Multi-Agent Reinforcement Learning for Task Open Systems

by

Gayathri Anil

(Under the direction of Prashant Doshi and Frederick Maier)

Abstract

We target task-open systems where tasks are introduced dynamically in a multi-agent system, requiring continuous learning and adaptation from agents. This scenario is challenging due to the evolving action space and reward function, which conventional RL algorithms cannot handle. We present a novel decision-making framework, Task-Open MDP (TaO-MDP), to capture dynamic task arrivals and evolving environments. We further introduce a multi-agent RL algorithm, Models of Hyper Interactions under Task Openness (MOHITO), that learns a generalized policy for task-open environments. It employs interaction graphs to link agents, tasks, and actions, processed via graph neural networks. MOHITO uses centralized training and decentralized execution to select the best action from the available action space. Evaluated in a task-open Ridesharing domain, our algorithm facilitates knowledge transfer and boosts rewards by enabling agents to pool multiple passengers simultaneously. Results show agents achieve higher rewards compared to baseline methods, demonstrating MOHITO's effectiveness in dynamic, task-open environments.

Index words: Multi-agent reinforcement learning, Open agent systems, Multi-agent systems, Graph neural networks, Hypergraph, Incidence graph, Centralised training decentralised execution, Markov decision process

Multi-Agent Reinforcement Learning for Task Open Systems

by

Gayathri Anil

Bachelor of Technology, SASTRA University, 2018

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Science

Athens, Georgia

2024

Multi-Agent Reinforcement Learning for Task Open Systems

by

Gayathri Anil

Approved:

Major Professors: Prashant Doshi
Frederick Maier

Committee: Adam Eck
Jin Sun

Electronic Version Approved:

Ron Walcott
Dean of the Graduate School
The University of Georgia
August 2024

Multi-Agent Reinforcement Learning for Task Open Systems

Gayathri Anil

July 13, 2024

Dedication

To Amma, Acha, and Nand. You mean the world to me.

Acknowledgments

I'm grateful to the many individuals who have supported and guided me throughout my academic journey, culminating in this thesis. First and foremost, I would like to express my gratitude to my advisor, Dr. Prashant Doshi, for his guidance throughout my research. His expertise and insights have been instrumental in shaping the direction and success of this work. I'm also immensely thankful to Dr. Adam Eck and Dr. Leen-Kiat Soh for their regular feedback and constructive inputs, which have greatly influenced and refined my research. Special thanks to Dr. Frederick Maier for his invaluable feedback and support, which have been crucial in completing my thesis.

I would like to extend my sincere appreciation to Dr. Jin Sun for his support during my Master's program, not only as a professor and committee member but also as a mentor. I would like to thank Dr. Neal Outland, Dr. Khaled Rasheed, and Evette for their unwavering support and assistance throughout the program.

I'm also grateful to all my labmates, especially Daniel, for being my rubber duck, friend, advisor, and editor. To all my friends, thank you for your unwavering support, especially Bhargav, whose efforts have kept me balanced during times when balance was incredibly difficult to maintain.

Lastly, I owe my parents and sister a profound debt of gratitude. Their love and encouragement have always been my greatest strength.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multi-agent systems (MAS) represent complex frameworks consisting of autonomous intelligent agents, which interact within an environment to achieve individual or collective goals. These agents exhibit intelligent behavior, autonomously executing actions and making decisions based on their perceptions and knowledge. MAS are particularly effective in scenarios where tasks are too complex for a single agent to handle or where cooperation can lead to more efficient problem-solving. The foundational concepts and potential of MAS are comprehensively explored in the work by Balaji and Srinivasan (2010), which serves as an important resource for understanding these dynamic systems.

In an era marked by increasing autonomy, MAS have significantly transformed various sectors, from manufacturing to healthcare. For instance, in robotics, multiple robots collaborate to assemble products, while in logistics, a fleet of autonomous vehicles coordinates to deliver goods efficiently. The decision-making processes of these agents are underpinned by a wealth of algorithms and strategies that enable them to navigate, interact, and adapt within their environments. These algorithms can be broadly categorised into two primary classifications: **learning-based methods** Canese et al. (2021) and **planning-based methods** Moroni and Chiffi (2022).

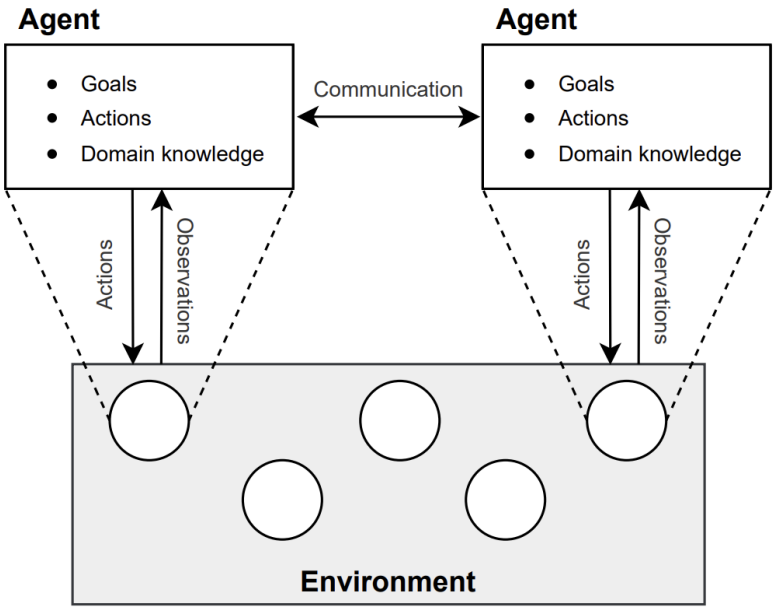

Figure 1.1: Schematic depicting a multi-agent system, characterised by an interactive domain inhabited by several autonomous agents (represented as circles). These agents have the capability to gather environmental data and execute decisions aimed at fulfilling their objectives. Source: Albrecht et al. (2024)

Most MAS under study are characterised by a fixed set of agents of a given type and a fixed number of tasks to be accomplished by these agents. These systems are called *closed* systems because both the agents and tasks are known in advance. In MAS, each decision-making agent must determine and continuously adapt strategies to achieve collective and/or individual objectives. Learning how to coexist and effectively operate within a shared environment that includes multiple other agents, even in a system where the agents and tasks are fixed, is challenging.

One property of real-world environments that further complicates multi-agent decision-making is *openness*, where some elements of the environment appear and disappear over time. Closed systems are limited in the sense that they do not model the dynamic nature of many real-world environments where openness can exist. To address these limitations, researchers have turned their attention to **open agent systems (OASYS)** Eck et al. (2023a). Rea-

soning in such open agent systems is more complicated than in closed environments because of the continuous changes in the set of agents, tasks, and the agents' capabilities.

Openness can occur due to changes in three different key components: the set of agents interacting in the environment, the set of tasks those agents aim to accomplish, and the types of agents (i.e., their capabilities)

- **Agent openness** occurs when the set of agents in the system changes over time. Existing agents may leave, never to return, or could rejoin the environment later, and new agents may join.
- **Task openness** occurs when the set of tasks that agents can work on changes over time as new tasks are introduced and prior tasks disappear.
- **Type openness** occurs when the capabilities and decision-making processes of agents change over time. Learning how to act in environments with openness is even more challenging since the set of objectives and/or agents is neither static nor predefined.

This thesis introduces the **Task-Open Markov Decision Process (TaO-MDP)**, a formal framework addressing the complexities of dynamic task-open systems. This framework is exemplified through the novel conceptualisation of the **Dynamic Ridesharing** domain, which is modeled and implemented as a TaO-MDP to effectively manage dynamic task assignments. Central to this thesis is the introduction of the multi-agent learning-based algorithm, **Models of Hyper Interactions under Task Openness (MOHITO)**. MOHITO employs centralised training and decentralised execution to develop generalised policies for task-open environments. By leveraging graph neural networks (GNNs) to process novel graph-based state-task representations—termed **interaction hypergraphs** and **interaction incidence graphs**—the algorithm facilitates sophisticated action selection based on the causal flow of information within these graphs.

Recent advances in **multi-agent reinforcement learning (MARL)** address related challenges. Multi-task learning algorithms (e.g., Tanaka and Yamamura (2003), Andreas et al. (2017); Rajeswaran et al. (2017); Sodhani et al. (2021)), for instance, excel at learning generalised policies across multiple related tasks simultaneously but falter when faced with tasks not seen during training. Lifelong learning (Thrun and Mitchell (1995); Ammar et al. (2014a); Chen and Liu (2018); Mendez et al. (2022)), on the other hand, is designed for continuous adaptation to new tasks over time but assumes a sequential, rather than simultaneous, arrival of these tasks. Out-of-distribution learning (e.g., Sedlmeier et al. (2020); Haider et al. (2023)) aims to enable agents to detect *when* their current tasks are different from those experienced during training and must adapt their behavior to new situations, but the question of *how* to adapt remains open. Despite these advancements, state-of-the-art MARL algorithms inherently struggle with such dynamic task openness due to their underlying design, which maps states to a static and dimensionally fixed action space.

The robustness and efficacy of the MOHITO algorithm have been validated through comprehensive empirical analysis within the novel dynamic ridesharing domain. The results highlight its practical applicability and innovative approach to navigating dynamic environments. Using the training algorithm, agents learn optimal policies, which are then tested in task-open ridesharing environments with various configurations, including different agent counts and task entry rates. Agents start from random locations and adapt to a continuously evolving set of tasks throughout each episode. Each training experiment involves agents initialised at random locations within the dynamic ridesharing grid, alongside a few passengers with varying start and endpoints. Throughout the episode, additional passengers with distinct start and endpoints are gradually introduced. At the start of training, agents are initialised with random policies in a task-open ridesharing environment. The training progresses as the episode unfolds following the novel training algorithm outlined in

this thesis.

As training progresses, after every few training steps, the learned policies are evaluated in predefined evaluation environments containing tasks that were not encountered by the agents during training. These evaluation environments consist of the same number of tasks as used during training, which is three times the number of agents. The rewards accumulated in these validation episodes are utilized to monitor policy convergence. As training epochs progress, when agents show stabilized rewards in the validation episodes, the policy is said to have converged.

On obtaining converged policies, they are evaluated to understand if the desired outcomes are present in them. Agents with the learned policies are deployed in several pre-determined test episodes containing previously seen and unseen tasks. The test environments are configured with varying levels of openness to see how well the policy can adapt to openness. This thesis reports performance across three predefined levels of openness: Openness Level 1 (OL1), Openness Level 2 (OL2), and Openness Level 3 (OL3).

Through these test runs, the intention is to see if the learned policies fulfill the basic expectations one has from an autonomous intelligent agent. These expectations are successful task execution, adaptability to dynamic previously unseen task set changes, strategic task selection to maximize rewards, and advanced planning abilities resulting in task pooling. The performance of the learned policy in the test episodes indicates that agents successfully satisfy all the above-listed expectations. The behavior of the learned policies in these environments even qualitatively resembles optimal human-like decision-making capabilities, incorporating long-term planning, such as pooling multiple tasks together whenever it is efficient. MOHITO-enabled agents learned policies that allowed them to efficiently tackle previously unseen tasks, maintaining low acceptance times for new tasks. Agents are observed evaluating the available set of actions at any given time rather than relying solely on pre-learned state-to-action mappings, thus effectively embracing task openness. The MOHITO-trained

policies exhibit long-term thinking capabilities, with agents pooling multiple tasks together when they fall along the same route.

The strategic decision-making capabilities of MOHITO-trained policies led to better performance compared to some baseline policies. This thesis identifies two static rule-based baselines—First-come first-serve and Nearest task first—apart from a dynamic decision-making based policy similar to MOHITO: an extension of PG-ELLA Ammar et al. (2014a) called Open Task PG-ELLA (OTPG-ELLA). Quantitatively, the policy learned by MOHITO performs better than the static baseline policies by an average of 29.78% and marginally better than the dynamic decision-making based baseline, demonstrating the algorithm's robustness and adaptability in dynamic, task-open environments.

The training of MOHITO-based policies encounters a limitation that results in a proportion of agents failing to converge to an optimal policy. This thesis also addresses this limitation and explores several directions to mitigate the issue.

1.1 Contributions

This thesis addresses the significant challenges associated with task-open multi-agent systems (MAS), specifically the dynamic changes to the action set, reward function, and environmental state, to accommodate dynamic and non-stationary environments. The contributions of this thesis include:

- This thesis introduces **Task-Open Markov Decision Process (TaO-MDP)**, a formal framework to represent the decision-making in task-open systems, capturing their unique characteristics and addressing the limitations of traditional MDPs in handling dynamic task arrivals and evolving environments. This formalisation represents a novel theoretical contribution to the field.

- Novel conceptualisation of a ridesharing domain as a task-open system called **Dynamic**

Ridesharing. This domain has been instantiated and implemented as a TaO-MDP, providing a structured approach to model dynamic task assignments and effectively addressing the challenges of a constantly evolving environment.

- Novel graph-based state-task representations as **interaction hypergraphs** and **interaction incidence Graphs**, are proposed. This representation captures a variation of the state and action variables typically used in reinforcement learning. It combines the agent's state information with the available tasks and the resulting action space at each time step. The graphical structure uses directional edges to allow information to flow from agent, task, and action nodes to a decision-making node.

- This thesis introduces a multi-agent reinforcement learning (MARL) algorithm called **Models of Hyper Interactions under Task Openness (MOHITO)**. MOHITO uses centralised training and decentralised execution to learn generalised policies for task-open environments. The algorithm utilises graph neural networks (GNNs) to process graphical state-task representations and select actions based on the causal flow of information within the graph-based representation. This approach offers a sophisticated and novel mechanism for action selection in dynamic environments.

- Comprehensive empirical results are provided, demonstrating the efficacy of the MOHITO algorithm in the Dynamic Ridesharing domain, instantiated as a TaO-MDP. These results validate the effectiveness of the proposed algorithm, highlighting its applicability and robustness in real-world scenarios.

1.2 Thesis Structure

The rest of the thesis is organised as follows:

- Chapter 2 reviews the existing literature on open-agent systems (OASYS), learning-based methods within OASYS, and related approaches to task openness, providing a context for the contributions of this thesis.

- Chapter 3 provides essential foundational concepts necessary for understanding the thesis.

- Chapter 4 presents the formal framework for task-open systems. Section 4.1 introduces the Task-Open MDP (TaO-MDP), a novel framework designed to represent decision-making in task-open systems. Section 4.2 introduces dynamic ridesharing and Section 4.3 applies this framework to this dynamic domain, demonstrating how TaO-MDP can effectively model and manage dynamic task assignments in real-world scenarios.

- Chapter 5 details the proposed multi-agent reinforcement learning algorithm for task-open systems. Section 5.1 discusses state-task representation using interaction incidence graphs, including the dynamics of state-task interactions in ridesharing. Section 5.2 introduces the MOHITO algorithm, explaining its architecture and training algorithm in detail.

- Chapter 6 presents the experimental setup and results for evaluating the MOHITO algorithm. Section 6.1 describes the training and evaluation setup, including the configurations used for testing. Section 6.2 outlines the performance baselines used for comparative analysis. Section 6.3 evaluates the performance of MOHITO, discussing key training metrics, policy analysis, and comparative analysis against baselines. The section also addresses training limitations and provides a discussion of the findings.

- Chapter 7 summarises the contributions of the thesis, discusses the limitations of the current work and suggests directions for future research.

Chapter 2

Related Work

In this chapter, we review the body of work that has contributed to the development and understanding of open-agent systems (OASYS) and related fields. The aim is to provide a comprehensive background that situates this thesis within the broader context of multi-agent systems (MAS) research. We explore various methodologies and approaches, particularly those that address the challenges posed by dynamic and open environments. This chapter is organised into sections that cover foundational concepts of OASYS, learning-based methods in OASYS, and other approaches relevant to task openness, highlighting how each of these areas contributes to the motivation and development of the proposed MOHITO algorithm.

2.1 Open-agent systems

An initial step towards understanding the extreme challenges brought about by openness for decision-making is to classify openness. Prior work Calmet et al. (2004),
Jumadinova et al. (2014); Shehory (2000) has recognised three types: **agent openness** manifests when the set of agents operating in the environment changes over time as individual agents temporarily or permanently join or leave the system; **task openness** where the set

of tasks for agents to accomplish changes over time; and **frame (or type) openness** where agents' frames (i.e., capabilities and reasoning processes) may change over time, such as when they acquire new abilities or change roles. These do not exhaustively represent all types of openness but they encompass the ones often seen in practice. Among these types of open systems, agent openness has received the most attention so far, *whereas task openness remains very much understudied.*

More than two decades ago, Shehory Shehory (2000) noted that agent openness is about introducing additional agents into the system in addition to the agents that comprise it initially. Calmet et al. Calmet et al. (2004) studied open societies of agents that are open to new agents either with no definite goal or with goals not exceedingly relevant to society. Both focused on the system and software architecture to support openness. Additional properties of agent openness were then reported after a gap of about ten years. Jamroga et al. Jamroga et al. (2013) defined the *degree of openness* of multiagency as the complexity of the minimal transformation that the system must undergo to add a new agent to the system or remove an existing one.

Jumadinova et al. Jumadinova et al. (2014) and Chen et al. Chen et al. (2016) extended the notion of openness to include both agent openness and task openness to model the dynamic nature of the agents and tasks in the environment, primarily considering the rates at which agents or tasks join and leave the system. As a first step towards reasoning about task openness, they reacted to such openness in ad-hoc and domain-specific manners, whereas we seek to develop principled learning solutions in this work that enable an agent to flexibly address changing tasks and the resulting changes to their action space and reward function being optimised.

Decision-making has predominantly focused on closed-world settings with a perfectly observed state, with a few exceptions. Whereas isolated discussions of open systems have appeared in the literature as we noted above, a more visible thrust into acting in such con-

texts has recently emerged. For example, Chandrasekaran et al. (2016) viewed an interacting agent's absence analogous to performing a *no-operation* and sought to evaluate whether actively predicting the agent's exit from the system is beneficial compared to post hoc inference from the no-operation. Observations of a clear improvement in global system behavior due to the former led to subsequent methods for better predicting agents' exits and re-entries even in MAS. Eck et al. (2020); Kakarlapudi et al. (2022) extended the notion of planning with communication in open MAS. They introduced a principled, decision-theoretic method that leverages the communicative interactive POMDP framework. This method enables agents to plan with both physical and communicative actions in open multiagent systems, addressing the nontrivial predictability of agent presence

2.2 Learning-based methods in OASYS

In addition to planning-based approaches that assume the agent has a mental model of the environment available a priori, recent research has also explored how agents can use reinforcement learning for decision-making in open systems.

(Jiang et al., 2020) addresses the concept of openness in multi-agent systems by proposing a flexible and adaptive framework that can handle the dynamic nature of agent interactions. The authors tackle agent openness by using graph neural networks (GNNs) to model the multi-agent environment as a graph, where each agent is a node, and the edges represent interactions between agents. The graph convolution process adapts to the changing graph structure, allowing the system to maintain an understanding of the inter-agent relationships even as agents enter or leave the system. This approach ensures that the reinforcement learning process remains robust to the open nature of the environment, enabling agents to learn effective cooperation strategies despite the fluidity of their operational context

Recently, Rahman et al. (2021) developed over the concept of graph convolutional rein-

forcement learning to introduce a policy learning approach for OASYS with agent openness by employing a Graph Neural Network (GNN) to dynamically model agent interactions as a graph, where nodes represent agents and edges their interplay. This graph-based framework facilitates the GNN's adaptation to fluctuating team structures and agent behaviors, fostering resilient cooperation within open, unpredictable environments. The architecture's design, which includes type embedding networks and parameterised models, allows for the computation of action-value functions that adapt to the team's current state. This adaptability is key to managing the complexities of open ad hoc teamwork. Figure 2.1 illustrates the solution architecture, highlighting how the GNN effectively addresses openness by adjusting to real-time changes in team dynamics and agent policies.

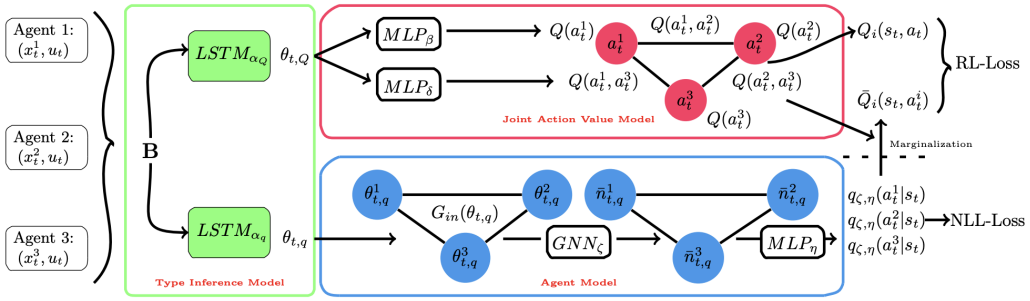

Figure 2.1: Overview of the architecture of Graph-based Policy Learning (GPL) introduced by (Rahman et al., 2021)

Instead of directly learning a model of the environment dynamics solved through planning as indicated by (Chandrasekaran et al., 2016), agents instead learn a deep neural network with components for (1) inferring the types of other agents, (2) approximating the other agent's policies, and (3) estimating the value of the joint action of all agents and the learning agent's contribution to cumulative rewards. Together, these components enable the agent to learn how to behave when the teammates of the learning agent change due to agent openness. Experiments on a few ad hoc teamwork benchmark problems demonstrated appropriate

adaptation to changing teams of agents and significant improvement in behavior compared to prior methods from both multiagent reinforcement learning and ad hoc reasoning.

Both (Jiang et al., 2020; Rahman et al., 2021) that contribute to RL in open MAS target agent openness without considering task openness. Both rely on the use of graph neural networks Wang et al. (2016); Battaglia et al. (2018a) that are resilient to changing input sizes Hamilton et al. (2017). These embed fully connected coordination graphs of the agents, and may intrinsically adapt to dynamic team sizes as agents depart or reenter.

We are aware of just two recent works, Jiang et al. (2020); Rahman et al. (2021), that contribute to RL in open MAS, both of which target agent openness without considering task openness. Both rely on the use of graph neural networks Wang et al. (2016); Battaglia et al. (2018a) that are resilient to changing input sizes Hamilton et al. (2017). These embed fully connected coordination graphs of the agents, and may intrinsically adapt to dynamic team sizes as agents depart or reenter.

2.3 Related Approaches to Task Openness

Other related topics have also inspired how to learn rules of behavior that generalise to novel situations due to changing sets of agents and tasks in OASYS. Out-of-distribution learning (e.g., Sedlmeier et al. (2020); Haider et al. (2023)), where agents detect that their current tasks are different from those experienced during training and must adapt their behavior to new situations,

Lifelong learning, as explored by (Thrun and Mitchell, 1995; Ammar et al., 2014a), (Chen and Liu, 2018; Mendez et al., 2022), involves agents continuously acquiring knowledge from past experiences and applying it to new tasks. This approach enables agents to learn more efficiently and effectively, as they can leverage previously gained knowledge to tackle future challenges. The work by (Abel et al., 2018) delves into the optimisation of policy

initialisation for lifelong learning1. The authors address the challenge of how to best use prior experience to bootstrap an agent's learning process when faced with a series of tasks drawn from a task distribution. They propose methods for initializing an agent's policy or value function that optimise expected performance over the distribution of tasks.

Multitask learning, investigated by (Tanaka and Yamamura, 2003; Andreas et al., 2017; Rajeswaran et al., 2017; Sodhani et al., 2021), focuses on agents learning to generalise across a set of tasks. By identifying similarities and differences between tasks, agents can develop strategies that improve their performance across the board, making them more versatile and capable of handling a variety of situations.

(Ammar et al., 2014a) present Policy Gradient Efficient Lifelong Learning Algorithm (PG-ELLA), a multi-task policy gradient method that allows for the consecutive learning of decision-making tasks. This method facilitates the transfer of knowledge between tasks, thereby accelerating the learning process. The approach is designed to be computationally efficient, enabling online learning with low overhead, which contrasts with the typically high computational cost of batch multi-task learning. This work is significant as it contributes to the development of more sample-efficient RL agents capable of quickly adapting to new tasks, which is particularly valuable in robotics and other areas where agents face multiple tasks over their operational lifetime.

