100.0\}$.

Table A.14: Maximum evaluation returns with a single standard deviation in Hallway $N_l = N_r = 10$ for varying λ.

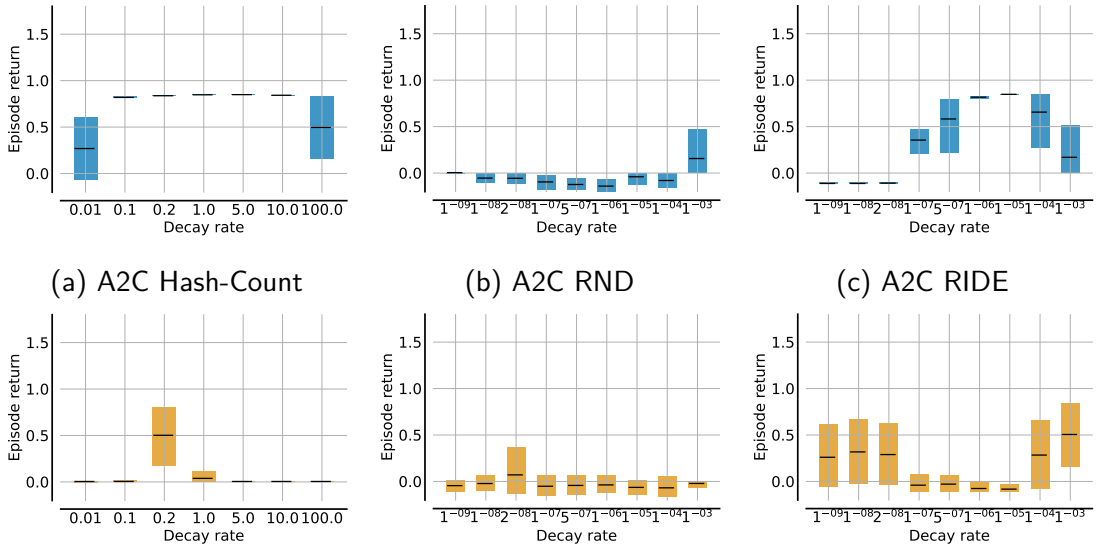
Algorithm \ λ	0.01	0.1	0.25	0.5	1.0	2.0	4.0	10.0	100.0
A2C Count	0.68 ± 0.26	0.51 ± 0.34	**0.85**	**0.85**	**0.85**	**0.85**	**0.85**	0.51 ± 0.34	0.80
A2C Hash-Count	0.68 ± 0.26	0.51 ± 0.34	**0.85**	**0.85**	**0.85**	**0.85**	**0.85**	0.51 ± 0.34	0.80
A2C ICM	0.17 ± 0.26	0.68 ± 0.26	**0.85**	**0.85**	0.51 ± 0.34	0.51 ± 0.34	**0.85**	0.89 ± 0.72	0.28 ± 0.40
A2C RND	0.00	0.34 ± 0.34	0.13 ± 0.32	-0.06 ± 0.07	-0.04 ± 0.06	-0.08 ± 0.08	-0.04 ± 0.06	-0.06 ± 0.07	-0.16 ± 0.06
A2C RIDE	**0.85**	**0.85**	**0.85**	**0.85**	**0.85**	**0.85**	**0.85**	0.85	**0.85**
PPO Count	0.00	0.00	0.00	0.00	0.00	0.00	-0.06 ± 0.04	0.00	-0.06 ± 0.07
PPO Hash-Count	0.00	0.00	0.00	0.00	0.17 ± 0.26	0.64 ± 0.24	0.32 ± 0.32	0.17 ± 0.25	0.14 ± 0.27
PPO ICM	0.00	0.51 ± 0.34	0.26 ± 0.38	0.32 ± 0.32	0.33 ± 0.33	0.32 ± 0.36	0.33 ± 0.34	0.42 ± 0.40	0.24 ± 0.40
PPO RND	0.16 ± 0.24	0.28 ± 0.37	0.32 ± 0.32	0.48 ± 0.32	0.80	0.61 ± 0.31	0.64 ± 0.24	0.60 ± 0.30	0.81 ± 0.01
PPO RIDE	0.10 ± 0.28	0.26 ± 0.37	0.26 ± 0.37	0.30 ± 0.57	0.43 ± 0.37	0.30 ± 0.57	0.29 ± 0.57	0.30 ± 0.57	0.26 ± 0.37
DeA2C Count	0.17 ± 0.26	0.68 ± 0.26	**0.85**	**0.85**	**0.85**	**0.85**	0.82 ± 0.02	0.84 ± 0.02	0.82 ± 0.02
DeA2C ICM	0.17 ± 0.26	**0.85**	0.68 ± 0.26	**0.85**	**0.85**	**0.85**	0.67 ± 0.26	0.34 ± 0.34	0.81 ± 0.01
DePPO Count	0.34 ± 0.34	0.68 ± 0.26	**0.85**	**0.85**	**0.85**	**0.85**	**0.85**	0.34 ± 0.34	0.80
DePPO ICM	0.17 ± 0.26	0.17 ± 0.26	0.68 ± 0.26	0.51 ± 0.34	**0.85**	$\mathbf{1.04 \pm 0.28}$	$\mathbf{0.85 \pm 0.01}$	1.04 ± 0.28	$\mathbf{0.85 \pm 0.56}$
DeDQN Count	0.26 ± 0.42	0.28 ± 0.40	0.32 ± 0.60	0.43 ± 0.42	0.43 ± 0.42	0.68 ± 0.76	0.47 ± 0.40	$\mathbf{1.60 \pm 0.30}$	$\mathbf{1.23 \pm 0.38}$
DeDQN ICM	0.32 ± 0.60	0.11 ± 0.31	0.25 ± 0.60	0.22 ± 0.61	0.34 ± 0.57	0.50 ± 0.63	0.66 ± 0.26	0.64 ± 0.57	0.70 ± 0.54



(a) A2C Hash-Count (b) A2C RND (c) A2C RIDE

(d) PPO Hash-Count (e) PPO RND (f) PPO RIDE

Figure A.8: Average evaluation returns with 95% confidence intervals for A2C and PPO with Hash-Count, RND and RIDE intrinsic rewards in DeepSea 10 with varying count increments and learning rates.



(a) A2C Hash-Count (b) A2C RND (c) A2C RIDE

(d) PPO Hash-Count (e) PPO RND (f) PPO RIDE

Figure A.9: Average evaluation returns with 95% confidence intervals for A2C and PPO with Hash-Count, RND and RIDE intrinsic rewards in Hallway $N_l = N_r = 10$ with varying count increments and learning rates.

Table A.15: Maximum evaluation returns with a single standard deviation in DeepSea 10 with count-based intrinsic rewards and varying count increments.

Algorithm \ Count increment	0.01	0.1	0.2	1.0	5.0	10.0	100.0
A2C Count	**0.99**	**0.99**	0.79 ± 0.30	**0.99**	**0.99**	**0.99**	**0.99**
A2C Hash-Count	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
PPO Count	0.59 ± 0.40	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	0.00
PPO Hash-Count	0.79 ± 0.30	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
DeA2C Count	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
DePPO Count	0.79 ± 0.30	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
DeDQN Count	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**

Table A.16: Maximum evaluation returns with a single standard deviation in DeepSea 10 with prediction-based intrinsic rewards and varying learning rates.

Algorithm \ Learning rate	1e-9	1e-8	2e-8	1e-7	5e-7	1e-6	1e-5	1e-4	1e-3
A2C ICM	**0.00**	**0.00**	**0.00**	**0.00**	0.39 ± 0.40	$\mathbf{0.79 \pm 0.30}$	**0.99**	**0.99**	**0.99**
A2C RND	$\mathbf{0.19 \pm 0.30}$	**0.00**	**−0.01**	$\mathbf{0.20 \pm 0.30}$	0.00	0.19 ± 0.30	0.39 ± 0.40	0.19 ± 0.30	0.39 ± 0.40
A2C RIDE	**0.00**	**0.00**	**0.00**	**0.00**	0.00	0.00	0.00	0.00	0.00
PPO ICM	**−0.01**	$\mathbf{0.19 \pm 0.30}$	**0.00**	$\mathbf{0.39 \pm 0.40}$	**0.99**	$\mathbf{0.59 \pm 0.40}$	**0.99**	**0.99**	**0.99**
PPO RND	$\mathbf{0.19 \pm 0.30}$	$\mathbf{0.19 \pm 0.30}$	$\mathbf{0.19 \pm 0.30}$	$\mathbf{0.39 \pm 0.40}$	0.59 ± 0.40	$\mathbf{0.59 \pm 0.40}$	0.59 ± 0.40	0.39 ± 0.40	0.39 ± 0.40
PPO RIDE	**0.00**	$\mathbf{0.19 \pm 0.30}$	$\mathbf{0.20 \pm 0.30}$	**0.00**	0.79 ± 0.30	$\mathbf{0.79 \pm 0.30}$	0.79 ± 0.30	0.39 ± 0.40	0.20 ± 0.30
DeA2C ICM	**0.00**	**0.00**	**0.00**	**0.00**	0.39 ± 0.40	$\mathbf{0.59 \pm 0.40}$	**0.99**	**0.99**	**0.99**
DePPO ICM	**0.00**	**0.00**	**0.00**	**0.00**	0.19 ± 0.30	0.39 ± 0.40	**0.99**	**0.99**	**0.99**
DeDQN ICM	**0.00**	$\mathbf{0.19 \pm 0.30}$	**0.00**	**0.00**	0.39 ± 0.40	$\mathbf{0.79 \pm 0.30}$	**0.99**	**0.99**	**0.99**

Table A.17: Maximum evaluation returns with a single standard deviation in Hallway $N_l = N_r = 10$ with count-based intrinsic rewards and varying count increments.

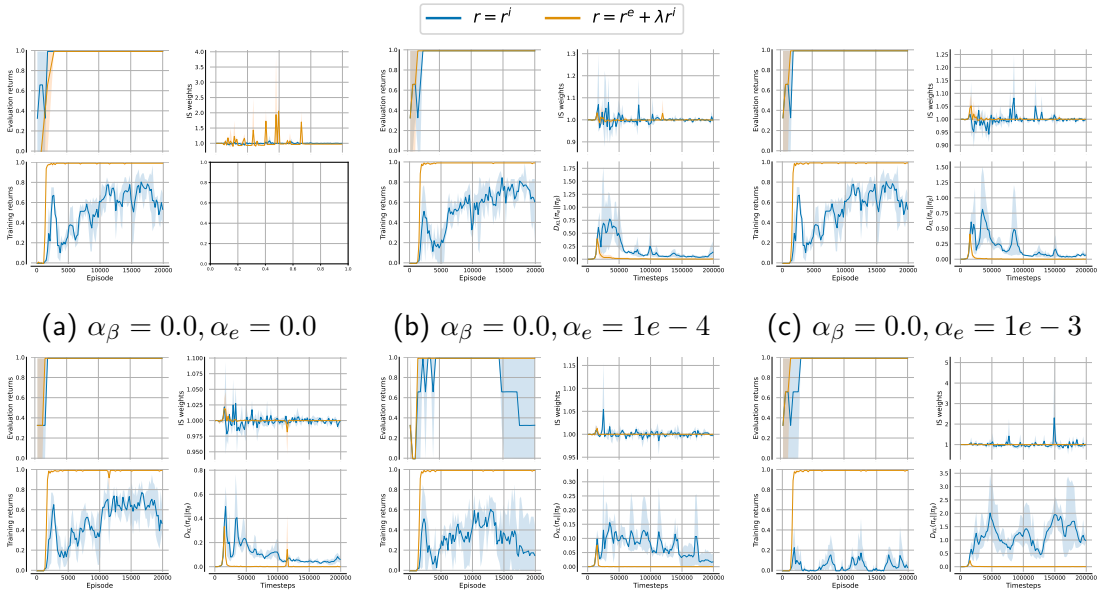
Algorithm \ Count increment	0.01	0.1	0.2	1.0	5.0	10.0	100.0
A2C Count	0.80	**0.80**	0.17 ± 0.26	**0.85**	0.68 ± 0.26	0.68 ± 0.26	$\mathbf{0.68 \pm 0.26}$
A2C Hash-Count	0.34 ± 0.34	**0.85**	**0.85**	**0.85**	**0.85**	**0.85**	$\mathbf{0.51 \pm 0.34}$
PPO Count	0.00	0.00	0.16 ± 0.25	0.00	0.00	0.00	0.00
PPO Hash-Count	0.16 ± 0.24	0.26 ± 0.38	0.80	0.17 ± 0.26	0.00	0.00	0.00
DeA2C Count	**0.85**	$\mathbf{1.00 \pm 0.30}$	0.83 ± 0.02	**0.85**	**0.85**	0.34 ± 0.34	0.17 ± 0.26
DePPO Count	$\mathbf{1.23 \pm 0.38}$	**0.80**	0.25 ± 0.37	**0.85**	0.68 ± 0.26	0.51 ± 0.34	0.00
DeDQN Count	$\mathbf{1.02 \pm 0.60}$	**0.85**	$\mathbf{1.23 \pm 0.38}$	0.43 ± 0.42	0.24 ± 0.40	0.05 ± 0.34	0.26 ± 0.42

Table A.18: Maximum evaluation returns with a single standard deviation in Hallway $N_l = N_r = 10$ with prediction-based intrinsic rewards and varying learning rates.

