CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Safe Multi-Robot Planning Via Long-Run Averages

Master's Thesis in Computer Science and Engineering

EYOB EMBAYE JOHAN DAUN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Safe Multi-Robot Planning Via Long-Run Averages

EYOB EMBAYE JOHAN DAUN

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Safe Multi-Robot Planning Via Long-Run Averages

EYOB EMBAYE JOHAN DAUN

Supervisor: Anna Louise Gautier, Computer Science
Examiner: Nir Piterman, Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2026

Safe Multi-Robot Planning Via Long-Run Averages

EYOB EMABYE
JOHAN DAUN
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothentburg

Abstract

Constrained reinforcement learning in Markov decision processes (MDPs) has received increasing attention for its use in sequential decision making problems with safety requirements. This study investigated safe planning via long run average reward using MDPs. This thesis uses grid-world environments and builds on the Triple-QA framework [1]. Three approaches are evaluated: A single-agent baseline and two multi agent extensions, a trivial joint-state extension, and separate Q-table approach.

The results show that the single agent algorithm reproduces the result found in the original framework and serves as a reliable baseline. The joint state space extension suffers from poor scalability due to exponential growth in the state action space, and therefore does not achieve comparable reward per agent as the baseline. In contrast, the separate Q-table approach scales significantly better and achieves a level comparable to the single agent case both in an environment with and without agent interaction. Although the result of two agents with trivial extension and separate Q-table was achieved to satisfy the constraint, the test with three agents did not satisfy for both algorithms.

Keywords: Computer, science, computer science, engineering, multi-agent, reinforcement learning, project, thesis.

Acknowledgements

We would like to give a big thanks to our supervisor, Anna Louise Gautier, for her continuous guidance, support, and valuable feedback throughout this thesis project. Her dedication and availability, often helpful with her insightful ideas, have been greatly appreciated and have contributed significantly to the development of this work. During this thesis, there were moments where we felt stuck and unable to move forward. In those moments, Anna encouraged us to think creatively, helped us to see the challenge from different perspectives, and guided us towards possible directions. Annas help was especially appreciated during the literature review phase; the continuous feedback and weekly meetings shaped both our ways of thinking and our work.

We also thank our examiner, Nir Piterman, for his time, insights, and valuable guidance during the thesis process.

Finally, we are grateful to all those who supported us during the project.

Eyob Embaye, Johan Daun, Gothenburg, 2026-06-20

Contents

List of Figures

List of Tables

1

Introduction

In today's world, the demand for intelligent systems is increasing rapidly and with the rapid growth of research in this field, many commercial and corporate entities are integrating intelligent systems such as warehouse robots, drones, and games. Their ability to adapt to environments that dynamically change and learn in the online manner has made artificial intelligence (AI) systems more effective and useful in environments that require simulating intelligent behaviors [2]. These AI systems are capable of integrating multi agent systems (MAS) [3] using reinforcement learning (RL). RL is a framework where an agent learns to make a sequential decision by interacting with the environment and receives a reward and the agent goal is to maximize the cumulative reward in the long term [4], [1].

An AI agent refers to a system that is capable of making decisions (actions) independently to perform a certain task, on behalf of a user, using the available environment [2], [4]. An AI environment is the context in which the agent takes actions and perceives. Everything except the agent itself, including other agents, data, rules, and physical or digital surroundings, is considered an environment. The environment is what shapes how the agent makes its decisions and learns [4]; see figure 1.1. An agent selects an action, receives a reward and cost, transitions to the next state, and updates the cost and reward. Now an MAS is one in which there are multiple of these autonomous entities (agent) sharing a common environment [2], working collectively to perform a task or collaborating on a common goal. Think of multiple robots operating in a formation or several autonomous vehicles driving through the same intersection.

The consequences of an action taken by the agent depend on the environment. The agent taking a wrong action in a video game may not affect the user as much, but if the system is engineered without careful attention, it could have outcomes such as collisions or fatal errors in autonomous driving [5] and in clinics to perform tasks such as surgery [6]. Therefore, it is important to balance between maximizing expected reward and minimizing the constraint violations that govern safety. A standard setting formulation for RL with constraints is the Constrained Markov Decision Process (CMDP) framework, where the agent aims to learn and perform a policy that maximizes the expected cumulative reward without going over the threshold, such as collisions or missing targets. CMDP is an extension of MDP with safety added on top of MDP [7].

Many studies have been conducted on safe planning in single and multi agent settings

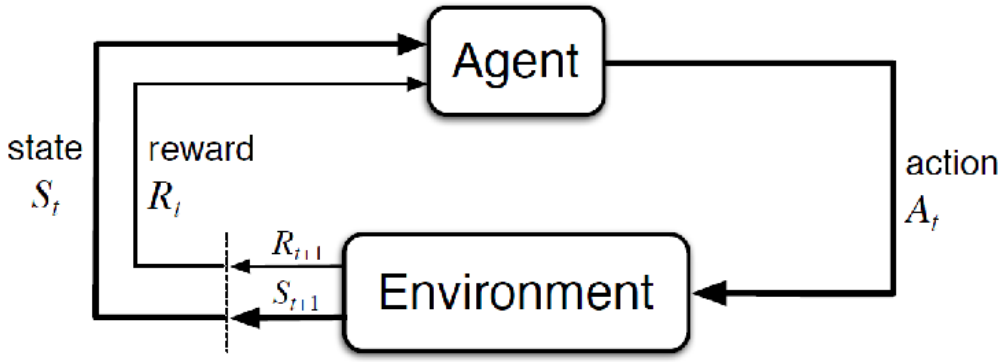

Figure 1.1: The agent-environment interaction in MDP for single-agent system, $t \in \{1, 2, 3, \dots\}$, $S_t \in \mathcal{S}$, $A_t \in \mathcal{A}$, $R_t \in \mathcal{R}$

for finite horizon using RL [2]. Safe planning in multi robot systems is often modeled through multi agent Markov Decision Processes (MMDPs) [8]. Many studies for multi agent planning focus on discounted cost MDPs which value immediate rewards rather than future rewards, or finite horizon MDPs which work only for a discrete time [1], [9]. While discounted formulations provide strong theoretical guaranties and computational tractability, they may not appropriately capture long term safety requirements in persistent multi agent systems. Giving more value to immediate rewards rather than future rewards makes an agent short sighted, which leads to poor long term policies and misses out large future gains or causes the agent to converge to a suboptimal policy. This is like a human consuming too much sugar for immediate gratification and risking their future health rather than consuming less and having a healthier life in the future. This prevents the agent from learning complex tasks that require the agent to focus on delayed gratification to achieve a certain goal. These approaches are not suitable for the safe planning of MAS.

Existing work extends the concept of long term average reward [9] to safe and constrained single agent planning [8], [10]. Using CMDP for an infinite horizon has been efficient [1]. Extending these concepts to MASs is an important step toward multi robot planning and decision making. Multi agents are powerful because they can learn through their interactions with the environment and share their experiences with other agents so that they do not repeat the learning process for no purpose. Now, think of the collective behavior of those agents. The behavior of a MAS increases the potential for accuracy as well as adaptability and, therefore, tends to outperform single agent systems due to the larger pool of shared optimization. Agents can share their experience to optimize the efficiency of the policy [11].

1.1 Problem Statement

Despite significant progress in safe reinforcement learning for single agent CMDPs with long run average reward [9] criteria, the extension of these frameworks to MASs remains limited. When scaling from a single agent system to a multi agent system

many challenges arise such as non stationarity, coordination between agents and, most critically, exponential growth of the joint state action space [12], [3]. Existing approaches in multi agent reinforcement learning primarily focus on finite horizon or discounted reward settings, which may not appropriately capture long term safety requirements. Therefore, this thesis aims to develop and analyze RL methods for constrained MAS under long run average reward performance while satisfying safety constraints and improving scalability.

1.2 Aim and Research Questions

The aim of this thesis is to develop an RL framework for safe planning in cooperative MASs under infinite horizon settings. More specifically, the work will extend existing constrained RL methods from previous research papers that was originally formulated for single agent CMDPs to MASs where agents interact within a shared state space and they must satisfy the constraints in the long run. The objective of this thesis is to formulate the multi agent constrained decision making problem, adapt a suitable learning algorithms, and analyzes their theoretical and empirical properties. We will pay more attention to challenges that are unique to MASs such as non stationary, scalability, and shared constraints, and the main goal is to provide a scalable approach for safe planning of decision making.

This thesis was conducted according to the following research questions:

- How can we adapt a single agent constrained reinforcement learning algorithm for long run average in CMDP to a cooperative constrained multi agent setting?
- How does the naive extension of a single agent CMDP algorithm for long run average scale when the number of agents increases?
- Can a multi agent extension preserve the long run average cost constraint while maintaining a reasonable reward performance?

1.3 Scope and Limitation

This thesis focuses on the extension to MAS under specific structural assumptions. The analysis will be limited to finite state and action spaces and to environments where the system dynamics satisfy the assumption required for theoretical guaranties.

The proposed methods will be validated only in simulation environments. Real world robotic deployment, hardware implementation and experimental validation on physical systems are outside the scope of this work.

Furthermore, due to time constraints, the algorithms implemented will mainly focus on methods related to reinforcement learning.

2

Background

This chapter introduces the technical concepts and technical background relevant to this thesis. Since many of the concepts are built upon each other, the chapter is structured progressively. The first concepts introduced are RL and constrained RL, followed by MDP, CMDP, and MMDP. These concepts provide the foundation for understanding constrained single agent and constrained multi agent systems. The chapter ends with a discussion of long run average reward formulations and the problem formulation considered in this thesis.