Recently, (Zhang et al., 2023) have also studied decision-making through multiagent RL when other agents' policies abruptly change during operations, which could be useful for guiding RL under task and type openness. At the same time, insights about the complexities caused by OASYS and how to reason with them could produce new methods that may generalise to settings considered in these related areas.

Chapter 3

Foundational Concepts

In this chapter, we define and clarify several foundational concepts and terminologies that form the basis for subsequent discussions throughout this thesis. Section 3.1 introduces the concept of task-open systems, which are central to our study. Section 3.2 discusses Markov Decision Processes (MDPs), essential for modeling stochastic environments in decision-making problems. In Section 3.3, we explore Multi-Agent Reinforcement Learning (MARL), focusing on methodologies where multiple agents interact and learn concurrently. Following that, Section 3.4 examines hypergraphs and incidence graphs, which are critical for analyzing the dynamics within task-open systems. Section 3.5 presents Graph Neural Networks, a method for processing graph-structured data effectively. Lastly, Section 3.6 describes the Dynamic Ridesharing toy domain, which serves as the practical test bed for the methodologies developed in this thesis.

3.1 Task-Open Systems

Extant learning generally assumes that the set of tasks that the agents seek to complete are known in advance and remain in the environment until completed. This assumption

is valid for many real-world environments. For example, (Kong et al., 2023) assumed an environment in their domain of multi-aircraft close-range air combat to train agents on three specific pre-determined types of shooting subtasks. Amongst popular MARL benchmark problems Papoudakis et al. (2021), level-based foraging Albrecht and Ramamoorthy (2013) tasks agents with collaboratively collecting food items that are present from the beginning of operations; problems within the multiagent particle environment (MAPE) Lowe et al. (2017) contain consistent tasks ranging from evading or catching one another (Predator-Prey) to spreading agents to cover a fixed set of landmarks. The StarCraft multiagent challenge problems Samvelyan et al. (2019) include multiple scenarios where two teams of agents have a single high-level task of competing to eliminate their opponents. In such *closed* environments, the agents' goals are fixed, the reward function optimised by each agent is also unchanging, and the set of relevant actions is generally static.

On the contrary, many real-world environments where MARL might be used for decision-making by intelligent agents are more complex. For example, teams in a business organisztion often experience a request to accept new projects, which must be completed concurrently with existing projects. Drivers in a ridesharing application (e.g., Uber, Lyft) with vacancies in their vehicle must decide whether to accept new hailing passengers that arrive dynamically and often unexpectedly within the environment. For larger vehicles (e.g., Uber Pool), the driver must balance the needs of multiple simultaneous tasks (different passengers occupying the same vehicle but desiring different destinations) while also considering the locations of competing drivers to maximise their income, as illustrated in Figure 4.1. As such, new tasks and goals are introduced over time, and their presence may alter previously planned or learned behavior (e.g., picking up a nearby passenger could alter the order in which the driver intended to drop off existing passengers).

In this thesis, we focus on task openness that is caused by *exogenous factors*, such as external market forces in business organizations or passengers appearing dynamically within

dynamic ridesharing applications (not due to decisions internal to the ridesharing drivers or supporting infrastructure). Such factors are impossible to plan for since they are usually unknown to the actors within a MAS. In some circumstances, the factors might be predicted from experience with learning. However, reactive behavior also typically requires learning since the range of possible tasks (e.g., diversity of dynamically arriving passengers) could be difficult to enumerate and anticipate.

It is acknowledged that task openness can also be caused by *endogenous factors*, such as one business organization deciding to merge with another to combine their product lines and teams, or a task being completed and thus exiting the environment. For such types of task openness, planning-based solutions have been previously explored, particularly decision-making under uncertainty with time-varying Markov models (e.g., Badings et al. (2023)), which do not necessarily require reinforcement learning. In this thesis, we address task openness due to endogenous factors to the extent of events like task completion. The study of task openness under other endogenous factors is left as future work.

3.2 Markov Decision Process

Markov Decision Processes (MDP) provide a mathematical framework for modeling stochastic environments for decision-making problems. An MDP is formally defined using the following 4-tuple

$$\mathsf{MDP} \stackrel{def}{=} \langle S, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$$

where S denotes the set of states called the *state space*, representing all possible configurations of the environment. A denotes the set of actions available to the agent called the *action space*, T is the *transition function*, and R is the *reward function*

- **State Space** - The configuration of the environment at any time step is called the

state s of the environment. The state space S denotes all possible states that the environment can be in. MDPs assume that the state is perfectly observable, meaning that the agent has complete and accurate knowledge of the current state s when making decisions.

- **Action Space** - Given the state s of the environment, an agent is required to take a decision on what action a to carry out at that time step. The Action Space A is the set of all possible actions that the agent can carry out in the environment.
- **Transition Function** - When an agent carries out an action a in a state s, the environment transitions into the next state *s'*. The transition function can be deterministic or stochastic and can be defined as

$$T(s, a, s')=Pr(s'|s, a)$$

- **Reward Function** - When an agent performs action a in state s, it receives a reward r. The reward function R determines the reward the agent receives at each time step. R can be function of only s, or *(s, a)*, or even *(s, a, s')*

MDPs operate on **Markovian Property** which states that the future state depends only on the present state and does not depend on the past history. That is,

$$Pr(s^{t+1}|s^t, a^t)=Pr(s^{t+1}|s^0, a^0, s^1, a^1, ..., s^t, a^t)$$

The decision horizon T of an MDP refers to the number of time steps into the future for which an agent considers the consequences of its actions when making decisions. The stochastic process is called a finite-horizon MDP if the number of time steps is finite. Otherwise, when the number of time steps is infinite, the stochastic process is referred to as an infinite-horizon MDP.

Policy

The objective of the MDP is to find a good policy π for the decision-making agent. A policy function π is a mapping from the state space S to the action space A, where $\pi(s)$ determines the best action that the agent can carry out in state s. The policy can be deterministic or stochastic. A deterministic policy maps states to actions as shown below. This means that for any given state, the policy will always produce the same action.

$$\pi : S \mapsto A$$

However, a stochastic policy maps states to actions via a probability density function. It assigns probabilities to each action that can be taken from a given state, allowing for a range of actions to be chosen under similar circumstances.

$$\pi : S \mapsto Pr(A)$$

Value Functions

Value functions are used to both evaluate the performance of policies and to optimise them. They are used to measure the expected return from states or state-action pairs under a specific policy.

The state-value function $V^{\pi}(s)$ measures the expected return starting from state s under policy π

$$V^{\pi}(s) = E^{\pi}\left[\sum_{t=0}^{T} R(s_t, a_t) \mid s_0 = s\right]$$

Here, γ is a discount factor with a value in the range $[0, 1)$, which balances the importance of immediate and future rewards.

The action-value function, or Q-function $Q^{\pi}(s, a)$, measures the expected return starting from state s, taking action a, and following policy π thereafter.

$$Q^{\pi}(s, a) = E^{\pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a\right]$$

Optimal value functions are derived from the best possible policy π^* that maximises the expected return

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

The value functions can be recursively defined using the Bellman equations. These equations express the value of a state or state-action pair in terms of the immediate reward and the value of subsequent states. The corresponding Bellman optimality equation is

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in S} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right]$$

Similarly, the optimal action-value function $Q^*(s, a)$ delivers the maximum return from state s after taking an action a, under the best policy:

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

The Bellman optimality equation for the action-value function is

$$Q^*(s, a) = \sum_{s' \in S} T(s, a, s') \left[R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \right]$$

3.3 Multi-Agent Reinforcement Learning

Within the realm of decision-making for autonomous agents, policies can be derived through two categories of approaches: planning and learning. The planning-based methods involve pre-computing strategies based on model predictions or simulations of the environment. On the other hand, learning-based methods adapt and evolve policies through the agent's interactions with the environment. A significant and increasingly prevalent subset of learning-based methods is Reinforcement Learning (RL), which enables agents to learn optimal behaviors via trial-and-error, by reinforcing them with positive and negative rewards corresponding to the actions taken.

In single-agent RL, an agent perceives the state of the environment and chooses the best action to execute at that time step. The execution of this action causes a transition to a new state within the environment, which consequently results in a positive or negative reward to the agent. Over time, through repeated interactions with the environment, the agent learns an optimal policy that maximises its total accumulated reward. Figure 3.1 illustrates the action-reward feedback loop of a generic single-agent RL model.

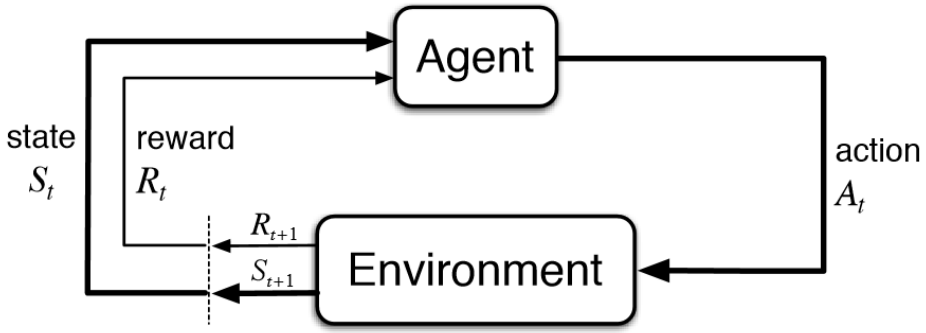

Figure 3.1: Action-reward feedback loop of a generic single-agent RL model where an agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent

Multi-agent Reinforcement Learning (MARL) addresses the problem of sequential decision-making for multiple autonomous agents in an environment that can interact with the environment as well as with each other. Each agent in MARL is modeled as an independent decision-maker that learns from its interactions with the environment and other agents. In such a setting, before taking an action, agents need to account for not only the dynamics of the environment but also the actions and policies of other agents. This introduces non-stationarity from the perspective of any individual agent, rendering the landscape of MARL inherently more complex than its single-agent counterpart.

Single-agent RL algorithms can be applied to train individual agents in a multi-agent system by having each agent treat the environmental changes due to others' actions as environmental stochasticity. While this can sometimes lead to meaningful policies, it is not

sufficient for effective learning. Figure 3.2 shows Independent learning in multi-agent systems where individual agents ignore the existence of other agents.

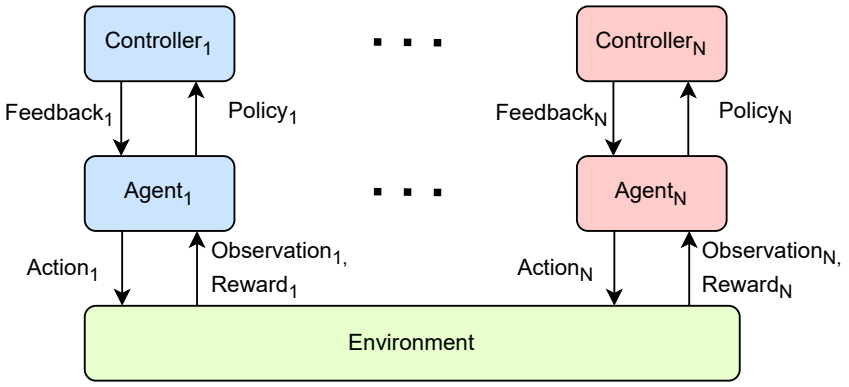

Figure 3.2: Independent Learning in Multi-Agent Systems

To aptly capture the intricacies of decision-making in multi-agent systems, sophisticated algorithms are required to learn within such environments. These algorithms must navigate scenarios, where the aim might involve attaining a collective objective or pursuing individual objectives that may conflict with one another.

One of the approaches in MARL is Centralised Training with Decentralised Execution (CTDE)Lowe et al. (2017); Foerster et al. (2017). CTDE aims to leverage the global information available during the training phase to learn decentralised policies that agents execute during operation. That is, individual agents pass their local information, such as their observation, action and reward, to a central controller. This central controller then collects joint actions, joint observations and joint rewards to update individual agent policies. Figure 3.3 shows this learning paradigm. This approach allows for coordination among agents by sharing information during training, thus enabling them to learn policies that are aware of the actions and potential strategies of their counterparts. During execution, each agent acts independently based on its policy, ensuring scalability and adaptability in diverse and dynamic environments.

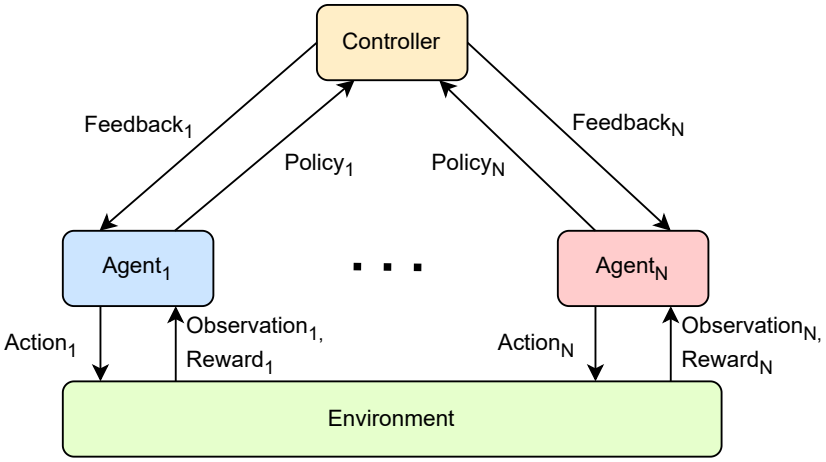

Figure 3.3: Centralised Training and Decentralised Execution learning paradigm with one controller for all agents

While the paradigm outlined in Figure 3.3 proves effective in scenarios focused on achieving a collective objective, a modified version of this approach is adopted when agents pursue individual objectives that may clash with each other. Figure 3.4 illustrates a learning paradigm wherein each agent possesses its own controller, yet still utilises joint actions observed in the environment to refine its individual policy. During execution, agents once again act autonomously based on their respective policies.

The **Actor-Critic framework** Lowe et al. (2017), a cornerstone of modern RL techniques, is a manifestation of the CTDE learning paradigm with controllers. In this framework, the 'Actor' component is responsible for selecting actions based on the current policy, while the 'Critic' assesses the taken actions by evaluating the potential long-term rewards. The Actor-Critic method thus combines the benefits of value-based and policy-based approaches, offering a powerful mechanism for policy improvement.

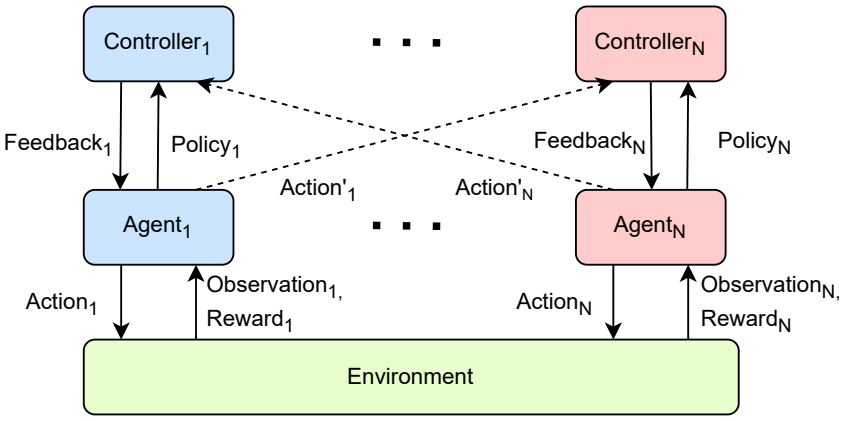

Figure 3.4: Centralised Training and Decentralised Execution learning paradigm with individual controllers for each agent. $Action'_i$ represents the inferred action of agent i, as indicated by the dotted line in the image.

3.4 Hypergraphs and Incidence Graphs

A graph is a structure containing a set of objects in which some pairs of the objects are in some sense "related". The objects in a graph are represented by nodes and the "relations" between the pairs of nodes are represented as edges between them. A **hypergraph** is an extension of a graph, where edges can connect any number of nodes. This generalisation allows us to represent more complex relationships in data. The edges in hypergraphs are called hyperedges. The cardinality of a hyperedge refers to the number of nodes connected by that hyperedge. A hypergraph where all hyperedges have the same cardinality k is called a **k-uniform hyperedge**

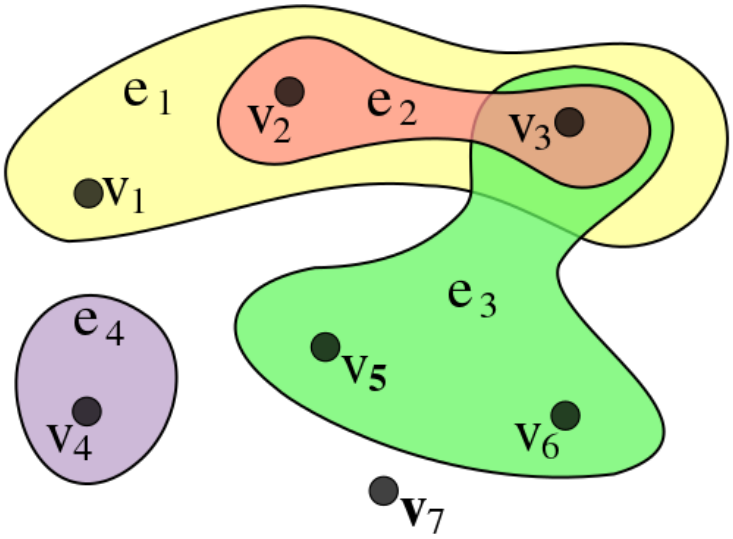

Figure 3.5: An example of an undirected hypergraph. Source: Wikipedia contributors (2024)

Formally, a hypergraph H can be defined as a pair $H := (X, E)$, where X is a set of elements called nodes, and E is a set of non-empty subsets of X called hyperedges.

An example of an undirected hypergraph can be seen in Figure 3.5. In the hypergraph, $X = \{v1, v2, v3, v4, v5, v6, v7\}$ and $E = \{e1, e2, e3, e4\} = \{\{v1, v2, v3\}, \{v2, v3\}, \{v3, v5, v6\}, \{v4\}\}$. In this hypergraph, the cardinality of edges $e1, e2, e3, e4$ are 3, 2, 3, and 1 respectively. Figure 3.6 (a) shows an alternate representation of the hypergraph.

Hypergraphs can also be viewed as **incidence structures**. For every hypergraph, there exists a corresponding bipartite **incidence graph** or **Levi graph**. A bipartite graph is a graph whose vertices can be divided into two disjoint sets such that no two graph vertices within the same set are adjacent. An incidence graph is a specific type of bipartite graph that represents the incidences or connections between nodes and hyperedges of a hypergraph.

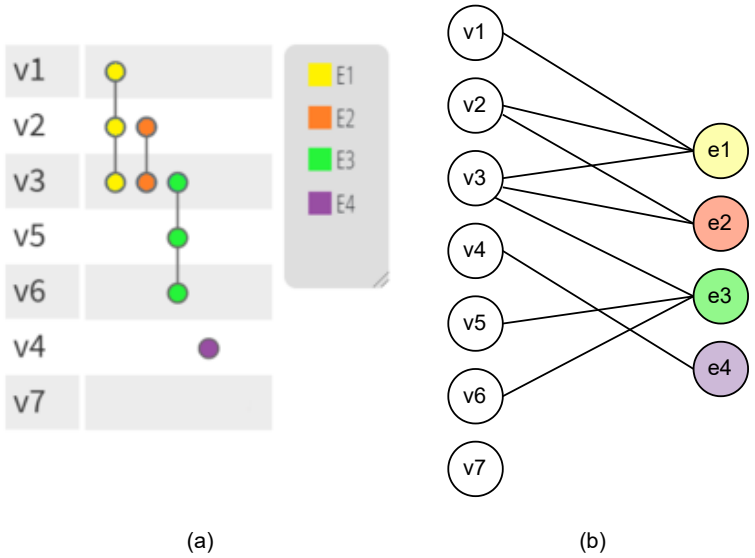

Figure 3.6: (a)Alternate representation of the hypergraph in Figure 3.5, (b)Incidence graph of the hypergraph in Figure 3.5

The process of converting a hypergraph to an incidence graph involves creating a node in the incidence graph for each hyperedge in the hypergraph. Edges are then drawn in the incidence graph between nodes representing nodes and those representing hyperedges that are incident in the hypergraph. This results in a bipartite graph where nodes representing nodes and hyperedges in the hypergraph are never adjacent.

Figure 3.6 (b) illustrates the incidence graph of the hypergraph depicted in Figure 3.5. In this incidence graph, the hyperedges $e1, e2, e3, e4$ are represented as nodes. The edges in the incidence graph connect the original nodes $v1, v2, v3, v4, v5, v6, v7$ to these hyperedge nodes $e1, e2, e3, e4$.

Chapter 4

Modeling Task-Open Systems

Traditional Markov Decision Processes (MDPs) struggle to handle the dynamic nature of task-open systems, as they rely on a static framework that requires fixed state and action spaces, along with fixed transition and reward functions. They cannot support systems where tasks arrive unpredictably, making it necessary to develop a more flexible framework to formalise these systems. In this chapter, in Section 4.1, we introduce our novel contribution: the Task-Open Markov Decision Process (TaO-MDP), which extends the traditional MDP framework to better accommodate the complexities of task-open environments. Specifically, we formalise the representation of decision-making in task-open systems through the TaO-MDP. In Section 4.2, we then introduce the novel conceptualisation of the task-open rideshare domain, called dynamic ridesharing, which models the unpredictable nature of real-world ridesharing scenarios. Finally, in Section 4.3, we instantiate dynamic ridesharing as a TaO-MDP, demonstrating how this framework can effectively manage dynamic task assignments and the evolving environment. This chapter provides a comprehensive understanding of the TaO-MDP and its application to dynamic ridesharing, setting the stage for further exploration and analysis.

4.1 Task-Open MDP

We propose **Task-Open MDP** or **TaO-MDP** to formalise the representation of decision-making in task-open environments where exogenous factors trigger the action space and hence the reward function of an environment to change from one time step to another. We propose to model the problem as consisting of a *two-step iteration* of a *base decision-making model* and a *generator*. The base decision-making model is an MDP and represents the parameter values of the initial decision-making model. The generator is responsible for modifying the decision-making model. At the occurrence of an exogenous factor that leads to a modification in the list of tasks present in the environment, the generator captures how the parameters of the newly added task(s) modify the *current decision-making* model to generate a new one for the agent.

In developing this novel decision-making framework, the following assumptions have been considered.

1. All tasks in the environment are homogeneous and share a common underlying structure, denoted as ϵ. This underlying structure ϵ defines the functional similarities across tasks. While each task introduces a specific set of actions, these actions are guided by the common structure ϵ ensuring that new actions align with the functional characteristics of existing actions.

2. When a new task is added to the environment, the mapping of the corresponding new actions to the underlying structure ϵ must be known. This mapping is represented as $a_T \times \epsilon \mapsto \{0, 1\}$, indicating that each new action a_T is defined in relation to the common underlying structure ϵ. This ensures that new actions are consistent with the existing functional framework, even as the specific action set evolves.

To formalise the decision-making process within a task-open system adhering to the

aforementioned assumptions, the following formalisation has been devised. Specifically, let TaO represent the challenge posed by task openness:

$$\textbf{TaO-MDP} \stackrel{def}{=} \langle \mathbf{M^T}, \mathbf{X^{Ta}}, \mathbf{G} \rangle$$

- $\mathbf{M^T}$ is the base decision-making model at the T^{th} timestep, which is a standard Markov Decision Process (MDP) defined by the state space, action space, transition function, and reward function available in the environment at timestep T. A detailed overview of MDP can be found in Section 3.2.

$$M^T \stackrel{def}{=} \langle S, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle^T$$

- $\mathbf{X^{Ta}}$ represents the outcomes of events, both exogenous and endogenous, that cause task openness in an environment. Exogenous variables are assumed to come from an external distribution independent of the agents' actions. Although endogenous variables can be influenced by agents' actions, they are not explicitly modeled; their outcomes simply update the task set.

$$X^{Ta} = \langle \tau, \omega \rangle$$

  $\tau \in \mathcal{X}^{Ta}$ includes parameters of tasks resulting from exogenous events, such as new tasks or tasks exiting the environment. Similarly, $\omega \in \mathcal{X}^{Ta}$ contains parameters of tasks affected by endogenous events. While τ is the outcome of exogenous variables(s), ω is the outcome of endogenous variable(s) present within M^T.