Algorithm \ Learning rate	1e-9	1e-8	2e-8	1e-7	5e-7	1e-6	1e-5	1e-4	1e-3
A2C ICM	-0.04 ± 0.06	-0.04 ± 0.06	-0.04 ± 0.06	0.15 ± 0.28	0.68 ± 0.26	0.51 ± 0.34	**0.85**	**0.85**	0.34 ± 0.34
A2C RND	0.00	0.00	0.00	0.00	0.00	0.00	-0.04 ± 0.06	-0.08 ± 0.08	0.17 ± 0.25
A2C RIDE	-0.11 ± 0.01	-0.11 ± 0.01	-0.11 ± 0.01	$\mathbf{0.84 \pm 0.02}$	0.66 ± 0.28	**0.85**	**0.85**	0.66 ± 0.28	0.17 ± 0.26
PPO ICM	0.40 ± 0.40	0.40 ± 0.40	0.22 ± 0.40	0.24 ± 0.40	0.40 ± 0.40	0.26 ± 0.38	0.33 ± 0.33	0.50 ± 0.34	0.67 ± 0.26
PPO RND	**0.80**	**0.80**	$\mathbf{0.64 \pm 0.24}$	$\mathbf{0.44 \pm 0.38}$	**0.80**	0.23 ± 0.39	0.12 ± 0.30	0.24 ± 0.40	0.44 ± 0.38
PPO RIDE	0.44 ± 0.36	0.62 ± 0.28	$\mathbf{0.65 \pm 0.24}$	$\mathbf{0.43 \pm 0.37}$	0.44 ± 0.37	0.10 ± 0.30	0.26 ± 0.37	0.46 ± 0.38	0.51 ± 0.34
DeA2C ICM	0.00	0.00	0.00	$\mathbf{0.49 \pm 0.33}$	0.67 ± 0.26	**0.85**	**0.85**	**0.85**	**0.85**
DePPO ICM	-0.04 ± 0.06	-0.04 ± 0.06	-0.04 ± 0.06	$\mathbf{0.42 \pm 0.40}$	0.48 ± 0.38	0.68 ± 0.26	**0.85**	0.49 ± 0.33	0.00
DeDQN ICM	-0.04 ± 0.06	0.00	-0.04 ± 0.06	$\mathbf{1.04 \pm 0.78}$	0.34 ± 0.57	0.31 ± 0.57	0.30 ± 0.38	0.13 ± 0.32	0.17 ± 0.26

A.4 KL-Divergence Constraint Regularisation

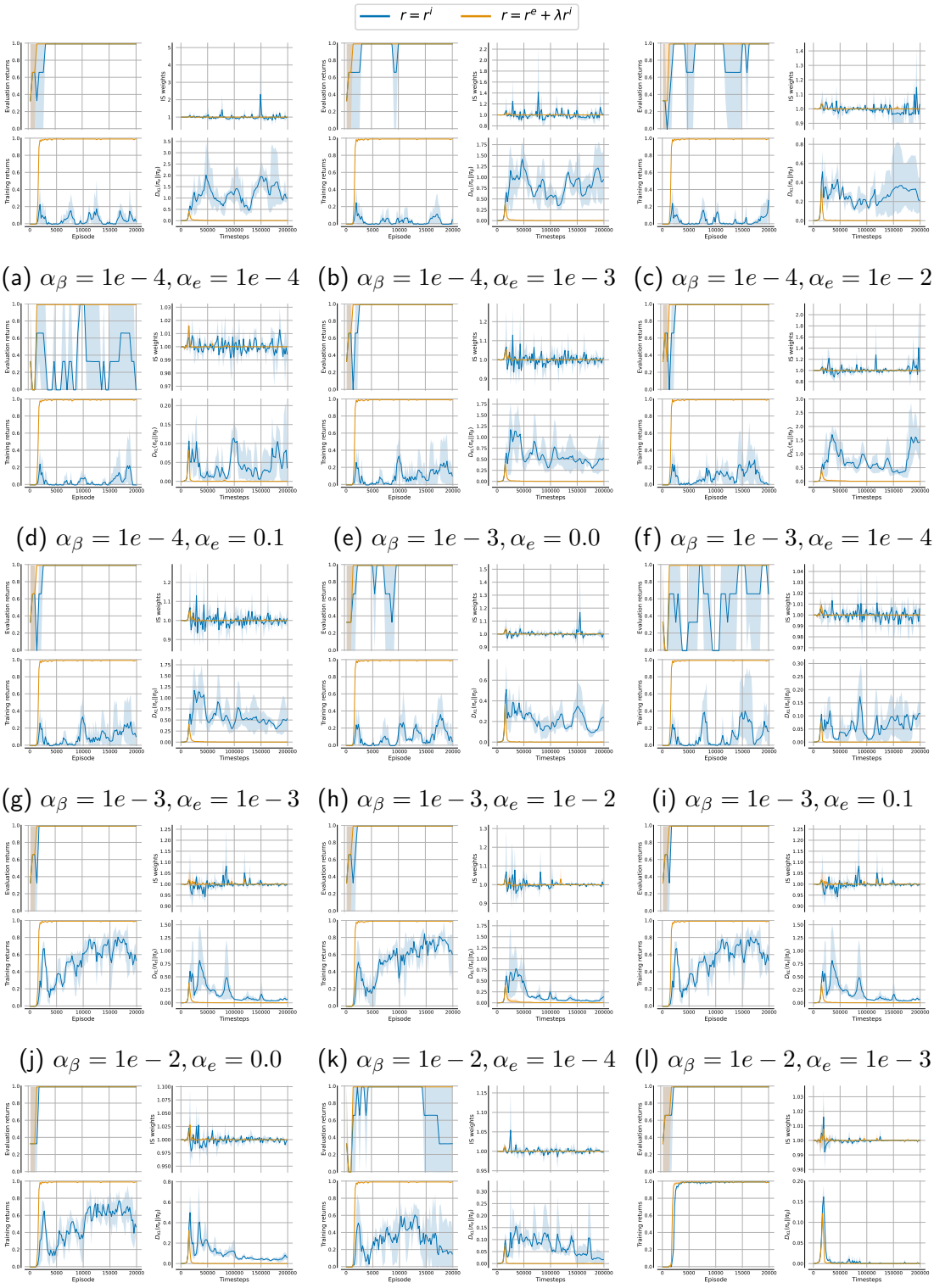
In this section, we first provide figures showing evaluation returns of the evaluation policy (top left), training returns of the exploration policy (bottom left), IS weights (top right) and KL-divergence of exploration and exploitation policy (bottom right) for DeA2C trained with Count intrinsic rewards and KL-divergence constraints in the DeepSea 10 and Hallway $N_l = N_r = 20$ tasks for 20,000 episodes. Each figure corresponds to DeA2C being trained with KL-divergence constraint coefficients α_β and α_e for the regulariser terms for the exploration policy π_β and exploitation policy π_e, respectively. We present training with the exploration policy being trained using only intrinsic rewards (blue) or using the sum of intrinsic and extrinsic rewards (orange). Figures A.10 to A.12 show evaluation returns with such divergence constraints in DeepSea 10, and Figures A.13 to A.15 show evaluation returns in Hallway $N_l = N_r = 20$. Average returns and 95% confidence intervals are computed over 1,000 samples across three seeds and we consider regularisation coefficients $\alpha_\beta, \alpha_e \in \{0.0, 0.0001, 0.001, 0.01, 0.1\}$.



(a) $\alpha_\beta = 0.0, \alpha_e = 0.0$ (b) $\alpha_\beta = 0.0, \alpha_e = 1e-4$ (c) $\alpha_\beta = 0.0, \alpha_e = 1e-3$

(d) $\alpha_\beta = 0.0, \alpha_e = 1e-2$ (e) $\alpha_\beta = 0.0, \alpha_e = 0.1$ (f) $\alpha_\beta = 1e-4, \alpha_e = 0.0$

Figure A.10: DeepSea 10 evaluation with divergence constraint regularisation coefficients α_β and α_e. Shading indicates 95% confidence intervals; Part 1



(a) $\alpha_\beta = 1e-4, \alpha_e = 1e-4$ (b) $\alpha_\beta = 1e-4, \alpha_e = 1e-3$ (c) $\alpha_\beta = 1e-4, \alpha_e = 1e-2$

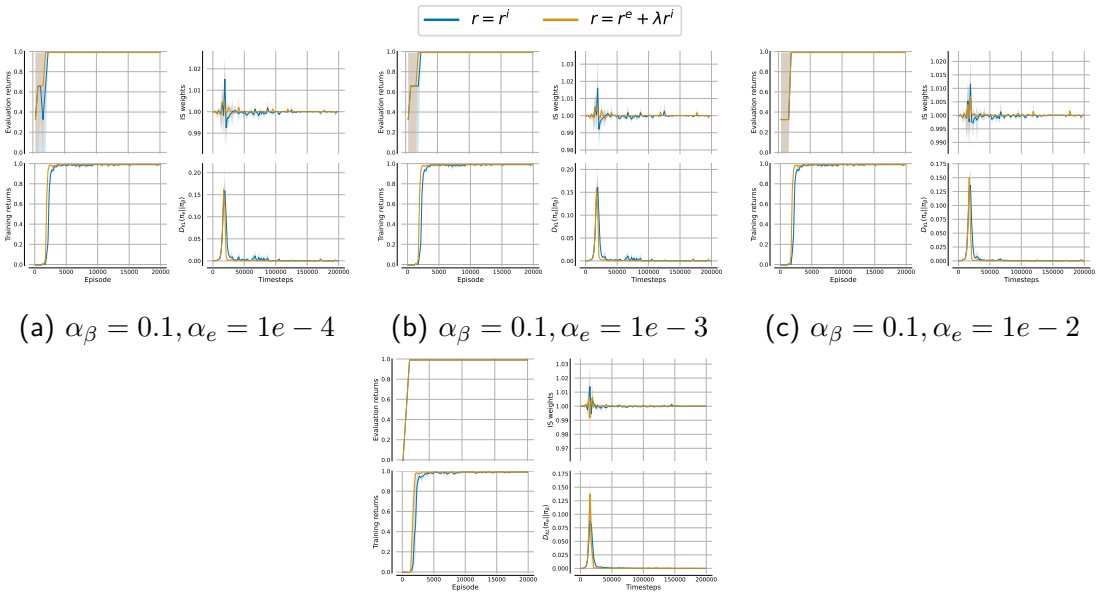
(d) $\alpha_\beta = 1e-4, \alpha_e = 0.1$ (e) $\alpha_\beta = 1e-3, \alpha_e = 0.0$ (f) $\alpha_\beta = 1e-3, \alpha_e = 1e-4$

(g) $\alpha_\beta = 1e-3, \alpha_e = 1e-3$ (h) $\alpha_\beta = 1e-3, \alpha_e = 1e-2$ (i) $\alpha_\beta = 1e-3, \alpha_e = 0.1$

(j) $\alpha_\beta = 1e-2, \alpha_e = 0.0$ (k) $\alpha_\beta = 1e-2, \alpha_e = 1e-4$ (l) $\alpha_\beta = 1e-2, \alpha_e = 1e-3$

(m) $\alpha_\beta = 1e-2, \alpha_e = 1e-2$ (n) $\alpha_\beta = 1e-2, \alpha_e = 0.1$ (o) $\alpha_\beta = 0.1, \alpha_e = 0.0$

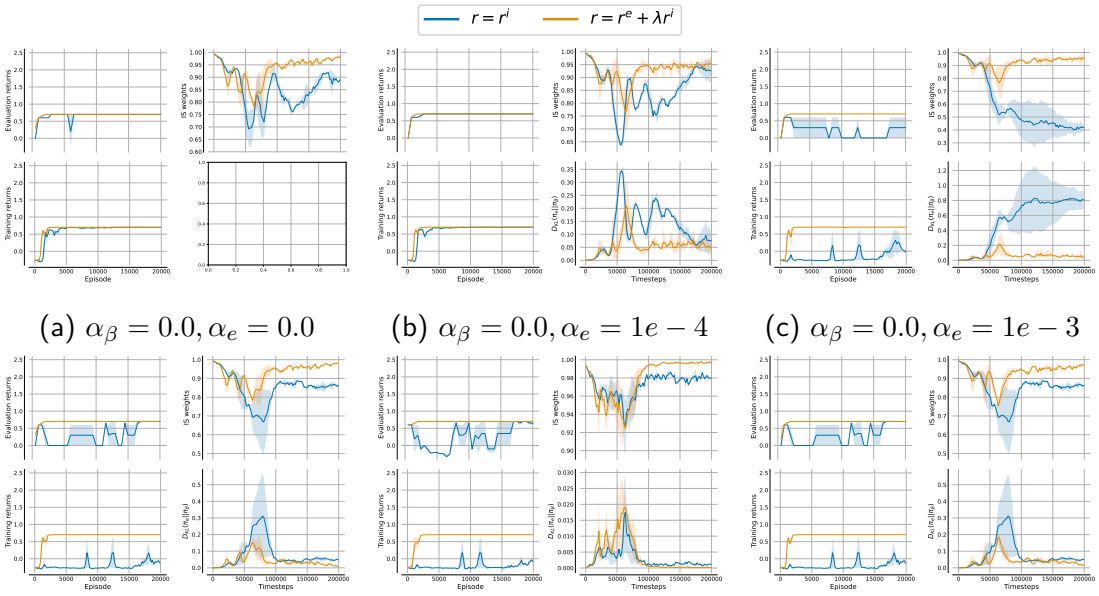
Figure A.11: DeepSea 10 evaluation with divergence constraint regularisation coefficients α_β and α_e. Shading indicates 95% confidence intervals; Part 2



(a) $\alpha_\beta = 0.1, \alpha_e = 1e-4$ (b) $\alpha_\beta = 0.1, \alpha_e = 1e-3$ (c) $\alpha_\beta = 0.1, \alpha_e = 1e-2$

(d) $\alpha_\beta = 0.1, \alpha_e = 0.1$

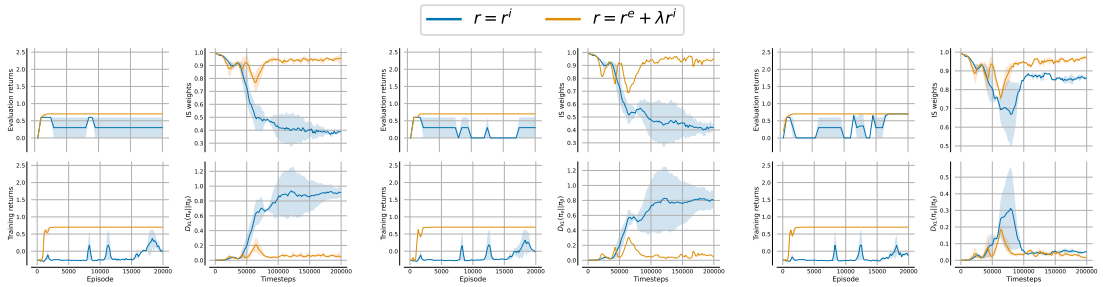
Figure A.12: DeepSea 10 evaluation with divergence constraint regularisation coefficients α_β and α_e. Shading indicates 95% confidence intervals; Part 3



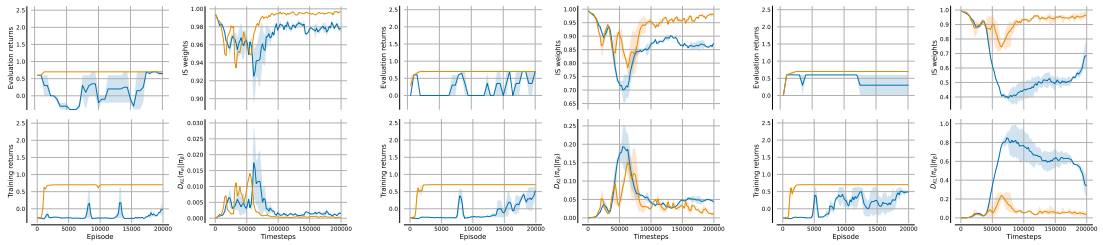
(a) $\alpha_\beta = 0.0, \alpha_e = 0.0$ (b) $\alpha_\beta = 0.0, \alpha_e = 1e-4$ (c) $\alpha_\beta = 0.0, \alpha_e = 1e-3$

(d) $\alpha_\beta = 0.0, \alpha_e = 1e-2$ (e) $\alpha_\beta = 0.0, \alpha_e = 0.1$ (f) $\alpha_\beta = 1e-4, \alpha_e = 0.0$