2.1 Reinforcement Learning (RL)

RL is concerned with sequential decision making in a state of uncertainty and is a branch of machine learning. The agent in RL learns by interacting with an environment and receives feedback in the form of a reward [4], [13]. At each time step t the agent observes a state denoted s_t, selects an action a_t, receives a reward r_t and transitions to a new state s_{t+1}; see figure 1.1. The next state the agent moves to are determined by the environment. The transition function specifies the probability of moving to each possible next state s_{t+1} for each pair of (s_t, a_t).

The two primary categories of policies are deterministic and stochastic. A deterministic policy is a policy that selects a single action in each state, while a stochastic action selects an action by defining a probability distribution over possible actions. In RL the goal is to find an optimal policy that maximizes the cumulative expected reward.

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right].$$

Where π denotes the policy that is being evaluated and $\mathbb{E}_\pi$ is the expectation generated by π over time. $\gamma \in [0, 1)$ is the discount factor. And $r(s_t, a_t)$ is the reward obtained after taking the action a_t at state s_t.

In RL, the quality of a policy is often evaluated using value functions. The state-value functions $V^\pi(s)$ estimate the expected reward obtained by starting in state s and following policy π, while the action value functions $Q^\pi(s, a)$ estimate the expected reward obtained by taking action a in state s and then following policy π. The state value function and the action value function are defined

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} r(s_t, a_t) \mid s_0 = s\right],$$

and

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} r(s_t, a_t) \mid s_0 = s, a_0 = a\right].$$

respectively.

These functions are used to evaluate the desirable states and actions with respect to the discounted objective. Many reinforcement learning algorithms, such as Q-learning are based on estimating or optimizing these value functions. Reinforcement learning algorithms are commonly categorized into model-free and model-based approaches depending on whether the agent explicitly models the environment dynamics.

- **Model-free**: learns directly from interaction by trial and error. They do not estimate the transition dynamics. Model-free approaches are favored in environments that are complex and where modeling the dynamics of environment accurately is challenging. They also require a large number of steps or interactions with the environment to learn optimal policies.
- **Model-based**: uses, or it attempts to learn a model of the environment dynamics and then uses the model to plan a route.

Another important way of evaluating a RL algorithms is also regret. It measures the performance loss difference the optimal policy and the current policy. In general RL, regret can be defined as a difference between the cumulative reward obtained by an optimal policy $\pi^{\star}$and the cumulative reward obtained by the current policy π [14]. The standard RL is designed to maximize the reward, but in many of the real world applications, maximizing the reward is not enough. Applications such as autonomous vehicles, autonomous surgery equipment, or in warehouse robots require additional safety operational constraints. The agent must learn to maximize the reward while also satisfying the constraint, and this is what motivates the study of constrained reinforcement learning.

2.1.1 Constrained Reinforcement Learning (CRL)

CRL is the standard RL but with safety added on top of it. Safety is incorporated in the standard framework of decision making, see Figure 2.1. As mentioned in section 2.1, in the standard RL the agents objective is to select the actions that maximize the reward but in CRL the agents objective is not only to maximize the reward but to also ensure the constraint is not violated or the cost is under a certain predefined threshold. The CRL setting is critical and relevant in safety critical applications where maximizing the reward is not sufficient.

Unlike in the standard RL where the agent observes a state, takes an action, gets a reward, and moves to the next state, in CRL the agent also receives a cost incurred

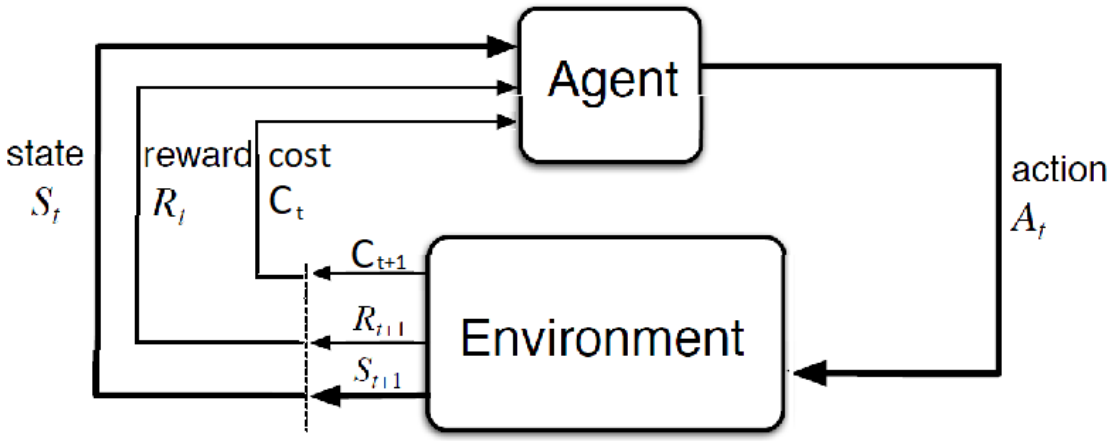

Figure 2.1: The agent-environment interaction in CMDP for single-agent system, $t \in \{1, 2, 3, \dots\}$, $S_t \in \mathcal{S}$, $A_t \in \mathcal{A}$, $R_t \in \mathcal{R}$, $C_t \in \mathcal{R}$

by the action taken and moves to the next state. The agents objective is to find the policy that maximizes the expected reward while satisfying the cost constraint.

In the formulation of CRL, the reward function captures the desired performance of the task and the cost function is for the undesirable behaviors such as collisions caused by the action that lead to violations of the system constraints.

In the setting of the average reward which is commonly used in the continuous settings, the objective of the agent is usually defined in terms of long run average reward, cost. This setting is useful when the problem does not have natural terminal state and the agent is expected to operate continuously over time.

One common approach to solving constrained reinforcement learning problems is to transform the constrained optimization problem into an unconstrained one using a Lagrangian formulation. In this approach, the violation of the constraint is penalized by introducing a Lagrange multiplier, which controls how strong constraint violations are penalized. The agent then optimizes a modified objective that balances reward maximization and cost minimization. If an agent violates a constraint then the penalty increases and actions with high cost becomes less attractive. On the other hand, if the constraint is satisfied then the agent shifts focus on maximizing the reward. Therefore, the Lagrangian formulation provides a mechanism of trade off between maximizing reward and safety.

A policy that achieves high reward may be violating the constraint and therefore incurs high cost, while an agent that aims to avoid cost overly may achieve poor reward, so the challenge is to learn a policy that balances between cost and reward. The agent should achieve strong performance while conserving the cost under the predefined threshold. In safety critical systems, this trade off is the central part of the design. The agent may do some violations that still satisfy the systems requirements. CRL is central and relevant because the agent is not only expected to achieve maximum reward but also satisfies a predefined cost constraint.

2.2 Markov Decision Process (MDP)

A Markov Decision Process (MDP) is a mathematical framework often used to model sequential decision making problems in reinforcement learning. The framework describes interaction between an agent and its environment, where the agent selects an action and, based on that action, receives a reward and transitions to a new state each step.

An important characteristic of MDPs is the Markov property, which states that the future state depends only on the current state and action, rather than the complete path of the previous state and action the agent has taken. Therefore, the agent's decision can be based solely on the current state, which significantly simplifies the modeling of the decision process. This assumption ensures that the value based formulations from reinforcement learning, such as the state value function $V^\pi(s)$ and the action value function $Q^\pi(s, a)$ become well defined.

To model the decision process formally, the system is simplified into four components, which are used to describe how the agent interacts with its environment over time. These components are the state, action, transition function, and reward function. Firstly, the state represents the current situation of the system. For example, the state may represent the position of the agent, but it could also represent something more complex. For example, in a grid world environment, the state may represent the position of the agent in the grid world, but in other more complex environment, the state might also include the information about other agents such as location, battery level of a robot or the current speed of a vehicle. If two scenarios in the system should be treated differently, then they should also be represented by different states. Based on the current state, the agent selects an action from a predefined set of possible actions. The result of the action on the environment is determined by the transition function, which gives the probability of transitioning to a new state. In addition, the agent receives a reward after each action to indicate how desirable the selected action was with respect to the overall objective.

An MDP can be formally defined by the tuple (S, A, P, R), where:

- S denotes the finite set of states in which the agent can be (the state space).
- A denotes the finite set of actions available to the agent.
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function, where $P(s' \mid s, a) = \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\}$.
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function.

A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ maps each state to a probability distribution on actions where $\Delta(A)$ denotes the set of all probability distributions on the action set A.

The environment model was defined in the MDP definition specifics, however, the objective depends on the chosen reward criterion. Two of the most common criteria are the long run average and the discounted reward. In the long run average reward setting, the objective is to maximize the long run average reward

$$J(\pi) = \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} r(s_t, a_t) \right]$$

Here, $\mathbb{E}_\pi$ denotes the expected value under the policy π, and $J(\pi)$ represents the performance of the policy. The objective is therefore

$$\max_\pi \quad J(\pi)$$

.

And in the discounted reward setting, the objective is to always maximize

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

where $\gamma \in [0, 1)$ is the discount factor.

2.3 Constrained Markov Decision Process (CMDP)

In many real world applications, solely maximizing reward is insufficient. In systems such as autonomous vehicles, warehouse robots and surgical robots must of course achieve high performance but also satisfy safety requirements during operation. A policy that achieves high reward may still be undesirable if it frequently violates safety constraints, for example a self driving car causing collisions. The standard MDP formulation 2.2 does not take safety into account for such real world constraints, which motivates the study of CMDP.

A Constrained Markov Decision Process is an extension of the MDP2.2 framework with safety constraint function on the accumulated expected cost added on top of it. While MDP focuses on maximizing reward, CMDP balances between reward and cost. The idea is to accumulate as high reward as possible preserving the safety constraint[15].