- $\mathbf{G}$ is the generator function that transforms the base decision-making model when an exogenous or endogenous event changes the set of available tasks, populating X^{Ta}. When X^{Ta} is set, G, along with ϵ, updates the current decision-making model M^T.

As mentioned in the assumptions, ϵ denotes the underlying action description of a task—a conceptual framework defining the functional similarities of actions across all tasks. This ensures that new tasks and their corresponding actions seamlessly integrate into the decision-making model.

$$M^{T+1} = \begin{cases} G(M^T, X^{Ta}) & \text{if } \tau \in X^{Ta} \text{ or } \omega \in X^{Ta} \text{ are non-empty} \\ M^T & \text{otherwise} \end{cases}$$

Components of $\mathcal{G}$ operate on parameters of the base model and generate new ones

- $S^{T+1} = G_S(S^T, X^{Ta})$: G updates the state space to include representations of newly added tasks and remove those of tasks that have exited, as indicated in X^{Ta}.
- $A^{T+1} = G_A(A^T, X^{Ta})$: G updates the action space using X^{Ta} and ϵ, combining existing actions with those required for new tasks, and removing actions associated with exited tasks.
- $T^{T+1} = G_T(T^T, X^{Ta})$: G adapts the transition function to incorporate the new states and actions while excluding those related to exited tasks. Transitions for new tasks in X^{Ta} are updated by comparing the underlying framework ϵ of the new tasks to the existing framework. This is present in X^{Ta}
- $R^{T+1} = G_R(R^T, X^{Ta})$: G similarly updates the reward function to account for the entry and exit of tasks using X^{Ta} and ϵ.

Consequently, the operational workflow for the agent consists of solving the current decision-making model either through planning or reinforcement learning, executing the obtained policy while simultaneously monitoring the exogenous variables in $\mathcal{X}^{Ta}$. If these variables indicate the arrival of new tasks, the agent pauses the execution of its current

policy and applies the generator $\mathcal{G}$ to the current model M^T to obtain a new model $\mathcal{G}(M^T)$, which the agent then solves to obtain the revised policy that is also cognizant of the new tasks.

In this thesis, the Task-Open variant of the rideshare domain serves as the experimental testbed. The subsequent sections offer an introduction to the task-open ridesharing domain, along with a formal representation of this domain defined as a TaO-MDP.

4.2 Example Domain: Dyamic Ridesharing

Ridesharing Qin et al. (2022) involves driver agents transporting passengers to their desired destinations. Traditionally, this domain has been modeled with a fixed set of passengers and predetermined tasks. In this section, we introduce a novel conceptualisation of task openness in the ridesharing domain: dynamic ridesharing. This task-open version allows new tasks (passenger requests) to appear dynamically, simulating real-world unpredictability. We discuss the design details of this dynamic environment, highlighting its complexities and the challenges it poses to the learning agents.

Each driver agent can transport up to k passengers in their vehicle at a time. Each passenger (task) has a pickup location, a desired drop-off location, and a fare assigned by a centralised algorithm external to the driver's decision-making. Thus, at any point in time, each driver agent must keep track of (1) how much room it has available to pick up additional passengers, (2) the drop-off location of each passenger currently being transported, and (3) the currently available tasks not yet collected by other drivers. This information is all fully observable to the driver.

Since the passengers carried by a driver change over time, and new passengers not yet assigned to a driver also appear dynamically, the action space of the high-level decision process of a driver agent also changes over time. As the specific passengers in the system

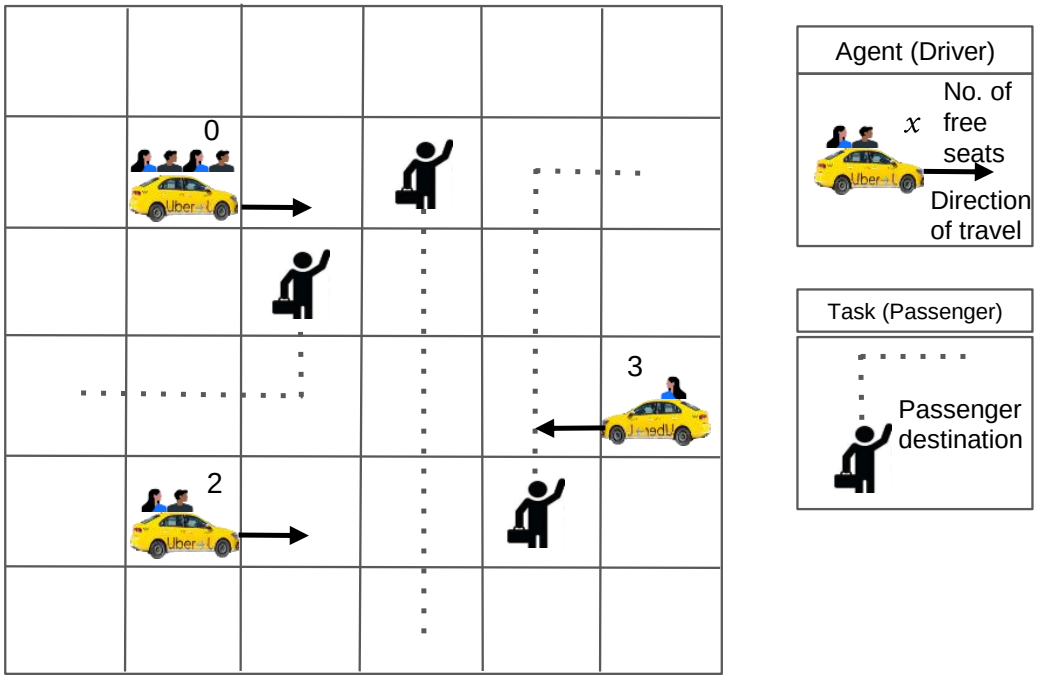

Figure 4.1: A dynamic ridesharing driver operates a vehicle in a task-open MAS where new tasks (passengers) suddenly appear and existing tasks complete leading to an *open* ground action set.

change over time (due to arrival or drop off), the actions relevant for each driver change, and the number of actions is not even constant but grows and shrinks depending on the current passengers in the system.

At any timestep, each driver must decide which passenger to serve, whether that is (1) a passenger currently occupying the driver's vehicle, (2) an outstanding passenger who recently appeared but has not yet been assigned to any driver, or (3) a newly assigned passenger who is not yet in the vehicle. Consequently, the action set associated with a passenger task includes accepting the passenger (*accept*), picking up the accepted passenger from its pick-up location (*pick*), and dropping the picked-up passenger at its drop-off location (*drop*). These actions must be carried out in the sequence of *accept-pick-drop* to complete a task.

Notably, we model *pick* and *drop* actions as one-step actions, rather than using temporally

extended actions as is common in the literature Jiao et al. (2021); Oda (2021). This design choice facilitates task pooling, allowing agents to handle multiple passengers concurrently. While temporally extended actions could simplify the learning process by bundling multiple steps into a single high-level action, our approach requires agents to perform *pick* and *drop* actions repeatedly over several timesteps to complete a task and realise the reward.

Changes in the passengers (tasks) both assigned to a driver agent and available for assignment also change the reward function optimised by each driver. This is because new passengers bring new actions that were not previously accounted for by the reward function and because each passenger has a new fare that they earn the driver when dropped off at their destination location (where fares generally increase with longer distances from pickup to drop-off locations).

Importantly, these changes to the reward function over time also cause the utility function (i.e., discounted cumulative rewards) of the driver agents to also change over time with task openness, including for actions that previously existed. For instance, at one point in time, a driver might currently have two passengers A and B that need to be dropped off at locations on opposite sides of the map, causing the driver's utility function to be higher for serving the closest passenger A (due to earning positive rewards for completing a task more quickly). However, suppose a new passenger C were to enter the system with a similar destination as the non-served passenger B. In that case, the driver might pause their current task (serving passenger A) because moving instead towards the destinations of B and C would now have higher utility due to task openness through the newly introduced task assigned to the driver.

Notably, changes to the set of passengers over time in dynamic ridesharing *cannot simply be modeled as* agent *openness*, for which existing planning and reinforcement learning solutions already exist (as described above). Once a passenger enters the system, they are generally not autonomous actors – each passenger's pick-up and drop-off locations and fares are constant, and they simply trust the drivers in the system to take them to their drop-off

location. Thus, drivers do not need to model them as autonomous actors for which they should adapt their own behavior; instead, the passengers represent unique tasks whose individual descriptions do not change while the set of available tasks changes over time. Thus, existing solutions for addressing agent openness cannot be reused in this domain, as reflected in the fact that those prior works do not consider dynamic action sets nor changing reward functions.

4.3 Modeling Dynamic Ridesharing as TaO-MDP

Having discussed the definition of TaO-MDP in the previous section, this section will instantiate the previously defined dynamic ridesharing domain as a TaO-MDP. Figure 4.2 illustrates a snapshot of the dynamic ridesharing environment at timestep t. Here, we formally define the TaO-MDP for the ridesharing domain, with a particular focus on the environment at timestep t.

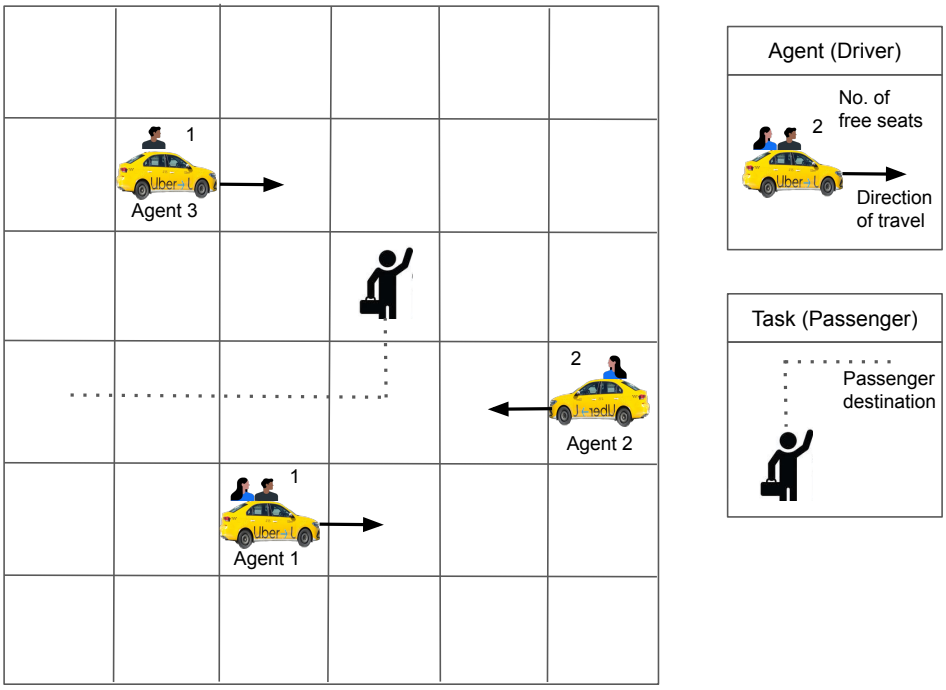

Figure 4.2: Snapshot of the Dynamic Ridesharing Domain at Time Step t

Formally, the TaO-MDP for the dynamic ridesharing problem can be defined as:

$$\textbf{TaO-MDP} \stackrel{def}{=} \langle \mathbf{M^T}, \mathbf{X^{Ta}}, \mathbf{G} \rangle$$

- The **base decision-making model $\mathbf{M^T}$** is an MDP describing the state space, the action space, the transition function and the reward function of dynamic ridesharing. Let us represent the MDP at timestep t as M^T

$$M^T \stackrel{def}{=} \langle S, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle^T$$

  - The **state** of the environment captures the location of all agents and detailed information about all tasks in the system, including the pick-up and drop-off lo-

cations of passengers and the ride fare. In dynamic ridesharing, we use a matrix to represent the state, where the grid cells indicate the agent's locations and passenger details. Information about a passenger traveling with an agent is represented within the same cell where the agent is present.

Below is an example representation of the state at timestep t as shown in Figure 4.2

$$s = \begin{bmatrix} (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) \\ (\{\},\{\}) & (\{Ag_3\},\{P_{31}\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) \\ (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{P_1\}) & (\{\},\{\}) & (\{\},\{\}) \\ (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) & (\{Ag_2\},\{P_{21}\}) \\ (\{\},\{\}) & (\{Ag_1\},\{P_{11},P_{12}\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) \\ (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) \end{bmatrix}$$

where, $s \in S^T$, $Ag_i = [passenger\_count, ride\_status, ...]$ and

$P_{ij} = [pick\_loc, drop\_loc, ride\_fare, servicing\_agent, ...]$

The agents at (4, 2), (3, 5), and (1, 1) are represented as Ag_1, Ag_2, and Ag_3, respectively. The passengers being serviced by Ag_1 are represented as P_{11} and P_{12}. The passenger being serviced by Ag_2 is represented as P_{21}, and the passenger being serviced by Ag_3 is represented as P_{31}. The new un-serviced passenger is represented as P_1. As shown above, the agent and passenger variables are vectors that capture more detailed information about them.

– The **action** space for each passenger includes *accept*, *pick*, and *drop* actions, along with a *no-operation* action allowing agents to do nothing for a timestep. Each action is specific to a particular passenger, meaning that an *accept*, *pick*, or *drop* action for one passenger will have a completely different effect than the same

action for another passenger. This is because the transition function and the effect on the environment are unique for each passenger based on their current state and location. For example, a *pick* action for P_{12} involves agent Ag_1 moving closer to P_{12}'s pick-up location, which may be different from P_{21}'s pick-up location, thus affecting the agent's travel path differently.

An example representation of the action space at timestep t is shown below:

$$\mathcal{A}^T = \{\text{accept}_{P_{01}}, \text{drop}_{P_{11}}, \text{drop}_{P_{12}}, \text{drop}_{P_{21}}, \text{drop}_{P_{31}}, \text{no-op}\}$$

Each action $a \in A^T$ is a vector describing parameters related to that action, such as pick-up or drop-off locations. For passengers already being serviced, only *drop* actions are available, as other actions have been completed. For the new task P_{01}, the only feasible action at this step is *accept*.

– The **transition** function determines the next state of the environment based on the actions carried out by the agents. In this domain, the transition function is deterministic.

  For example, if an agent Ag_i performs a *drop* action for passenger P_{ij}, the transition function updates the state by moving Ag_i and P_{ij} both one step closer to P_{ij}'s drop-off location. The combined actions of all agents collectively influence the environment, resulting in a new state that reflects the updated positions and statuses of both agents and passengers.

– The **reward** function determines the reinforcements agents receive for different actions. Shown below is a representation of the reward function for the dynamic ridesharing. Agents receive a negative move cost for every step they take towards the pick-up and drop-off locations of the passengers, but receive a relatively huge positive reward when they finally drop off the passenger.

$$R(s, a_i, s') = \begin{cases} r_{\text{fare}_{P_{ij}}} & \text{if } a_i = \text{drop}_{P_{ij}} \text{ and } \text{Ag}_i \text{ is at } P_{ij}\_drop\_loc \text{ in } s' \\ -c_{\text{move}} & \text{if } a_i = \text{pick}_{P_{ij}} \\ -c_{\text{move}} & \text{if } a_i = \text{drop}_{P_{ij}} \text{ and } \text{Ag}_i \text{ is not at } P_{ij}\_drop\_loc \text{ in } s' \\ -c_{\text{overload}} & \text{if } a_i = \text{accept}_{P_{ij}} \text{ and } Ag_i\text{'s pooling limit is exceeded in } s' \\ -c_{\text{conflict}} & \text{if } a_i = \text{accept}_{P_{ij}} \text{ and } P_{ij} \text{ is also accepted by a nearer agent} \\ 0 & \text{if } a_i = \text{no-op or } a_i = \text{accept}_{P_{ij}} \end{cases}$$

where, a_i is the action carried out by Ag_i. $-c_{move}, -c_{overload}, -c_{conflict}$ are the costs for moving, exceeding pooling limits, and accepting passengers closer to other agents, respectively. $r_{\text{fare}_{P_{ij}}}$ is the ride fare for the respective passenger.

- The **task openness events $\mathbf{X^{Ta}}$** capture the arrival and departure of passengers in the system. $\tau \in X^{Ta}$ includes all passengers entering or leaving the environment due to exogenous factors, such as new passenger requests or cancellations. $\omega \in X^{Ta}$ encompasses passengers entering or exiting the environment due to endogenous factors, such as dropping off the passenger. Each task in τ and ω contains the pick-up and drop-off locations and the ride fare. This thesis addresses task entry due to exogenous factors and task exit due to endogenous factors. Thus, in the version of dynamic rideshare in this thesis, ω only includes tasks leaving the environment through arriving at their destination, and τ only includes tasks entering the environment through calling for a ride.

  For example, in state s shown in Figure 4.2, if Ag_3 executes $drop_{P_{31}}$ and reaches its drop-off location in s', the completion of task P_{31} is triggered. Suppose a new passenger P_{02} enters the environment in the next time step. The endogenous task completion

event and the exogenous new passenger event modify the decision-making model's components as follows:

$$\mathcal{T} = \{(task\_entry\_bool, P_{02\_pick\_loc}, P_{02\_drop\_loc}, r_{fare_{P_{02}}}, \epsilon_{P_{02}})\}$$
$$\mathcal{W} = \{(task\_entry\_bool, P_{31\_pick\_loc}, P_{31\_drop\_loc}, r_{fare_{P_{31}}}, \epsilon_{P_{31}})\}$$

The **underlying action description** ϵ contains the vectors for actions required to complete the task and a mapping of how these new actions correspond to existing ones in the action space. For instance, the action vector of $accept_{P_{02}}$ maps to other *accept* actions in the action space as they are functionally similar, despite differing immediate transitions in the environment.

- The **generator** (**G**) in dynamic ridesharing updates the decision-making model when new tasks (passengers) arrive. This function updates the state space, action space, transition function, and reward function based on the new task information in τ and ω. The pick-up and drop-off locations, along with the ride fare information in X^{Ta}, update the state to reflect the new passengers available for the agents. The actions accompanying each new passenger, as specified in ϵ, update the action space. Consequently, the reward and transition functions are revised based on the new state and action space.

  In the running example, X^{Ta} is populated due to both endogenous and exogenous events, triggering the generator G to create a new decision-making model M^{T+1}. The next state s' is updated by G_S to include a new passenger P_{02}, while passenger P_{31}, whose service is completed, is removed. The action space is updated as follows:

$$A^{T+1} = G_A(A^T, X^{Ta}) = \{\text{accept}_{P_{01}}, \text{accept}_{P_{02}}, \text{drop}_{P_{11}}, \text{drop}_{P_{12}}, \text{drop}_{P_{21}}, \text{no-op}\}$$

With the updated state and action space, the reward and transition functions are also updated using ϵ in τ and ω through G_R and G_T, respectively. The updated reward function R^{T+1} will adjust to include or exclude the ride fares of entering or exiting passengers. The newly introduced actions, indicated by ϵ, will be mapped to the existing actions, extending the costs of moving, picking, and other actions to the new tasks. Additionally, the transition function is expanded to accommodate the transitions for the modified set of tasks.

Modeling the dynamic ridesharing domain as a TaO-MDP provides a structured and comprehensive framework to address the complexities introduced by task openness. By formalizing the state, action, transition, and reward functions, alongside the integration of task openness events and the generator function, this approach captures the dynamic nature of ridesharing environments. This framework not only facilitates the development of robust multi-agent reinforcement learning algorithms but also sets a foundation for further research into dynamic, task-open systems.

Chapter 5

MARL for Task-Open Systems

In the realm of task-open environments, the dynamic nature of the action space presents a significant challenge for developing effective learning algorithms. Traditional Reinforcement Learning (RL) algorithms, which are designed to learn a static mapping from a stable state space to a fixed action space, are inadequate for handling the ever-changing action dimensions inherent in task-open systems. This chapter explores an innovative approach that selects the optimal action based on a relative evaluation of the set of actions available at a specific time step, contingent upon the current state.

This chapter is structured as follows: Section 5.1 discusses the state-task representation in task-open systems by introducing interaction hypergraphs and interaction incidence graphs. These representations capture the present state, along with the existing set of tasks and associated actions, in a graphical format, serving as a crucial component of the approach. Section 5.1.2 provides a concrete example of constructing an interaction hypergraph and interaction incidence graph for a specific state in the dynamic ridesharing domain at a given time step. Moving forward, Section 5.2 introduces the novel MARL algorithm, MOHITO, which selects the optimal action through a relative evaluation of the current set of actions. Section 5.2.1 discusses the detailed architecture of the MOHITO algorithm, while Section

5.2.2 focuses on the training algorithm, outlining the steps and methodologies used to train agents effectively in task-open environments.

5.1 State-Task Representation in Task-Open Systems

The cornerstone of our approach is the creation of a generalised policy that is informed by relative evaluations. This necessitates the policy to consider the present state in tandem with the existing set of tasks and the associated actions. We encapsulate this approach within the framework of *interaction hypergraphs*.

At each time step, this graphical representation encapsulates the agent's observable state and tasks, along with the associated action set. These elements are graphically combined into the interaction hypergraph and subsequently interpreted through a Graph Neural Network. By employing this approach, we overcome the limitations imposed by pre-defined action spaces, or even agent spaces in environments exhibiting Agent Openness, resulting in a more adaptable and dimensionally agile RL paradigm.

As in any graph, a hypergraph comprises 3 entities - nodes, edges, and global attributes. However, in a hypergraph, an edge can join any number of nodes. The interaction hypergraph comprises three distinct types of nodes: *Agent nodes* that encapsulate state-specific information of agents; *Task nodes* that store information about the tasks available in the environment; and corresponding *Action nodes* that hold information about the actions that are required to complete each task. We adopt a 3-uniform hypergraph to represent an interaction graph, where each edge connects exactly three nodes. To be precise, each edge in the interaction hypergraph connects an Agent node, a Task node, and an Action node. Edges are introduced between all Agent, Task, and corresponding Action nodes to capture the comprehensive set of potential interactions that agents may have with extant tasks in the environment. Incorporating these edges serves a pivotal role in quantifying the effectiveness

of any Agent-Task-Action combination in the given state.

5.1.1 Interaction incidence graphs

Hypergraphs, while powerful for capturing complex relationships, can introduce various computational complexities, including but not limited to high computational overhead. These factors can limit their practical applicability to large-scale scenarios. Consequently, we transform the interaction hypergraphs into their corresponding incidence graphs, which serve as the bipartite equivalent of a hypergraph. This transformation represents the same combinatorial data as the original graph but with 2-uniform edges instead.

To facilitate this transformation, we introduce a new set of nodes, designated as *Hyperedge nodes*. Each 3-uniform edge that previously connected Agent, Task, and Action nodes in the original hypergraph is now represented by a dedicated Hyperedge node. Subsequently, 2-uniform edges are established to connect this Hyperedge node with each of the pre-existing Agent, Task, and Action nodes. Notably, the edges must be directed inwardly, from the Agent nodes, Task nodes, and Action nodes converging upon the corresponding Hyperedge nodes. Such inward orientation highlights the aggregation of information into the Hyperedge node, reflecting the fusion of information from the state and available task and action set a singular, analytical entity.

The resulting bipartite graph, termed an *interaction incidence Graph*, allows for the assessment of the efficacy of any Agent-Task-Action combination within the environment through the node embedding of the connecting Hyperedge node. The methodology to compute the node embedding of a Hyperedge node is discussed in the subsequent section. Note that with an incidence graph, the computational complexity is markedly reduced without compromising the integrity of the information represented, allowing for a more scalable and efficient analysis in vast and dynamic environments.

5.1.2 State-Task dynamics in Ridesharing

To elucidate the manner in which a state, its tasks, and the relevant actions are encapsulated within an interaction hypergraph—and consequently, an incidence Graph—let us consider a specific timestep within the Dynamic Ridesharing domain, as depicted in Figure 4.2. At this juncture, the environment comprises three agents and five passengers, four of whom are engaged in transit, with one passenger awaiting acceptance.

Focusing on eastbound Agent 1, positioned at cell (2, 3) and currently accommodating passengers A and B, we denote the newly available passenger as passenger C. Each passenger represents a distinct task, replete with a suite of potential actions; thus, Agent 1's purview encompasses Tasks A, B, and C, as well as their corollary Actions A, B, and C. For instance, implementing Action A would entail progressing towards Passenger A's designated drop-off point, while selecting Action C would involve the acceptance of Passenger C. It is pivotal to acknowledge that Agent 1's perception of the state excludes passengers presently in the care of other agents. While each agent maintains awareness only of its accepted and in-service passengers, all unaccepted passengers within the environment remain observable to the collective.