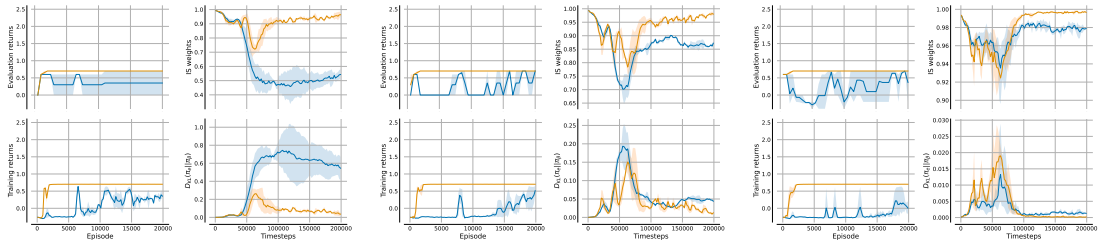
Figure A.13: Hallway $N_l = N_r = 20$ evaluation with divergence constraint regularisation coefficients α_β and α_e. Shading indicates 95% confidence intervals; Part 1



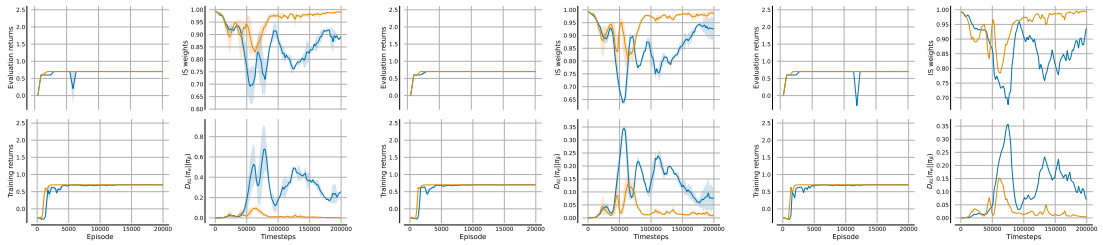
(a) $\alpha_\beta = 1e-4, \alpha_e = 0.0001$ (b) $\alpha_\beta = 1e-4, \alpha_e = 1e-3$ (c) $\alpha_\beta = 1e-4, \alpha_e = 1e-2$



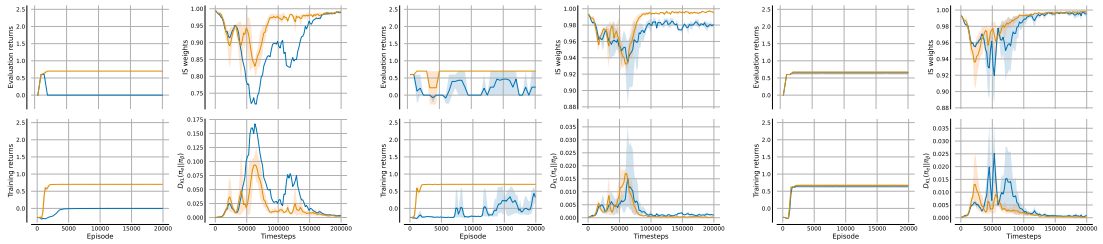
(d) $\alpha_\beta = 1e-4, \alpha_e = 0.1$ (e) $\alpha_\beta = 1e-3, \alpha_e = 0.0$ (f) $\alpha_\beta = 1e-3, \alpha_e = 0.0001$



(g) $\alpha_\beta = 1e-3, \alpha_e = 1e-3$ (h) $\alpha_\beta = 1e-3, \alpha_e = 1e-2$ (i) $\alpha_\beta = 1e-3, \alpha_e = 0.1$

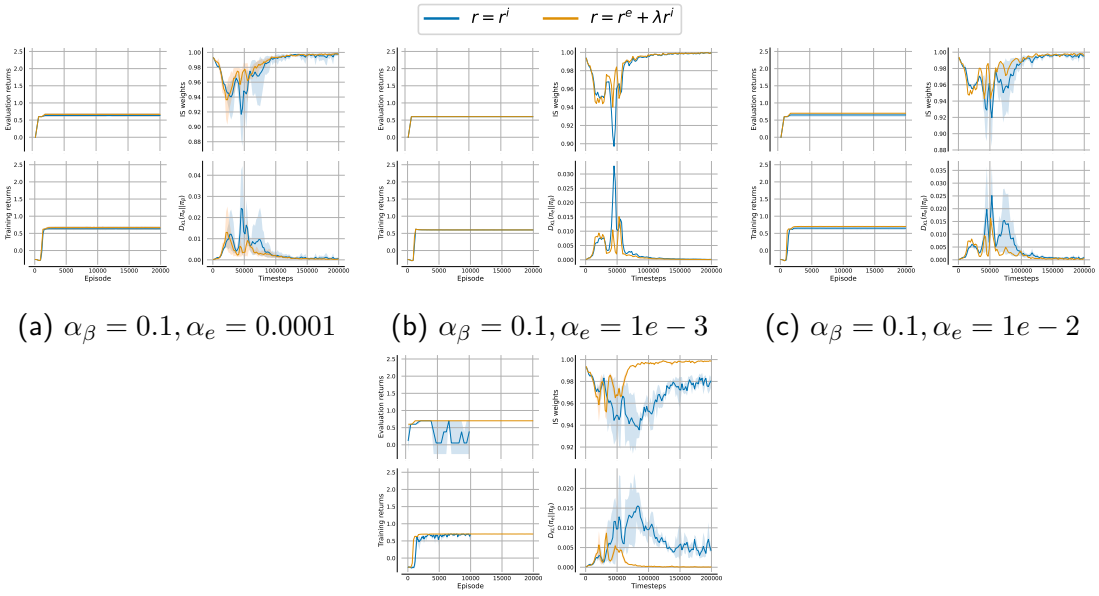


(j) $\alpha_\beta = 1e-2, \alpha_e = 0.0$ (k) $\alpha_\beta = 1e-2, \alpha_e = 0.0001$ (l) $\alpha_\beta = 1e-2, \alpha_e = 1e-3$



(m) $\alpha_\beta = 1e-2, \alpha_e = 1e-2$ (n) $\alpha_\beta = 1e-2, \alpha_e = 0.1$ (o) $\alpha_\beta = 0.1, \alpha_e = 0.0$

Figure A.14: Hallway $N_l = N_r = 20$ evaluation with divergence constraint regularisation coefficients α_β and α_e. Shading indicates 95% confidence intervals; Part 2

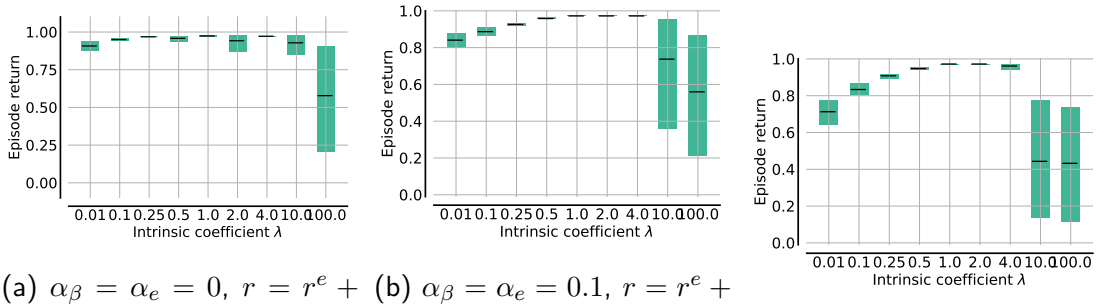


(a) $\alpha_\beta = 0.1, \alpha_e = 0.0001$ (b) $\alpha_\beta = 0.1, \alpha_e = 1e-3$ (c) $\alpha_\beta = 0.1, \alpha_e = 1e-2$

(d) $\alpha_\beta = 0.1, \alpha_e = 0.1$

Figure A.15: Hallway $N_l = N_r = 20$ evaluation with divergence constraint regularisation coefficients α_β and α_e. Shading indicates 95% confidence intervals; Part 3

Following these results, we identify the best performing regularisation coefficients for DeA2C trained with Count intrinsic rewards according to evaluation returns in both DeepSea and Hallway tasks and evaluate their sensitivity to varying values of λ and rates of decay in DeepSea 10 and Hallway $N_l = N_r = 20$ tasks following the evaluation procedure outlined in Section 3.4.1. We consider both cases in which the exploration policy π_β is trained using extrinsic and intrinsic rewards or is trained using only intrinsic rewards. All results show learning curves with average evaluation curves and 95% bootstrapped confidence intervals computed over five random seeds. Figure A.16 and Figure A.17 show the sensitivity results to varying λ values in DeepSea 10 and Hallway $N_l = N_r = 10$, respectively. Similarly, Figure A.18 and Figure A.19 show the sensitivity results to varying decay rates in DeepSea 10 and Hallway $N_l = N_r = 10$, respectively.

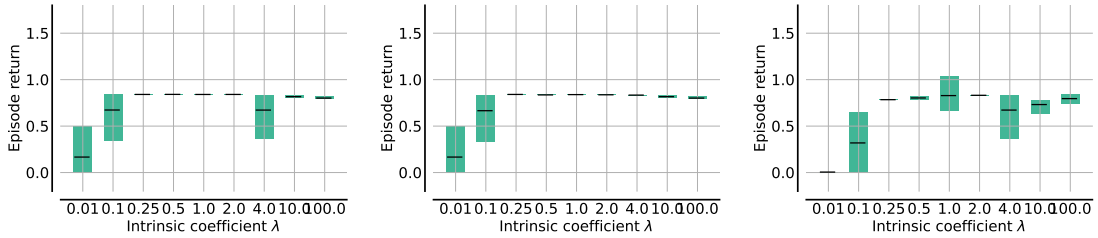


(a) $\alpha_\beta = \alpha_e = 0$, $r = r^e + \lambda r^i$

(b) $\alpha_\beta = \alpha_e = 0.1$, $r = r^e + \lambda r^i$

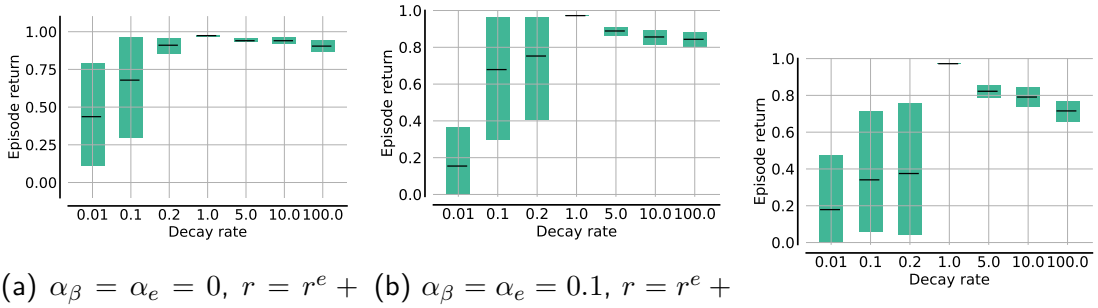
(c) $\alpha_\beta = \alpha_e = 0.1$, $r = \lambda r^i$

Figure A.16: Average evaluation returns with 95% confidence intervals for DeA2C with Count in DeepSea 10 with varying λ (a) without divergence constraints, (b) with divergence constraints and the exploration policy trained on both extrinsic and intrinsic rewards, and (c) with divergence constraints and the exploration policy only trained on intrinsic rewards.



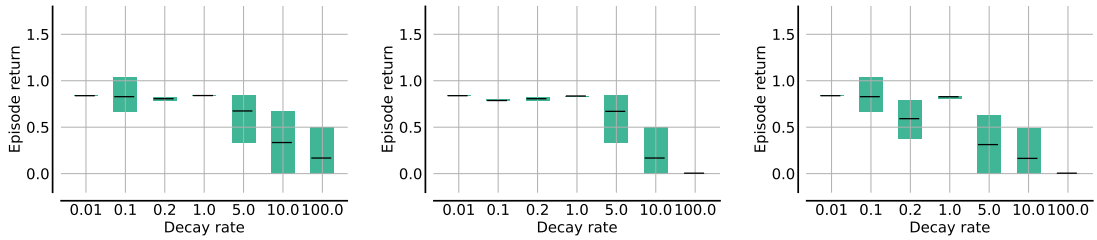
(a) $\alpha_\beta = \alpha_e = 0$, $r = r^e + \lambda r^i$ (b) $\alpha_\beta = 1e{-}2, \alpha_e = 0.0001$, $r = r^e + \lambda r^i$ (c) $\alpha_\beta = 1e{-}2, \alpha_e = 0.0001$, $r = \lambda r^i$

Figure A.17: Average evaluation returns with 95% confidence intervals for DeA2C with Count in Hallway $N_l = N_r = 10$ with varying λ (a) without divergence constraints, (b) with divergence constraints and the exploration policy trained on both extrinsic and intrinsic rewards, and (c) with divergence constraints and the exploration policy only trained on intrinsic rewards.



(a) $\alpha_\beta = \alpha_e = 0$, $r = r^e + \lambda r^i$ (b) $\alpha_\beta = \alpha_e = 0.1$, $r = r^e + \lambda r^i$ (c) $\alpha_\beta = \alpha_e = 0.1$, $r = \lambda r^i$

Figure A.18: Average evaluation returns with 95% confidence intervals for DeA2C with Count in DeepSea 10 with varying count increments (a) without divergence constraints, (b) with divergence constraints and the exploration policy trained on both extrinsic and intrinsic rewards, and (c) with divergence constraints and the exploration policy only trained on intrinsic rewards.



(a) $\alpha_\beta = \alpha_e = 0$, $r = r^e + \lambda r^i$

(b) $\alpha_\beta = 1e{-}2, \alpha_e = 0.0001$, $r = r^e + \lambda r^i$

(c) $\alpha_\beta = 1e{-}2, \alpha_e = 0.0001$, $r = \lambda r^i$

Figure A.19: Average evaluation returns with 95% confidence intervals for DeA2C with Count in Hallway $N_l = N_r = 10$ with varying count increments (a) without divergence constraints, (b) with divergence constraints and the exploration policy trained on both extrinsic and intrinsic rewards, and (c) with divergence constraints and the exploration policy only trained on intrinsic rewards.

Appendix B

Multi-Agent Deep Reinforcement Learning Benchmark

B.1 The EPyMARL Codebase

As part of this work we extended the well-known PyMARL codebase (Samvelyan et al., 2019) to include more algorithms, support more environments as well as allow for more flexible tuning of the implementation details.

All code for EPyMARL is publicly available open-source on GitHub under the following link: `https://github.com/uoe-agents/epymarl`.

Details on how to install and use EPyMARL are provided in the provided documentation under the GitHub repository, and blog posts for further information can be found under the following link: `https://agents.inf.ed.ac.uk/blog/epymarl/` and `https://agents.inf.ed.ac.uk/blog/epymarl-v2/`.

All source code that has been taken from the PyMARL repository was licensed under the Apache License v2.0. Any new code is also licensed under the Apache License v2.0. The NOTICE file in the GitHub repository contains information about the files that have been added or modified compared to the original PyMARL codebase.