An CMDP can be formally defined by the tuple (S, A, P, R, C), where:

- S denotes the finite set of states in which the agent can be (the state space).
- A denotes the finite set of actions available to the agent.
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function, where $P(s' \mid s, a) = \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\}$.
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function.
- $C^k : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the cost function for constraint $k \in [K] = \{1, 2, \ldots, K\}$, (denotes the cost incurred by the agent for constraint k)

A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ maps states to distributions over actions[2], [15]. Where $\Delta(\mathcal{A})$ is the set of all probability distributions over the action space $\mathcal{A}$. The goal

of an agent in CMDP with the long run average criterion is to find a policy that maximizes the long-run average reward while maintaining the constraint cost under a certain threshold [10], [11], [15].

- **Long-run average reward:** $\mathrm{J}_r(\pi) = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_\pi \left[\sum_{t=0}^{T-1} r(s_t, a_t)\right]$
- **Long-run average cost for constraint** k**:** $\mathrm{J}_c^k(\pi) = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_\pi \left[\sum_{t=0}^{T-1} c^k(s_t, a_t)\right]$

Where r_t is the reward at time t, π is the policy used and $\mathbb{E}_\pi[x]$ denotes the expected value of x given policy π. J refers to performance metric J_{r/c^k} for reward R and constraint (cost) C under policy π. The constrained optimization problem is:

$$\begin{aligned} \max_{\pi} \quad & J_r(\pi) \\ \text{s.t.} \quad & J_c^k(\pi) \leq \tau_k, \quad \forall k \in [K] \end{aligned}$$

where τ_k is the threshold for the constraint k.

2.4 Multi-agent MDP (MMDP)

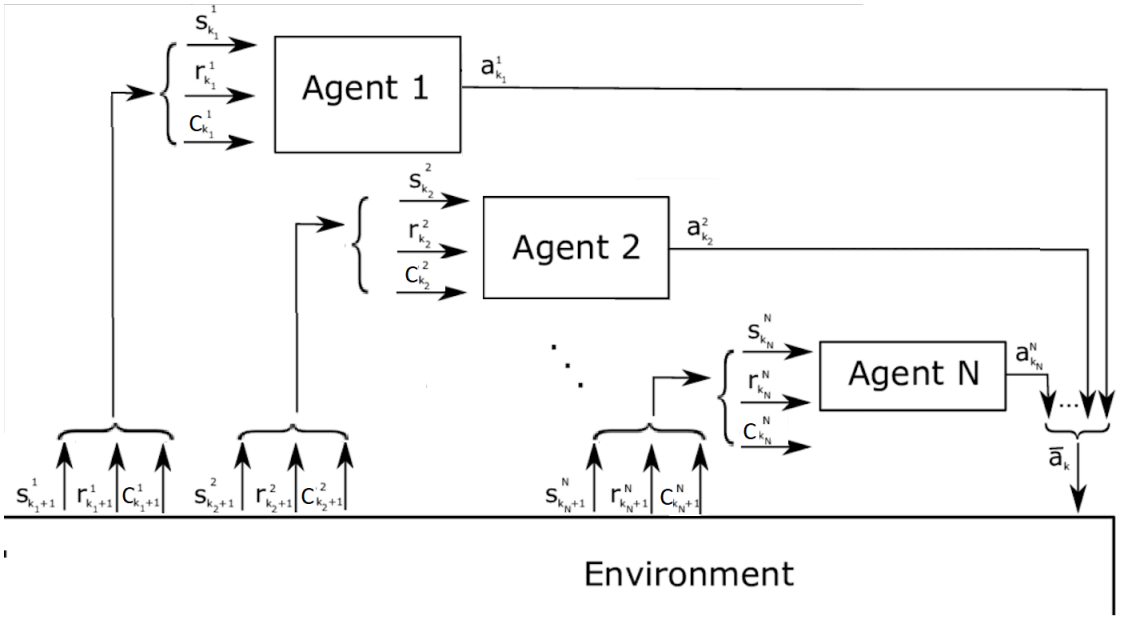

Figure 2.2: Agent–environment interaction in separable state space MDP for MAS, where $k_1 \in \{1, 2, 3, \ldots\}$, $s_{k_i}^i \in \mathcal{S}$, $a_{k_i}^i \in \mathcal{A}$, $r_{k_i}^i \in \mathcal{R}$, and $c_{k_i}^i \in \mathcal{R}$.

Just as the introduction of safety constraints motivated the extension of the standard MDP framework to Constrained Markov Decision Processes (CMDPs), MDPs can also be extended to Multi-agent Markov Decision Process (MMDP) framework, see 2.2. In this framework, multiple agents operate simultaneously within a shared environment and therefore must coordinate their behavior to achieve a common objective. In such a setting, the outcome of one agent action depends not only on

its own state and action but also on the state and action of the other agents in the system. The standard single agent framework is therefore insufficient for modeling the interactions of multiple agents that operate in the same system.

In MMDP the environment is described using a global state that contains information about all agents in the system. Rather than a single agent interacting with the environment, the decision making progress is determined collectively by all the agents. The global state space is therefore be represented as the cartesian product of the individual state space of each agent. Here is how it looks in the general case:

$$\mathcal{S} = S_1 \times S_2 \times ... \times S_N \tag{2.1}$$

where $\mathcal{S}$ denotes the joined state space, S_i denotes the state space of agent i and N is the number of agents in the system. In the same way, each agent selects an action from its own action space and the resulting decision of the system is represented by a joined action space

$$\mathcal{A} = A_1 \times A_2 \times ... \times A_N \tag{2.2}$$

where $\mathcal{A}$ denotes the joined action space, A_i denotes the action space of agent i.

The trainsition function is then extended to depend on the joint state and the joint action of all agents,

$$P(s' \mid s, a) \tag{2.3}$$

where

$$s = (s_1, ..., s_N), a = (a_1, ..., a_N) \tag{2.4}$$

and s' represent the new states for the agents. This means that the probability of transitioning to a new global state may depend on the action chosen by all agents in the system.

The reward function is generated in a similar way in the general case and is defined as:

$$R = R(s_1, ..., s_N, a_1, ..., a_N) \tag{2.5}$$

Here, there is a joined reward for the entire system as the reward depends on the collective behavior of the agents. In some settings, the reward function can be written as the sum of local reward functions.

$$R = R_1(s_1, a_1) + ... + R_N(s_N, a_N) \tag{2.6}$$

If the reward function can be written in this way, then it is referred to as a separable reward function. Separable rewards simplifies the learning process greatly since each agent contributes independently to the global objective, whereas non separable rewards often require stronger coordination between agents.

Introducing multiple agents significantly increases the complexity of the decision making problem. If a single agent system contains $\mid S \mid$ states, then an MMDP with N agents may contain up to $\mid S \mid^N$ joint states, assuming that all combinations of states are possible. This results that an algorithm which requires O(f($\mid S \mid$)) time to evaluate the best strategy in the single agent setting, then extending the same algorithm directly to an MMDP may require O(f($\mid S \mid^N$)) time. Consequently, algorithms that are polynomial in the single agent setting often become exponential in the number of agents when extended directly to MMDPs [16]. This scalability challenge is one of the major difficulties in multi-agent reinforcement learning and is a central challenge problem in this thesis.

Depending on how the agents learn over time and how they share information, multi-agents learning can be divided into different categories. The most common approaches are centralized training and execution (CTE), centralized training and decentralized execution (CTDE), and decentralized training and execution (DTE)[17]. In CTE learning, agents use global information from the system, information such as joint state, joint action, shared reward, and cost signal. This approach is usually difficult to scale when the number of agents increases because the joint state space and joint action space grows exponentially. The naive approach in this thesis has this setting.

Unlike in the CTE case, in DTE, there is no a central controller and the agents must chose actions on their own. In DTE learning, agents learn using only their own local information without getting access to the joint action, joint state or a centralized training process. This setting is easier to implement but can be more difficult in terms of coordination because each agent must learn while the other agents are learning and changing their policies [17], [18].

A standard middle ground or compromise for this approach is CTDE where agents share global data during training but act independently during execution. The agents find their own policies that can be executed individually, while agents use the centralized information during training [17]. When improving the naive approach, we used CTDE where each agent has its own Q-table and selects actions independently from its local state. However, the agents receive a shared reward and cost during training and a shared queue Z for tracking the constraint violation. This allows the agent to learn from the combined behavior of the system, allowing coordination while not having to deal with the joint state-action space.

2.5 Problem formulation

When scaling a system from single to multi agent, each agent interacts with an environment; see Figure 2.2 using RL and updating their policy over time. This introduces a few challenges to deal with.

The study of extending single agent to multi agent systems in RL is great for solving larger and complex tasks [2]. However, the extension presents significant scientific and engineering challenges that have not been fully resolved in the existing literature. Although significant progress has been made in the planning of single agents with limited resources using the CMDP framework, the planning achieved bounded regret and violation of zero constraint [1]. Extending these approaches to MASs presents complexities in theoretical guaranties, coordination and scalability.

Model based approaches have demonstrated the theoretical guaranties for single agent planning for infinite horizon and average reward CMDPs through the value iteration methods [9] and a recent study on model free RL algorithms has demonstrated a promising capability for learning safe policies without the complete knowledge of the environment model [1]. However, this approaches do not address the exponential state space growth that are presented in MASs.

In the domain of multi agent reinforcement leaning (MARL) study, existing work has primarily been focusing on unconstrained settings or finite horizon problems. Centralized training with decentralized execution has shown good performance in complex domains such as StarCraft and achieved very impressive wins through cumulative reward[11]. However, these approaches do not take into account safety guaranties, which makes them unsuitable for multi robot applications that require safety.