From Agent 1's point-of-view, the corresponding incidence hypergraph is articulated in Figure 5.1. Within this framework, the 3-way Hyperedge 1 connects Agent 1, the ongoing Task A, and the consequential Action A. Hyperedges 1, 2, and 3 are shown in red to signify the tasks under Agent 1's consideration. Additionally, the illustration shows Hyperedge 4, which interconnects Agent 2 with Task C and the respective Action C, and Hyperedge 5, which aligns Agent 3 with Task C and Action C, highlighting the visibility of these tasks to all agents within the purview of the environment. Given that this hypergraph represents Agent 1's observations, it intentionally excludes the tasks being serviced by Agents 2 and 3.

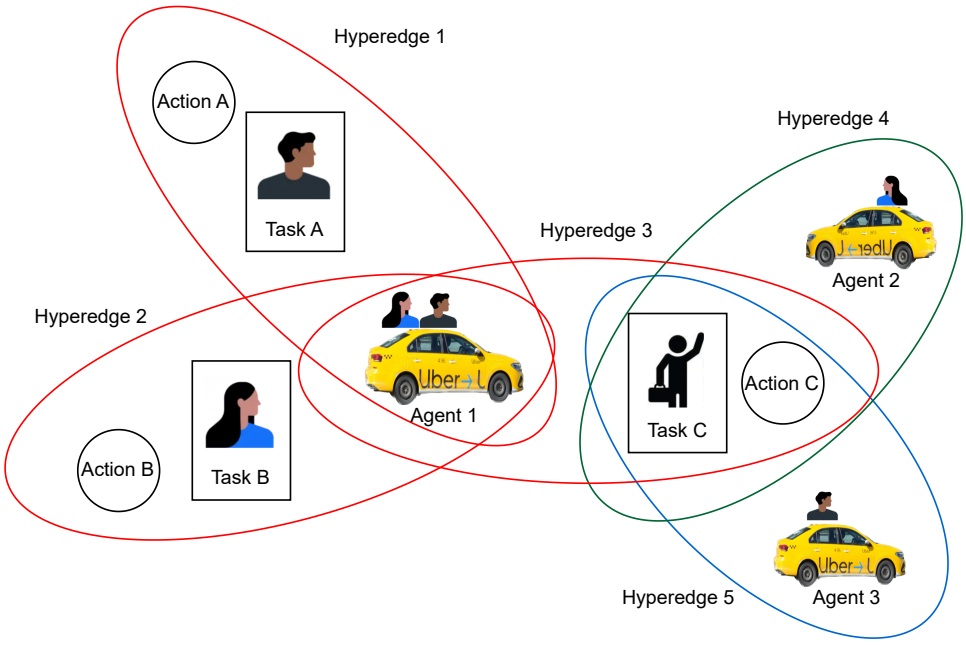

Figure 5.1: Observation interaction hypergraph capturing the state of the Dynamic Ridesharing domain at timestep t

The transformed interaction incidence graph, derived from the hypergraph depicted in Figure 5.1, is exhibited in Figure 5.2. The original hyperedges have been replaced by distinct Hyperedge nodes. These nodes form the new nexus of the graph, each branching out via directed edges to a specific agent, task, and action triad formerly unified by the corresponding hyperedge. In this graphic representation, the Hyperedge nodes of Agent 1 are emphasised in red, while those correlated with Agents 2 and 3 are distinguished in green and blue, respectively. It is important to observe that all edges are consistently oriented inwards, converging upon the Hyperedge nodes, reinforcing the directionality of the relationships within the graph.

While only Hyperedges 1, 2, and 3 are pivotal for Agent 1 in determining the optimal

action to execute at this timestep, the inclusion of Hyperedges 4 and 5 is crucial in illustrating that the unaccepted Task C is also under consideration by other agents within the environment. We can see that Task C and Action C are connected to 3 different Hyperedge nodes and thus are in consideration by 3 decision-making agents. Note that, only nodes that are directly connected to an agent's hyperedge influence the agent's decision-making.

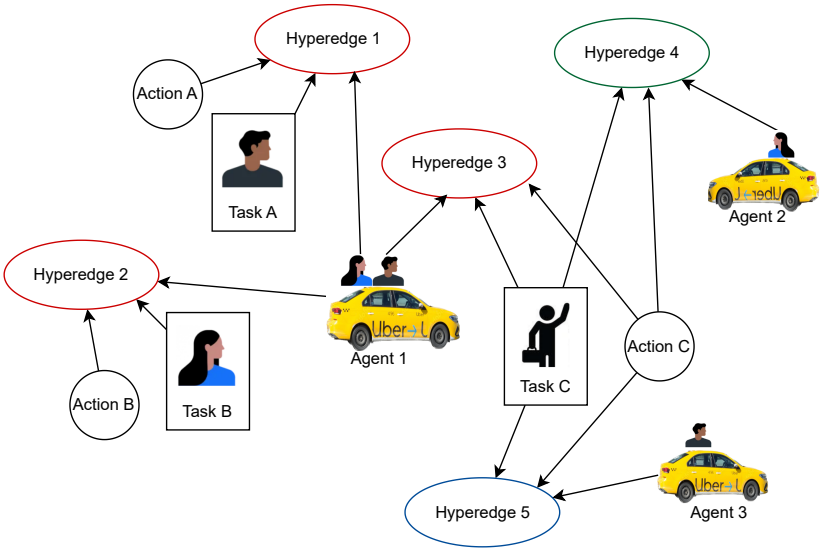

Figure 5.2: Observation interaction incidence graph capturing the state of the Dynamic Ridesharing domain at timestep t

While the above figures show the interaction incidence graph of one agent's observation of the environment, Figure 5.3 shows the interaction incidence graph of the state, where task associations of all agents are indicated

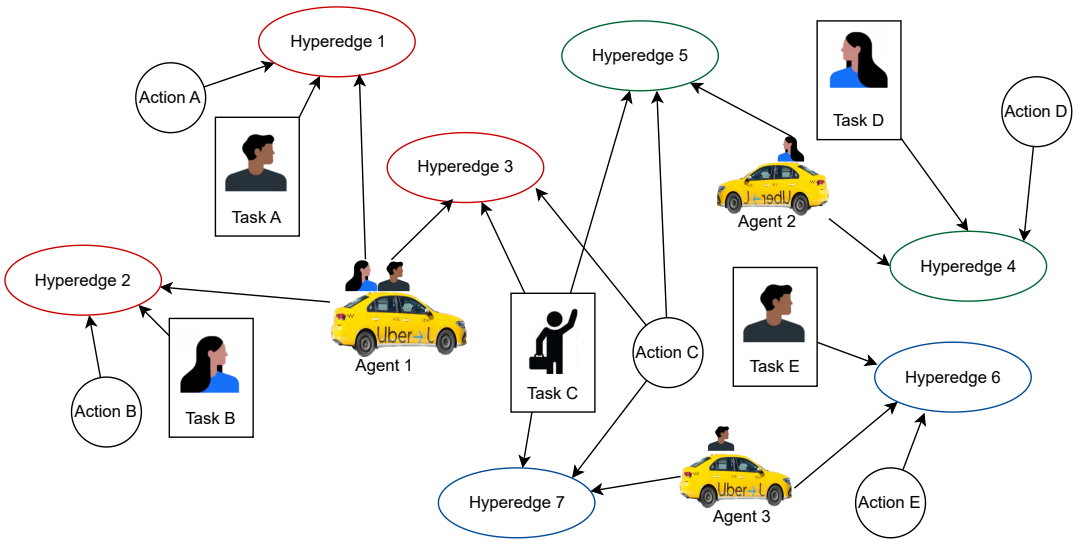

Figure 5.3: State interaction incidence graph capturing the state of the Dynamic Ridesharing domain at timestep t

The interaction incidence Graph is invariably accompanied by a feature matrix, serving as a way to integrate state, task, and action attributes into the respective Agent nodes, Task nodes, and Action nodes. In the Ridesharing domain, Agent nodes can be used to encapsulate the agent's positional data within the environment, alongside metrics such as the number of tasks presently assigned to them or their remaining fuel reserves. Task nodes can be used to capture pertinent task details, including passengers' origin and destination points, as well as the allocated ride fare. Similarly, Action nodes can be used to capture the specific action details to be executed concerning designated tasks. Given that Hyperedge nodes initially lack a concrete representation, they can be initially populated with infinitesimal random values, setting the stage for subsequent refinement and application.

5.2 MOHITO

Having delineated the challenges posed by dynamic, non-stationary environments, we now turn our attention to the architecture that enables agents to make optimal decisions within

such complex settings. In this section, we explore the details of the novel Multi-Agent Reinforcement Learning (MARL) algorithm - Models of Hyper Interactions under Task Openness (MOHITO). The primary objective of this algorithm is to evaluate the environment via the interactions among agents, tasks, and actions at each time step, thereby enabling the selection of the most suitable action for that particular time step.

The algorithm utilises an Actor-Critic framework and engages in learning based on the Centralised Training and Decentralised Execution (CTDE) strategy, and is designed specifically to leverage interaction graph representations of the environment. Like all CTDE approaches, each agent has a local actor (or policy) and a local critic. During the training phase, agents' policies are learned in a centralised manner by harnessing the joint observations of the state, and the joint action set that is carried out in the environment. This allows each agent to learn optimal strategies by considering *both* individual and collective objectives, respectively. Post-training, the algorithm transitions to a decentralised execution mode, where each agent makes an autonomous decision based on its own local observations and policies. This ensures coordinated learning while maintaining the real-world applicability that comes with decentralised operation.

5.2.1 Architecture

In this deep learning-based framework, both the actor and critic are defined as deep networks. To leverage the benefits of the graphical representation of interactions in the environment, we propose the use of Graph Neural Networks (GNN), specifically Graph Attention Networks (GAT), for both the actor and critic networks. GNNs are a class of deep learning techniques designed to perform inference on data described by graphs by performing optimizable transformations on graph attributes (e.g., nodes, edges, global attributes).

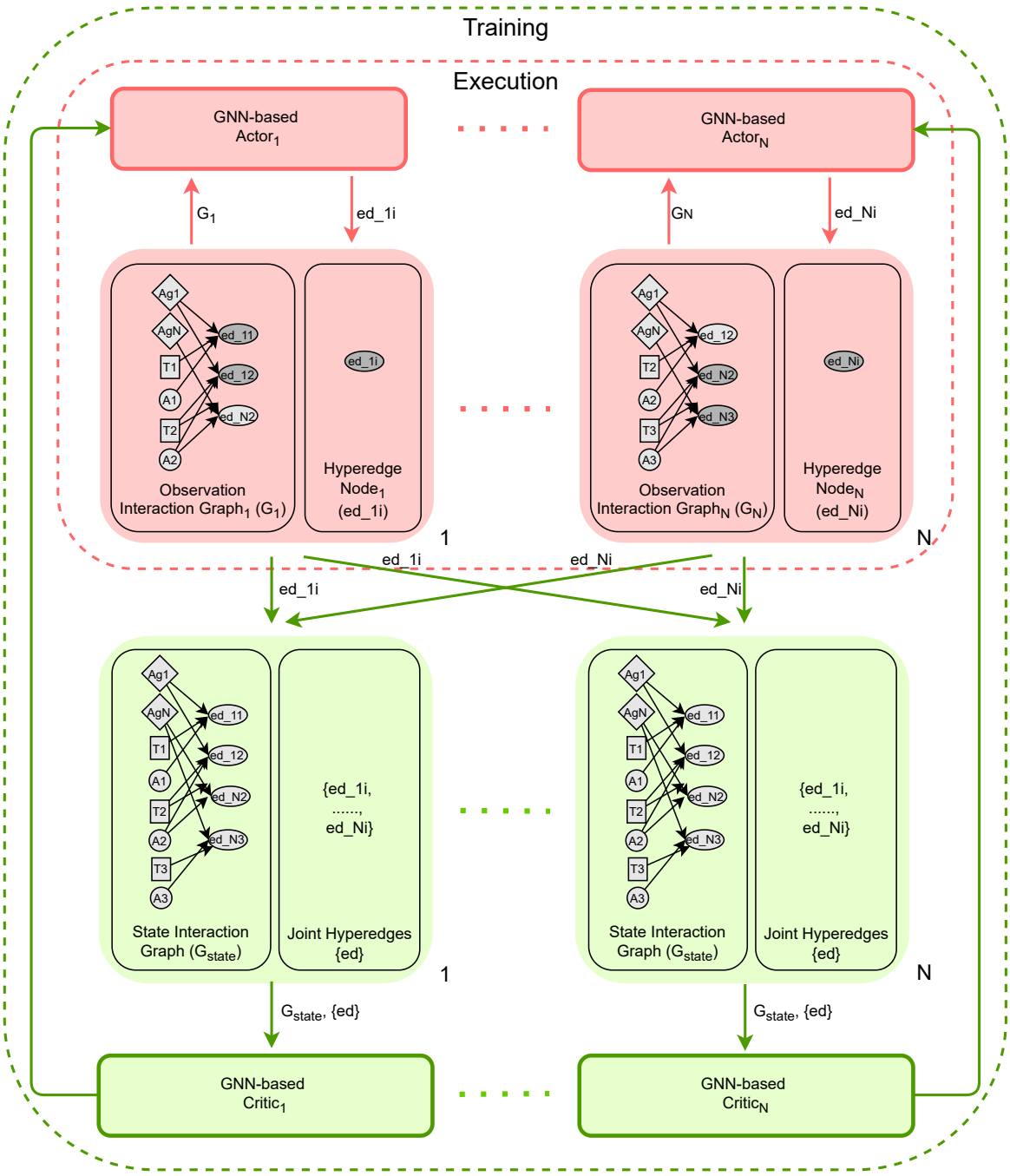

Figure 5.4: Overview of MOHITO Algorithm

We use the message-passing neural network framework proposed by (Gilmer et al.,

2017) with the Graph Networks architecture schematics introduced by (Battaglia et al., 2018b). Within this framework, each node in the graph communicates with its neighbors by exchanging and transforming information through message passing. Specifically, when a node transmits a message, it conveys a modified version of its own features to all neighboring nodes. Conversely, upon receiving messages from its neighbors, a node aggregates this information, incorporating it with its own features to enrich its understanding of the local graph structure.

The directionality of edges in the graph plays a crucial role in the message-passing process. In graphs with unidirectional edges, messages are passed in a single direction from the source node to the target node. This can be useful for tasks where the flow of information is inherently directional, such as modeling causal relationships or hierarchical structures. In contrast, graphs with bidirectional edges allow messages to be passed in both directions between connected nodes.

Furthermore, the integration of GAT layers into the network is pivotal. GAT layers calculate attention coefficients between a node and its neighbors, effectively quantifying the significance of features from neighboring nodes. This mechanism allows for a weighted emphasis on the most relevant neighboring nodes, enhancing the overall message-passing process.

Figure 5.4 shows an overview of the architecture of the MOHITO algorithm, featuring GNN-based actor and critic networks. The actor network processes an observation interaction incidence Graph (similar to the one in Figure 5.2), with directed edges converging from *Agent* nodes, *Task* nodes, and *Action* nodes towards the *Hyperedge* nodes. Through GNN processing, this network engages in message-passing, culminating in the aggregation of data from the *Agent*, *Task*, and *Action* nodes into the *Hyperedge* node—thus positioning the *Hyperedge* nodes as the principal decision-making entities.

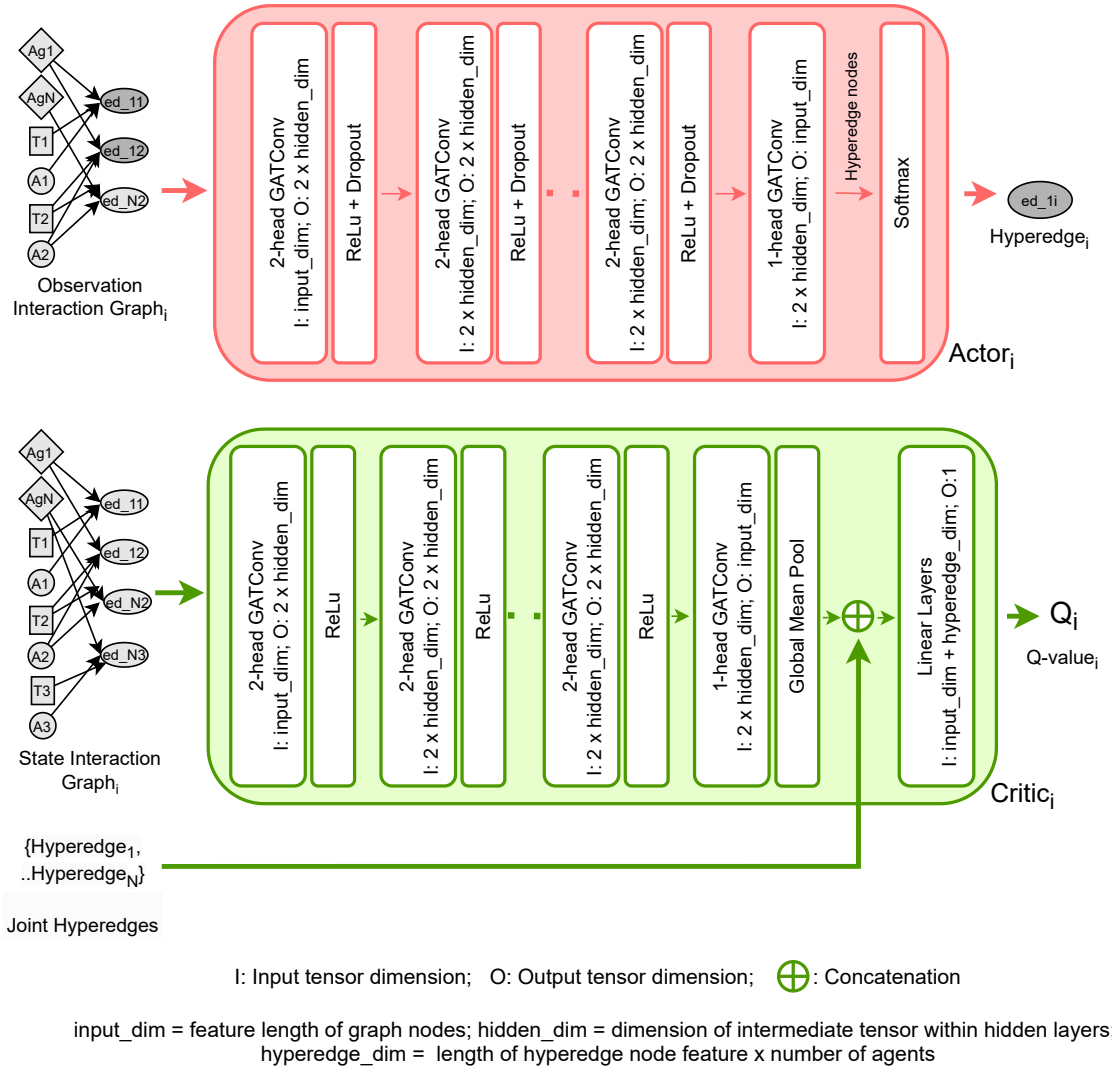

Figure 5.5: Architecture of the Actor and Critic Networks

The actor-GNN is designed to select one of the *Hyperedge* nodes connected to the learner agent, serving as the output. The detailed architecture of the actor network in Figure 6.1 shows this node filtering step, which occurs prior to the softmax operation on the learner agent's *Hyperedge* node features. Once the *Hyperedge* node with the highest value is selected, the corresponding action is obtained for updating the environment through a straightforward mapping of the *Action* node connected to it.

The critic network processes the state interaction incidence graph, similar to the one in Figure 5.3, along with the vector of values of the *Hyperedge* nodes generated by all agents' actor networks. The state interaction graph is processed by successive GAT layers, after which the node features are pooled and concatenated with the vector of *Hyperedge* node values. The new vector of values is then passed through a series of linear layers to compute the *Q-value*. The detailed architecture of the critic network can be found in Figure 6.1.

Note that the dropout layers are only used during training to avoid overfitting, and are turned off during execution. More details of the network architecture, including the dimensions and the number of layers used to generate the reported results, can be found in the Appendix section.

5.2.2 Training Algorithm

The proposed MARL architecture, Models for Hyper Interactions under Task Openness (MOHITO), is a cohesive integration of the advanced components described previously. It is designed to operate under the principles of CTDE, where the environmental parameters accessible to the agents are graphically encapsulated and subsequently processed to determine their course of action. An overview of MOHITO's training algorithm can be found in Figure 5.4.

Given a task-open environment (*env*) with N agents, at time step t, the environmental state (X_t) and set of available tasks ($T_t = \{T_0, .., T_k\}_t$) are defined. While agents can fully observe the environmental state, they can only observe a subset of the total tasks, the ones available to them for execution. Each agent i in the environment represents the multifaceted information of the state and the observed task set as an observation interaction incidence graph (Obs_i), containing *Agent, Task, Action*, and *Hyperedge* nodes, each described by a fixed-length feature vector. As *Hyperedge* nodes are generated nodes not associated with innate features, they are initialised with infinitesimal random values to seed the learning

process. This is indicated in lines $2-4$ in *Algorithm 2*

The GNN-based actor network (π_i) processes agent i's observation graph to return the *Hyperedge* node (ed_i) connected to it with the highest value. The corresponding action (a_i) that the *Hyperedge* node is connected to is also determined. Action selection using the policy can be seen in line 5 and the subroutine is described in *Algorithm 1.*

Upon determining the best *Hyperedge* nodes for all agents, a joint hyperedge set ($\{ed_0, ..., ed_N\}$) is created. This set represents the set of decisions made by the agents in state X at timestep t. The critic network now evaluates the quality of the decisions made by agents. The state (X_t), along with the complete set of tasks available in the environment (T_t), is converted into a state interaction incidence graph (G_{critic}), as shown in lines $12-13$. The critic network of each agent (Q_i) processes the state graph along with the joint hyperedge set to return a Q-value (q_i). This can be seen in lines $15-16$. The Q-value serves as an evaluative metric, indicating the expected cumulative future rewards that an agent can accrue from executing the selected decisions in the current state, under the current policy.

The joint set of actions carried out by each agent in the environment is used to determine the next state (X_{t+1}), the next set of tasks (T_{t+1}), and the reward set ($r_0, .., r_{N_t}$). Since the environment is task-open, the next set of tasks depends not only on the actions carried out by the agents but also on exogenous factors. This transition is indicated in line 6 of *Algorithm 2.*

In line with the principles of CTDE architectures, the centralised training phase in the algorithm allows the critic to leverage collective information for the evaluation of selected actions through Q-values. Meanwhile, during the decentralised execution phase, each agent relies exclusively on its own local observations and policy to make real-time decisions. The Q-values output by the critics, along with the reward set, are used to update the agents' actor and critic networks. The loss functions used to update the two networks are discussed in the subsequent sections. The update of the actor network is specified in lines $21-29$, and

that of the critic network is in lines $11 - 20$.