B.2 Computational Cost

Approximately 138,916 CPU hours were spent for executing the experiments presented in the paper without considering the CPU hours required for the hyperparameter search. Figure B.1 presents the cumulative CPU hours required to train

each algorithm in each environments (summed over the different tasks and seeds) with and without parameter sharing using the best identified hyperparameter configurations reported in Appendix B.6. We observe that the computational cost of running experiments in SMAC is significantly higher compared to any other environment. Finally, the CPU hours required for training the algorithms without sharing is slightly higher compared to training with parameter sharing.

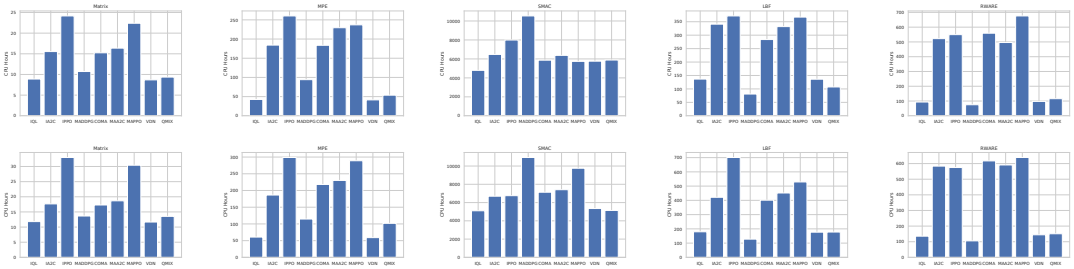


Figure B.1: CPU hours required to execute the experiments for each algorithm and environment with (top row) and without (bottom row) parameter sharing.

B.3 SMAC Win-Rates

Table B.1: Maximum win-rate and 95% confidence interval over five seeds for all nine algorithms with parameter sharing in all SMAC tasks.

Tasks \Algs.	IDQN	IA2C	IPPO	MADDPG	COMA	MAA2C	MAPPO	VDN	QMIX
2s_vs_1sc	0.61 ± 0.04	1.00	1.00	0.21 ± 0.20	0.34 ± 0.41	1.00	1.00	0.74 ± 0.04	0.85 ± 0.03
3s5z	0.39 ± 0.04	0.72 ± 0.23	0.17 ± 0.13	0.15 ± 0.30	0.81 ± 0.19	0.99 ± 0.01	0.96 ± 0.01	0.92 ± 0.05	0.94 ± 0.01
corridor	0.44 ± 0.20	0.80 ± 0.08	0.82 ± 0.25	0.00	0.00	0.00	0.68 ± 0.34	0.44 ± 0.37	0.53 ± 0.29
MMM2	0.27 ± 0.08	0.14 ± 0.17	0.15 ± 0.15	0.00	0.00	0.01 ± 0.01	0.73 ± 0.07	0.89 ± 0.04	0.89 ± 0.04
3s_vs_5z	0.67 ± 0.17	0.00	0.72 ± 0.43	0.00	0.00	0.00	0.41 ± 0.43	0.62 ± 0.31	0.43 ± 0.37

Table B.2: Maximum win-rate and 95% confidence interval over five seeds for all nine algorithms without parameter sharing in all SMAC tasks.

Tasks \Algs.	IDQN	IA2C	IPPO	MADDPG	COMA	MAA2C	MAPPO	VDN	QMIX
2s_vs_1sc	0.49 ± 0.12	1.00	0.99 ± 0.01	0.06 ± 0.10	0.79 ± 0.40	0.96 ± 0.04	1.00	0.64 ± 0.11	0.84 ± 0.02
3s5z	0.48 ± 0.23	0.27 ± 0.25	0.27 ± 0.25	0.13 ± 0.09	0.91 ± 0.08	0.87 ± 0.07	0.94 ± 0.03	0.81 ± 0.06	0.70 ± 0.11
corridor	0.05 ± 0.06	0.31 ± 0.39	0.17 ± 0.30	0.00	0.01 ± 0.02	0.06 ± 0.12	0.01 ± 0.01	0.08 ± 0.11	0.40 ± 0.32
MMM2	0.04 ± 0.08	0.06 ± 0.13	0.00	0.00	0.00	0.00	0.00	0.58 ± 0.04	0.23 ± 0.16
3s_vs_5z	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.11 ± 0.22	0.23 ± 0.24

B.4 Learning Curves in All Tasks

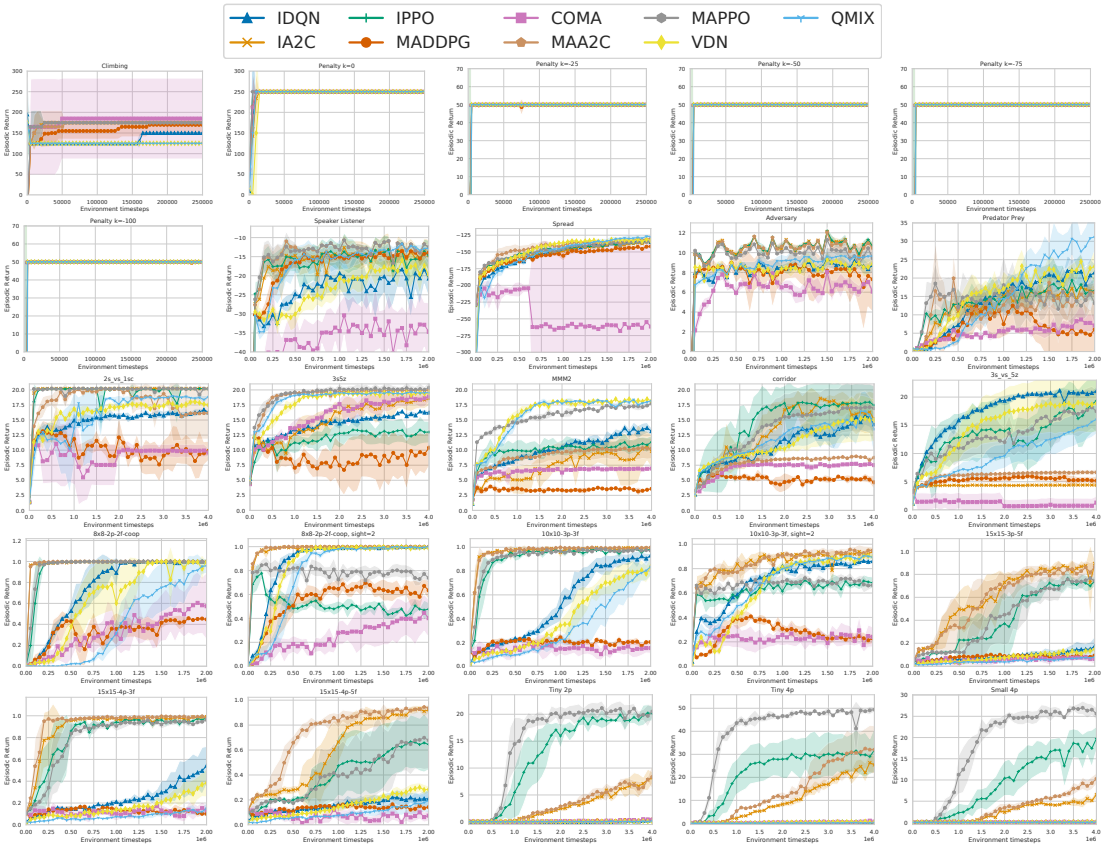


Figure B.2: Episodic returns of all algorithms with parameter sharing in all environments showing the mean and the 95% confidence interval over five different seeds.

B.5 Hyperparameter Optimisation

The parameters of each algorithms are optimised for each environment in one of its tasks and are kept constant for the rest of the tasks within the same environment. Each combination of hyperparameters is evaluated for three different seeds. The combination of hyperparameters that achieved the maximum evaluation, averaged over the three seeds, is used for producing the results presented in this work. Table B.3 presents the range of hyperparameters we evaluated in each environment, on the respective applicable algorithms. In general, all algorithms were evaluated in approximately the same number of hyperparameter combination for each environment to ensure consistency. To reduce the computational cost, the hyperparameter search was limited in SMAC compared to the other environments. However, several of the evaluated algorithms have been previously evaluated in SMAC and their best hyperparameters are publicly available in their

Table B.3: Range of hyperparameters that was evaluated in each environment. N/A means that this hyperparameter was not optimised, and that we used one that was either proposed in the original paper or was found to be the best in the rest of the environments. If only one value is presented it means that this hyperparameter was used for all algorithms in this task.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64/128	64/128	64/128	64/128	64/128
learning rate	0.0001/0.0003/0.0005	0.0001/0.0003/0.0005	0.0005	0.0001/0.0003/0.0005	0001/0.0003/0.0005
reward standardisation	True/False	True/False	True/False	True/False	True/False
network type	FC	FC/GRU	FC/GRU	FC/GRU	FC/GRU
evaluation epsilon	0.0/0.05	0.0/0.05	0.0/0.05	0.0/0.05	0.0/0.05
epsilon anneal	50,000/200,000	50,000/200,000	50,000	50,000/200,000	50,000/200,000
target update	200(hard)/0.01(soft)	200(hard)/0.01(soft)	N/A	200(hard)/0.01(soft)	200(hard)/0.01(soft)
entropy coefficient	0.01/0.001	0.01/0.001	N/A	0.01/0.001	0.01/0.001
n-step	5/10	5/10	N/A	5/10	5/10

respective papers.

B.6 Selected Hyperparameters

In this section we present the hyperparameters used in each task. In the off-policy algorithms we use an experience replay to break the correlation between consecutive samples (Lin, 1992b; Mnih et al., 2015). In the on-policy algorithms we use parallel synchronous workers to break the correlation between consecutive samples (Mnih et al., 2015). The size of the experience replay is either 5K episodes or 1M samples, depending on which is smaller in terms of used memory. Exploration in Q-based algorithms is done with epsilon-greedy, starting with $\epsilon = 1$ and linearly reducing it to 0.05. Additionally, in Q-based algorithms we select action with epsilon-greedy (with a small epsilon value) to ensure that the agents are not stuck. The evaluation epsilon is the hyperparameter that is optimised during the hyperparameter optimisation, with possible values between 0 and 0.05. In the stochastic policy algorithms, we perform exploration by sampling their categorical policy. During execution, in the stochastic policy algorithms, we sample their policy instead of computing the action that maximises the policy. The computation of the temporal difference targets is done using the Double Q-learning (van Hasselt, 2010) update rule. In IPPO and MAPPO the number of update epochs per training batch is 4 and the clipping value of the surrogate objective is 0.2.

Table B.4: Hyperparameters for IDQN with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	128	128	64
learning rate	0.0003	0.0005	0.0005	0.0003	0.0005
reward standardisation	True	True	False	True	True
network type	FC	FC	GRU	GRU	FC
evaluation epsilon	0.0	0.0	0.05	0.05	0.05
epsilon anneal	50,000	200,000	50,000	200,000	50,000
target update	200 (hard)	0.01 (soft)	200 (hard)	200 (hard)	0.01 (soft)

Table B.5: Hyperparameters for IDQN without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	128	64	64	64
learning rate	0.0001	0.0005	0.0005	0.0003	0.0005
reward standardisation	True	True	True	True	True
network type	FC	FC	GRU	GRU	FC
evaluation epsilon	0.0	0.0	0.05	0.05	0.05
epsilon anneal	50,000	200,000	50,000	50,000	50,000
target update	0.01 (soft)	0.01 (soft)	200 (hard)	200 (hard)	0.01 (soft)

Table B.6: Hyperparameters for IA2C with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	64	128	128	64
learning rate	0.0005	0.0005	0.0005	0.0005	0.0005
reward standardisation	True	True	True	True	True
network type	FC	GRU	FC	GRU	FC
entropy coefficient	0.01	0.01	0.01	0.001	0.01
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)
n-step	5	5	5	5	5

Table B.7: Hyperparameters for IA2C without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	64	64	64
learning rate	0.0001	0.0005	0.0005	0.0005	0.0005
reward standardisation	True	True	True	True	True
network type	FC	FC	FC	GRU	FC
entropy coefficient	0.01	0.01	0.01	0.01	0.01
target update	200 (hard)	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)
n-step	5	10	5	5	5

Table B.8: Hyperparameters for IPPO with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	64	128	128	128
learning rate	0.0005	0.0003	0.0005	0.0003	0.0005
reward standardisation	True	True	False	False	False
network type	FC	GRU	GRU	FC	GRU
entropy coefficient	0.001	0.01	0.001	0.001	0.001
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	200 (hard)	0.01 (soft)
n-step	5	5	10	5	10

Table B.9: Hyperparameters for IPPO without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	128	64	128	128
learning rate	0.0005	0.0001	0.0005	0.0001	0.0005
reward standardisation	True	True	True	False	False
network type	FC	FC	FC	GRU	FC
entropy coefficient	0.001	0.01	0.001	0.001	0.001
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	200 (hard)	0.01 (soft)
n-step	5	10	10	5	10

Table B.10: Hyperparameters for MADDPG with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	128	64	64
learning rate	0.0003	0.0005	0.0005	0.0003	0.0005
reward standardisation	True	True	False	True	False
network type	FC	GRU	GRU	FC	FC
actor regularisation	0.001	0.001	0.01	0.001	0.001
target update	200 (hard)	200 (hard)	0.01 (soft)	200 (hard)	0.01 (soft)

Table B.11: Hyperparameters for MADDPG without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	128	64	64
learning rate	0.0005	0.0005	0.0005	0.0003	0.0005
reward standardisation	True	True	True	True	False
network type	FC	GRU	FC	FC	FC
actor regularisation	0.001	0.01	0.001	0.001	0.001
target update	200 (hard)	0.01 (soft)	0.01 (soft)	200 (hard)	0.01 (soft)

Table B.12: Hyperparameters for COMA with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	64	128	128	64
learning rate	0.0005	0.0003	0.0005	0.0001	0.0005
reward standardisation	True	True	True	True	True
network type	FC	GRU	FC	GRU	FC
entropy coefficient	0.01	0.001	0.01	0.001	0.01
target update	0.01 (soft)	200 (hard)	0.01 (soft)	200 (hard)	0.01 (soft)
n-step	5	10	5	10	5

Table B.13: Hyperparameters for COMA without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	128	128	64
learning rate	0.0003	0.0005	0.0005	0.0001	0.0005
reward standardisation	True	True	True	True	False
network type	FC	GRU	GRU	GRU	FC
entropy coefficient	0.01	0.01	0.01	0.001	0.01
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)
n-step	10	10	5	5	5

Table B.14: Hyperparameters for MAA2C with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	128	128	64
learning rate	0.003	0.0005	0.0005	0.0005	0.0005
reward standardisation	True	True	True	True	True
network type	FC	GRU	FC	GRU	FC
entropy coefficient	0.001	0.01	0.01	0.01	0.01
target update	0.01	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)
n-step	10	5	5	10	5

Table B.15: Hyperparameters for MAA2C without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	128	128	128	64
learning rate	0.0005	0.0003	0.0005	0.0005	0.0005
reward standardisation	True	True	True	True	True
network type	FC	GRU	FC	GRU	FC
entropy coefficient	0.001	0.01	0.01	0.01	0.01
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)
n-step	10	5	5	5	5