The challenge in this thesis is to develop algorithms that can scale the properties of a safe single-agent RL while addressing the challenge of dimensionality that comes from the joint state space. This thesis will find studies on safe single agent algorithms in RL to address for multi robot RL. The scientific and engineering challenge in this thesis is in closing the gap in theoretical for safe RL in single agent to a practical method of MASs in an infinite horizon.

The goal for this thesis is to extend long run average CMDP methods from single agent systems to multi agent systems. This will be done while preserving their constrain properties and avoiding the exponential increase in computational complexity that arises from the trivial joint state extension. To ensure that the proposed multi agent CMDP implementations is fast and accurate enough, they will be benchmarked in a different modification of safety grid environments.

3

Literature Review

This chapter discusses the literature which is the basis of this thesis, theoretically and algorithmically. The main goal reviewing the literature is to understand the different approaches used and to identify reinforcement learning methods that are suitable for safe planning under long run average criteria. Additionally to identify and evaluate single agent constrained methods that can be extended to multi agent constrained settings. List of the papers and key ideas are summarized in Table **??**

Paper	**Problem addressed & key idea**
Wei et al [1]. (2022)	Constrained average-reward learning in CMDPs with unknown dynamics. Uses a model-free primal-dual Q-learning algorithm.
Zheng & Ratliff [7]. (2020)	Safe exploration in constrained reinforcement learning. Uses a model-based optimistic-reward/pessimistic-cost approach.
Agarwal et al [19]. (2022)	Learning CMDPs while minimizing regret and constraint violations. Uses posterior sampling in a model-based setting.
Wei et al [20]. (2020)	Average-reward learning in MDPs with unknown dynamics. Develops a model-free algorithm without constraints.
Ashok et al [21]. (2017)	Planning in long-run average reward MDPs. Studies value-iteration methods for continuing tasks.
Brazdil et al [22]. (2014)	Optimization of multiple long-run objectives in MDPs. Studies reward-cost trade-offs and multiple objectives.
Forejt et al [23]. (2011)	Verification of multiple objectives in probabilistic systems. Develops methods for multi-objective verification.
Quatmann et al [24]. (2021)	Optimization of multiple long-run and total reward objectives. Studies trade-offs between competing objectives.

Table 3.1: Comparison of literature considered for algorithm selection

3.1 Regret and Constraint Violation Bounds

In MDP when comparing reinforcement learning algorithms different approaches are used but the central criterion for comparing the algorithms is regret. Regret is the difference in performance between the optimal policy and the learning algorithm [14]. When an algorithm has a low regret, the difference in cumulative reward is growing slowly over time. If regret is 0 the the algorithm is performing as good as the oracle but if the regret is growing linear with the horizontal time T then the algorithm is

making mistakes at every step, thus doesn't learn the optimal behavior. The goal is to have low regret because it reflects that the learning algorithm is approaching the optimal policy.

In constrained RL, regret alone is not enough because the agent has also to satisfy the cost constraint. Therefore, in constrained RL, the performance is evaluated using both regret and constraint violations. Constraint violation bound describes how bad the agent violates the safety criterion [1].

3.2 Candidate Papers

One of the main challenges in CRL is the learning an optimal policy while also satisfying safety constraint while the cost function and the reward function are not known. Many of the approaches that exist today leads to unsafe behavior during the learning process because of the challenge of balancing safety constraint satisfaction and reward optimization. To address this challenge, Zheng and Ratliff [7] presented a model-based algorithm based on the constrained upper confidence reinforcement learning (C-UCRL), model-based algorithm for CMDPs with a known transition dynamics but unknown reward and cost functions. The clever approach in this paper is that it is optimistic in reward uncertainty and pessimistic in cost uncertainty, it follows a strategy of "if the state is unexplored then assume that it is a good state and also think that it could be a dangerous state".

Achieving low regret and and constraint violation with algorithms that are computationally efficient is one of the main challenges of CMDP. To address this challenge, Agarwal, Bai and Aggarwal propose CMDP-PSRL [19], a posterior sampling algorithm for ergodic CMDPs with long-term average constraints. The CMDP-PSRL algorithm maintains a Dirichlet prior over multiple possible models and samples from the posterior at the start of each epoch, saves the CMDP optimization for this model using linear programming, and executes the resulting policy until the next epoch. An epoch refers to the period of interaction time where during this time the agent follows the policy that is computed from the previous sampled CMDP model. This approach is very smart compared to the optimistic approach because it has $S\text{x}A$ variables while optimistic has $S^2\text{x}A$ variables. Compared to the model-free [1] this paper has better regret and constraint violations, see Table 3.2. Under Slater's condition, they prove $T_M S\sqrt{AT}$ regret and constraint violation bounds, which makes it the first $\sqrt{T}$ type bounds for this setting using posterior sampling.

Wei et al. [20] introduce two model free algorithms for learning infinite horizon average reward MDPs. The First algorithm, optimistic Q-Learning, reduces the problem to a discounted-reward MDP and directly learns Q-values for each state-action pair. This algorithm achieves regret of $T^{2/3}$ under the minimal assumption of weakly communicating MDPs. The second algorithm, MDP-OOMD require the stricter ergodic MDPs and learns the optimal policy via a novel application of adaptive adversarial multi armed bandit algorithms. Each state maintains a bandit

algorithm to estimate action values. MDP-OOMD achieves improved regret of $\sqrt{T}$, as the cost of the more restrictive MDP variant.

The trade of between model-free and model-based RL is that while mode-based [19] is more sample efficient, which is found to be better for regret, the model-free [1] scales better to large problems and takes less memory.

Quatmann and Katoen [24] focus on multi-objective model checking for Markov automata, which combines long-run average and total reward objectives. Their contribution is that they present a computationally efficient procedure for multi-objective model checking of long-run average (LRA) reward and total reward objectives, plus the mixture of both. This is a generalization of the iterative approach of Forejt et al. [23], which is for the total and LRA reward of MDPs. Their procedure is that they reduce the complex multi-objective problem into a series of simple single objective problems by taking a weighted sum of the different reward functions. The method decomposes the problem using maximal end components (MEC) and solves for reachability on the new quotient model, similar to [21]. An MEC in MDP is a closed part where the agents cannot escape no matter what policy is chosen. once the agent enters it, there is a strategy that keeps the agent inside forever[21]. Both [[21], [24]], deal with long-run average rewards with MDPs but they approach the problem from different angles and both analyze systems that behave randomly. Understanding the studies of optimization of a system with multiple criteria is helpful to this thesis due to the nature of the environment that are used in this thesis. Understanding trade offs between multiple objectives can be related to the different constraints the agents has for example: the limited interactions between the agents or avoiding the obstacles in the environments.

Similarly to [24], Ashok et al [21] also first find all MECs. It has two phases. In the first phase, they find MECs. Run a value iteration locally and find the reward for each MEC. In the second phase they collapse the model to a quetient model (with MECs compressed to a single state) and find the best strategy to reach the good MEC. This problem is now a reachability problem in ending up in good MEC and focuses only on the important part of the MDP. Their method significantly outperforms linear programming approaches in benchmarks.

Both [24] and [22] address the computation of ϵ-approximate Pareto front for multiple long run average objectives. ϵ-approximate means that even tho the Pareto front is not exact, it is guaranteed to be in the small error margin ϵ. Pareto front represents the set of achievable expected long run reward vectors under a single strategy. Reward vector means, instead of having a reward value each strategy will have multiple reward values for each of the objectives. However the underlying models differ significantly. [22] study discrete time MDPs whereas [24] extend the analysis to Markov automata. Algorithmically, [22] reduce the problem to multi objective linear programming over steady state distribution while [24] combine linear programming with structural decomposition techniques such as MEC analysis. Furthermore, [24] supports combinations of long run average and total reward objectives, thus handling a richer class of specifications. Finally, unlike the primarily theoretical focus, [[22], [24]] provide an experimental evaluation that demonstrates practical applicability.

3.3 Selected algorithm

Another main challenge in CRL is the study of designing an algorithm which is model free and in the infinite horizon average reward and is in CMDPs, additionally this algorithms can learn the optimal policy while also satisfying the infinite horizon average cost constraint. Prior studies focused on approaches that are model based or unconstrained settings which have limits such as applying them when the environments dynamics are not known. To address this issue, In contrast to unconstrained average reward MDP algorithms such as [20], Wei, Liu and Ying present an algorithm called Triple-QA[1] which is the first model-free algorithm for infinite horizon average reward constrained Markov decision process. Unlike the prior Model based approaches, their method directly learns action value functions (Q-functions) for both reward and utility, whereas [20] considers only reward maximization without constraints. The algorithm uses a primal-dual approach. The primal part is for maximizing a weighted combination of the q-values of reward and utility. The dual part is for tracking constraint violations by maintaining a virtual queue Z; see section 2.1.1. In order to prevent oscillations, the queue Z is updated slowly (in K^B steps) while the q-functions are updated at every step where K is the total number of training steps and B is a parameter that controls how often the virtual queue Z is updated [1]. This algorithm proved regret of $\tilde{O}(K^{5/6})$ and zero constraint violations, see Table 3.2.

Algorithm [7] constructs confidence intervals and solves for robust linear programs and achieves a bounded regret, see Table 3.2, while guaranteeing that constraint violation is below the threshold with probability less than one and greater than or equal to zero. This paper sits between the model based approach which is based on a posterior sampling with unknown transitions and rewards, as discussed in Agarwal et al. [19], and the model free approach which is based on the standard Q-learning with unknown transitions and rewards, as discussed by Wei et al. [1]. Both papers are discussed in more detail below. [7] use confidence intervals directly instead of sampling, making the algorithm computationally light.