Algorithm 2 details the training process of MOHITO comprehensively. While the above description provides an overview, several crucial components are integral to the training algorithm. The epsilon-greedy action selection Sutton and Barto (1998) strategy, depicted in *Algorithm 1*, ensures exploration during learning. A random policy (π_{rand}) is employed to select a random hyperedge from the interaction graph. Additionally, the principle of double networks van Hasselt et al. (2016) is utilised to stabilise network updates, with the target actor and critic networks (π'_i, Q'_i) being updated at predetermined rates (τ_π, τ_Q). This can been in lines $30 - 32$ in *Algorithm 2*

MOHITO - Online training parameters

Input: A set of N agents in an episodic environment
Parameters:
  π^θ, π'^θ: Main and Target Actor networks with parameters θ and θ' for each agent
  Q^ϕ, Q'^ϕ: Main and Target Critic networks with parameters ϕ and ϕ' for each agent
  π^{rand}: Random Actor network for action exploration
  S: Minibatch size
  γ: Discount factor
  λ_π, λ_Q: Regularisation coefficient for Actor and Critic networks
  τ_π, τ_Q: Actor and Critic target network update rates

Algorithm 1 MOHITO - Action selection subroutine $ps(Obs, \pi) \rightarrow a, ed$

Input: Observations Obs, policy π
Output: action a, maximum hyperedge ed
for agent i **do**
  $Graph_i \leftarrow$generateIncidenceGraph(Obs$_i$) #see Section 5.1.2
  $Edges_i \leftarrow \pi_i(Graph_i)$
  $MaxEdgeIndex_i \leftarrow argmax_a(\sum Edges_i[a])$
  $ed_i \leftarrow Edges_i[MaxEdgeIndex_i]$
  $a_i \leftarrow \epsilon\text{-}greedy(MaxEdgeIndex_i, argmax_a(\sum \pi^{rand}(Graph_i)[a]), \epsilon)$
end for

Algorithm 2 MOHITO - Online Training

for *episode* $\leftarrow 1$ to E **do**
  Get start state X with tasks $\{T_0, \ldots, T_t\}$
  while *episode* is not terminated **do**
    Get observation incidence graphs $Obs_0, Obs_1, \ldots, Obs_{N-1}$
    $a, ed^{\theta} \leftarrow ps(Obs, \pi^{\theta})$
    $Obs', r \leftarrow env(a)$
    $a', ed'^{\theta'} \leftarrow ps(Obs', \pi^{\theta'})$
    Add $(X, ed^{\theta}, r, X', ed'^{\theta'})$ to buffer
    if $size$(buffer) $= S$ **then**
      $Loss_{Q_i}, Loss_{\pi_i} \leftarrow 0 \ \forall i \in$ agents
      for $(Obs, ed^{\theta}, r, Obs', ed'^{\theta'})$ in buffer **do**
        $G_{critic} \leftarrow generateIncidenceGraph(X)$
        $G'_{critic} \leftarrow generateIncidenceGraph(X')$
        for agent i **do**
          $q_i^{\phi} \leftarrow Q_i^{\phi}(G_{critic}, ed^{\theta})$ # **q actual**
          $q_i'^{\phi'} \leftarrow Q_i'^{\phi'}(G'_{critic}, ed'^{\theta'})$ # **q expected**
          $Loss_{Q_i} \leftarrow Loss_{Q_i} + MSE[q_i^{\phi}, r_i + \gamma * q_i'^{\phi'}] + \lambda_Q * |\phi - \phi'|$
        end for
      end for
      Backpropagate $Loss_{Q_i}$ to update Q_i^{ϕ}
      for $(X, ed^{\theta}, r, X', ed'^{\theta'})$ in buffer **do**
        $G_{critic} \leftarrow generateIncidenceGraph(X)$
        for agent i **do**
          $q_i^{\phi} \leftarrow Q_i^{\phi}(G_{critic}, \{ed^{\theta}\})$
          $Loss_{\pi_i} \leftarrow Loss_{\pi_i} - mean(q_i^{\phi}) + \lambda_{\pi} \times |\theta_t - \theta_{t-1}|$
        end for
      end for
      Backpropogate $Loss_{\pi_i}$ to update π_i^{θ} and clear buffer
    end if
    for each agent i after every K episodes **do** #**slow update**
      $\theta'_i \leftarrow \tau \times \theta_i + (1 - \tau) \times \theta'_i$
      $\phi'_i \leftarrow \tau \times \phi_i + (1 - \tau) \times \phi'_i$
    end for
  end while
end for

The training losses used in the MOHITO algorithm are similar to those used for the actor and critic networks in the MADDPG algorithm [Lowe et al. (2020)]. The actor loss function is defined as:

$$Loss_{\pi_i} = -E\left[Q_i^{\phi}(G_{\text{critic}}, \{ed_i^{\theta}\})\right] + \lambda_{\pi} \times |\theta_t - \theta_{t-1}|$$

In addition to the mean Q-value evaluation of the critic graph and joint hyperedges, a regularization term is included to minimize changes in the network parameters from one training update to the next. This regularization ensures that the network does not overfit to the non-stationarity of the environment, which is particularly caused by task openness.

The critic loss function is defined as:

$$Loss_{Q_i} = MSE\left[q\_\text{actual}, r_i + \gamma \times q\_\text{expected}\right] + \lambda_{\pi} \times |\phi_t - \phi_{t-1}|$$

$$\text{where,} \quad q\_\text{actual} = Q_i^{\phi}(G_{\text{critic}}, \{ed^{\theta}\})$$

$$q\_\text{expected} = Q_i^{\phi'}(G'_{\text{critic}}, \{ed'^{\theta'}\})$$

The critic loss is based on the Temporal Difference (TD) error, which measures the difference between the actual Q-value and the expected Q-value. The expected Q-value is calculated by evaluating the next state and the next action resulting from the current action taken at this time step. A similar regularization term is included in the critic's loss equation to ensure stability and prevent overfitting to transient changes in the environment.

Chapter 6

Experiments and Results

In this chapter, we present the experimental results of our novel MARL technique, MOHITO, and provide a comprehensive analysis of these results. Section 6.1 outlines the experimental setup, detailing the configurations and parameters used to train and evaluate agents' policies across various open systems. Section 6.2 introduces the performance baselines used for comparative analysis, providing a reference point for the evaluation of the MOHITO algorithm.

Section 6.3 evaluates the performance of the MOHITO algorithm through various subsections. Section 6.3.1 discusses key training metrics, such as reward curves, to provide insights into the learning process. Section 6.3.2 offers a detailed policy analysis, examining the behavioral outcomes of the learned policies and how agents handle dynamic tasks. Section 6.3.3 compares the performance of MOHITO-trained policies against the baseline policies, highlighting the algorithm's strengths and areas for improvement. Finally, Section 7.1 addresses the training limitations, discussing challenges like non-convergence for some agents and the complexities introduced by dynamic task sets, and outlining steps taken to enhance robustness. The results demonstrate that MOHITO significantly outperforms baseline policies, showing superior adaptability and efficiency in dynamic, task-open envi-

ronments. Additionally, MOHITO-trained agents quickly adapt to new tasks as they arrive, maintaining high performance and demonstrating the algorithm's robustness in handling the unpredictability inherent in task-open systems.

6.1 Experimental Setup

This section discusses the dynamic ridesharing training and testing setups employed for the results presented in this chapter.

Section 6.1.1 details the training and evaluation setup, describing the configurations and parameters used to train the agents. This includes an explanation of the environment, the initialisation of agents and tasks, and the criteria used to evaluate the performance of the learned policies during training. Section 6.1.2 focuses on the testing setup, outlining the procedures and configurations used to test the trained agents in various task-open scenarios.

6.1.1 Training and Evaluation Setup

Each training experiment involves agents being initialised at random locations within the dynamic ridesharing grid, alongside a few passengers with varying start and endpoints. Throughout the episode, additional passengers with distinct start and endpoints are gradually introduced. At the start of training, agents are initialised with random policies in a task-open ridesharing environment. The training progresses as the episode unfolds, following the algorithm outlined in Section 5.2. Each episode lasts for 100 steps, with tasks introduced during the first 70% of the episode's duration. The number of tasks is kept proportional to the number of agents present in the environment to ensure that agents aren't overloaded with options during training. The results reported in subsequent sections have used environments containing three times the number of passengers for training. At the end of each episode, the environment is reset for a new episode, while the agents' policies carry forward.

Although agents are trained to navigate task-open environments where they may encounter an unpredictable set of tasks, the training process is carried out within a singular, consistent environment. This means that every time the environment is reset during training, it is reset with the same episode as before. Agents face a fixed set of tasks that recur at the same locations, episode after episode, during training. This consistency helps focus the training on fostering the ability to interpret graphs and make informed decisions based on them. Particularly, considering that agents must figure out how to repeatedly perform a *pick* or *drop* action for several timesteps to effectively complete the activity, the consistent task setup aids in honing these skills.

As training progresses, after every few training steps, the learned policies are evaluated in predefined evaluation environments containing tasks that were not encountered by the agents during training. These evaluation environments consist of the same number of tasks as used during training, which is three times the number of agents. The rewards accumulated in these validation episodes are utilised to monitor policy convergence.

6.1.2 Testing Setup

Upon obtaining converged policies for agents, the performance of these policies are evaluated in pre-determined test environments. Similar to the evaluation environments, the test episodes contain tasks that were previously unseen by the agents in both training and validation setups. The test environments are configured with varying levels of openness. This thesis reports performance across three predefined levels of openness:

- *Openness Level 1 (OL1)* - This setting has the lowest task entry rate and has about 6 tasks entering the environment during the episode.
- *Openness Level 2 (OL2)* - This setting has a medium level of task entry rate, with about 9 tasks entering the environment during the episode.

- *Openness Level 3 (OL3)* - This setting has the highest task entry rate and is the toughest setup that we have used for testing in this thesis. This setting has about 12 tasks entering the environment during the episode.

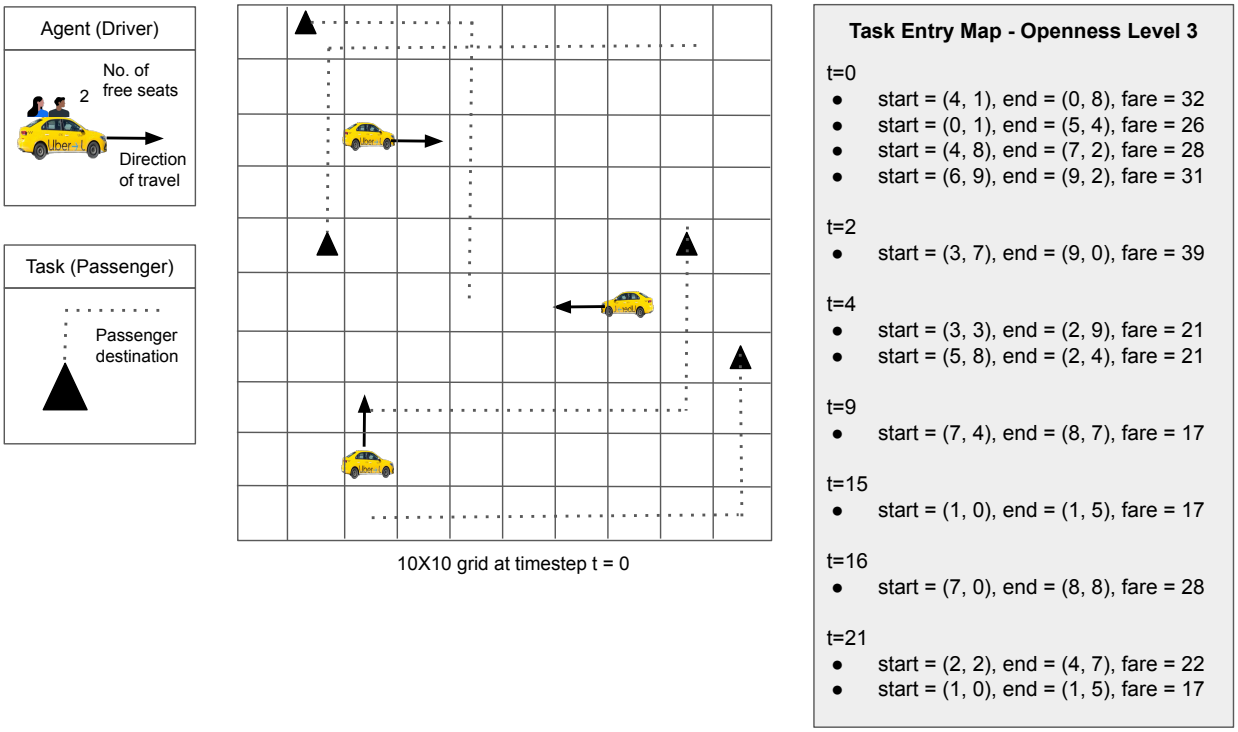

Figure 6.1: Task entry map for an example test episode with 3 agents operating at openness level 3

We test multi-agent setups involving 2, 3, and 4 agents across the three levels of openness mentioned above. The results of these tests are detailed in the subsequent sections. Note that agents have only been trained with a fixed number of tasks in the environment. The levels of openness are only for testing to evaluate how well agents are able to adapt to varying task entry rates and handle the dynamic nature of task-open environments. This approach helps assess the robustness and flexibility of the learned policies in managing different degrees of environmental complexity and task dynamism.

6.2 Performance Baselines

To establish the comparative efficacy of the MOHITO algorithm within the context of task openness in multi-agent environments, we require baseline policies. Since there is no prior research that explores learned policies in environments characterised by task openness, there isn't a state-of-the-art policy for the dynamic ridesharing domain that can be directly used as a baseline. Thus, the aim is to find a reference point for the most straightforward autonomous behaviors. The following static task allocation policies have been chosen due to their elemental and interpretable nature, serving as benchmarks for autonomous decision-making within the rideshare domain. They rely on predefined rules to distribute tasks among agents. By comparing and contrasting the behaviors learned by agents using the MOHITO algorithm against those exhibited by these baseline policies, we aim to emphasise the strategic depth and adaptability of the model in handling the complexities and dynamic aspects of task-open systems.

- **First-come First-serve Policy (FCFS):** Under this policy, the actions of the agents are determined by the order of task entry in the environment. Agents give precedence to tasks in the sequence they emerge, favoring the earliest tasks over newer ones. Agents take actions to complete one task, before switching to the next one. This policy encapsulates a rudimentary yet intuitive approach, aligning with human heuristics of task prioritisation and is reflective of traditional queue-based processing systems.

- **Greedy Distance Policy (NTF):** In this policy, agents always choose the nearest task, thereby minimising the distance traveled and consequently the effort expended. Once an agent picks the nearest available task, it performs all actions required to complete the task at hand, before considering the next task to take up. This policy encapsulates an approach grounded in the principle of immediate gratification, disregarding longer-term strategic considerations for real-time, distance-based expediency.

While the static task allocation policies listed above serve as a baseline for simple autonomous behaviors, they are not adequate for evaluating the performance of MOHITO-trained policies, which incorporate task pooling. The areas of research closest to task openness are lifelong learning and multi-task learning, as discussed in Chapter 2. Policy Gradient Efficient Lifelong Learning Algorithm (PG-ELLA) Ammar et al. (2014a) is primarily a lifelong learning algorithm, though it also incorporates elements of multi-task learning. Thus, the following adaptation of PG-ELLA has also been considered as a baseline:

- **Open-task Policy Gradient Efficient Lifelong Learning Algorithm (OTPG-ELLA):** This algorithm is an adaptation of the original PG-ELLA to address environments with dynamically changing task sets, enabling agents to continuously adapt and optimise their policies as new tasks emerge. OTPG-ELLA employs a policy gradient framework enhanced for efficient lifelong learning, facilitating the integration of new tasks without compromising performance on established ones. It excels in scenarios where task predictability is low and adaptability is essential.

Implementation details of OTPG-ELLA can be found in the Appendix section. OTPG-ELLA contrasts with static task allocation strategies by prioritising adaptability and comprehensive task engagement over simpler, more immediate task resolution strategies. Consequently, it provides a good benchmark for evaluating MOHITO's performance in dynamic task-oriented environments.

6.3 Performance Evaluation

This section assesses the efficacy of the novel multi-agent RL algorithm MOHITO. Central to the examination is the performance of the algorithm in the task-open Rideshare domain. In this domain, each passenger dynamically added to the environment system equates to a new

task, creating a problem where agents' adaptability and strategic foresight are paramount. We subject the algorithm to a series of scenarios, gauging its capacity for knowledge transfer, its proficiency in handling simultaneous tasks, and its overall reward optimisation.

We conduct a detailed analysis of trajectories to highlight the behavioral superiority imparted by this training algorithm within the learned policies. This section emphasises both the quantitative benefits, such as reward maximisation, and the qualitative improvements in agent decision-making, establishing MOHITO as a significant advancement for multi-agent task-open systems.

Section 6.3.1 discusses key training metrics, such as reward curves, to provide insights into the learning process. Section 6.3.2 offers a detailed policy analysis, examining the behavioral outcomes of the learned policies and how agents handle dynamic tasks. Section 6.3.3 compares the performance of MOHITO-trained policies against the baseline policies, highlighting the algorithm's strengths and areas for improvement. Finally, Section 7.1 addresses the training limitations, discussing challenges like non-convergence for some agents and the complexities introduced by dynamic task sets, and outlining steps taken to enhance robustness.

6.3.1 Training Metrics

In this subsection, we discuss the key metrics that provide insights into the training process of our novel multi-agent RL algorithm, MOHITO. We focus on two primary aspects: reward-based convergence analysis and training losses.

The evaluation of MOHITO's training process reveals a significant challenge: not all agents converge to optimal policies. This challenge is illustrated in the following figures and discussed in detail.

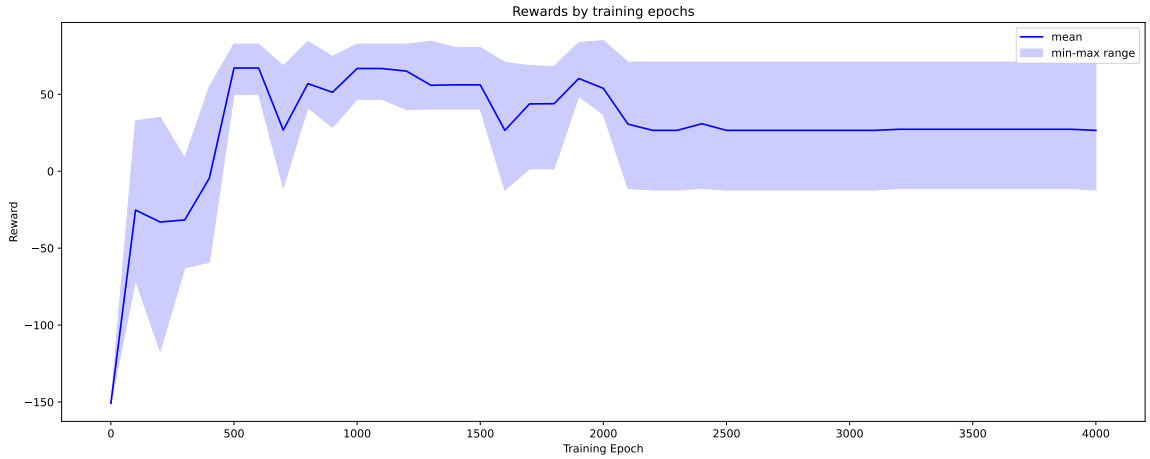

Figure 6.2: Mean rewards achieved in validation episodes through training epochs for a setup with 4 agents

Figure 6.2 shows the rewards obtained by agents in predetermined validation environments throughout training epochs. This figure reports the mean rewards, along with the standard deviation, that the agents achieved by the agents during evaluation episodes in a 4-agent setup. The evaluation episodes used to record the rewards comprised a combination of openness levels 1 and 2.

While the rewards appear to be converging to a stable point, the level of reward at the point of convergence seems to be slightly lower than the reward before convergence. To gain further understanding, Figure 6.3 displays the accumulated rewards by each agent in multi-agent environments with 2-, 3-, and 4-agent setups. Note that the bottom plot, showing rewards for a 4-agent setup, is the same as in Figure 6.2. This figure helps to understand the reward levels that each agent converged to. It is observable that not all agents converged to an improved policy.

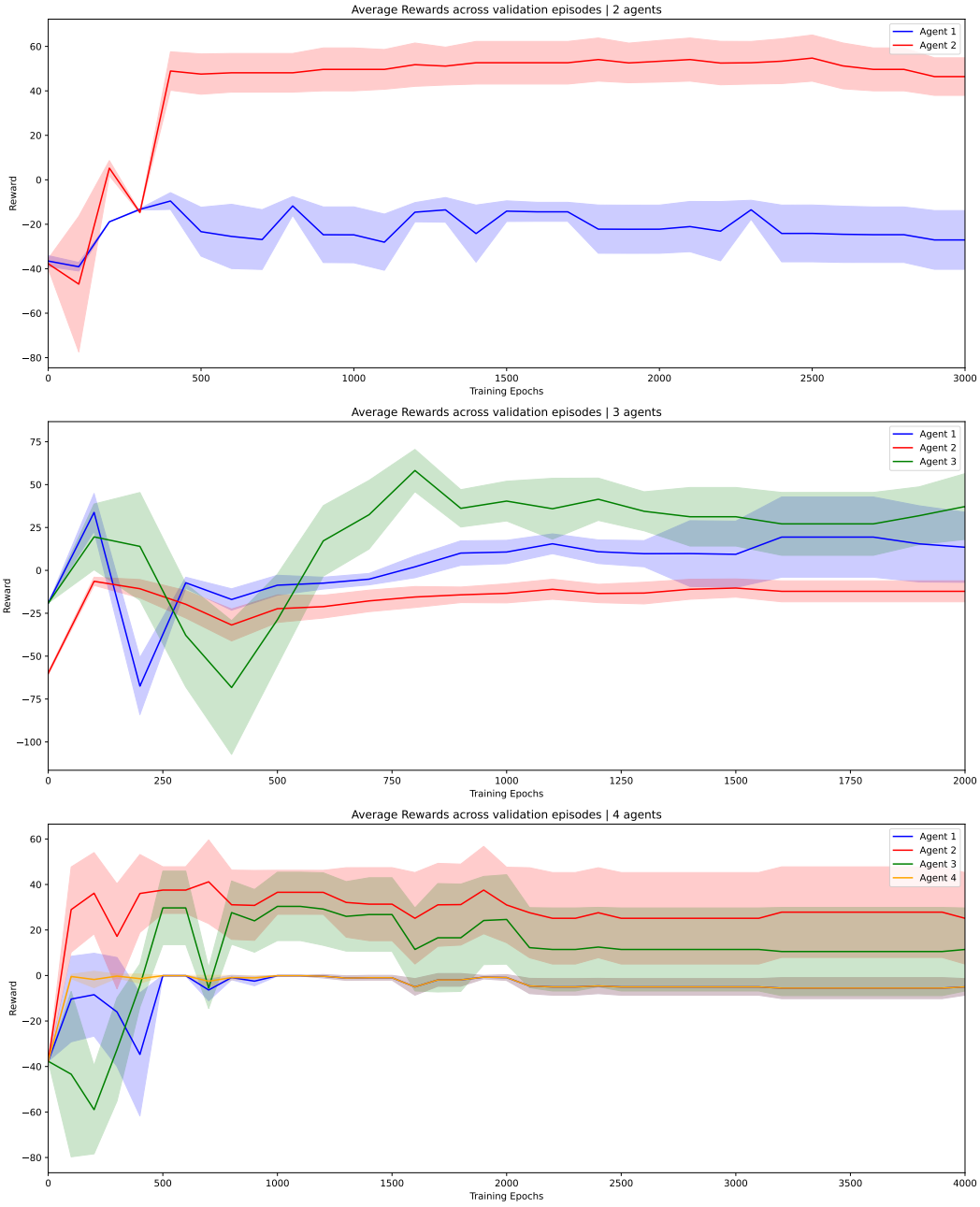

Figure 6.3: Validation rewards across training epochs for 2, 3, and 4-agent setups, illustrating the average rewards obtained by agents during validation episodes. The shaded region around the mean denotes the standard deviation of rewards across different validation episodes. These setups were trained with triple the number of passengers as the number of agents.

The bottom plot in Figure 6.3 shows the rewards obtained by the individual agents in the same setup shown in Figure 6.2. Agents 2 and 3 demonstrate convergence to an improved

policy, while agents 1 and 4 converge to sub-optimal policies, with their rewards trending towards zero or negative values at the point of convergence. As agents settle into a learned policy, those that effectively learn to complete tasks manage to pick an optimal number of tasks and complete them. Conversely, the agents that do not learn effectively resort to performing *no-operation* action, resulting in penalties for all the incomplete tasks that have remained in the environment for an extended duration. This behavior explains the drop in overall reward levels at the point of convergence shown in Figure 6.2.

Similar behavior can be observed in the top and middle plots in Figure 6.3, which show the rewards for a 2-agent and 3-agent setup, respectively. In the plot for the 2-agent setup, agent 2 converges to an optimal policy, while agent 1 explores various policies before ultimately settling on a *no-operation* only policy. Similarly, in the plot for the 3-agent setup, agents 2 and 3 converge to an improved policy, while agent 1 fails to do so.

This inconsistent behavior is observed across all training setups, regardless of the number of agents in the environment. In every training run, some agents converge to improved policies, while others settle into policies characterised by frequent execution of the *no-operation* action. Initially, these agents attempt to complete tasks during the early training epochs but almost immediately settle into learning a policy that predominantly performs *no-operation* action. Additional reward convergence plots for setups with 2 and 3 agents are included in the Appendix, demonstrating this behavior.