Table B.16: Hyperparameters for MAPPO with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	64	64	128	128
learning rate	0.0005	0.0005	0.0005	0.0003	0.0005
reward standardisation	True	True	False	False	False
network type	FC	FC	GRU	FC	FC
entropy coefficient	0.001	0.01	0.001	0.001	0.001
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)
n-step	5	5	10	5	10

Table B.17: Hyperparameters for MAPPO without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	128	64	128	128
learning rate	0.0005	0.0001	0.0005	0.0001	0.0005
reward standardisation	True	True	True	False	False
network type	FC	FC	GRU	FC	FC
entropy coefficient	0.001	0.01	0.001	0.001	0.001
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	200 (hard)	0.01 (soft)
n-step	5	5	10	10	10

Table B.18: Hyperparameters for VDN with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	128	128	128	64
learning rate	0.0001	0.0005	0.0005	0.0003	0.0005
reward standardisation	True	True	True	True	True
network type	FC	FC	GRU	GRU	FC
evaluation epsilon	0.0	0.0	0.05	0.0	0.05
epsilon anneal	200,000	50,000	50,000	200,000	50,000
target update	0.01 (soft)	200 (hard)	200 (hard)	0.01 (soft)	0.01 (soft)

Table B.19: Hyperparameters for VDN without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	64	64	64
learning rate	0.0005	0.0005	0.0005	0.0001	0.0005
reward standardisation	True	True	True	True	True
network type	FC	FC	GRU	GRU	FC
evaluation epsilon	0.0	0.0	0.05	0.05	0.05
epsilon anneal	50,000	50,000	50,000	50,000	50,000
target update	0.01 (soft)	200 (hard)	200 (hard)	200 (hard)	0.01 (soft)

Table B.20: Hyperparameters for QMIX with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	64	128	64	64
learning rate	0.0003	0.0005	0.005	0.0003	0.0005
reward standardisation	True	True	True	True	True
network type	FC	GRU	GRU	GRU	FC
evaluation epsilon	0.0	0.0	0.05	0.05	0.05
epsilon anneal	200,000	200,000	50,000	200,000	50,000
target update	0.01 (soft)	0.01 (soft)	200 (hard)	0.01 (soft)	0.01 (soft)

Table B.21: Hyperparameters for QMIX without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	64	64	64
learning rate	0.0005	0.0003	0.0005	0.0001	0.0003
reward standardisation	True	True	True	True	True
network type	FC	GRU	GRU	GRU	FC
evaluation epsilon	0.0	0.0	0.05	0.05	0.05
epsilon anneal	50,000	200,000	50,000	50,000	50,000
target update	0.01 (soft)	0.01 (soft)	200 (hard)	0.01 (soft)	0.01 (soft)

Appendix C

Ensemble Value Functions for Multi-Agent Exploration

C.1 Hyperparameter Optimisation

For IDQN, VDN, QMIX and extensions with EMAX, we conduct a gridsearch to identify best hyperparameters in one selected task within each environment by evaluating each algorithm configuration for three runs and selecting the hyperparameter configuration which led to highest average evaluation returns throughout training. We largely based our configurations on the reported hyperparameters from Papoudakis et al. (2021) with minimal hyperparameter tuning. Our implementation of IDQN, VDN, QMIX, and EMAX are based on the EPyMARL codebase[1]. For the baseline of MAVEN, CDS, and EMC, we migrated the provided codebase from the authors[2] into EPyMARL to support all environments. For MAVEN, CDS, and EMC, we use the hyperparameters identified for QMIX for each environment with the algorithm-specific hyperparameters provided by the authors. For IPPO and MAPPO, we use the best identified hyperparameters reported in Papoudakis et al. (2021).

[1]Available at `https://github.com/uoe-agents/epymarl`.

[2]Available at `https://github.com/AnujMahajanOxf/MAVEN`, `https://github.com/lich14/CDS` and `https://github.com/kikojay/EMC`.

Table C.1: Hyperparameters for IDQN, VDN, QMIX and extensions with EMAX in LBF. The gridsearch was conducted in Foraging-10x10-4p-3f-coop for 4M time steps, and the bold entries corresponding to the best identified configuration.

Algorithm	Hyperparameter	Value
Shared	γ	0.99
	Activation function	ReLU
	Parameter sharing	True
	Optimiser	Adam
	Maximum gradient norm	5
	Minimum ϵ	0.05
	Evaluation ϵ	0.05
	Learning rate	e^{-4}
	Target update frequency	200
	Replay buffer capacity (episodes)	5,000
	Batch size (episodes)	32
QMIX	Mixing embedding size	32
	Hypernetwork embedding size	64
IDQN	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	ϵ decay steps	**50,000**, 200,000
VDN	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	ϵ decay steps	50,000, **200,000**
QMIX	Network architecture	**FC**, FC + GRU
	Network size	64, **128**
	Reward standardisation	False, **True**
	ϵ decay steps	50,000, **200,000**
IDQN-EMAX	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	UCB uncertainty coefficient β	0.1, 0.3, **1**
VDN-EMAX	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	UCB uncertainty coefficient β	**0.1**, 0.3, 1
QMIX-EMAX	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	UCB uncertainty coefficient β	0.1, **0.3**, 1

Table C.2: Hyperparameters for IDQN, VDN, QMIX and extensions with EMAX in BPUSH. The gridsearch was conducted in BPUSH 12×12 2ag for 7.5M time steps, and the bold entries corresponding to the best identified configuration.

Algorithm	Hyperparameter	Value
Shared	γ	0.99
	Activation function	ReLU
	Parameter sharing	True
	Optimiser	Adam
	Maximum gradient norm	5
	Minimum ϵ	0.05
	Evaluation ϵ	0.05
	Learning rate	e^{-4}
	Target update frequency	200
	Replay buffer capacity (episodes)	5,000
	Batch size (episodes)	32
QMIX	Mixing embedding size	32
	Hypernetwork embedding size	64
IDQN	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	ϵ decay steps	**50,000**, 200,000
VDN	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	ϵ decay steps	50,000, **200,000**
QMIX	Network architecture	**FC**, FC + GRU
	Network size	64, **128**
	Reward standardisation	False, **True**
	ϵ decay steps	50,000, **200,000**
IDQN-EMAX	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	UCB uncertainty coefficient β	0.1, 0.3, **1**
VDN-EMAX	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	UCB uncertainty coefficient β	**0.1**, 0.3, 1
QMIX-EMAX	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	UCB uncertainty coefficient β	0.1, **0.3**, 1

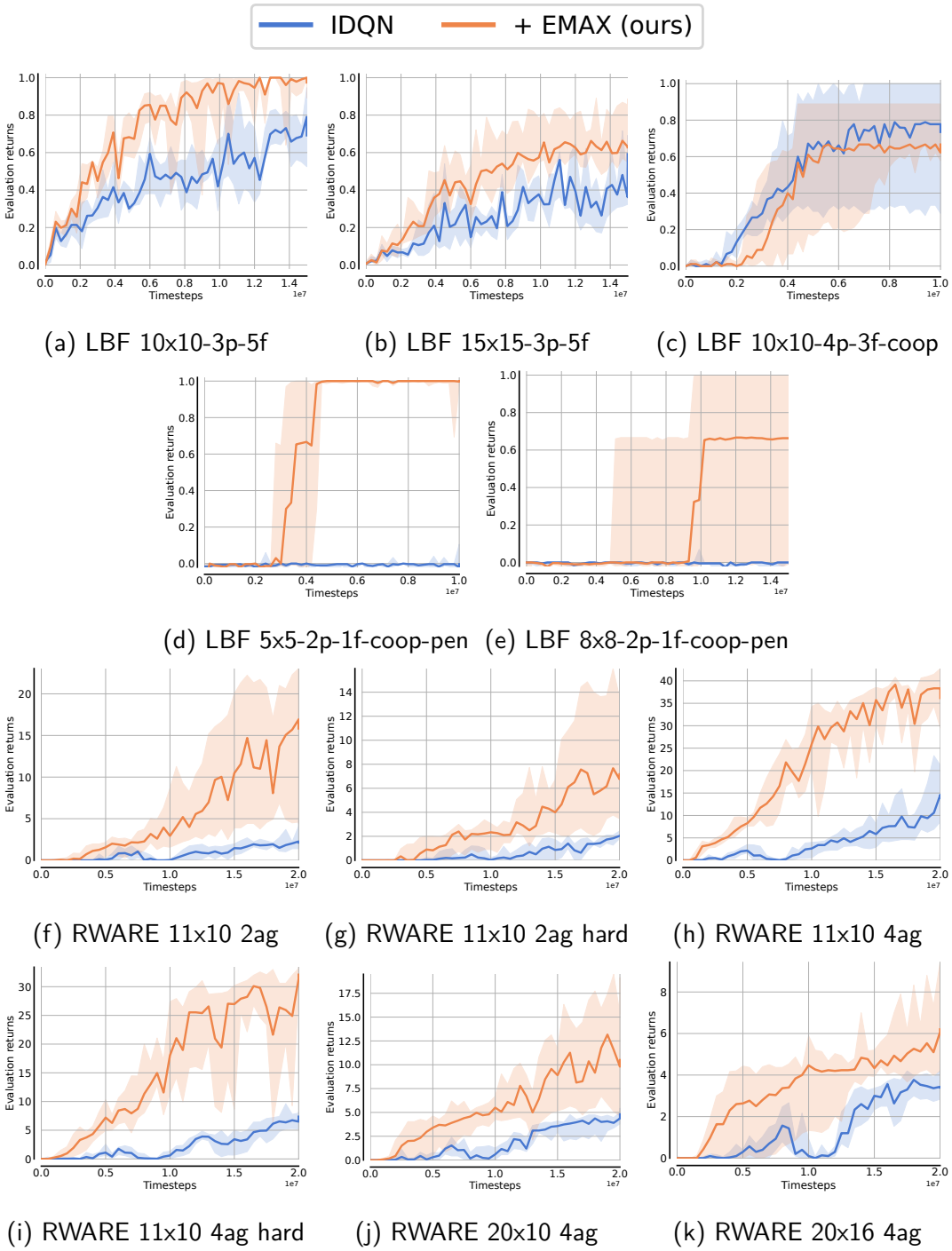
Table C.3: Hyperparameters for IDQN, VDN, QMIX and extensions with EMAX in RWARE. The gridsearch was conducted in RWARE 11×10 4ag for 5M time steps, and the bold entries corresponding to the best identified configuration.

Algorithm	Hyperparameter	Value
Shared	γ	0.99
	Activation function	ReLU
	Parameter sharing	True
	Optimiser	Adam
	Maximum gradient norm	5
	Minimum ϵ	0.05
	Evaluation ϵ	0.05
	Learning rate	e^{-4}
	Target update frequency	200
	Replay buffer capacity (episodes)	5,000
	Batch size (episodes)	32
QMIX	Mixing embedding size	32
	Hypernetwork embedding size	64
IDQN	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	ϵ decay steps	**50,000**, 200,000
VDN	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	ϵ decay steps	**50,000**, 200,000
QMIX	Network architecture	**FC**, FC + GRU
	Network size	64, **128**
	Reward standardisation	False, **True**
	ϵ decay steps	**50,000**, 200,000
IDQN-EMAX	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	UCB uncertainty coefficient β	0.1, **0.3**, 1
VDN-EMAX	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	UCB uncertainty coefficient β	0.1, **0.3**, 1
QMIX-EMAX	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	UCB uncertainty coefficient β	0.1, **0.3**, 1

Table C.4: Hyperparameters for IDQN, VDN, QMIX and extensions with EMAX in MPE. The gridsearch was conducted in Spread for 1M time steps, and the bold entries corresponding to the best identified configuration.

Algorithm	Hyperparameter	Value
Shared	γ	0.99
	Activation function	ReLU
	Parameter sharing	True
	Optimiser	Adam
	Maximum gradient norm	5
	Minimum ϵ	0.05
	Evaluation ϵ	0.05
	Learning rate	e^{-4}
	Target update frequency	200
	Replay buffer capacity (episodes)	5,000
	Batch size (episodes)	32
QMIX	Mixing embedding size	32
	Hypernetwork embedding size	64
IDQN	Network architecture	**FC**, FC + GRU
	Network size	64, **128**
	Reward standardisation	False, **True**
	ϵ decay steps	**50,000**, 200,000
VDN	Network architecture	**FC**, FC + GRU
	Network size	64, **128**
	Reward standardisation	False, **True**
	ϵ decay steps	**50,000**, 200,000
QMIX	Network architecture	**FC**, FC + GRU
	Network size	64, **128**
	Reward standardisation	False, **True**
	ϵ decay steps	**50,000**, 200,000
IDQN-EMAX	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	UCB uncertainty coefficient β	0.1, 0.3, **1**
VDN-EMAX	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	UCB uncertainty coefficient β	**0.1**, 0.3, 1
QMIX-EMAX	Network architecture	FC, **FC + GRU**
	Network size	64, **128**
	Reward standardisation	False, **True**
	UCB uncertainty coefficient β	0.1, **0.3**, 1