Both [1] and [19] deal with the problem of balancing reward maximization with safety constraints, but take a fundamentally different approach and are model-based and model-free, respectively.

Table 3.2: Constraint violation bounds and regret of selected algorithm and two most related papers

RL algorithms regret and constraint bound			
Algorithms	Regret Bound	Constraint Bound	Model
CMDP-PSRL in [19]	$\sqrt{T}$	$\sqrt{T}$	Model-Based
Triple-QA in [1]	$\tilde{O}(K^{5/6})$	Zero	Model-Free
C-UCRL in [20]	$T^{3/4}$	zero with prob $1-\eta$	Known

As shown in table 3.2, the posterior sampling algorithm has the best regret, but it must learn everything and Triple-QA has zero violations, but worse regret than the posterior sampling, plus it also needs to learn everything. C-UCRL has regret in the middle of the other two algorithms, but it requires less learning because the dynamics are known. This algorithm has different approaches, but they are very good in their own way.

4

Methods

This chapter describes the environment used for evaluating the reinforcement learning algorithms and the implemented learning algorithms themselves. The algorithms are presented progressively, starting from the single agent baseline and extending toward more advanced multi agent approaches.

4.1 Environment

The environment plays an important role in the evaluation of RL algorithms, since it defines how agents interact with the system, receive rewards, and violate safety constraints. In this thesis, a grid world based environment is used to evaluate both the constrained single agent and constrained multi agent implementation under the long run average reward setting.

The implemented environment used in this thesis is closely based on the grid world environment presented in the Triple-QA paper [1]; see figure 4.1. Since the original Triple-QA implementation and its environment are not open source, the environment was reconstructed based on the descriptions and figures in the paper. The choice of this environment was for two main reasons. First, using a similar environment allows the result of the single agent implementation in this thesis to be compared against the results reported in the original work, making it easier to identify potential implementation errors or deviations. Second, the structure of the environment is well designed for evaluating both the impact of constraint and the scalability in multi agent settings, since the path which receives the highest reward also violates the safety constraints and creates a sort of trade off between reward maximization and safety.

The grid world is defined over a finite set of states representing the positions of the agents together with four actions, one for each of the cardinal directions, that determine how the agent can move within the area.

The states (cells) are categorized into four different groups with different roles. The first two categories, the goal state and the start state are represented as a red star and a blue triangle respectively as illustrated in 4.1. The goal state is the state the agents aim to reach, and the agents will grant 1 reward if they manage to do so. In addition to the goal reward, there is also a small reward depending on the euclidean distance to the goal normalized to 0.1. This is to guide the agents toward

the goal state by providing incremental feedback during navigation. Secondly, the environment also contains one or more start states, depending on the number of agents. Those states are where the agents are initially placed at the beginning of each simulation. Also, if an agent reaches the goal state, then the agent will be returned to one of the unoccupied start states. The last two categories are the obstacle cells (yellow cells) and safe cells (black). Transitioning to an obstacle state incurs a cost of 1 for the agent, and otherwise the cost received is 0.

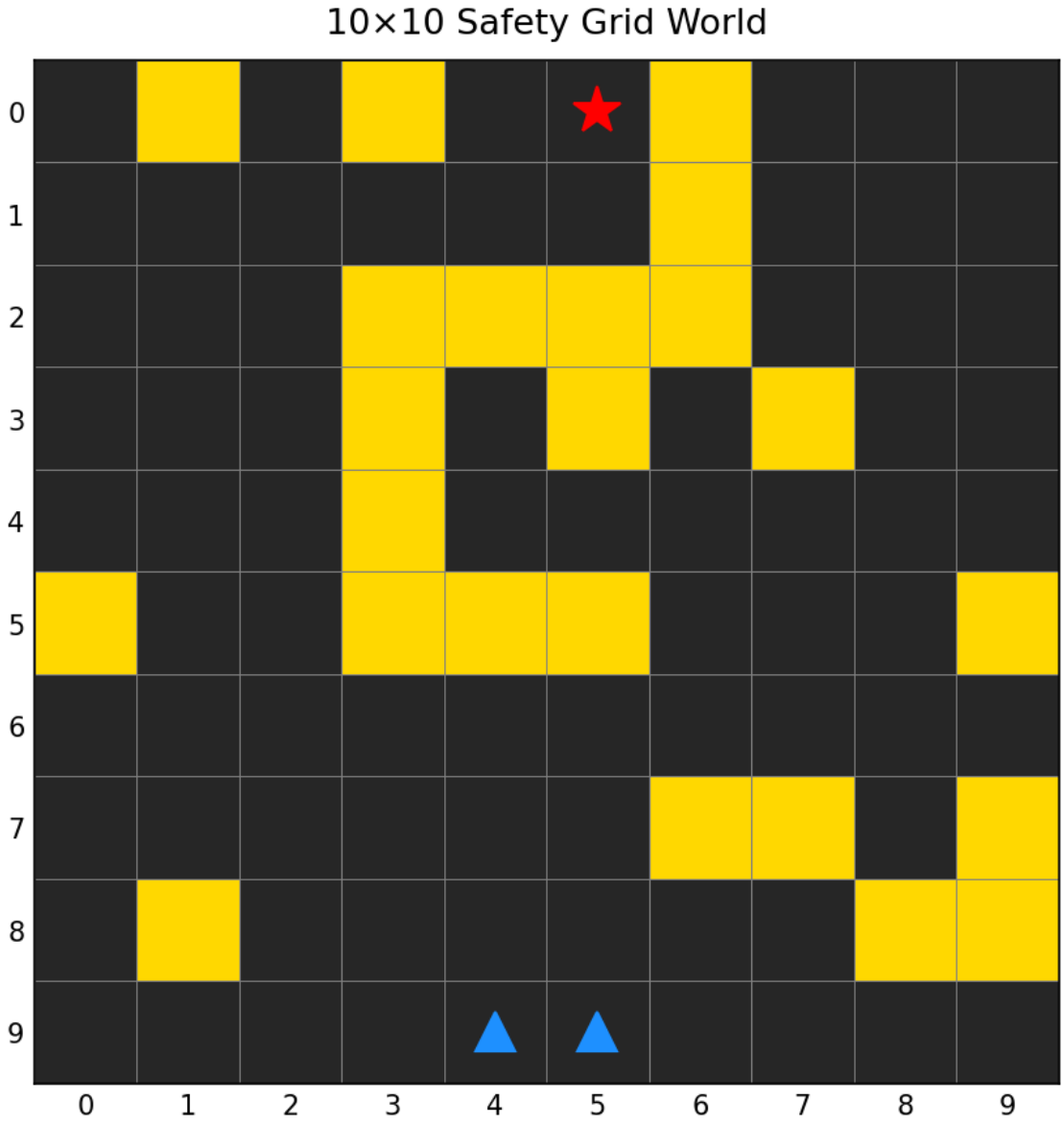

Figure 4.1: The multi agent grid world environment. The blue triangles are the start states, the yellow squares are obstacles and the red star is the goal

When extending the environment to MAS, additional interaction rules between agents must be considered. In this paper, two different interaction settings are studied. In the first approach, agents do not interact with each other at all. However, agents aim to maximize the joint reward while preserving the joint cost under the

predefined threshold, so the the objective is still defined at the system level. The entire reward and cost functions are separable and each of the agents chooses their actions independently based on their own state. The second approach, limited interaction between agents, is introduced by making it impossible for two agents to occupy the same state. This coordination restricts agents, since the available action may now depend on the positions of other agents. These two settings are compared to evaluate how agent interaction affects scalability, reward performance, and constraint satisfaction.

4.2 Algorithms

The development of the algorithms was based on the objectives and research questions; see section 1.2. First, a single agent constrained reinforcement learning algorithm is implemented. This algorithm is the baseline of the extension. The single agent is extended to a naive approach, joint state action multi agent system, and the third stage was a more scalable multi agent variant where the agents use separate Q-tables and share a constraint.

The starting point of this thesis is Triple-QA, a model free primal dual RL algorithm for infinite horizon average reward CMDPs [1]. In the literature, the constraint is formulated as a utility lower bound, but in this thesis the same idea is written as a cost form.

$$\begin{aligned} \max_{\pi} \quad & J_r(\pi) \\ \text{subject to} \quad & J_c(\pi) \leq \rho \end{aligned}$$

where

$$J_r(\pi) = \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} r(s_t, a_t) \right],$$

and

$$J_c(\pi) = \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} c(s_t, a_t) \right].$$

Here $J_r(\pi)$ is the long run average reward and $J_c(\pi)$ is the long run average cost for the policy π and ρ is the predefined constraint threshold.

In the grid world experiment, the cost is incurred when the agent enters states that are perceived as unsafe or obstacles; see figure 4.1.

This single agent implementation is used as a baseline to confirm that the algorithm behaves correctly as stated in the paper [1] in the constrained long run average setting before extending it to the multi agent setting. A pseudocode of the implemented algorithm is shown in algorithm 1. In this implementation, the agent maintains two

state action value tables, one for reward and one for cost. These tables are denoted by

$$\hat{Q}(s,a) \quad \text{and} \quad \hat{C}(s,a),$$

The update rules for these tables are shown on lines 12 to 18. The reward table estimates the long run expected reward of selecting a particular action in a given state, while the cost table estimates the corresponding long run expected cost associated with that action.