This convergence issue is a notable shortcoming of the algorithm, and further discussion on this limitation is provided in the subsequent section on Limitations. However, the subset of agents that successfully converge to an improved policy exhibit enhanced decision-making capabilities. An analysis of the performance of the learned policies, along with a comparison of their performance in various task-open dynamic ridesharing domains, is presented in the subsequent sections.

6.3.2 Policy Analysis

While the previous subsection discussed training and convergence, in this subsection we get into the behavioral analysis of the learned policy using MOHITO.

In evaluating the learned policies for the dynamic ridesharing domain, certain fundamental expectations are established to determine the effectiveness and optimality of the agent behaviors. Firstly, an optimal agent policy should demonstrate the capability to learn and complete tasks efficiently. In dynamic ridesharing, this involves understanding and executing the necessary steps to pick up and drop off passengers accurately. Secondly, since the algorithm specifically addresses task-open environments, agents should be adept at handling previously unseen tasks. They need to adapt quickly to new tasks introduced dynamically during an episode, ensuring they maintain high performance without prior knowledge of these tasks.

In addition to these fundamental expectations, there are advanced capabilities that can help distinguish the performance of the algorithm from others. Agents should exhibit the ability to commit to the most rewarding tasks based on their positioning within the environment. This strategic task selection is crucial for maximizing rewards and minimizing inefficiencies. Furthermore, agents should display advanced planning abilities, particularly in scenarios where they can complete multiple tasks simultaneously. The environment allows for task pooling, and an optimal policy would enable agents to plan their routes efficiently to pick up and drop off multiple passengers in a single journey, thereby enhancing overall performance.

The following are the behaviors observed in the learned policies in the dynamic ridesharing domain:

- **Behavioral outcomes in the learned policy:** Agents successfully navigate the accept-pick-drop regime, understanding that executing pick or drop actions multiple

times consecutively is necessary to reach a passenger's pick-up or drop-off location. The pooling limit—the total number of passengers an agent can service at once—is enforced not physically but through a significant penalty. Evidence of the model's learning is observed in the agents' ability to avoid exceeding this limit, thereby avoiding substantial penalties, and focusing on maximizing their rewards by not overly accepting passengers.

- **Handling unseen tasks in a dynamic environment:** Although agents were trained in a singular environment (as described in Section 6.1.1), the MOHITO-trained policy exhibits the ability to take up and complete previously unseen tasks in a dynamic ridesharing environment. Agents utilizing the MOHITO-trained policy accept passengers as they arrive and complete these tasks efficiently compared to agents following rule-based baselines, which struggle with the dynamic and unpredictable nature of task arrivals. While agents were trained in environments with a fixed number of passengers, they can also be seen performing well in environments with a higher number of passengers than what they have seen in training. This behavior is not limited to tasks present at the beginning of an episode but also includes tasks that appear dynamically during the episode. This is evidenced by the high rewards accumulated by optimally performing agents during the validation episodes, as shown in Figure 6.3.

- **Task selection based on proximity:** Beyond merely accepting and completing tasks, agents demonstrate the ability to select the most advantageous tasks based on their proximity to the task start points and/or the task endpoints that fall within their current route. This behavior indicates a degree of long-term planning ability. We showcase the outcome of this learned behavior by comparing the time taken to reach passengers in the environment with that of the baselines in the subsequent section.

- **Exhibiting long-term planning and ride pooling:** Agents exhibit long-term

thinking and planning capabilities, notably in their approach to ride pooling. Rather than completing one task at a time, agents are observed accepting multiple passengers if they fall along their current route, effectively pooling tasks together. When opportunities arise, agents pick up multiple passengers at once until they reach their pooling limit. This behavior showcases the agents' ability to optimise their routes and maximise efficiency by integrating multiple tasks into a single journey. We demonstrate this behavior in the following paragraphs.

To understand the task pooling behavior among the agents, we conduct evaluation episodes for setups involving 2, 3, and 4 agents, each with varying degrees of openness as described in the section discussing Experimental Setup. We then compare the average number of steps in which agents either perform a single task or pool multiple tasks together. Figure 6.4 provides insight into the average number of steps per agent for both single-task execution and pooled-task execution. The proportion of pooled to single-task execution can be seen increasing as the level of openness in the environment increases. This increase in pooled task execution is beneficial as it indicates that agents are effectively optimizing their routes and maximizing efficiency, thereby reducing idle time and improving overall service quality in more dynamic and unpredictable environments.

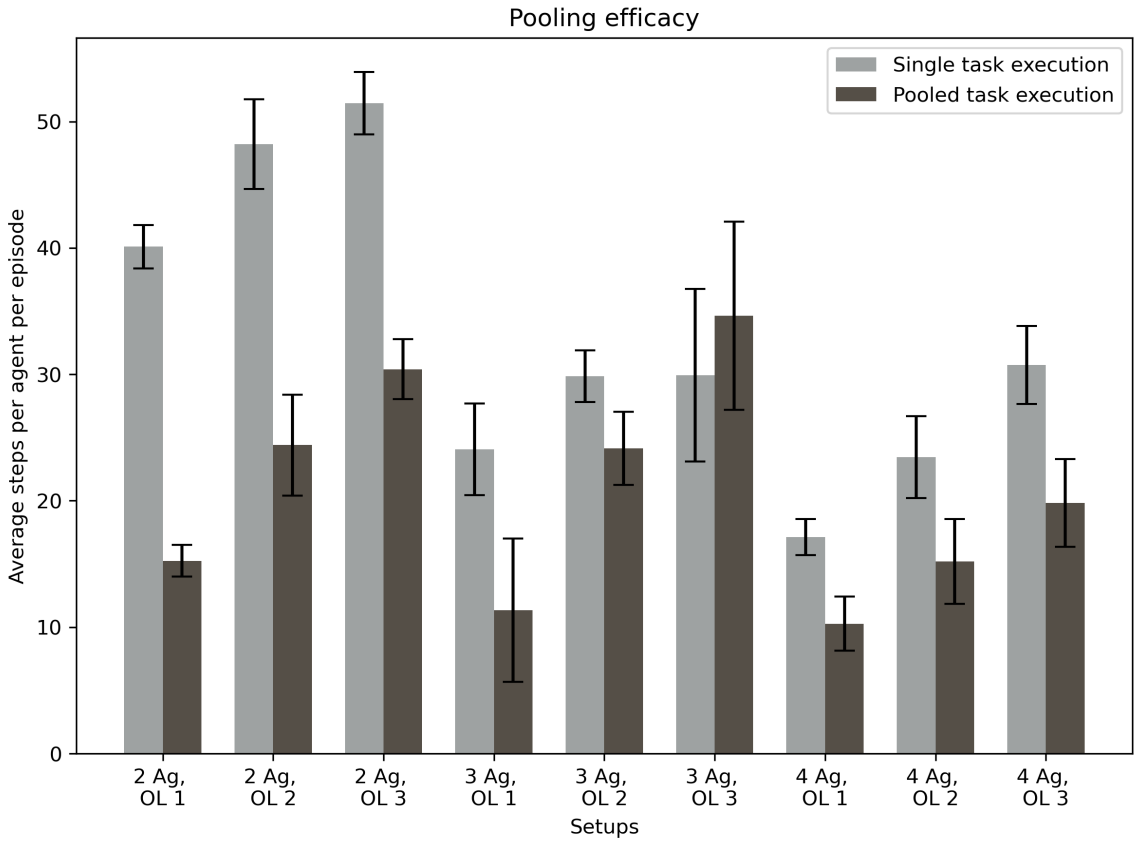

Figure 6.4: Comparison of steps involved in task pooling versus single-task execution by agents across various setups

Agents in EnvAgents in Env	Pooling-to-Single Task Ratio		
	OL1	OL2	OL3
2	0.38	0.51	0.59
3	0.47	0.81	1.16
4	0.60	0.65	0.65

Table 6.1: Table displaying the pooling-to-single task ratio for agents across various openness levels (OL)

The increase in the ratio of pooled-task to single-task execution steps can further be observed in Table 6.1. This suggests that as the environment becomes more open, MOHITO agents exhibit increased pooling behavior. This aligns with our expectations, as the algorithm is designed to encourage long-term thinking and efficient task completion, allowing

agents to effectively manage and complete tasks in the shortest possible time.

These findings highlight the strengths of the MOHITO algorithm in creating intelligent, adaptive agents capable of strategic planning and efficient task management in dynamic environments.

6.3.3 Comparative analysis against Baselines

To assess the performance of the policy learned by the MOHITO algorithm in comparison to baseline policies, a series of predefined test episodes are created across varying levels of openness, as mentioned in 6.1.2. We then obtain rewards from learned policies by deploying agents in these environments. Concurrently, rewards are also logged for agents operating in the environment according to the rule-based baseline policies introduced in the previous section.

Note that due to the training limitation discussed in the previous sections, where not all agents converged to an improved policy, the baseline comparisons were conducted in environments with optimally trained agents. Specifically, optimally trained agents from different training runs were combined to work in the environment. This ensures a fair comparison between the MOHITO algorithm and the baselines.

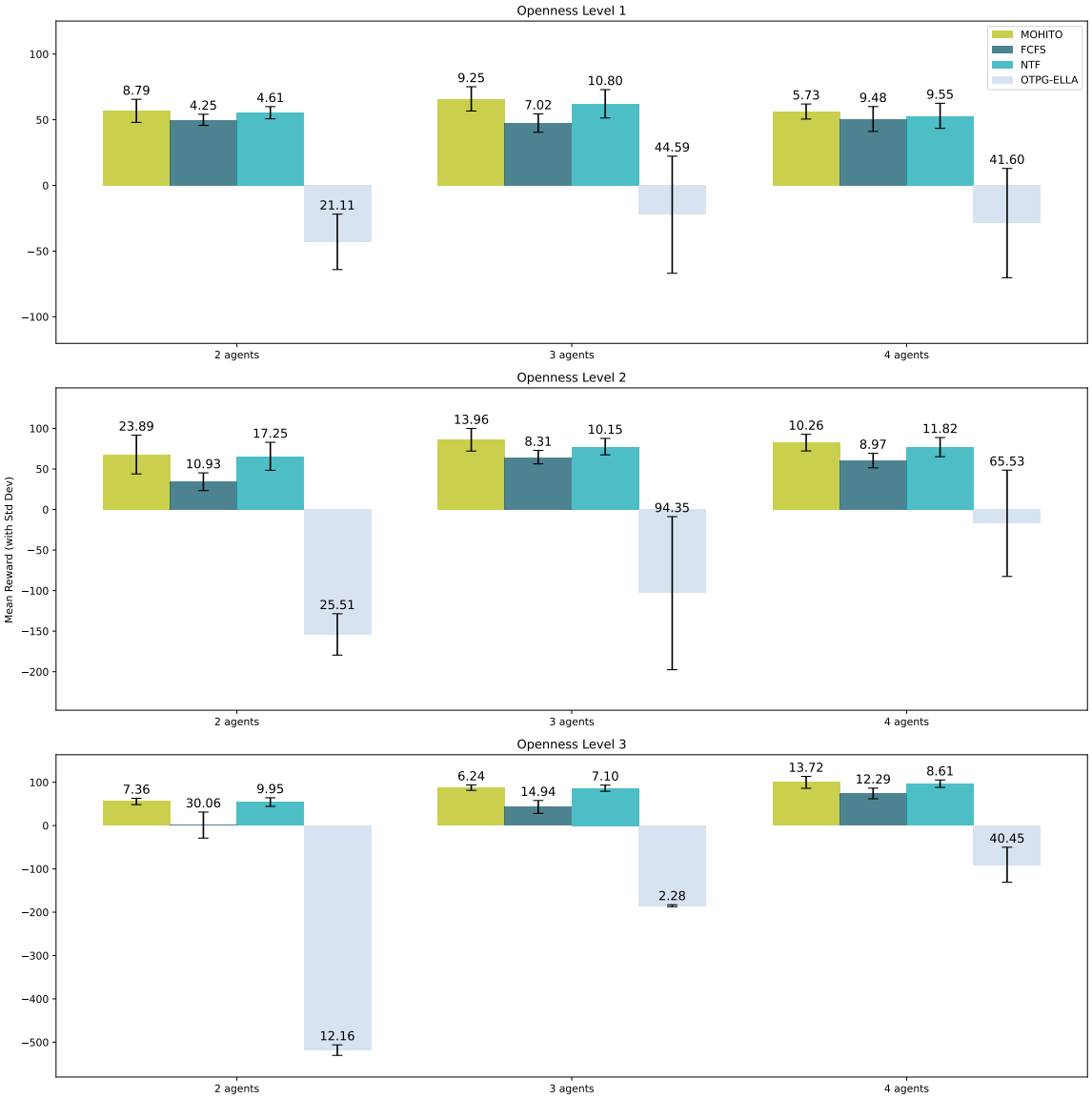

Figure 6.5: Comparison of average episodic rewards across different openness levels for policies based on MOHITO algorithm, First-come First-serve (FCFS), Nearest Task First (NTF), and Open-task Policy Gradient Efficient Lifelong Learning Algorithm (OTPG-ELLA)

Figure 6.5 shows the average episodic rewards resulting from several experiments. This plot shows the performance of the 3 policies in 2, 3, and 4 agent setups, across varying levels of openness. The evident observation is that MOHITO performs better than the static task allocation baselines (FCFS and NTF) on average in all setups. Additionally, MOHITO also performed better than OTPG-ELLA, which allows agents to pool multiple passengers. Not

only does MOHITO generally achieve higher average rewards, but it also often exhibits lower standard deviation in rewards, indicating more consistent performance.

While the difference in performance between MOHITO and OTPG-ELLA is evidently significant, we conducted a Wilcoxon significance test to validate the observed differences in rewards between MOHITO and the two static baselines (NTF and FCFS). The test confirmed that the differences between MOHITO's mean rewards and those of the other policies (NTF and FCFS) are statistically significant, with p-values of 1.25e-12 and 1.73e-13, respectively. This confirms that MOHITO's superior performance is not incidental. Quantitatively, MOHITO achieves average rewards that are 54.73% and 4.83% higher than the two static baseline policies, FCFS and NTF, respectively.

The following observations can be drawn from the comparisons illustrated in Figure 6.5

- **Analysis based on Openness Level:** In the 2-agent setup, the average rewards of the three policies (MOHITO, FCFS, NTF) are comparable. This similarity can be attributed to the limited number of agents and the sparse distribution of tasks, where the distance needed to reach the passengers outweighs the advantage gained by a superior decision-making policy. However, as the level of openness increases, the difference between the rewards of MOHITO and its baselines becomes more pronounced. This trend suggests that MOHITO's decision-making abilities are superior to those of the rule-based baselines, a pattern that is also evident in the 3 and 4-agent setups. Interestingly, OTPG-ELLA shows a significant deviation, with notably lower performance compared to MOHITO, especially at higher levels of openness. This highlights MOHITO's robustness and effectiveness in handling dynamic task environments where traditional algorithms like OTPG-ELLA struggle.

- **Analysis based on Agent Count:** Within each level of openness, the difference in rewards between MOHITO and the baselines tends to decrease as the number of agents

increases. This observation implies that rule-based baselines can perform comparably well in environments with a higher number of agents. While MOHITO handles increased complexity effectively and maintains efficient task pooling and coordination among multiple agents, there is still room for improvement to consistently outperform the baselines in all scenarios. Notably, OTPG-ELLA, despite its poor performance in lower agent setups, shows slightly better results in the 4-agent setup at higher openness levels. However, it still lags behind MOHITO, underscoring MOHITO's superior adaptability and efficiency in dynamic, task-open environments.

While reward comparison is crucial in multi-agent reinforcement learning (MARL) systems, MOHITO, being a MARL algorithm designed for task-open systems, necessitates evaluating how effectively agents process new tasks arriving in the environment. This evaluation is essential because task-open systems continuously introduce new and dynamic tasks, challenging agents to adapt and respond efficiently. Understanding how well agents handle these new tasks provides insight into the algorithm's ability to manage the dynamic nature of task-open environments, ensuring robustness and flexibility in real-world applications. One way to quantify this effectiveness is by examining the rate at which agents respond to new tasks, specifically the average time agents take to accept a newly arrived task, pick up a passenger, and complete the task.

Figure A.2 shows a comparison of the average task acceptance time, average task pick-up time, and average task completion time across the three policies. This figure illustrates the duration from task entry to the respective action points. In contrast, Figure A.1 in the Appendix provides a stacked plot, depicting the average number of steps agents spend on each of these actions.

MOHITO shows competitive average task pick-up times, especially outperforming the baselines by a lot at higher levels of openness. This indicates MOHITO's efficiency in not only accepting tasks quickly but also in moving towards and picking up passengers effectively.

Beyond merely accepting and completing tasks, agents demonstrate the ability to select the most advantageous tasks based on their proximity to the task start points and/or the task endpoints that fall within their current route. This behavior indicates a degree of long-term planning ability.

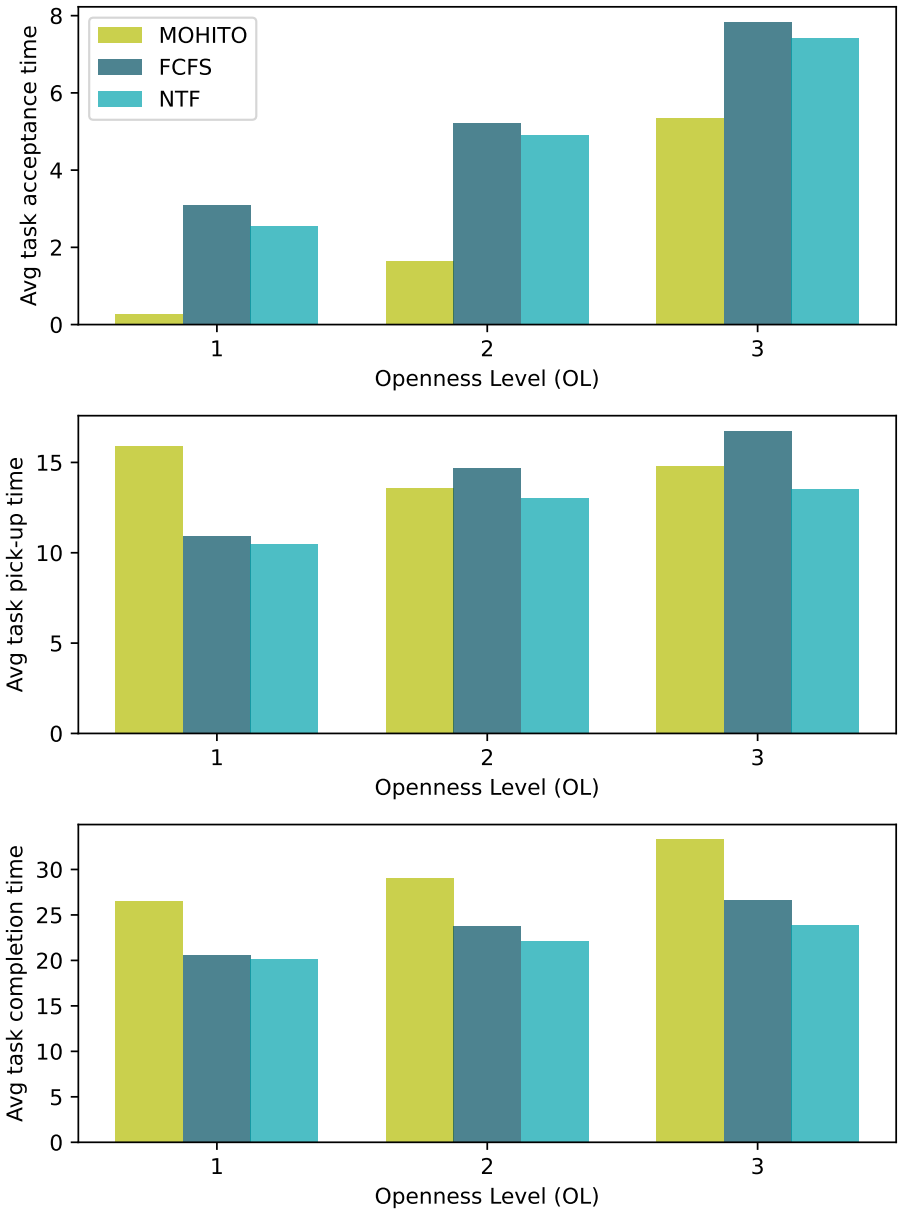

Figure 6.6: Average task acceptance, task pick-up and task completion time in MOHITO, First-Come First-Serve (FCFS) and Nearest Task First (NTF) policies

MOHITO consistently shows lower task acceptance times across all levels of openness, indicating its superior efficiency in quickly accepting tasks compared to the baseline policies. This is especially beneficial in a domain like dynamic ridesharing because passengers typically expect prompt service. Faster task acceptance times translate to shorter waiting periods for passengers, leading to higher customer satisfaction. However, MOHITO exhibits a higher average task completion time across all levels of openness. This can be attributed to MOHITO's complex and strategic approach to task management, which involves more planning and coordination. In contrast, the baselines, being rule-based policies, follow simpler and more direct strategies, resulting in faster task completion times.

Additionally, MOHITO-trained agents are, to some extent, competing with each other to maximise their rewards. There are instances where multiple agents accept the same passenger, but the passenger is only allocated to the closest driver. As a result, the other agents lose a timestep. It is also noteworthy that while rule-based baselines avoid confusion or incorrect decision-making, decision-making policies like MOHITO may occasionally make mistakes. A mistake, in this context, can occur when an agent's evaluation of which task to prioritise changes from one step to the next. For example, an agent might decide that task 1 is the best focus at one timestep but then switch to task 2 in the subsequent step. These decisions can lead to inefficiencies, but MOHITO consistently recovers from such occasional mistakes and completes the tasks efficiently. This adaptability is a significant advantage over rule-based policies, which do not experience such decision-making fluctuations but also lack the flexibility to optimize in a dynamic environment.

There is still room for improvement in MOHITO regarding smoother task completion, which could result in comparable completion times with the baselines. Despite this, MOHITO's strategic approach and recovery ability highlight its robustness in dynamic and task-open environments.

Chapter 7

Conclusion

The property of openness in real-world environments complicates multi-agent decision-making due to the continuous changes in the set of agents, tasks, and the agents' capabilities. Dynamic ridesharing exemplifies these challenges, as it requires efficient decision-making in real time amidst the unpredictability of passenger requests. Traditional approaches in multi-agent systems and reinforcement learning struggle to effectively address these dynamic aspects.

First, we introduced the concept of Task-Open Markov Decision Processes (TaO-MDPs) to formally represent decision-making in environments where the task set can dynamically change. This framework extends traditional MDPs to accommodate task openness, with detailed descriptions of the state, action, transition, and reward functions. We then instantiated the dynamic ridesharing domain as a TaO-MDP, illustrating how this framework can handle the complexities of real-world ridesharing scenarios, including adaptive action spaces and dynamic task entry and exit.

We introduced the novel MOHITO (Models of Hyper Interactions under Task Openness) algorithm, which leverages centralised training and decentralised execution to enable agents to efficiently learn in dynamic environments. The algorithm utilises interaction graphs to represent the state, task, and action information of an environment. The graph convolutional

networks, integral to the MOHITO algorithm, process the interaction graphs to select the best action from the available actions at each time step. This represents a paradigm shift from conventional RL, which relies on a fixed action space. We detailed the architecture and training process of MOHITO, demonstrating its ability to adapt to new tasks and optimise decision-making in a dynamic environment.

MOHITO was used to train agent policies in the dynamic ridesharing environment, where passenger requests can appear dynamically. Agents trained with MOHITO were not only able to complete previously seen and unseen tasks, but also strategised to maximise their rewards by pooling multiple passenger requests. The agents exhibited advanced planning capabilities, such as task pooling, and showed superior adaptability and efficiency. Our experimental results demonstrated that MOHITO significantly outperforms baseline policies in dynamic, task-open environments.

7.1 Training Limitations

The training process of MOHITO revealed a significant challenge: not all agents converged to an optimal policy. This issue, previously discussed and illustrated in Figure 6.3, shows that while some agents converge to optimal policies, others settle into a *no-operation* action policy.