C.2 Individual Task Evaluation Returns in Mixed-Objective Tasks



(a) LBF 10x10-3p-5f (b) LBF 15x15-3p-5f (c) LBF 10x10-4p-3f-coop

(d) LBF 5x5-2p-1f-coop-pen (e) LBF 8x8-2p-1f-coop-pen

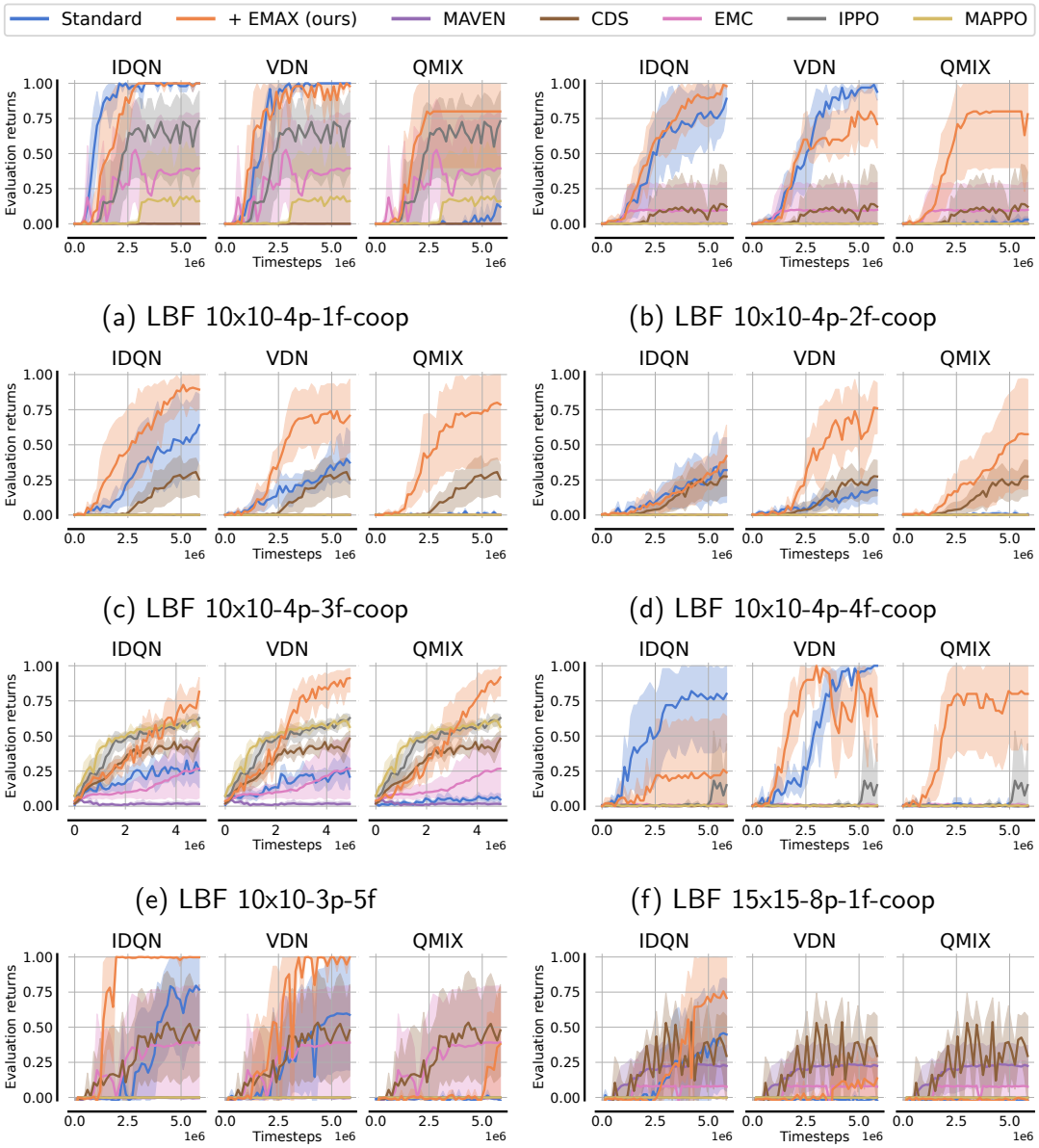
(f) RWARE 11x10 2ag (g) RWARE 11x10 2ag hard (h) RWARE 11x10 4ag

(i) RWARE 11x10 4ag hard (j) RWARE 20x10 4ag (k) RWARE 20x16 4ag

Figure C.1: Average evaluation returns and 95% confidence intervals for IDQN with and without EMAX in mixed-objective LBF and RWARE tasks.

C.3 Individual Task Evaluation Returns in Cooperative Tasks

C.3.1 Level-Based Foraging



(a) LBF 10x10-4p-1f-coop

(b) LBF 10x10-4p-2f-coop

(c) LBF 10x10-4p-3f-coop

(d) LBF 10x10-4p-4f-coop

(e) LBF 10x10-3p-5f

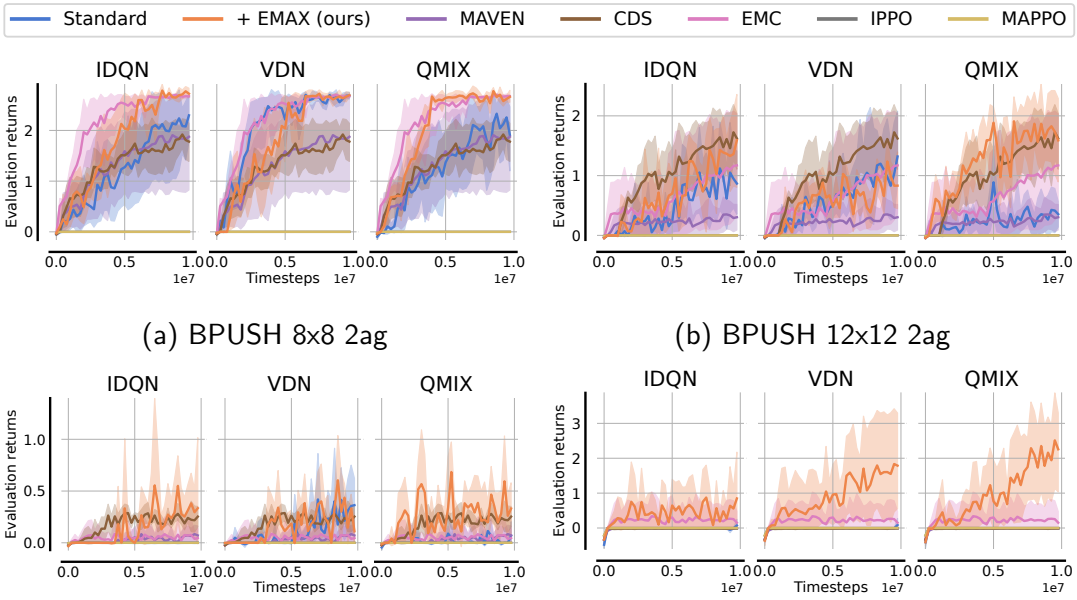
(f) LBF 15x15-8p-1f-coop

(g) LBF 5x5-2p-1f-coop-pen

(h) LBF 5x5-2p-2f-coop-pen

Figure C.2: Average evaluation returns and 95% confidence intervals for all algorithms in LBF tasks.

C.3.2 Boulder-Push



(a) BPUSH 8x8 2ag

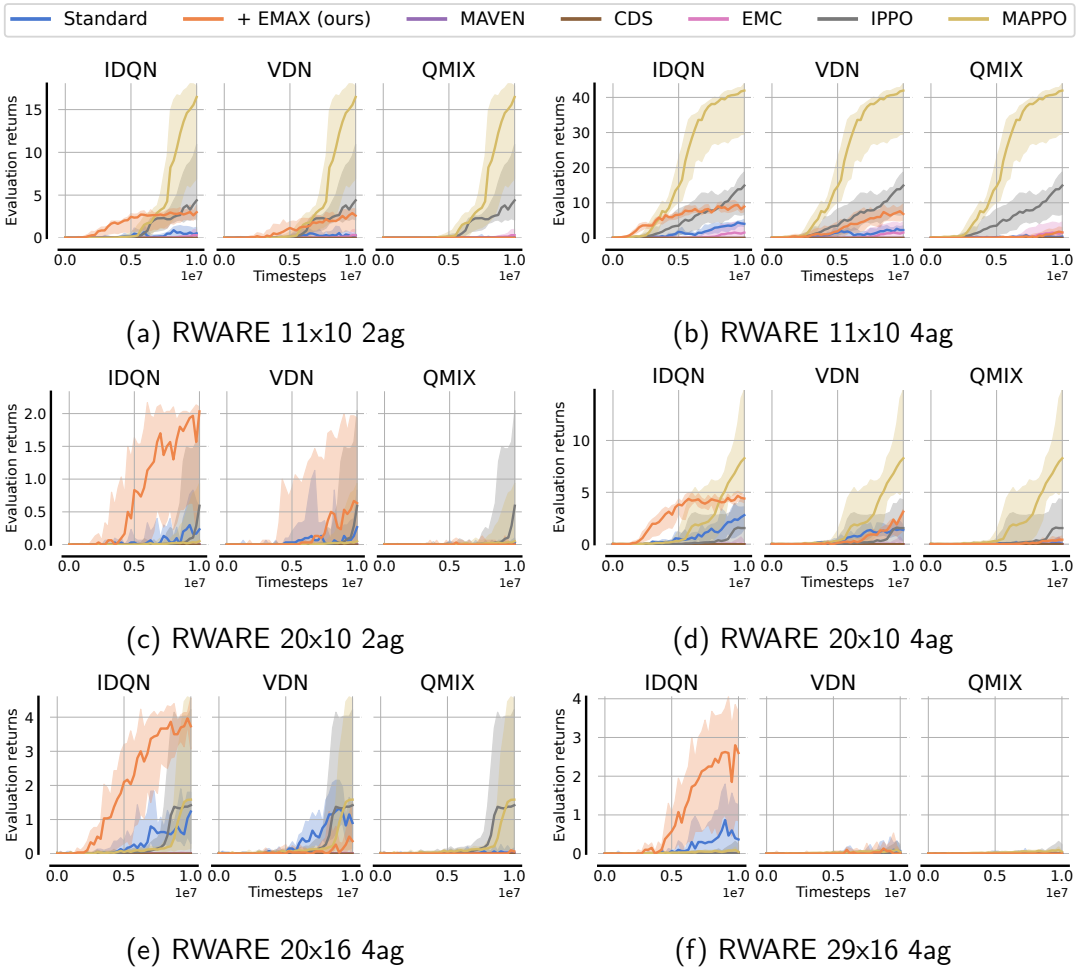
(b) BPUSH 12x12 2ag

(c) BPUSH 20x20 2ag

(d) BPUSH 5x5 4ag

Figure C.3: Average evaluation returns and 95% confidence intervals for all algorithms in BPUSH tasks.

C.3.3 Multi-Robot Warehouse



(a) RWARE 11x10 2ag

(b) RWARE 11x10 4ag

(c) RWARE 20x10 2ag

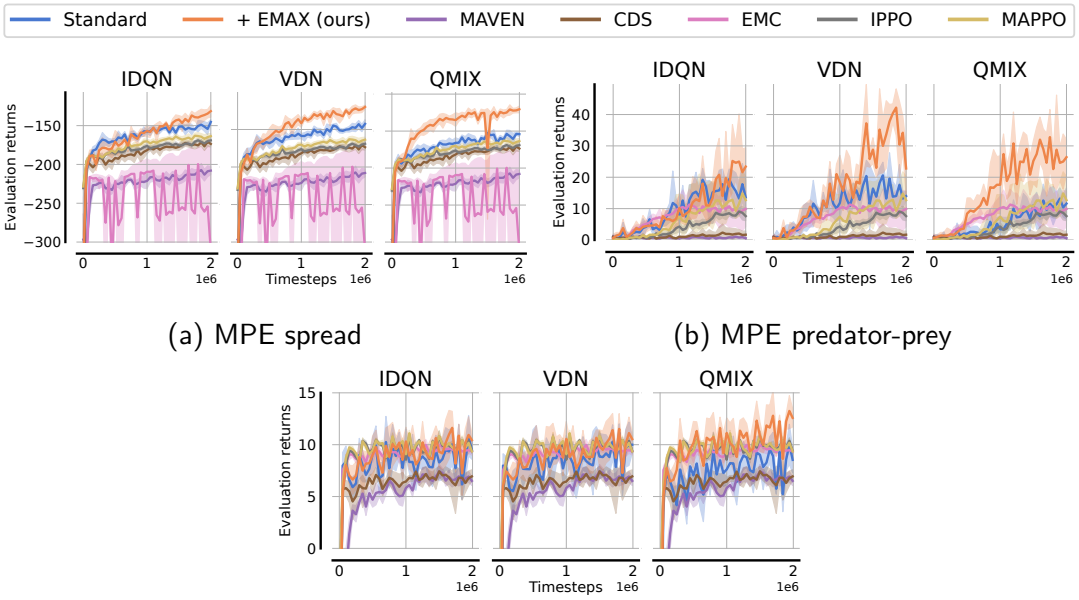
(d) RWARE 20x10 4ag

(e) RWARE 20x16 4ag

(f) RWARE 29x16 4ag

Figure C.4: Average evaluation returns and 95% confidence intervals for all algorithms in RWARE tasks.

C.3.4 Multi-Agent Particle Environment



(a) MPE spread

(b) MPE predator-prey

(c) MPE adversary

Figure C.5: Average evaluation returns and 95% confidence intervals for all algorithms in MPE tasks.

Bibliography

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. (2021). Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*.

Agogino, A. K. and Tumer, K. (2008). Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *International Conference on Autonomous Agents and Multi-Agent Systems*.

Ahilan, S. and Dayan, P. (2019). Feudal multi-agent hierarchies for cooperative reinforcement learning. In *International Conference on Learning Representations Workshop on Structure & Priors in Reinforcement Learning*.

Akkaya, I., Andrychowicz, M., Chociej, M., teusz Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N. A., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. M. (2019). Solving rubik's cube with a robot hand. *ArXiv preprint 1910.07113*.

Albrecht, S. V., Christianos, F., and Schäfer, L. (2024). *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press.

Albrecht, S. V. and Ramamoorthy, S. (2013). A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. In *International Conference on Autonomous Agents and Multi-Agent Systems*.

Albrecht, S. V. and Stone, P. (2017). Reasoning about hypothetical agent behaviours and their parameters. In *International Conference on Autonomous Agents and Multi-Agent Systems*.

Andoni, A. and Indyk, P. (2008). Near-optimal hashing algorithms for approx-

imate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., Gelly, S., and Bachem, O. (2021). What matters in on-policy reinforcement learning? A large-scale empirical study. In *International Conference on Learning Representations*.

Anschel, O., Baram, N., and Shimkin, N. (2017). Averaged-DQN: Variance reduction and stabilization for deep reinforcement learning. In *International Conference on Machine Learning*.

Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256.

Azadeh, K., Roy, D., De Koster, R., and Khalilabadi, S. M. G. (2023). Zoning strategies for humanrobot collaborative picking. *Decision Sciences*.

Azagirre, X., Balwally, A., Candeli, G., Chamandy, N., Han, B., King, A., Lee, H., Loncaric, M., Martin, S., Narasiman, V., Qin, Z., Richard, B., Smoot, S., Taylor, S., Ryzin, G., Wu, D., Yu, F., and Zamoshchin, A. (2024). A better match for drivers and riders: Reinforcement learning at Lyft. *Informs Journal on Applied Analytics*, 54:71–83.

Bakhtin, A., Brown, N., Dinan, E., Farina, G., Flaherty, C., Fried, D., Goff, A., Gray, J., Hu, H., Jacob, A. P., Komeili, M., Konath, K., Kwon, M., Lerer, A., Lewis, M., Miller, A. H., Mitts, S., Renduchintala, A., Roller, S., Rowe, D., Shi, W., Spisak, J., Wei, A., Wu, D. J., Zhang, H., and Zijlstra, M. (2022). Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science*, 378:1067 – 1074.

Barto, A. G. (2013). Intrinsic motivation and reinforcement learning. In *Intrinsically motivated learning in natural and artificial systems*, pages 17–47. Springer.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*.

Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., and Wang, Z. (2020). Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82.

Bellman, R. (1957). A Markovian decision process. *Indiana University Mathematics Journal*, 6:679–684.

Bellman, R. (1966). Dynamic programming. *Science*, 153(3731):34–37.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., J ózefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pondé de Oliveira Pinto, H., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint 1912.06680*.

Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*.

Böhmer, W., Kurin, V., and Whiteson, S. (2020). Deep coordination graphs. In *International Conference on Machine Learning*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems*.

Brys, T., Harutyunyan, A., Suay, H. B., Chernova, S., Taylor, M. E., and Nowé, A. (2015). Reinforcement learning from demonstration through shaping. In *International Joint Conference on Artificial Intelligence*.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2019a). Large-scale study of curiosity-driven learning. In *International Conference on Learning Representations*.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2019b). Exploration by random network distillation. In *International Conference on Learning Representations*.