4.2.1 Single Agent Algorithm

At each time step, the agent greedily chooses an action with respect to the queue Z and $\eta > 0$ (lines 7). Z acts as a regulator that balances reward and cost. η is used to control the influence of the virtual queue Z when selecting an action. While a high value for Z increases the penalty for an action that incurs a high cost, a high value of η reduces the effect of the penalty. Therefore, the algorithm favors actions with high estimated reward and low cost, in particular, when the constraint is at risk of being violated. Additionally, in order to encourage exploration, the agent selects a random action with probability $p = 0.03$.

After executing the selected action, the agent observes the reward, the cost, and the next state (line 8). Then the reward and cost estimates are updated. The algorithm updates both the reward and cost estimates after each transition. In the update; observed reward and cost, an estimate of the future value of the next state, and a bonus used for exploration are used. The bonus is added to encourage the exploration of state action pairs that are not visited often. The optimistic tables $\hat{Q}$ and $\hat{C}$ are updated only when the new calculated estimate is equal to or smaller than the previous optimistic estimates.

Unlike the value function tables, the virtual queue is updated more slowly. The algorithm first accumulates costs over a fixed length of a frame, and at the end of the frame, the virtual queue is updated (line 24). The value function tables are updated at each time step, while Z are updated after each frame. The approach is preferred to reduce the instability caused by the rapidly changing penalty. This allows the estimated to have a fixed penalty after each time frame.

The algorithm can be summarized as follows;

- Initialize the tables of value function, visit counts, virtual queue $Z = 0$ and all the other global variables. (line 1-4)
- Observe the current state and select an action using the virtual queue and the penalized objective. (line 7)
- Execute the selected action and receive reward, cost, and the next state. (line 8)
- Update the cost and reward estimates based on the received values of the reward, cost, and next state. (line 10-11)
- Collect all costs within the time frame. (line 19)
- At the end of each frame, update the virtual queue. (line 24)

The correctness of this baseline implementation is important for a couple of reasons. First, it verifies that the implementation learns the correct policy that maximizes the reward while satisfying the cost constraint in the same environment as the original implementation in the Triple-QA implementation; see environment 4.1. In addition, this baseline implementation serves as a reference when evaluating the effect of adding multiple agents to the system.

Algorithm 1 Triple-QA

Initialize $Q_1(s,a) = K^{1/6}$, $\hat{Q}_1(s,a) = K^{1/6}$, $\gamma = 1 - \frac{1}{H}$, $\chi = K^{1/3}$, $\eta = K^{1/6}$, $\iota = 8\log(\sqrt{2K})$, $\beta = \frac{2}{3}$, $\epsilon = \frac{9\kappa}{\sqrt{SA\iota}K^{1/6}}$
Initialize $n_1(s,a) = 0$, $\forall (s,a) \in \mathcal{S} \times \mathcal{A}$
Initialize $\hat{V}_1(s) \leftarrow H$, $\forall s \in \mathcal{S}$
Initialize $\bar{C} \leftarrow 0$, $Z_1 \leftarrow 0$
Define $\alpha_\tau = \frac{\chi+1}{\chi+\tau}$, $b_\tau = \kappa\sqrt{\frac{(\chi+1)\iota}{\chi+\tau}}$, $frameLength = K^\beta$
for $k = 1$ to K **do**
    $a_k \leftarrow \arg\max_a \left(\hat{Q}_k(s_k,a) + \frac{Z}{\eta}\hat{C}_k(s_k,a)\right)$
    Observe s_{k+1}
    $n_{k+1}(s_k,a_k) \leftarrow n_k(s_k,a_k) + 1$, $\tau \leftarrow n_{k+1}(s_k,a_k)$
    Update:

$$Q_{k+1}(s_k,a_k) \leftarrow (1-\alpha_\tau)Q_k(s_k,a_k) + \alpha_\tau\Big[r(s_k,a_k) + \gamma\hat{V}_k(s_{k+1}) + b_\tau\Big]$$

    Update:

$$C_{k+1}(s_k,a_k) \leftarrow (1-\alpha_\tau)C_k(s_k,a_k) + \alpha_\tau\Big[g(s_k,a_k) + \gamma\hat{W}_k(s_{k+1}) + b_\tau\Big]$$

    if $Q_{k+1}(s_k,a_k) \le \hat{Q}_k(s_k,a_k)$ **and** $C_{k+1}(s_k,a_k) \le \hat{C}_k(s_k,a_k)$ **then**
        $\hat{Q}_{k+1}(s_k,a_k) \leftarrow Q_{k+1}(s_k,a_k)$
        $\hat{C}_{k+1}(s_k,a_k) \leftarrow C_{k+1}(s_k,a_k)$
    else
        $\hat{Q}_{k+1}(s_k,a_k) \leftarrow \hat{Q}_k(s_k,a_k)$
        $\hat{C}_{k+1}(s_k,a_k) \leftarrow \hat{C}_k(s_k,a_k)$
    end if
    $\bar{C} \leftarrow \bar{C} + (1-\gamma)\hat{C}_k(s_k,a_k)$
    $a^0 \leftarrow \arg\max_a \left(\hat{Q}_{k+1}(s_k,a) + \frac{Z}{\eta}\hat{C}_{k+1}(s_k,a)\right)$
    $\hat{V}_{k+1}(s_k) \leftarrow \hat{Q}_{k+1}(s_k,a^0)$
    $\hat{W}_{k+1}(s_k) \leftarrow \hat{C}_{k+1}(s_k,a^0)$
    if $k \bmod frameLength = 0$ **then**
        $Z \leftarrow Z + \rho + \epsilon - \frac{\bar{C}}{K^\beta}$
        $\bar{C} \leftarrow 0$, $n_t(s,a) \leftarrow 0$
        Reset $Q, \hat{Q}, V, C, \hat{C}, W$ to H
    end if
end for

4.2.2 Trivial Joint State Action Extension

The first approach for the multi agent method is a direct extension of the single agent method; see section 4.2.1 to a joint state action space. In this setting, for a multi agent system with N agents, the state space is S^N and the global state is defined as

$$s = (s^1, s^2, \ldots, s^N),$$

and the global action action is defined as

$$a = (a^1, a^2, \ldots, a^N)$$

. s^i and a^i is the state of the agent i and the action selected by the agent, respectively. And the joint state is

$$S_{\text{joint}} = S_1 \times S_2 \times \cdots \times S_N,$$

and the joint action is

$$A_{\text{joint}} = A_1 \times A_2 \times \cdots \times A_N.$$

In this trivial extension approach, the full multi agent system is treated as a single CMDP, therefore, the algorithm maintains one table for reward and one table for cost over joint state action pairs. The tables can be formulated similarly to the single agent as

$$\hat{Q}(s, a) \quad \text{and} \quad \hat{C}(s, a),$$

but here the state is $s \in S_{\text{joint}}$. Similarly, the action is $a \in A_{\text{joint}}$. The more agents are added to the system, the state and action space grows exponentially.

In this implementation, the predefined constraint threshold grows by a multiple of the number of agents the system has. Thus, agents share the same goal and actions are selected with the same penalized objectives as for a single agent 4.2.1. The reward and cost are additive over agents and therefore separable, as defined in Section 2.4. The system reward and cost are given by

$$r_t = \sum_{i=1}^{N} r_t^i$$

and

$$c_t = \sum_{i=1}^{N} c_t^i$$

The main advantage of this implementation is that it serves as a benchmark for the improved implementation, see subsection 4.2.3. Plus, this trivial extension also preserves the structure of the single agent CMDP formulation which makes this method a good baseline for studying the multi agent extension. But, this method does not scale well to the joint state action table growing exponentially with the number of agents added to the system. This makes the direct extension from single agent to multi agent exponentially expensive in training time and in memory. For this reasons, this extension mainly serves as demonstration of the scalability problem described in 2.5.

4.2.3 Centralized Training Decentralized Execution

In order to address the scalability problem, this thesis implemented another multi agent method with centralized training and decentralized execution (CTDE). Unlike the trivial extension where the system had to learn the full joint state action space, in this method, each agent maintains their own reward and cost tables, and during training they use shared reward, shared cost and the virtual queue Z. Each agent updates its local tables using the shared reward, cost, and also the virtual queue.

The algorithm maintains two tables,

$$\hat{Q}^i(s^i, a^i) \quad \text{and} \quad \hat{C}^i(s^i, a^i),$$

for each agent i where s^i and a^i are the local state of the agent and the action of the agent i. The agents select their own action using the local penalized objectives of their own reward and cost tables:

$$a_i = \arg\max_{a_i \in \mathcal{A}_i} \left(\hat{Q}_i(s_i, a_i) - \frac{Z}{\eta} \hat{C}_i(s_i, a_i) \right)$$

This correspond to the decentralized execution component of the CTDE framework, since each agent selects actions using only local information. The Joint action at time t is given by

$$a_t = (a_t^1, a_t^2, \ldots, a_t^N)$$

.

Training uses centralized information such as shared reward, shared cost, and shared virtual queue even though the agents select actions using their own tables. After the selected actions are executed then the environment returns the next state of each agent and the accumulated reward and cost. Similarly to the single agent implementation 4.2.1, the shared cost is accumulated in each frame and it is used to update the virtual queue at the end of each frame.

Here, if the collective behavior of the agents violates the constraint, the same queue affects all agents of the system. The constraint causes each agent to select an action that maximizes their own reward but is also compatible with the global safety constraint of the system.

The CTDE approach helps reduce the size of the value representation from a single joint state action table to multiple separate tables. For N agents, there are N separate tables. When the agents have their own local tables, then the number of value entries becomes

$$N|S||A|$$

instead of the problematic state space growth in the trivial extension which is

$$|S|^N|A|^N$$

The CTDE approach has changed the exponential growth in memory with the number of agents problem in the trivial extension to linear in the number of agents. Therefore, the method is more scalable than the trivial extension.