In the Dynamic Ridesharing domain, actions such as *accept*, *pick*, and *drop* are used. Unlike previous literature Jiao et al. (2021); Oda (2021), which has modeled ridesharing with temporally extended actions through Semi-Markov Decision Processes or SMDPs Baykal-Gürsoy (2010), we have modeled *pick* and *drop* as one-step actions. This design choice facilitates task pooling, allowing agents to handle multiple passengers concurrently. While temporally extended actions could simplify the learning process by bundling multiple steps into a single high-level action, our approach requires agents to perform *pick* and *drop* actions

repeatedly over several timesteps to complete a task and realise the reward. This results in a traditionally hard-to-manage delayed reward environment, significantly increasing the complexity of the learning process. The requirement for multiple consecutive actions to complete a task introduces a much larger state space, as agents must track and optimise each step in the sequence over time, making efficient learning and decision-making more challenging.

Agents encounter only negative rewards for moving until a task is completed. To achieve a positive reward, an agent must successfully execute the *accept-pick-drop* sequence for a passenger, particularly mastering the repeated selection of *pick* or *drop* actions to see the task to completion. This challenge is compounded by the presence of several other actions for different passengers in the agent's action space, especially as new passengers continue to appear. While the domain already has a delayed reward problem, the design of one-step actions significantly reduces the probability of encountering the delayed reward, increasing the complexity of the learning process.

Some agents, due to early lucky explorations, learn to effectively transport passengers despite these complications. Changing the seed affects the number and the set of agents that reach a converged policy. Once a few agents figure out the accept-pick-drop sequencing, they dominate the environment, grabbing all passengers and reducing learning opportunities for slower agents. Consequently, these slower agents adopt a *no-operation* policy, as it offers a higher reward (0) compared to the negative costs associated with task-related actions like fuel cost for moving. Even epsilon-greedy exploration fails to shift these agents from the *no-operation* policy, as the potential rewards from exploration are lower than the rewards from sticking to their current policy. This makes it nearly impossible for exploration alone to lead these agents to successful task completion at this stage.

We investigate many training strategies to address this issue:

- **Increasing task count to avoid agent dominance:** We attempt to mitigate the

convergence issue by increasing the number of passengers in the environment. The idea is to ensure that even if some agents dominate, others will still have opportunities to learn and improve. However, this approach results in extremely large interaction incidence graphs, significantly increasing the complexity of the learning process. We also experimented with a gradual increase in passenger count during training, allowing agents ample time to adapt to the increasing size of the interaction incidence graphs. Specifically, the passenger count was increased from twice the number of agents to 4.5 times the number of agents over the training period, with the number of introduced passengers gradually increasing every few episodes. Despite this adjustment, slow-learning agents that had already resorted to a *no-operation* policy did not benefit from the gradual increase. Instead, their failed explorations only reinforced their inclination to do nothing.

- **Use of expert trajectories:** Similar to the exploration component introduced by epsilon-greedy training, we introduce an expert trajectory component where, for a small proportion of steps spread throughout training, an expert policy determines the action for the agent. Specifically, for 10% of the steps in an episode, an expert agent dictates the action, providing additional guidance. This proportion remains constant while epsilon is reduced exponentially as training progresses. In another schedule, both epsilon and the expert-guided proportion are reduced over time. Despite these adjustments, some agents still resort to a *no-operation* policy for reasons similar to those discussed earlier

- **Non-Singular Training Environments:** Section 6.1.1 describes the training setup used to learn the optimal policies discussed in the previous sections. A singular environment is used. To address the convergence limitations, we also explore training agents in non-singular environments with varying task sets. However, this approach

does not yield significant improvements. The inherent complexity of the tasks and the need to understand them from start to finish prove too challenging, and the changing task sets further complicate learning. Another strategy we employ is gradually changing the singular environment. However, this leads to sudden changes in agent behavior during training, indicating that the approach might introduce additional instability rather than promoting steady learning.

Ultimately, the limitations observed appear to be closely tied to the inherent complexity of the tasks themselves. Modeling *pick* and *drop* actions as one-step actions, rather than using temporally extended actions, creates a challenging learning environment. This design choice, aimed at facilitating task pooling, introduces a significant delay in realizing the benefits of actions, making it difficult for agents to consistently converge to optimal policies. The delayed rewards inherent in this setup complicate the learning process, as agents must navigate an environment where the positive outcomes of their actions are not immediately apparent, especially when the probability of successfully executing the rewarding sequence of actions is very low.

7.2 Future Work

This thesis has laid the foundation for understanding and addressing the complexities of task openness in a multi-agent system using the novel TaO-MDP framework and the MOHITO algorithm. However, several avenues for future research can further enhance and expand upon these initial findings.

One promising direction for future work is to modify the ridesharing domain by replacing the one-step actions with temporally extended actions. Simplifying the action space in this manner could reduce the complexity of the domain, making it easier for agents to learn optimal policies. Future research should carry out comprehensive tests to determine if

these modifications can solve the convergence issues currently observed with the MOHITO algorithm.

The principles and algorithms developed in this thesis can be extended to other dynamic and open environments. One such domain is wildfire firefighting, where fires (tasks) appear and evolve unpredictably, requiring quick and strategic responses from firefighters (agents). Unlike the ridesharing domain, this domain will be a fully co-operative one. Applying the TaO-MDP framework and MOHITO algorithm to this domain could provide valuable insights and further validate the robustness of these methods in diverse scenarios.

Another vital area for future research is to extend the architecture of MOHITO to handle not just task openness but also agent openness and type openness. Developing a comprehensive framework that accommodates these additional dimensions of openness will significantly enhance the applicability of MOHITO to a wider range of real-world multi-agent systems.

These directions offer exciting opportunities to build upon the current work, addressing its limitations and expanding its applicability to more complex and varied real-world scenarios. Future research in these areas promises to further advance the field of multi-agent reinforcement learning in dynamic and open environments.

In conclusion, this thesis has made significant strides in understanding and solving the complexities of task openness in multi-agent systems. The development of the TaO-MDP framework and the MOHITO algorithm represents a substantial contribution to the field of multi-agent reinforcement learning, offering a robust solution for dynamic task environments and paving the way for future advancements.

Bibliography

David Abel, Alekh Agarwal, Fernando Diaz, Akshay Krishnamurthy, and Robert E Schapire. 2018. Policy and Value Transfer in Lifelong Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80. PMLR, 20–29.

Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. 2024. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press. `https://www.marl-book.com`

Stefano V. Albrecht and Subramanian Ramamoorthy. 2013. A Game-Theoretic Model and Best-Response Learning Method for Ad Hoc Coordination in Multiagent Systems. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems* (St. Paul, MN, USA) *(AAMAS '13)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1155–1156.

Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. 2014a. Online Multi-Task Learning for Policy Gradient Methods. In *Proceedings of the 31st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 32)*, Eric P. Xing and Tony Jebara (Eds.). PMLR, 1206–1214.

Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. 2014b. Online Multi-Task Learning for Policy Gradient Methods. In *Proceedings of the 31st International Conference on Machine Learning (Proceedings of Machine Learning Research,*

Vol. 32), Eric P. Xing and Tony Jebara (Eds.). PMLR, Bejing, China, 1206–1214. https://proceedings.mlr.press/v32/ammar14.html

Jacob Andreas, Dan Klein, and Sergey Levine. 2017. Modular Multitask Reinforcement Learning with Policy Sketches. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia) *(ICML'17)*. JMLR.org, 166–175.

Thom Badings, Thiago D. Simão, Marnix Suilen, and Nils Jansen. 2023. Decision-making under uncertainty: beyond probabilities. *International Journal on Software Tools for Technology Transfer* (2023). https://doi.org/10.1007/s10009-023-00704-3

P. G. Balaji and D. Srinivasan. 2010. *An Introduction to Multi-Agent Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–27. https://doi.org/10.1007/978-3-642-14435-6_1

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew M. Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018a. Relational inductive biases, deep learning, and graph networks. *CoRR* abs/1806.01261 (2018). arXiv:1806.01261 http://arxiv.org/abs/1806.01261

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew M.

Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018b. Relational inductive biases, deep learning, and graph networks. *CoRR* abs/1806.01261 (2018). arXiv:1806.01261 http://arxiv.org/abs/1806.01261

Melike Baykal-Gürsoy. 2010. Semi-Markov Decision Processes. *Wiley Encyclopedia of Operations Research and Management Science* (2010).

Samir Bouabdallah. 2007. *Design and control of quadrotors with application to autonomous flying*. Ph. D. Dissertation. École Polytechnique Fédérale De Lausanne. https://doi.org/10.5075/epfl-thesis-3727

Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. 2010. *Reinforcement Learning and Dynamic Programming Using Function Approximators* (1st ed.). CRC Press, Inc., USA.

J. Calmet, A. Daemi, R. Endsuleit, and T. Mie. 2004. A Liberal Approach to Openness in Societies of Agents. In *Engineering Societies in the Agents World IV, Lecture Notes in Computer Science*, Vol. 3071. 81–92.

Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. 2021. Multi-Agent Reinforcement Learning: A Review of Challenges and Applications. *Applied Sciences* 11, 11 (2021). https://doi.org/10.3390/app11114948

Muthukumaran Chandrasekaran, Adam Eck, Prashant Doshi, and Leenkiat Soh. 2016. Individual Planning in Open and Typed Agent Systems. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence* (Jersey City, New Jersey, USA) *(UAI'16)*. AUAI Press, Arlington, Virginia, United States, 82–91. http://dl.acm.org/citation.cfm?id=3020948.3020958

charlesdognin. 2018. Online Multi Task Learning. https://github.com/cdcsai/Online_Multi_Task_Learning.

B. Chen, A. Eck, and L.-K. Soh. 2016. Collaborative Human Task Assignment for Open Systems (Extended Abstract). In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 1441–1442.

Zhiyuan Chen and Bing Liu. 2018. *Lifelong Reinforcement Learning*. Springer International Publishing, Cham, 139–152. https://doi.org/10.1007/978-3-031-01581-6_9

Adam Eck, Maulik Shah, Prashant Doshi, and Leenkiat Soh. 2020. Scalable Decision-Theoretic Planning in Open and Typed Multiagent Systems. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence* (New York City, New York, USA) *(AAAI'20)*.

Adam Eck, Leen-Kiat Soh, and Prashant Doshi. 2023a. Decision making in open agent systems. *AI Magazine* 44, 4 (2023), 508–523. https://doi.org/10.1002/aaai.12131

Adam Eck, Leen-Kiat Soh, and Prashant Doshi. 2023b. Decision making in open agent systems. *AI Magazine* (09 Oct 2023). https://doi.org/10.1002/aaai.12131

EllaBot. 2015. PG-ELLA. https://github.com/EllaBot/PG-ELLA.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2017. Counterfactual Multi-Agent Policy Gradients. arXiv:1705.08926 [cs.AI]

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 1263–1272. https://proceedings.mlr.press/v70/gilmer17a.html

Tom Haider, Karsten Roscher, Felippe Schmoeller da Roza, and Stephan Günnemann. 2023. Out-of-Distribution Detection for Reinforcement Learning Agents with Probabilistic Dynamics Models. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems* (London, United Kingdom) *(AAMAS '23)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 851–859.

William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of the 31st Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. 1025–1035.

W. Jamroga, A. Meski, and M. Szreter. 2013. Modularity and Openness in Modeling Multi-Agent Systems. In *Fourth International Symposium and Games, Automata, Logics and Formal Verification (GandALF)*. 224–239.

Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. 2020. Graph Convolutional Reinforcement Learning. In *International Conference on Learning Representations*.

Yan Jiao, Xiaocheng Tang, Zhiwei Qin, Shuaiji Li, Fan Zhang, Hongtu Zhu, and Jieping Ye. 2021. Real-world Ride-hailing Vehicle Repositioning using Deep Reinforcement Learning. *arXiv preprint arXiv:2103.04555* (2021). `https://doi.org/10.48550/arXiv.2103.04555`

J. Jumadinova, P. Dasgupta, , and L.-K. Soh. 2014. Strategic Capability-Learning for Improved Multi-agent Collaboration in Ad-hoc Environments. *IEEE Transactions on Systems, Man, and Cybernetics-Part A* 44, 8 (2014), 1003–1014.

Anirudh Kakarlapudi, Gayathri Anil, Adam Eck, Prashant Doshi, and Leen-Kiat Soh. 2022. Decision-theoretic planning with communication in open multiagent systems. In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*. 938–948.

W.-R. Kong, D.-Y. Zhou, Y.-J. Du, Y. Zhou, and Y. y Zhao. 2023. Hierarchical multi-agent reinforcement learning for multi-aircraft close -range air combat. *IET Control Theory and Applications* 17 (2023), 1840–1862.

Martin Lauer and Martin A. Riedmiller. 2000. An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, Pat Langley (Ed.). Morgan Kaufmann, 535–542.

Hui Li, Xuejun Liao, and Lawrence Carin. 2009. Multi-task Reinforcement Learning in Partially Observable Stochastic Environments. *J. Mach. Learn. Res.* 10 (jun 2009), 1131–1186.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *Neural Information Processing Systems (NIPS)* (2017).

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2020. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. arXiv:1706.02275 [cs.LG]

Laëtitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. 2012. Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems. *The Knowledge Engineering Review* 27 (2012), 1 – 31. `https://api.semanticscholar.org/CorpusID:220758222`

Laetitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. 2007. Hysteretic Q-learning : an algorithm for Decentralized Reinforcement Learning in Cooperative Multi-Agent Teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 64–69. `https://doi.org/10.1109/IROS.2007.4399095`

J.A. Mendez, H. van Seijen, and E. Eaton. 2022. Modular Lifelong Reinforcement Learning via Neural Composition. In *Proceedings of the Tenth International Conference on Learning Representations (ICLR'2022)*.

Stefano Moroni and Daniele Chiffi. 2022. Uncertainty and Planning: Cities, Technologies and Public Decision-Making. *Perspectives on Science* 30, 2 (04 2022), 237–259. https://doi.org/10.1162/posc_a_00413 arXiv:https://direct.mit.edu/posc/article-pdf/30/2/237/2005006/posc_a_00413.pdf

Takuma Oda. 2021. Equilibrium Inverse Reinforcement Learning for Ride-hailing Vehicle Network. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 2281–2290. https://doi.org/10.1145/3442381.3449935

Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P. How, and John Vian. 2017. Deep Decentralized Multi-task Multi-Agent Reinforcement Learning under Partial Observability. arXiv:1703.06182 [cs.LG]

Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. 2021. Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms in Cooperative Tasks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*. https://openreview.net/forum?id=cIrPX-Sn5n

Jan Peters and Stefan Schaal. 2008. Natural Actor-Critic. *Neurocomputing* 71, 7 (2008), 1180–1190. https://doi.org/10.1016/j.neucom.2007.11.026 Progress in Modeling, Theory, and Application of Computational Intelligenc.

Zhiwei (Tony) Qin, Hongtu Zhu, and Jieping Ye. 2022. Reinforcement learning for ridesharing: An extended survey. *Transportation Research Part C: Emerging Technologies* 144 (2022), 103852. https://doi.org/10.1016/j.trc.2022.103852

Muhammad A Rahman, Niklas Hopner, Filippos Christianos, and Stefano V Albrecht. 2021. Towards Open Ad Hoc Teamwork Using Graph-based Policy Learning. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 8776–8786.

Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. 2017. EPOpt: Learning Robust Neural Network Policies Using Model Ensembles. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. `https://openreview.net/forum?id=SyWvgP5el`

Stuart Russell and Peter Norvig. 2010. *Artificial Intelligence: A Modern Approach*. Pearson Education.

Paul Ruvolo and Eric Eaton. 2013. ELLA: An Efficient Lifelong Learning Algorithm. In *Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 28)*, Sanjoy Dasgupta and David McAllester (Eds.). PMLR, Atlanta, Georgia, USA, 507–515. `https://proceedings.mlr.press/v28/ruvolo13.html`

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. 2019. The StarCraft Multi-Agent Challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems* (Montreal QC, Canada) *(AAMAS '19)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2186–2188.

Andreas Sedlmeier, Thomas Gabor, Thomy Phan, Lenz Belzner, and Claudia Linnhoff-Popien. 2020. Uncertainty-based Out-of-Distribution Classification in Deep Reinforce-

ment Learning. In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART'2020)*. SCITEPRESS - Science and Technology Publications. https://doi.org/10.5220/0008949905220529

O. Shehory. 2000. Software Architecture Attributes of Multi-agent Systems. In *1st International Workshop on Agent-Oriented Software Engineering, Revised papers*. ACM, 77–89.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. 2021. Multi-Task Reinforcement Learning with Context-based Representations. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 9767–9779. https://proceedings.mlr.press/v139/sodhani21a.html

Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller (Eds.), Vol. 12. MIT Press. https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf

F. Tanaka and M. Yamamura. 2003. Multitask reinforcement learning on the distribution of MDPs. In *Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation. Computational Intelligence in Robotics and Automation for the New Millennium (Cat. No.03EX694)*, Vol. 3. 1108–1113 vol.3. https://doi.org/10.1109/CIRA.2003.1222152

S. Thrun and T.M. Mitchell. 1995. Lifelong Robot Learning. *Robotics and Autonomous Systems* 15, 1-2 (1995), 25–46.

Prashant Trivedi and Nandyala Hemachandra. 2021. Multi-agent Natural Actor-critic Reinforcement Learning Algorithms. *CoRR* abs/2109.01654 (2021). arXiv:2109.01654 https://arxiv.org/abs/2109.01654

Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-learning. *arXiv preprint arXiv:1509.06461* (2016). https://doi.org/10.48550/arXiv.1509.06461

Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural Deep Network Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. Association for Computing Machinery, 1225–1234.

Wikipedia contributors. 2024. Hypergraph — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Hypergraph. [Online; accessed 29-May-2024].

R. J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8 (1992), 229–256.

Lixiang Zhang, Jingchen Li, Yi'an Zhu, Haobin Shi, and Kao-Shing Hwang. 2022. Multi-agent reinforcement learning by the actor-critic model with an attention interface. *Neurocomputing* 471 (2022), 275–284. https://doi.org/10.1016/j.neucom.2021.06.049

Ziqian Zhang, Lei Yuan, Lihe Li, Ke Xue, Chengxing Jia, Cong Guan, Chao Qian, and Yang Yu. 2023. Fast teammate adaptation in the presence of sudden policy change. In *Uncertainty in Artificial Intelligence*. PMLR, 2465–2476.

Appendix A

Task Completion Time Metrics

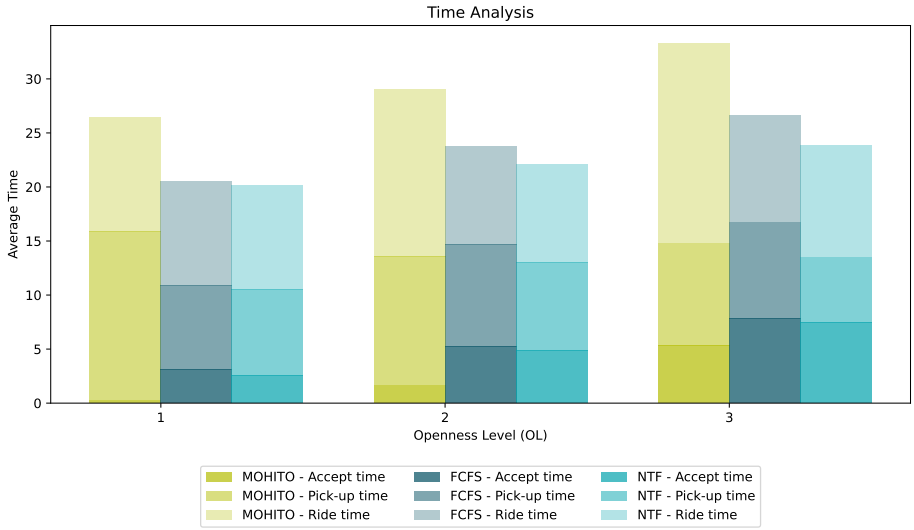

Figure A.1: Stacked bar plot showing the average time spent on each task phase across the three policies (MOHITO, FCFS, NTF). The plot illustrates the breakdown of time from task entry to acceptance, from acceptance to pick-up, and from pick-up to completion

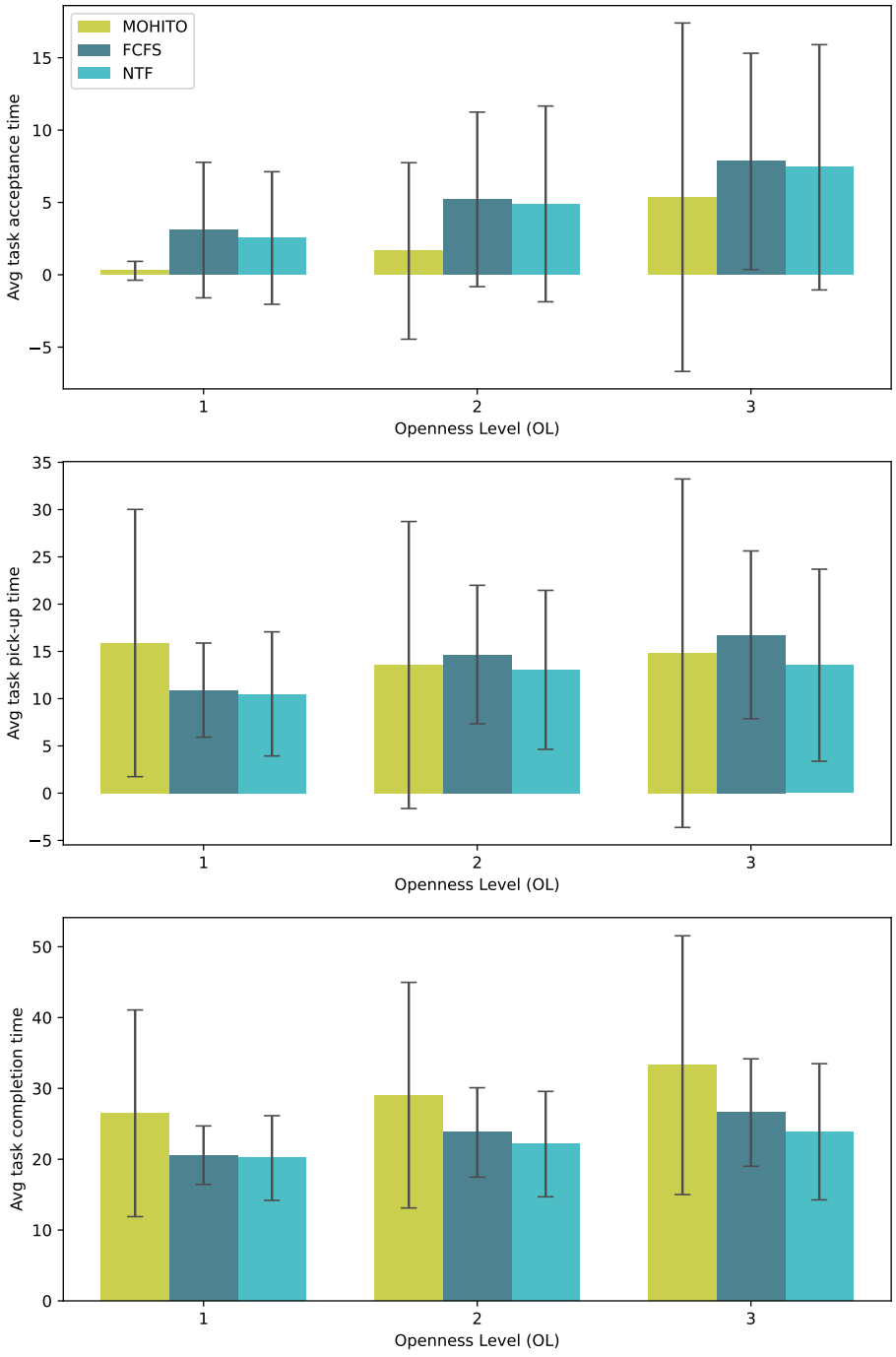

Figure A.2: Average task acceptance, task pick-up and task completion time in MOHITO, First-Come First-Serve (FCFS) and Nearest Task First (NTF) policies. The standard deviation has been marked as error bars

Appendix B

Training parameters

This section lists the training parameters used to attain the converged policies used in this results' section of this thesis

- Number of agents = 2, 3, 4
- Number of training epochs = 10000
- Length of each training episode = 100 steps
- Batch size = 20
- Grid dimensions = 10x10
- Rewards
  - Cost to accept a passenger = 0
  - Cost to move to pick/drop = -1.2/(number of active passengers)
  - Cost to pick up a passenger = -0.1
  - Cost to drop off a passenger = ridefare

  - Cost to an agent if it accepts a passenger that was accepted by other agents as well and didn't get the passenger = -2
  - Cost to accept a passenger beyond the agent's pooling limit = -2

- Gamma (for loss function) = 0.9
- Range of epsilon for epsilon-greedy = [0.9, 0.05]
- Epsilon decay rate = 0.00035
- Epsilon decay function [/epsilon_fn = lambda epoch: min(epsilon_range) + (max(epsilon_range) - min(epsilon_range)) * math.exp(-epsilon_decay * epoch)/]
- Actor parameters
  - Number of hidden layers = 20
  - Learning rate = 0.001
  - Regularisation lamdba = 0.1
  - Soft update rate = 0.05
- Critic parameters
  - Number of hidden layers = 20
  - Learning rate = 0.01
  - Regularisation lamdba = 0.01
  - Soft update rate = 0.05
  - Gradient clip = 5

The implementation of MOHITO and Dynamic Ridesharing can be found at **https://github.com/oasys-mas/MOHITO/tree/Gayathri-Anil**

Appendix C

OTPG-ELLA

C.1 PG-ELLA

First described by (Ammar et al., 2014b), PG-ELLA, as depicted in Alg. 3, is an extension of the authors' previous work in linear regression Ruvolo and Eaton (2013) which greatly expands the variety of practical applications by using an internal policy gradient approach Sutton et al. (1999); Peters and Schaal (2008); Li et al. (2009). Notably, it features a modular structure capable of utilizing learning a variety of base learners Williams (1992); Peters and Schaal (2008) to individually learn policies for like tasks Busoniu et al. (2010); Bouabdallah (2007). As they are learned, these task-specific policies are efficiently combined by the main algorithm to form a partially unified set that shares some policy information across tasks. This partial unification comes from the structure of the algorithm's latent-space-defined policy parameters, which are divided into one large latent space L shared between tasks and another, smaller latent space $s^{(t)}$ unique to each task t. For each learned task, the policy parameters $\theta^{(t)}$ are defined as follows: $\theta^{(t)} = Ls^{(t)}$. (Ammar et al., 2014b) describe the benefits of this shared latent space in greater detail, but for purposes related to open-task environments, this efficient and task-based approach to learning presents an opportunity to

lead into learning a policy capable of operating in environments that have a variety of tasks present at any given moment.