Chan, S. C., Fishman, S., Canny, J., Korattikara, A., and Guadarrama, S. (2020). Measuring the reliability of reinforcement learning algorithms. In *International Conference on Learning Representations*.

Chapelle, O. and Li, L. (2011). An empirical evaluation of Thompson sampling. In *Advances in Neural Information Processing Systems*.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Annual ACM Symposium on Theory of Computing*.

Chen, E., Hong, Z.-W., Pajarinen, J., and Agrawal, P. (2022). Redeeming intrinsic rewards via constrained optimization. In *Advances in Neural Information Processing Systems*.

Cho, K., Van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*.

Christianos, F., Papoudakis, G., and Albrecht, S. V. (2023). Pareto actor-critic for equilibrium selection in multi-agent reinforcement learning. *Transactions on Machine Learning Research*.

Christianos, F., Papoudakis, G., Rahman, M. A., and Albrecht, S. V. (2021). Scaling multi-agent reinforcement learning with selective parameter sharing. In *International Conference on Machine Learning*.

Christianos, F., Schäfer, L., and Albrecht, S. V. (2020). Shared experience actor-critic for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*.

Ciosek, K., Vuong, Q., Loftin, R., and Hofmann, K. (2019). Better exploration with optimistic actor critic. In *Advances in Neural Information Processing Systems*.

Claes, D., Oliehoek, F., Baier, H., Tuyls, K., et al. (2017). Decentralised online planning for multi-robot warehouse commissioning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 492–500.

Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI Conference on Artificial Intelligence*.

Clouse, J. A. (1996). Learning from an automated training agent. *Adaptation and Learning in Multiagent Systems*.

Covington, P., Adams, J., and Sargin, E. (2016). Deep neural networks for youtube recommendations. In *ACM Conference on Recommender Systems*, pages 191–198.

Da Silva, F. L., Glatt, R., and Costa, A. H. R. (2017). Simultaneously learning and advising in multiagent reinforcement learning. In *International Conference on Autonomous Agents and Multi-Agent Systems*.

Da Silva, F. L., Taylor, M. E., and Costa, A. H. R. (2018). Autonomously reusing knowledge in multiagent reinforcement learning. In *International Joint Conference on Artificial Intelligence*.

Da Silva, F. L., Warnell, G., Costa, A. H. R., and Stone, P. (2020). Agents teaching agents: A survey on inter-agent transfer learning. *Autonomous Agents and Multi-Agent Systems*, 34(1).

Damani, M., Luo, Z., Wenzel, E., and Sartoretti, G. (2020). PRIMAL$_2$: Pathfinding via reinforcement and imitation multi-agent learning - lifelong. *Robotics and Automation Letters*, 6:2666–2673.

Dayan, P. and Hinton, G. E. (1992). Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*.

de Witt, C. S., Gupta, T., Makoviichuk, D., Makoviychuk, V., Torr, P. H., Sun, M., and Whiteson, S. (2020). Is independent learning all you need in the StarCraft multi-agent challenge? *arXiv preprint 2011.09533*.

Dearden, R., Friedman, N., and Russell, S. (1998). Bayesian Q-learning. *AAAI*.

Degris, T., White, M., and Sutton, R. S. (2012). Off-policy actor-critic. In *International Conference on Machine Learning*.

Dematic (2024). Autostore storage and retrieval system. accessed: 2024-02-24.

Dematic (2024). Dematic multishuttle 2. accessed: 2024-02-24.

Dennis, M., Jaques, N., Vinitsky, E., Bayen, A., Russell, S., Critch, A., and Levine, S. (2020). Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in Neural Information Processing Systems*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint 1810.04805*.

Devlin, S. M. and Kudenko, D. (2012). Dynamic potential-based reward shaping. In *International Conference on Autonomous Agents and Multiagent Systems*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Du, Y., Han, L., Fang, M., Liu, J., Dai, T., and Tao, D. (2019). LIIR: Learning individual intrinsic reward in multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*.

Dwaracherla, V., Lu, X., Ibrahimi, M., Osband, I., Wen, Z., and van Roy, B. (2020). Hypermodels for exploration. In *International Conference on Learning Representations*.

Espeholt, L., Marinier, R., Stanczyk, P., Wang, K., and Michalski, M. (2020). SEED RL: Scalable and efficient deep-RL with accelerated central inference. In *International Conference on Learning Representations*.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. (2018). IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *International Conference on Machine Learning*.

Fachantidis, A., Taylor, M. E., and Vlahavas, I. (2019). Learning to teach reinforcement learning agents. *Machine Learning and Knowledge Extraction*, 1(1):21–42.

Flet-Berliac, Y., Ferret, J., Pietquin, O., Preux, P., and Geist, M. (2021). Adversarially guided actor-critic. In *International Conference on Learning Representations*.

Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*.

Fu, H., Yu, S., Littman, M., and Konidaris, G. (2022a). Model-based lifelong reinforcement learning with Bayesian exploration. In *Advances in Neural Information Processing Systems*.

Fu, W., Yu, C., Xu, Z., Yang, J., and Wu, Y. (2022b). Revisiting some common practices in cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

Gallici, M., Fellows, M., Ellis, B., Pou, B., Masmitja, I., Foerster, J. N., and Martin, M. (2024). Simplifying deep temporal difference learning. *arXiv preprint 2407.04811*.

Gao, Y., Xu, H., Lin, J., Yu, F., Levine, S., and Darrell, T. (2018). Reinforcement learning from imperfect demonstrations. In *International Conference on Machine Learning*.

Gerstgrasser, M., Danino, T., and Keren, S. (2024). Selectively sharing experiences improves multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep Learning*, volume 1. MIT Press.

Greshler, N., Gordon, O., Salzman, O., and Shimkin, N. (2021). Cooperative multi-agent path finding: Beyond path planning and collision avoidance. In *International Symposium on Multi-Robot and Multi-Agent Systems*.

Guestrin, C., Koller, D., and Parr, R. (2001). Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems*.

Guestrin, C., Lagoudakis, M. G., and Parr, R. (2002). Coordinated reinforcement learning. In *International Conference on Machine Learning*.

Gupta, J. K., Egorov, M., and Kochenderfer, M. (2017). Cooperative multi-agent control using deep reinforcement learning. In *Autonomous Agents and Multiagent Systems Workshops*, pages 66–83. Springer.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*.

Hansen, E. A., Bernstein, D. S., and Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *AAAI Conference on Artificial Intelligence*.

Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Hausknecht, M. and Stone, P. (2015). Deep recurrent Q-learning for partially observable MDPs. In *AAAI Fall Symposium Series*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. (2017). Neural collaborative filtering. In *International Conference on World Wide Web*, pages 173–182.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *AAAI Conference on Artificial Intelligence*.

Hernandez-Leal, P., Kartal, B., and Taylor, M. E. (2019). A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Howard, R. A. (1964). *Dynamic programming and Markov processes*. New York: John-Wiley.

Huang, S. and Ontañón, S. (2022). A closer look at invalid action masking in policy gradient algorithms. In *International FLAIRS Conference*.

Iqbal, S. and Sha, F. (2019). Coordinated exploration via intrinsic rewards for multi-agent reinforcement learning. *arXiv preprint 1905.12127*.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castañeda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K., and Graepel, T. (2019). Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865.

Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*.

Janz, D., Hron, J., Mazur, P., Hofmann, K., Hernández-Lobato, J. M., and Tschiatschek, S. (2019). Successor uncertainties: Exploration and uncertainty in temporal difference learning. In *Advances in Neural Information Processing Systems*.

Jaques, N., Ghandeharioun, A., Shen, J. H., Ferguson, C., Lapedriza, A., Jones, N., Gu, S., and Picard, R. (2019). Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint 1907.00456*.

Jiang, M., Grefenstette, E., and Rocktäschel, T. (2021). Prioritized level replay. In *International Conference on Machine Learning*.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134.

Kang, W.-C. and McAuley, J. (2018). Self-attentive sequential recommendation. In *International Conference on Data Mining*, pages 197–206. IEEE.

Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.-M., Lam, V.-D., Bewley, A., and Shah, A. (2019). Learning to drive in a day. In *International Conference on Robotics and Automation*.

Kim, J.-B., Choi, H.-B., Hwang, G.-Y., Kim, K., Hong, Y.-G., and Han, Y.-H. (2020). Sortation control using multi-agent deep reinforcement learning in N-grid sortation system. *Sensors*, 20(12).

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Kok, J. R. and Vlassis, N. (2005). Using the max-plus algorithm for multiagent decision making in coordination graphs. In *Belgium-Netherlands Conference on Artificial Intelligence*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*.

Krnjaic, A., Steleac, R. D., Thomas, J. D., Papoudakis, G., Schäfer, L., To, A. W. K., Lao, K.-H., Cubuktepe, M., Haley, M., Börsting, P., and Albrecht,

S. V. (2024). Scalable multi-agent reinforcement learning for warehouse logistics with robotic and human co-workers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Pérolat, J., Silver, D., and Graepel, T. (2017). A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in neural information processing systems*.

Lee, K., Laskin, M., Srinivas, A., and Abbeel, P. (2021). SUNRISE: A simple unified framework for ensemble learning in deep reinforcement learning. In *International Conference on Machine Learning*.

Leibo, J. Z., Perolat, J., Hughes, E., Wheelwright, S., Marblestone, A. H., Duéñez-Guzman, E., Sunehag, P., Dunning, I., and Graepel, T. (2019). Malthusian reinforcement learning. In *International Joint Conference on Autonomous Agents and Multi-Agent Systems*.

Leroy, P., Morato, P. G., Pisane, J., Kolios, A., and Ernst, D. (2023). IMP-MARL: A suite of environments for large-scale infrastructure management planning via MARL. In *Advances in Neural Information Processing Systems, Track on Datasets and Benchmarks*.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint 2005.01643*.

Li, C., Wang, T., Wu, C., Zhao, Q., Yang, J., and Zhang, C. (2021a). Celebrating diversity in shared multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*.

Li, J., Tinka, A., Kiesel, S., Durham, J. W., Kumar, T. K. S., and Koenig, S. (2021b). Lifelong multi-agent path finding in large-scale warehouses. In *AAAI Conference on Artificial Intelligence*.

Li, Q., Gama, F., Ribeiro, A., and Prorok, A. (2020). Graph neural networks for decentralized multi-robot path planning. In *International Conference on Intelligent Robots and Systems*.

Li, W., Chen, H., Jin, B., Tan, W., Zha, H., and Wang, X. (2022). Multi-agent path finding with prioritized communication learning. In *International Conference on Robotics and Automation*.

Li, X., Luo, W., Yuan, M., Wang, J., Lu, J., Wang, J., Lu, J., and Zeng, J. (2021c). Learning to optimize industry-scale dynamic pickup and delivery problems. In *International Conference on Data Engineering*, pages 2511–2522. IEEE.

Liang, L., Xu, Y., McAleer, S., Hu, D., Ihler, A., Abbeel, P., and Fox, R. (2022). Reducing variance in temporal-difference value estimation via ensemble of deep networks. In *International Conference on Machine Learning*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *Conference on Learning Representations*.

Lin, L.-J. (1992a). *Reinforcement learning for robots using neural networks*. Carnegie Mellon University.

Lin, L.-J. (1992b). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning*, pages 157–163. Elsevier.

Liu, E. Z., Raghunathan, A., Liang, P., and Finn, C. (2021a). Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. In *International Conference on Machine Learning*.

Liu, I.-J., Jain, U., Yeh, R. A., and Schwing, A. (2021b). Cooperative exploration for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*.

Löffler, M., Boysen, N., and Schneider, M. (2022). Picker routing in AGV-assisted order picking systems. *Informs Journal on Computing*, 34(1):440–462.

Long, Q., Zhou, Z., Gupta, A., Fang, F., Wu, Y., and Wang, X. (2020). Evolutionary population curriculum for scaling multi-agent reinforcement learning. In *International Conference on Learning Representations*.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*.

Ma, H., Li, J., Kumar, T. K. S., and Koenig, S. (2017). Lifelong multi-agent path finding for online pickup and delivery tasks. In *International Conference on Autonomous Agents and Multiagent Systems*.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2017). The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*.

Mahajan, A., Rashid, T., Samvelyan, M., and Whiteson, S. (2019). MAVEN: Multi-agent variational exploration. In *Advances in Neural Information Processing Systems*.

Mark, M. S., Sharma, A., Tajwar, F., Rafailov, R., Levine, S., and Finn, C. (2023). Offline retraining for online RL: Decoupled policy learning to mitigate exploration bias. *arXiv preprint 2310.08558*.

McAleer, S., Farina, G., Lanctot, M., and Sandholm, T. (2023). ESCHER: Eschewing importance sampling in games by computing a history value function to estimate regret. In *International Conference on Learning Representations*.

McInroe, T., Jelley, A., Albrecht, S. V., and Storkey, A. (2024). Planning to go out-of-distribution in offline-to-online reinforcement learning. In *Reinforcement Learning Conference*.

Melo, F. S. (2001). Convergence of Q-learning: A simple proof. *Institute Of Systems and Robotics*, pages 1–4.

Mguni, D. H., Jafferjee, T., Wang, J., Perez-Nieves, N., Slumbers, O., Tong, F., Li, Y., Zhu, J., Yang, Y., and Wang, J. (2022). LIGS: Learnable intrinsic-reward generation selection for multi-agent learning. In *International Conference on Learning Representations*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Mordatch, I. and Abbeel, P. (2018). Emergence of grounded compositional language in multi-agent populations. In *AAAI Conference on Artificial Intelligence*.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. G. (2016). Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*.

Nash, J. (1951). Non-cooperative games. *Annals of mathematics*, pages 286–295.

Oliehoek, F. A. (2010). *Value-based planning for teams of agents in stochastic partially observable environments*. PhD thesis, University of Amsterdam.

Oliehoek, F. A. and Amato, C. (2016). *A Concise Introduction to Decentralized POMDPs*, volume 1. Springer.

Oliehoek, F. A., Spaan, M. T. J., and Vlassis, N. (2008). Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353.

Oliehoek, F. A., Whiteson, S., and Spaan, M. T. J. (2013). Approximate solutions for factored Dec-POMDPs with many agents. In *International Conference on Autonomous Agents and Multiagent Systems*.

Oliehoek, F. A., Witwicki, S. J., and Kaelbling, L. P. (2012). Influence-based abstraction for multiagent systems. In *AAAI Conference on Artificial Intelligence*.

Osband, I., Blundell, C., Pritzel, A., and van Roy, B. (2016a). Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems*.

Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvari, C., Singh, S., Roy, B. V., Sutton, R. S., Silver, D., and Hasselt, H. V. (2020). Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*.

Osband, I., Russo, D., and van Roy, B. (2013). (More) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*.

Osband, I. and van Roy, B. (2017). Why is posterior sampling better than optimism for reinforcement learning? In *International Conference on Machine Learning*.