The single agent implementation is evaluated against this approach and the trivial joint state action extension. This comparison is used to guide the thesis to develop according the research questions such as can the extension preserve the long run average cost constraint while maintaining a reasonable reward performance.

5

Results and Discussion

This chapter presents the results of the simulations that were performed in this thesis. All algorithms considered in this chapter are modifications of the Triple QA algorithm from the base paper [1] and the results are evaluated with respect to two main criteria: long run average reward and long run average cost. The reward is used to measure how well the agents learn to reach the goal state, while the cost is used to evaluate whether the learned behavior satisfies the safety constraint.

The figures presented show the running average reward and the running average cost during training, represented by the blue and red curves, respectively. In total, there were five runs with different seeds of 2,500,000 steps for a single agent and multi agent systems with 2 agents and 100 million steps for multi agent systems with 3 agents. The results of these five runs were then averaged into curves. The shaded area is the standard deviation from the mean. The threshold used for all runs was 0.15 times the number of agents, to match the value used in the base paper, which allows a direct comparison of the results.

The results of the simulation are presented in four stages. First, the results of the simulations of a single agent will be presented. This work was used as a baseline and to verify that this implementation is consistent with the original article [1]. Second, the result of the simulation from the trivial extension. Third, the result of separate Q-table (CTDE). The separate Q-table implementation is evaluated and compared with the trivial extension. Finally, we look at how the trivial approach and the CTDE approach scale with three agents. The primary objective is to assess how the different algorithms scale with the number of agents and how this affects the reward and the rate of convergence.

5.1 Single Agent

Figure 5.1 shows that the average reward and cost received by the agent match the value reported on the base paper, which are around 0.11 and 0.15, respectively. This indicates that the implementation is consistent with the original algorithm and that there are no significant deviations or implementation errors. Both reward and cost converge over time, with a gradual reduction in variance across seeds throughout training. This suggests that the learning process is stable and results in an extremely similar strategy across seeds. As a result, the single agent implementation can be considered a correct baseline for the later implementations to be compared against.

In the extension, the cost and reward depend on how many agents are in the system. For example, a system with 2 agents should have twice as much long run average reward as the baseline while the long run average cost is below the threshold.

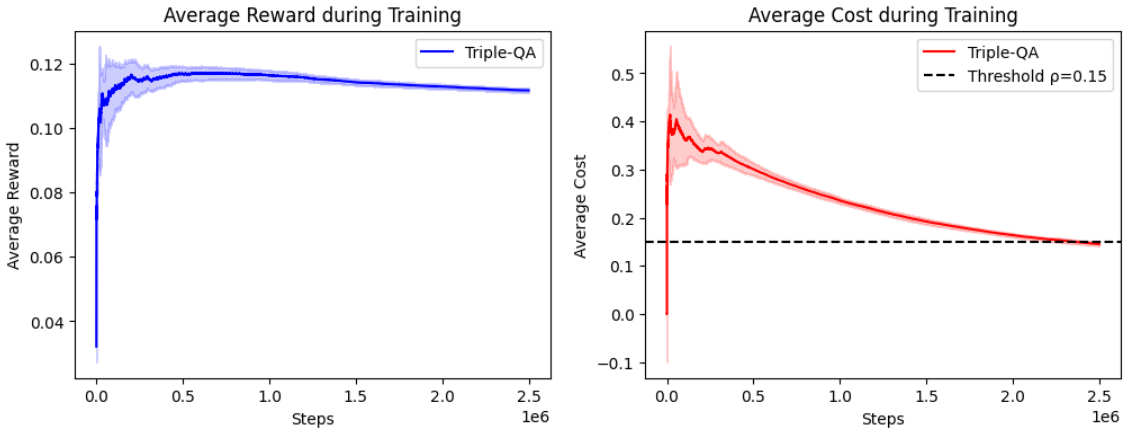

Figure 5.1: Both reward and cost as a function of steps for a constrained single agent in a grid world environment. The doted line is the threshold.

5.2 Multi Agent

The multi agent experiment is to evaluate how the constrained learning behaves when the system has more than a single agent in the environment. This thesis considers two different extensions. The first is the trivial extension of the joint state action 4.2.2, where the entire system is treated as a single agent CMDP system due to the extension that preserves the CMDP formulation of the constrained single agent system. The second is separate Q-table (CTDE) where each agent maintains their own separate Q-table while using shared information such as reward, cost, and the virtual queue; see section 4.2.3.

The main reason for these experiments is to evaluate the effect of adding multiple agents to a single agent constraint system. The effect can be evaluated in scalability, performance in long run average reward and constraint satisfaction.

5.2.1 Trivial Extension

Figure 5.2 shows the result of the extension of the trivial joint state, where the joint state of all agents is treated as a single state and the joint action of all agents is treated as a single action. The simulation occurs when there are two agents in the environment. The size of the joint state action space increases in this setting exponentially with the number of agents, but the structure of the single agent CMDP formulation is maintained.

From the observation in figure 5.2, the reward curve initially increases, showing that the agents are learning in the joint state action environment. Even though agents are learning, the reward at the beginning of the learning is lower compared to the

single agent system, see figure 5.1. One possible explanation is that this happened because of the increased difficulty of the learning problem. One explanation is that in the single agent, there were fewer state action pairs, which makes it easier for the combinations to be visited frequently. But in the joint state action pairs the state action combinations is visited less frequently and therefore the learning process is slower and less efficient.

As shown in the cost curve of figure 5.2, the policy is overly conservative and learns to overly avoid unsafe states compared to the optimal policy that balances long run average reward and cost. As a lack of balancing between reward and cost, the long run average cost is reduced way below the constraint cost. This reduction resulted in low long run average reward performance during training.

As discussed in section 4.2, the trivial extension as a direct multi agent baseline demonstrates the scalability problem. It becomes increasingly inefficient as the number of agents increases, and this was a motivation for the use of a more scalable approach based on centralized training and decentralized execution; see section 4.2.3.

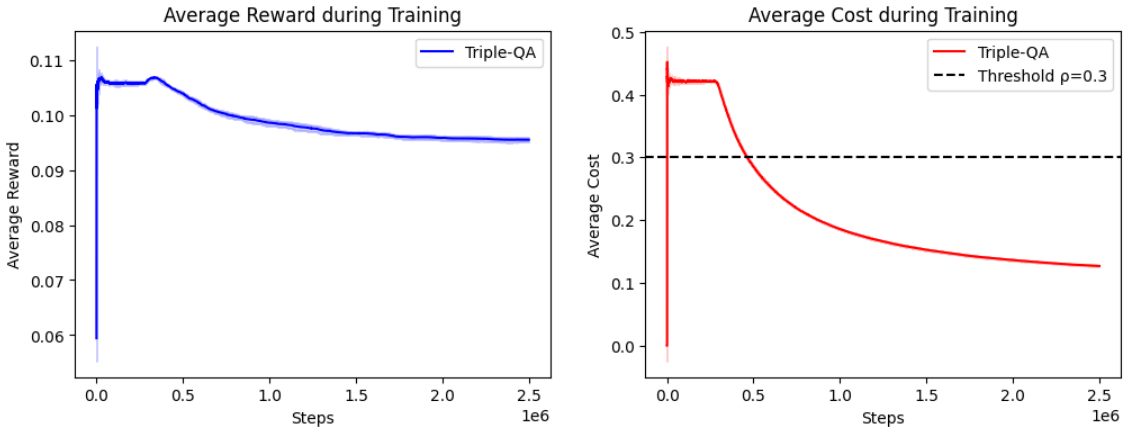

Figure 5.2: Both reward and cost as a function of steps for multi agent system with the trivial extension algorithm in a grid world environment. The doted line is the threshold.

5.2.2 Centralized Training Decentralized Execution

The results of the separate CTDE extension, see figure 5.3, show that the average reward converges to around 0.213, which is close to twice the single agent constrained reward. This indicates that the separate Q-table algorithm can find a strategy for two agents which is as good as the single agent algorithm. This suggests that the separate Q-table algorithm can scale the reward approximately linearly with the number of agents in the absence of interaction while preserving the safety constrained threshold.

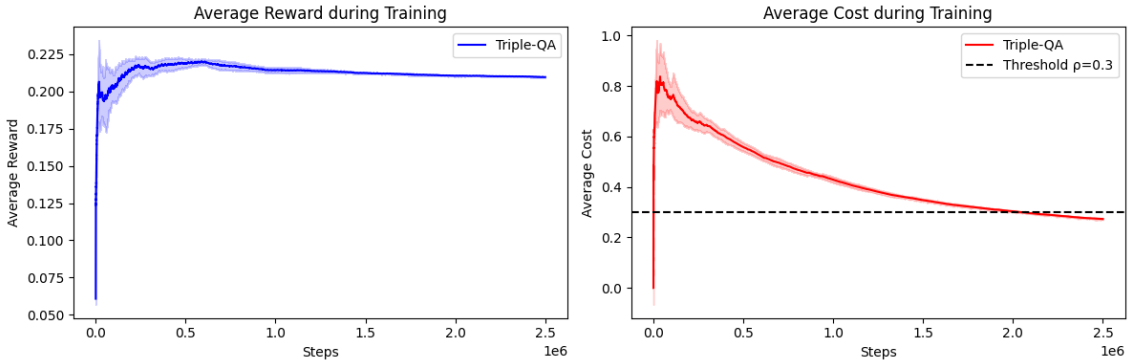

Figure 5.3: Both result and cost as a function of steps for multi agent system with the separate Q-table algorithm in a grid world environment. The doted line is the threshold.