Algorithm 3 PG-ELLA (k, λ, μ)

$T \leftarrow 0$ $A \leftarrow \mathbf{zeros}_{k\times d,k\times d}$ $b \leftarrow \mathbf{zeros}_{k\times d,1}$ $L \leftarrow \mathbf{zeros}_{d,k}$ some task t is available isNewTask(t) $T \leftarrow T + 1$ $(T^{(t)}, R^{(t)}) \leftarrow$ getRandomTrajectories() $(T^{(t)}, R^{(t)}) \leftarrow$ getTrajectories$(\alpha^{(t)})$ $A \leftarrow A - (s^{(t)}s^{(t)^\top}) \otimes \Gamma^{(t)}$ $b \leftarrow b - \text{vec}(s^{(t)^\top} \otimes (\alpha^{(t)^\top}\Gamma^{(t)}))$ compute $\alpha^{(t)}$ and $\Gamma^{(t)}$ from $(T^{(t)}, R^{(t)})$ $L \leftarrow$ reinitializeZeroColumns(L) $s^{(t)} \leftarrow \text{argmin}_s \ell(L, s, \alpha^{(t)}, \Gamma^{(t)})$ $A \leftarrow A + (s^{(t)}s^{(t)^\top}) \otimes \Gamma^{(t)}$ $b \leftarrow b + \text{vec}(s^{(t)^\top} \otimes (\alpha^{(t)^\top}\Gamma^{(t)}))$ $L \leftarrow \text{mat}((\frac{1}{T}A + \lambda I_{k\times d,k\times d})^{-1}\frac{1}{T}b)$

C.2 Adapting PG-ELLA

As presented by (Ammar et al., 2014b), PG-ELLA falls short of direct open-task application in three key aspects. First, the base learners described (Williams, 1992; Peters and Schaal, 2008) and implemented (Peters and Schaal, 2008; EllaBot, 2015; charlesdognin, 2018) are all specific to single-agent environments. While these are certainly capable of learning and operating in multi-agent environments, they lag behind newer methods of addressing multi-agency Lowe et al. (2020); Trivedi and Hemachandra (2021); Zhang et al. (2022) by only considering other agents as part of the environment, rather than as agents in their own right Russell and Norvig (2010). This limits the efficacy of any learned policy Matignon et al. (2012). Second, the nature of this single-agent implementation and traditional single-task environments trivialises trajectory generation explicitly featuring one agent working on a specific task, but such triviality is lost when attempting to account for the actions of other agents or the presence of other tasks. This makes iterating by task and generating trajectories over specific tasks impractical, resulting in both inefficiency should each agent need to generate a trajectory set for each task where they must work on that task, failing to learn from other tasks present in the environment or trajectories in which that task was taken by another agent. Third, the trajectory-related complications just described extend

further when the exact tasks to be present in a newly-generated trajectory are uncertain, as in an open-task environment Eck et al. (2023b). This also limits the ability for task-by-task iteration as it is difficult to generate trajectories that contain a particular task if the tasks present in a given trajectory cannot be known prior to generation.

C.2.1 Multi-agency

Beginning with the change that requires the least modification to PG-ELLA, multi-agency can largely be addressed through the base learner, with minimal alterations to the larger algorithm. Existing implementations of PG-ELLA EllaBot (2015); charlesdognin (2018) have generally used REINFORCE Williams (1992) as a simple base-learning algorithm, with the original authors Ammar et al. (2014b) having utilised a natural actor-critic implementation Peters and Schaal (2008). The former is a purely single-agent learning approach, and while actor-critic strategies Lowe et al. (2020); Trivedi and Hemachandra (2021); Zhang et al. (2022) have been developed for multi-agent learning, such adaptations are not discussed in Ammar et al. (2014b). As described previously, this does not prohibit an agent's ability to learn and operate in a multi-agent environment, but it does limit the potential efficacy of any policy learned Matignon et al. (2012). Looking to resolve this through modern policy-gradient approaches that both effectively and efficiently learn multiple agents' worth of policies in a multi-agent setting, multi-agent deep deterministic policy gradient (MADDPG) Lowe et al. (2020) presents itself as an actor-critic derived strategy that allows for centrally-trained yet independent critics with decentrally-executing actors as final in-environment policies. This methodology permits the use of a base-learning algorithm that can learn task-specific policies through an actor-critic approach, similar to the original paper's PG-ELLA implementation Ammar et al. (2014b), while gaining the benefits of multi-agent learning and improving the resulting policies Russell and Norvig (2010). Furthermore, MADDPG Lowe et al. (2020) in particular makes use of the distinction between the training and operating

environments to gain the benefits of centralised training, normally only available in co-operative settings where agents share rewards Lauer and Riedmiller (2000); Matignon et al. (2007); Omidshafiei et al. (2017), and still use an individual policy for each agent.

This centralised training with decentralised execution (CTDE) is achieved by taking trajectories generated through the independent actions of all the agents therein, and generating critical inputs across the joint observations and actions of all agents. These joint inputs are then used with each agent's individual rewards to create an agent-specific score, or Q-value, for a critic unique to that agent. This differs from traditional centralised training which would have a single score over all agents using a shared reward value, thus limiting its effectiveness to co-operative environments Lauer and Riedmiller (2000); Matignon et al. (2007); Omidshafiei et al. (2017), and extends those benefits to competitive or mixed co-opetitive settings Lowe et al. (2020). Each agent's actor can then learn a separate policy from their individual observation-action inputs and their respective critic's joint score. These separate policies are then used in the next round of trajectory generation to continue the cycle until convergence or end of iteration. This strategy, depicted in Fig. C.1, has been shown by Lowe et al. Lowe et al. (2020) to have great improvements in policy efficacy and agent coordination when compared to individually-learned agents.

C.2.2 Environments as Sets of Tasks

With multi-agent environments now addressed through the base learner, the next step in adapting PG-ELLA to open-task environments is enabling the operation of PG-ELLA in environments with multiple, simultaneously-existing tasks, and efficiently learning from such environments. For closed environments, the simplest approach does not require any alterations to the main algorithm, as any group of tasks can be presented as a single joint task Russell and Norvig (2010). That said, in the context of PG-ELLA, defining learnable tasks as the joint set of smaller, functional tasks that can be found in an environment containing

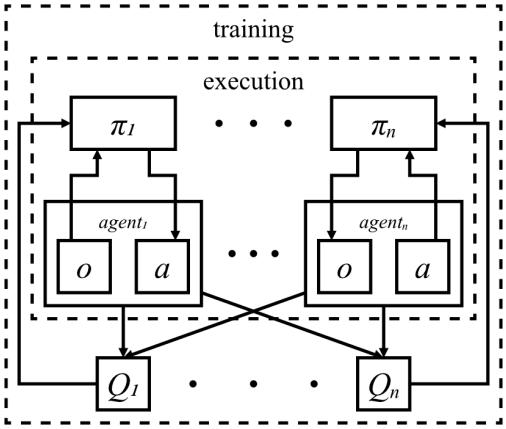

Figure C.1: Centralised training of critics over the joint-actors' observations and action choices with decentralised execution of actors from their individual observations, as described for MADDPG Lowe et al. (2020).

many is inefficient. For example, using the ride-sharing domain, if an environment contains three passengers t_1, t_2, and t_3, then PG-ELLA would learn task $t^*_{1,2,3}$. This simple solution does provide a working approach to the closed version of the domain, but falls prey to limited practical applicability Eck et al. (2023b) and excessive learning complexity. For the former, this clearly falls within the problems of a closed domain, not permitting the entry or exit of passengers, but it also limits the scope of applicable domains to those with a specific number of passengers. This occurs due to the structure of PG-ELLA's latent spaces, which requires all policies to have the same sized observation and action spaces Ammar et al. (2014b). As such, either agents would be unable to observe any passenger-specific information, or they would need to always have the same fixed number of passengers to preserve observation size. Problems with this way of defining tasks continue into the learning aspect, where a greater number of tasks must be learned overall with more complex policies. Continuing the previous example and assuming no relevance to the ordering of passengers represented in the environment, giving a mere two new passengers, t_4 and t_5, the possibility of appearing in the

environment would require PG-ELLA to learn an additional nine tasks to ensure operability in all possible environments. Furthermore, because the tasks PG-ELLA aims to learn are defined by the set of several passengers, the policies required to operate in this environment must similarly be more complex than would be the case for learning individual passengers, resulting in more policies overall that are each more complex than if tasks were instead defined by individual passengers. This all-as-one approach rapidly becomes intractable with either an increasing pool of possible passengers or an increasing number of passengers defining a task.

A better solution, one that makes use of PG-ELLA's task-based architecture, would be to define the environment not by one large task, but by a set of smaller tasks. These task sets can be retained for trajectory generation, but PG-ELLA is able to learn simpler task-specific policies, then join them to form a larger policy over the whole set only in execution. This alternate strategy allows for both a reduced learning load, as sufficient coverage of the base tasks may be achieved without exhaustive exploration of all possible joint task sets Russell and Norvig (2010), and a broader selection of operable environments, as combining individual tasks need not be limited to a specifically-sized joint set. Additionally, this approach acts as a first step toward adapting PG-ELLA for open environments, as they necessarily require this task set representation to depict the dynamic changes therein. However, the current design of PG-ELLA limits this approach by lacking a way to join these task-specific policies to form a larger joint policy and requiring task-by-task iteration with task-specific trajectories generated for each learned task-specific policy.

C.2.3 Forming Joint Policies

Before trajectories can be generated in environments with these sets of tasks, PG-ELLA needs a policy form capable of combining several task-specific policies into a joint policy over the full set of tasks present. To this end, one naive approach is to concatenate individually-

learned task-specific policies. In practice, this means taking the shared latent space L and, for each task t_i present in an environment with n tasks, multiplying L by the relevant task-specific latent space $s^{(t_i)}$ as described by Ammar et al. Ammar et al. (2014b) for a set of task-specific policy parameters $\theta^{(t_{[1:n]})}$. From there, the agents' observations can be passed through the corresponding policies $\pi_{\theta^{(t_{[1:n]})}}$ and the resulting action values concatenated to form the final joint policy $\pi(o)$ of each agent. This overall joint policy design can be seen in Fig. C.2.

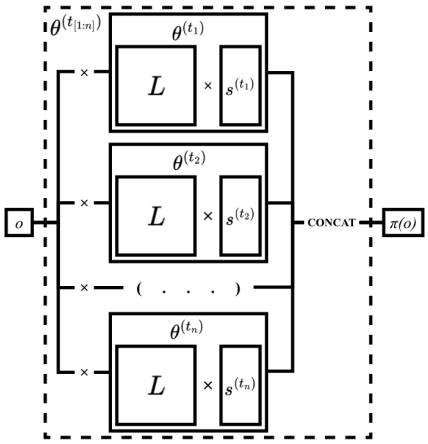

Figure C.2: Joining PG-ELLA's task-specific policies together to create a single policy for an environment with a set of tasks. Note that this initial approach requires a fixed observation space, to be addressed later.

Relativizing Action Values

Notably, this design relies heavily upon maintaining action values that are not only relative to the others within a task, but also to those of other tasks. Thankfully, PG-ELLA already utilises a large latent space shared between tasks which can be used in tandem with base learner design and implementation to enforce an effective relativity between tasks. Using the base learner model previously described in section C.2.1, two such strategies have been designed.

The first of these relies upon the shared latent space between tasks to maintain relative action scores over the course of learning. This includes exploiting the overall task policy shaping from the natural convergence of the shared latent space as a greater portion of possible tasks are learned Ruvolo and Eaton (2013), and using those same values to initialise both the actor and critic policies of newly-encountered tasks. For the applications explored in section 6.3, this solution demonstrated sufficient efficacy, allowing agents to make decisions with multiple simultaneous tasks available. That said, the environment examined therein utilises a relatively simple action scheme, so in consideration of the potential need for greater interconnection a more robust solution has also been devised.

In this second approach, rather than rely on the similarities of otherwise-independent task-specific critics merely initialised through the actors' shared latent space, two instances of PG-ELLA are run in tandem. The first, as before, is used to learn actor policies for execution with their shared latent space and task-specific elements, gaining all the benefits of the previously described strategy. The second, co-running implementation of PG-ELLA maintains another shared latent space among critics, each bearing their own additional task-specific elements. This further increases the inter-connectivity of each task-specific policy by connecting not only the actors operation in execution, but also the critics as policies are learned. One consequence of this second relativisation strategy is a substantial increase in the memory required for learning each task. Before, while a critic either had to be kept in connection with each task policy or reinitialised each encounter, this new strategy requires the full suite of additional information required for PG-ELLA be stored for each critic. This includes the local optimal, hessian gradient thereon, and a task-specific latent space. These can be estimated Ammar et al. (2014b), but that requires effectively re-learning them from scratch over new trajectories each time their respective task is encountered during learning, substantially increasing both the time needed for each encounter, and it is not guaranteed that the local optimal or gradient thereover will be the same for a given task within one

set to another. Such cost increases did not go unnoticed in evaluation, and the greater requirements consistently exceeded that of practical use, resulting in frequent crashes. This, and the sufficient relativity from the first strategy, ultimately led to the second approach's disuse.

Fixing Size Variability

This overall joint policy design works well so long as the observation space is fixed between sets. This limitation is not strictly a problem in closed environments, but to broaden applicability to a greater number of environments containing either variously-sized task sets or task openness, the fixed-size observations require substantial limitation over the available task-specific information. At worst, it must rely on features shared between or derived from the collectivisation of tasks, in which case many potential state spaces may risk indistinguishability. To avoid that and retain some task-specific information in a fixed-sized observation space, the space may instead afford some maximum number of observable tasks, resulting in a large observation space that has the potential to be largely wasted in any given application should there be far fewer tasks than the maximum observable, or worse: insufficient for applications containing more tasks than was space prepared for. As such, operation in an unfixed-size observation space is a more general and practical approach, but some alteration will be needed to adapt an uncertainly-sized observation to a fixed-size observation for use in PG-ELLA's task-specific policies.

To that end, one solution is to treat the observation as PG-ELLA treats its latent spaces. Using similar terminology, this involves dividing the observation space between shared features O_L and those that are task-specific $O_s^{(t_{[1:n]})}$, then passing only the shared and task-relevant parts to each task such that $O^{(t_i)} = \text{concat}(O_L, O_s^{(t_i)})$. This strategy is demonstrated below in Fig. C.3.

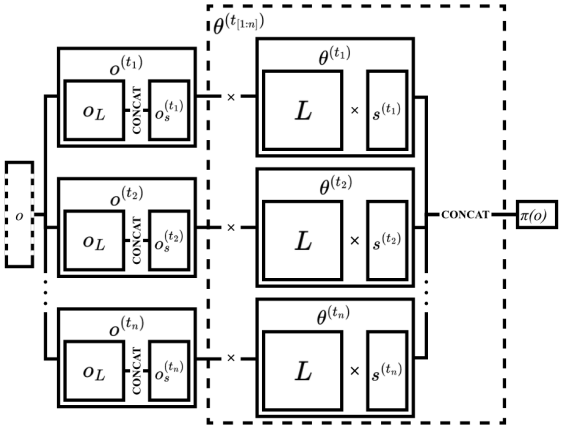

Figure C.3: In OTPG-ELLA, a variable observation space is partitioned into a set of fixed-size task-specific observations. These are defined by the concatenation all elements not specific to any task and the task-specific elements relevant to one task each of those present in the environment. After that, each observation can be passed through the appropriate task-specific policy for task-specific action values. Combining those results in the full policy for the agent at that step in the environment.

Joining Learned and Unlearned Policies

One final note in regards to this joint policy definition, it may be the case over the course of operation that an environment is encountered in which only a subset of the tasks therein actually have learned policies. In this case the joint policy can either ignore new tasks for safe operation over known tasks, or it can use a random policy for each unknown task t_i^{new} as a basis for learning as in the work of Ammar et al. Ammar et al. (2014b). To preserve inter-task relativity, these random policies can be formed through the existing shared latent space L and a randomly-initialised task-specific latent space $s^{(r)}$ for the policy parameters $\theta^{(t_i^{new})} = Ls^{(r)}$.

C.2.4 Learning Tasks by Set

With the updated policy form now operable in environments defined by sets of tasks, trajectories may be generated and learned for the tasks therein. However, PG-ELLA has a particular inefficiency with regards to joint-set trajectories which becomes apparent here. As presented, PG-ELLA iterates task-by-task and generates trajectories for only the relevant task. In a closed environment, this behavior can be replicated directly, by generating trajectories using only sets that contain task in question, but this leaves a glaring waste of trajectories for all the other tasks present, and it further leaves open the possibility of actions only being chosen for unrelated tasks, effectively wasting the trajectory. Considering the previous decision to adapt PG-ELLA from learning one large task into a set of several smaller ones, using the older behavior of iterating over the whole, set-by-set would prevent the need to regenerate trajectories for tasks already visited, and allow each trajectory to effect learning of at least some task in the set. While that iteration alone is a direct reflection of PG-ELLA operating over the one joint task, learning the smaller tasks internal to the broader set requires some modification. Looking at what else can be moved to this set-by-set iteration, trajectory generation is the only step that needs to move, but the zero-column re-initialisation and final reconstruction of the shared latent space L can also reasonably be calculated at this frequency, as shown by Ruvolo and Eaton Ruvolo and Eaton (2013). Given this basis of set-by-set iteration, the tasks internal to each set $t_{[1:n]}$ can then be iterated over to perform for each task: base learning, defining their task-specific latent spaces $s^{(t_i)}$, and adjusting the task-relevant elements of the shared latent space components T, A, and b.

Trajectory Filtering

From this set-task divided iteration, generated trajectories still require one further step to preserve task-relevance: trajectory filtering. This concept is necessary, due to the task-divided action space, to ensure trajectories contain only action choices for the task currently

being learned. It also avoids wasting algorithmic time that would be spent attempting to evaluate the generated trajectories with respect to tasks for which no relevant actions were chosen. Trajectories can be filtered simply by removing steps with foreign actions, those chosen for tasks other than the one in question. Such a simple filter does preserve immediate rewards and task-cumulative rewards, but may be inaccurate to the joint set and environment-cumulative rewards as the full, unfiltered trajectory's rewards are not necessarily represented. An alternative which can preserve the full trajectory for each task is possible through limiting task definition such that each individual task has a no-op action, then converting any foreign actions into no-ops of the current task. This second approach also puts forward another point of contact between tasks that may contribute towards inter-task relativisation.

Learning Uncertain Tasks

With multiple tasks now learnable through a single trajectory generated from a known set and iteration split between that set and its internal tasks, a greater problem presents itself: how is an agent to know which tasks are present in an open environment? Unlike the closed environments described previously which may be initialised with a particular task set, open-task environments can, in addition to their initialised task set, add tasks in execution. The resulting uncertainty regarding tasks present emerges both in operation, from the step-by-step observation, and in learning, through task-by-task iteration over generated trajectories. One option would be to include an additional task space in the environment of which the current subset is stored and passed to the agents alongside their observations, but this greatly limits the applicability of the resulting agents to environments explicitly designed for them. To avoid such limited usability, a method of interpreting tasks provided an observation or trajectory as a whole would be preferable. In cases where there exist discrete elements in task-specific parts of the observation, such interpretation can be achieved by

selecting a particular subset of those elements and defining tasks by that subset. This method was utilised successfully for the experiments conducted in section 6.3. Unfortunately, this strategy falls short for environments where the observation relies exclusively on continuous task-specific elements, though a similar effect may be achievable through discretisation, or environments which lack task-specific observation.

Regardless, the implementation of such an interpreter then provides a starting point for the observational divisions described in section C.2.3. From there, trajectories may be generated without a pre-planned task set using the joint policy method depicted in Fig. C.3 to divide the variably-sized observations according to the interpreted tasks. Said trajectories can then be expanded into a set of iterable tasks from those interpreted observations or a separate task list. Observation matching and action filtering can then be performed over the generated trajectories using the now-identified set of tasks present, creating fully task-specific learnable trajectories for each task encountered over the course of operation in an open-task environment. The full process of this open-task PG-ELLA (OTPG-ELLA) can be seen in Alg. 4.

Algorithm 4 OTPG-ELLA (k, λ, μ)

$T \leftarrow 0$ $A \leftarrow \mathbf{zeros}_{k\times d,k\times d}$ $b \leftarrow \mathbf{zeros}_{k\times d,1}$ $L \leftarrow \mathbf{zeros}_{d,k}$ some task t is available $(T, R) \leftarrow$ getTrajectories(α) $L \leftarrow$ reinitializeZeroColumns(L) $t_{[1:n]} \leftarrow$ getTasks(T, R) $t_i \in t_{[1:n]}$ $(T^{(t_i)}, R^{(t_i)}) \leftarrow$ filterTrajectories(T, R, t_i) isNewTask(t_i) $T \leftarrow T + 1$ $A \leftarrow A - (s^{(t_i)}s^{(t_i)^\top}) \otimes \Gamma^{(t_i)}$ $b \leftarrow b - \text{vec}(s^{(t_i)^\top} \otimes (\alpha^{(t_i)^\top}\Gamma^{(t_i)}))$ compute $\alpha^{(t_i)}$ and $\Gamma^{(t_i)}$ from $(T^{(t_i)}, R^{(t_i)})$ $s^{(t_i)} \leftarrow \text{argmin}_s \ell(L, s, \alpha^{(t_i)}, \Gamma^{(t_i)})$ $A \leftarrow A + (s^{(t_i)}s^{(t_i)^\top}) \otimes \Gamma^{(t_i)}$ $b \leftarrow b + \text{vec}(s^{(t_i)^\top} \otimes (\alpha^{(t_i)^\top}\Gamma^{(t_i)}))$ $L \leftarrow \text{mat}((\frac{1}{T}A + \lambda I_{k\times d,k\times d})^{-1}\frac{1}{T}b)$

- The implementation of OTPG-ELLA can be found at

  https://github.com/oasys-mas/otpg-ella