Osband, I., van Roy, B., Russo, D. J., and Wen, Z. (2019). Deep exploration via randomized value functions. *Journal of Machine Learning Research*, 20(124).

Osband, I., van Roy, B., and Wen, Z. (2016b). Generalization and exploration via randomized value functions. In *International Conference on Machine Learning*.

Ostrovski, G., Bellemare, M. G., van den Oord, A., and Munos, R. (2017). Count-based exploration with neural density models. In *International Conference on Machine Learning*.

Oudeyer, P.-Y. and Kaplan, F. (2008). How can we define intrinsic motivation. In *Conference on Epigenetic Robotics*.

Oudeyer, P.-Y. and Kaplan, F. (2009). What is intrinsic motivation? A typology of computational approaches. *Frontiers in Neurorobotics*, 1:6.

Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

Papoudakis, G., Christianos, F., Rahman, A., and Albrecht, S. V. (2019). Dealing with non-stationarity in multi-agent deep reinforcement learning. *arXiv preprint 1906.04737*.

Papoudakis, G., Christianos, F., Schäfer, L., and Albrecht, S. V. (2021). Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Advances in Neural Information Processing Systems, Track on Datasets and Benchmarks*.

Parker-Holder, J., Jiang, M., Dennis, M., Samvelyan, M., Foerster, J., Grefenstette, E., and Rocktäschel, T. (2022). Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, volume 2017.

Pérolat, J., de Vylder, B., Hennes, D., Tarassov, E., Strub, F., de Boer, V., Muller, P., Connor, J. T., Burch, N., Anthony, T., McAleer, S., Elie, R., Cen, S. H., Wang, Z., Gruslys, A., Malysheva, A., Khan, M., Ozair, S., Timbers, F., Pohlen, T., Eccles, T., Rowland, M., Lanctot, M., Lespiau, J.-B., Piot, B., Omidshafiei, S., Lockhart, E., Sifre, L., Beauguerlange, N., Munos, R., Silver, D., Singh, S., Hassabis, D., and Tuyls, K. (2022). Mastering the game of Stratego with model-free multiagent reinforcement learning. *Science*, 378(6623):990–996.

Petersen, C. G. and Schmenner, R. W. (1999). An evaluation of routing and volume-based storage policies in an order picking operation. *Decision Sciences*, 30(2):481–501.

Precup, D. (2000). Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80.

Quicktron (2024). Quicktron QuickBin. accessed: 2024-02-24.

Raileanu, R. and Rocktäschel, T. (2020). RIDE: Rewarding impact-driven exploration for procedurally-generated environments. In *International Conference on Learning Representations*.

Rashid, T., Peng, B., Boehmer, W., and Whiteson, S. (2020a). Optimistic exploration even with a pessimistic initialisation. In *International Conference on Learning Representations*.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. (2020b). Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(1).

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Ryu, H., Shin, H., and Park, J. (2022). REMAX: Relational representation for multi-agent exploration. In *International Conference on Autonomous Agents and Multiagent Systems*.

Samvelyan, M., Rashid, T., de Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., and Whiteson, S. (2019). The StarCraft multi-agent challenge. *International Conference on Autonomous Agents and Multi-Agent Systems*.

Sartoretti, G., Kerr, J., Shi, Y., Wagner, G., Kumar, T. K. S., Koenig, S., and Choset, H. (2018). PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning. *Robotics and Automation Letters*, 4:2378–2385.

Schaal, S. (1997). Learning from demonstration. In *Advances in Neural Information Processing Systems*.

Schäfer, L. (2019). Curiosity in multi-agent reinforcement learning. Master's thesis, University of Edinburgh.

Schäfer, L., Christianos, F., Hanna, J. P., and Albrecht, S. V. (2022). Decoupled reinforcement learning to stabilise intrinsically-motivated exploration. In *International Conference on Autonomous Agents and Multiagent Systems*.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. In *International Conference on Learning Representations*.

Schmidhuber, J. (1991). Curious model-building control systems. In *International Joint Conference on Neural Networks*. IEEE.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning*.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint 1707.06347*.

Scott, S. L. (2010). A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6).

Sessa, P. G., Kamgarpour, M., and Krause, A. (2022). Efficient model-based multi-agent reinforcement learning via optimistic equilibrium computation. In *International Conference on Machine Learning*.

Shapley, L. S. (1953). Stochastic games. *National Academy of Sciences of the United States of America*, 39(10):1095–1100.

Shen, M. and How, J. P. (2023). Implicit ensemble training for efficient and robust multiagent reinforcement learning. *Transactions on Machine Learning Research*.

Shetty, N., Sah, B., and Chung, S. H. (2020). Route optimization for warehouse order picking operations via vehicle routing and simulation. *SN Applied Sciences*, 2(2):1–18.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning*.

Sokota, S., D'Orazio, R., Kolter, J. Z., Loizou, N., Lanctot, M., Mitliagkas, I., Brown, N., and Kroer, C. (2023). A unified approach to reinforcement learning, quantal response equilibria, and two-player zero-sum games. In *International Conference on Learning Representations*.

Son, K., Kim, D., Kang, W. J., Hostallero, D. E., and Yi, Y. (2019). QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*.

Song, Y., Romero, A., Müller, M., Koltun, V., and Scaramuzza, D. (2023). Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82).

Strehl, A. L. and Littman, M. L. (2008). An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*, 74:1309–1331.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V. F., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. (2018). Value-decomposition networks for cooperative multi-agent learning. In *International Conference on Autonomous Agents and Multi-Agent Systems*.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press.

Taïga, A. A., Fedus, W., Machado, M. C., Courville, A., and Bellemare, M. G. (2020). On bonus-based exploration methods in the arcade learning environment. In *International Conference on Learning Representations*.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *International Conference on Machine Learning*.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2017). # Exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*.

Taylor, M. E., Suay, H. B., and Chernova, S. (2011). Integrating reinforcement learning with human demonstrations of varying ability. In *International Conference on Autonomous Agents and Multi-Agent Systems*.

Team, A. A., Bauer, J., Baumli, K., Baveja, S., Behbahani, F. M. P., Bhoopchand, A., Bradley-Schmieg, N., Chang, M., Clay, N., Collister, A., Dasagi, V., Gonzalez, L., Gregor, K., Hughes, E., Kashem, S., Loks-Thompson, M., Openshaw, H., Parker-Holder, J., Pathak, S., Nieves, N. P., Rakicevic, N., Rocktäschel, T., Schroecker, Y., Sygnowski, J., Tuyls, K., York, S., Zacherl, A., and Zhang, L. M. (2023a). Human-timescale adaptation in an open-ended task space. In *International Conference on Machine Learning*.

Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., et al. (2023b). Gemini: A family of highly capable multimodal models. *arXiv preprint 2312.11805*.

Team, S., Raad, M. A., Ahuja, A., Barros, C., Besse, F., Bolt, A., Bolton, A., Brownfield, B., Buttimore, G., Cant, M., Chakera, S., Chan, S. C. Y., Clune, J., Collister, A., Copeman, V., Cullum, A., Dasgupta, I., de Cesare, D., Trapani, J. D., Donchev, Y., Dunleavy, E., Engelcke, M., Faulkner, R., Garcia, F., Gbadamosi, C. T. T., Gong, Z., Gonzales, L., Gregor, K., Hallingstad, A. O., Harley, T., Haves, S., Hill, F., Hirst, E., Hudson, D. A., Hughes-Fitt, S., Rezende, D. J., Jasarevic, M., Kampis, L., Ke, R., Keck, T., Kim, J., Knagg, O., Kopparapu, K., Lampinen, A. K., Legg, S., Lerchner, A., Limont, M., Liu, Y., Loks-Thompson, M., Marino, J., Cussons, K. M., Matthey, L., Mcloughlin, S., Mendolicchio, P., Merzic, H., Mitenkova, A., Moufarek, A., and Oliveira, V. (2024). Scaling instructable agents across many simulated worlds. *arXiv preprint 2404.10179*.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.

Tilbury, C. R., Christianos, F., and Albrecht, S. V. (2023). Revisiting the gumbel-softmax in maddpg. In *International Conference on Autonomous Agents and Multi-Agent Systems Workshop on Adaptive and Learning Agents*.

Torbati, R. J., Lohiya, S., Singh, S., Nigam, M. S., and Ravichandar, H. (2023). MARBLER: An open platform for standardized evaluation of multi-robot reinforcement learning algorithms. In *International Symposium on Multi-Robot and Multi-Agent Systems*. IEEE.

Touvron, H., Martin, L., Stone, K. R., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D. M., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A. S., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I. M., Korenev, A. V., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X., Tang, B., Taylor, R., Williams, A., Kuan,

J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. (2023). LLAMA 2: Open foundation and fine-tuned chat models. *arXiv preprint 2307.09288*.

Tumer, K. and Agogino, A. (2007). Distributed agent-based air traffic flow management. In *International Joint Conference on Autonomous Agents and Multiagent Systems*.

Tyrrell, T. (1993). *Computational mechanisms for action selection*. PhD thesis, University of Edinburgh.

van den Berg, J. P., Guy, S. J., Lin, M. C., and Manocha, D. (2011). Reciprocal n-body collision avoidance. In *International Symposium of Robotics Research*.

van der Pol, E. (2016). *Deep reinforcement learning for coordination in traffic light control*. PhD thesis, University of Amsterdam.

van Hasselt, H. (2010). Double Q-learning. *Advances in Neural Information Processing Systems*.

van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. (2018). Deep reinforcement learning and the deadly triad. *arXiv preprint:1812.02648*.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Le Paine, T., Gülç ehre, Ç., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T. P., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. (2023). Voyager: An open-ended embodied agent with large language models. *arXiv preprint Arxiv-2305.16291*.

Wang, J. X., Hughes, E., Fernando, C., Czarnecki, W. M., Duéñez Guzmán, E. A., and Leibo, J. Z. (2019a). Evolving intrinsic motivations for altruistic behavior. In *International Conference on Autonomous Agents and Multi-Agent Systems*.

Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Langlois, E., Zhang, S., Zhang, G., Abbeel, P., and Ba, J. (2019b). Benchmarking model-based reinforcement learning. *arXiv preprint 1907.02057*.

Wang, T., Dong, H., Lesser, V., and Zhang, C. (2020a). ROMA: Multi-agent reinforcement learning with emergent roles. In *International Conference on Machine Learning*.

Wang, T., Wang, J., Wu, Y., and Zhang, C. (2020b). Influence-based multi-agent exploration. In *International Conference on Learning Representations*.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4):279–292.

Weyns, D., Holvoet, T., Schelfthout, K., and Wielemans, J. (2008). Decentralized control of automatic guided vehicles: applying multi-agent systems in practice. In *ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications*, pages 663–674.

Whitney, W. F., Bloesch, M., Springenberg, J. T., Abdolmaleki, A., and Riedmiller, M. (2021). Decoupled exploration and exploitation policies for sample-efficient reinforcement learning. *arXiv preprint 2101.09458*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

Wolpert, D. H. and Tumer, K. (1999). An introduction to collective intelligence. *arXiv preprint cs/9908014*.

Wolpert, D. H. and Tumer, K. (2002). Optimal payoff functions for members of collectives. In *Modeling Complexity in Economic and Social Systems*.

Wolpert, D. H., Tumer, K., and Swanson, K. (2000). Optimal wonderful life utility functions in multi-agent systems.

Wu, P., Escontrela, A., Hafner, D., Abbeel, P., and Goldberg, K. (2023). Daydreamer: World models for physical robot learning. In *Conference on Robot Learning*.

Wu, Y., Tucker, G., and Nachum, O. (2019). Behavior regularized offline reinforcement learning. *arXiv preprint 1911.11361*.

Wurman, P. R., Barrett, S., Kawamoto, K., MacGlashan, J., Subramanian, K., Walsh, T. J., Capobianco, R., Devlic, A., Eckert, F., Fuchs, F., Gilpin, L., Khandelwal, P., Kompella, V., Lin, H., MacAlpine, P., Oller, D., Seno, T., Sherstan, C., Thomure, M. D., Aghabozorgi, H., Barrett, L., Douglas, R., Whitehead, D., Dürr, P., Stone, P., Spranger, M., and Kitano, H. (2022). Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228.

Wurman, P. R., D'Andrea, R., and Mountz, M. (2008). Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1):9.

Xiao, Y., Hoffman, J., and Amato, C. (2020). Macro-action-based deep multi-agent reinforcement learning. In *Conference on Robot Learning*.

Xu, Q., Li, J., Koenig, S., and Ma, H. (2022). Multi-goal multi-agent pickup and delivery. In *International Conference on Intelligent Robots and Systems*.

Yan, Y., Chow, A. H., Ho, C. P., Kuo, Y.-H., Wu, Q., and Ying, C. (2022). Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities. *Transportation Research Part E: Logistics and Transportation Review*, 162:102712.

Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., and Wu, Y. (2022). The surprising effectiveness of PPO in cooperative multi-agent games. In *Advances in Neural Information Processing Systems, Track on Datasets and Benchmarks*.

Zheng, L., Chen, J., Wang, J., He, J., Hu, Y., Chen, Y., Fan, C., Gao, Y., and Zhang, C. (2021). Episodic multi-agent reinforcement learning with curiosity-driven exploration. In *Advances in Neural Information Processing Systems*.

Zhong, R., Hanna, J. P., Schäfer, L., and Albrecht, S. V. (2021). Robust on-policy data collection for data-efficient policy evaluation. In *Neural Information Processing Systems Conference Workshop on Offline Reinforcement Learning*.

Zhou, M., Luo, J., Villela, J., Yang, Y., Rusu, D., Miao, J., Zhang, W., Alban, M., Fadakar, I., Chen, Z., ping Huang, C., Wen, Y., Hassanzadeh, K., Graves, D., Zhu, Z., Ni, Y., Nguyen, N. M., Elsayed, M., Ammar, H., Cowen-Rivers, A. I., Ahilan, S., Tian, Z., Palenicek, D., Rezaee, K., Yadmellat, P., Shao, K., Chen, D., Zhang, B., Zhang, H., Hao, J., Liu, W., and Wang, J. (2021). SMARTS: An open-source scalable multi-agent RL training school for autonomous driving. In *Conference on Robot Learning*.

Zhou, Y., Li, J., and Zhu, J. (2020). Posterior sampling for multi-agent reinforcement learning: Solving extensive games with imperfect information. In *International Conference on Learning Representations*.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, volume 8, pages 1433–1438.

Zimmer, M., Viappiani, P., and Weng, P. (2014). Teacher-student framework: A reinforcement learning approach. In *International Conference on Autonomous Agents and Multi-Agent Systems Workshop*.

Zong, Z., Feng, T., Xia, T., Jin, D., and Li, Y. (2021). Deep reinforcement learning for demand driven services in logistics and transportation systems: A survey. *arXiv preprint 2108.04462*.

Žulj, I., Salewski, H., Goeke, D., and Schneider, M. (2022). Order batching and batch sequencing in an AMR-assisted picker-to-parts system. *European Journal of Operational Research*, 298(1):182–201.