5.2.3 Environmental Change

In MAS, the assumption of independent agents with no interaction between them is often unrealistic, as agents typically share the same environment and may influence each others possible actions. The split Q-table algorithm implicitly assumes that the optimal global strategy is close to the combined local optimal strategy for each individual agent though separating the Q-tables. However, these assumptions begin to break down when interactions between agents are introduced.

To investigate the impact of interaction between agents, the separate Q-table algorithm was evaluated in two different environments; One with no interactions and one where the agents are not allowed to occupy the same state (position) at the same time. Figure 5.4 compares the long run average reward and cost for these two settings.

The results of the simulations; see figures 5.3, 5.4 shows slight differences in long run average and reward. Both approaches converge to similar reward and cost values. This suggests that the introduction of limited interaction does not significantly affect the performance of the separate Q-learning algorithm in this setting.

One possible explanation is that the interaction between agents is relatively weak, since conflicts between agents occur infrequently and therefore only restrict a small subset of available combination of states and actions. As a result, agents are still able to behave approximately independently, meaning, the assumption underlying of the split Q-table algorithm still holds.

These results suggest that separating the Q-table between agents can still be a space and time effective way to find a strategy for agents with limited interaction by avoiding the rapid growth with the joint state action space present in the trivial extension approach.

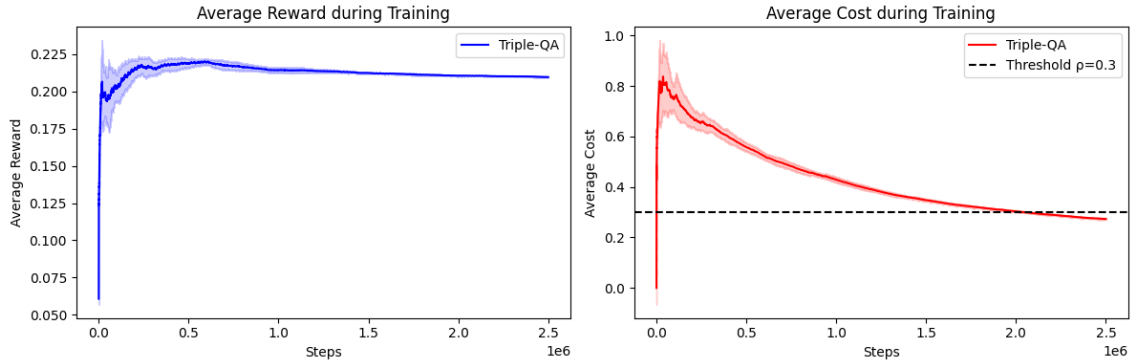

Figure 5.4: Result of the simulation where agents are not allowed to be on the same state. Both reward and cost as a function of steps for multi-agent system with the separate Q-table algorithm in the modified grid world environment. The doted line is the threshold

5.2.4 Effect of Increasing the Number of Agents

The result of the training performance of the trivial joint state action space extension and the separate Q-table (CTDE) extension can be shown in figure 5.5. The goal of this experiment is to study whether the observed behavior in the joint state action space extension and the separate Q-table (CTDE) extension with two agents remained stable when more agents were added to the system. In this evaluation, both the trivial extension and the separate Q-table (CTDE) converged quickly, but they were unable to satisfy the constraint. The initial step taken to solve this problem was to increase the number of steps to 100 million steps, and both remained above the threshold 0.45. The separate Q-table achieved substantial higher long run average reward, but this reward achievement came with a higher long run average cost, and the trivial extension achieved lower long run average reward and cost, but still remained above the threshold.

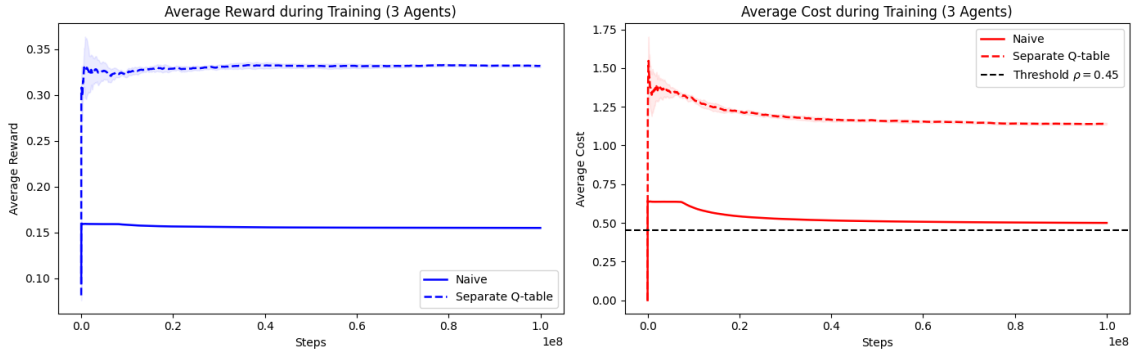

Figure 5.5: Result of the trivial joint state action extension and separate Q-table (CTDE) extension with three agents and 100 million running steps. The black doted line is the threshold 0.45 In the left we have long run average reward and in the right long run average cost.

6

Conclusion

This chapter presents the key findings and learning during this thesis, summarizes and interprets their meaning and significance, and answers the research questions 1.2. This chapter also includes the future work of this thesis which aims to reflect on what the next step is in this work, studies that could have been done, but limited due to time constraint.

6.1 Summary

From the results and discussion; see chapter 5, we can observe four main conclusions. First, the single agent algorithm works as intended. The result shows that the single agent implementation correctly learns the intended behavior. The reward is high when the cost constraint is violated, but the implementation decreases the cost until it reaches the threshold. As a result of the agent avoiding unsafe states, the reward decreases; see section 5.1 and figure 5.1. This baseline confirms that the single agent implementation is a proper algorithm for studying the extension from a single agent setting to a multi agent setting.

Second, as shown in section 5.2.1 and figure 5.2, we can observe that the trivial extension learns in the multi agent setting, however, the performance of the multi agent system is limited by the joint state action space growth. This method maintains the original formulation of CMDP but the exponential growth of the joint state action space makes this method less efficient.

Third, the CTDE approach with separate Q-table for each agent addresses the joint state growth problem that was introduced in the trivial extension. This method reduces the memory and joint state action space challenge by allowing each agent to learn from their own local state action space, while sharing global information such as long run average cost, long run average reward and the virtual queue. The result of CTDE approach shows performance improvement in the long run average reward compared to the trivial extension while maintaining the long run average cost close to the constraint. See section 5.2.2 and figure 5.3.

The separate Q-table (CTDE) extension provides a better trade off between scalability and reward performance and constraint satisfaction than the trivial join state action space extension. The trivial extension was able to learn to reduce the long run average cost while it was way above the constraint threshold, but this came at the

cost of the long run average reward performance 5.2, 5.3. From this experiment, we can observe that the policy learned to avoid unsafe states and was unable to let the agent get higher rewards. In contrast to the performance of the trivial joint state action space extension, the separate Q-table (CTDE) extension achieved higher reward while keeping the long run average cost very close to the constraint threshold. This result suggests that using the separate Q-table where the agents use a local Q-table while sharing global information such as the long run average reward, the long run average cost and the virtual queue Z was a better extension than the trivial joint state action space extension.

Fourth, in order to study whether the behavior that was observed with two agents remained stable, the methods were evaluated with three agents. In this evaluation both the trivial extension and the separate Q-table (CTDE) converged quickly, but they were unable to satisfy the constraint; see figure 5.5. The evaluation was further tested even with the 100 million number of steps, and both remained above the threshold 0.45. The separate Q-table achieved substantial higher long run average reward, but this higher long run average reward achievement came with a higher long run average cost, and the trivial extension achieved lower long run average reward and cost, but still remained above the threshold. From this result, we can observe that adding additional agents makes it more difficult to solve problems that are not solvable by just adding more number of training steps. In particular, the shared constraint becomes more difficult to satisfy as more agents are added and interacted with the same environment. The two multi agent extensions do not preserve the constraint satisfaction property from the observed results of the experiment.

6.2 Future Work

Although multiple comparisons of extensions have been demonstrated, several important challenges remain unsolved. In particular, when interaction constraints were introduced between agents in the separate Q-table setting 5.2.3, the results indicated only minor performance differences compared to the setting without interaction. In more strongly coupled systems, where an agent action depends more directly on other agents decisions and positions, the assumption underlying the separate Q formulation, that the global optimal policy can be approximated by the independent local policies, may start to break down. Future work could therefore investigate how increasing levels of interaction affect the performance of the separate Q-table method and find if and when decentralized learning no longer is a reasonable approximation.

Another important direction is to further investigate the behavior of the trivial joint state action extension. This extension has the same CMDP formulation as the single agent implementation since it treats the full system as a single system with large CMDP. Therefore, this extension is the most straightforward method. However, from the observed result; see figure 5.2, the policy is overly conservative and learns to overly avoid states that are not safe and as a result of that, the long run average cost is reduced way below the constraint cost. This resulted in low long run average reward performance. So future work should debug whether this unwanted behavior was caused by the exponential growth of the joint state action space, the global

variables, tightness, or the virtual queue update. Reward and cost aggregation could also be the issue.

As discussed in sections 5.2.4 and 6.1, the result of experimenting with three agents in both multi agent extension methods introduced additional difficulties. Although both methods converge quickly, both methods were unable to preserve the long run average cost under the predefined constrained threshold. What we can observe from the result is that increasing the number of agents does not preserve the safety of these methods. So, future work should include further investigation on how to handle constraint satisfaction for multi agent systems with multiple agents added in the system. This could vary from testing agent specific virtual queue to more coordination between agents and additional safety mechanism.

The environment used in this thesis; see figure 4.1 is useful for separating the reward and the cost, and it is a good representation to evaluate scalability in multi agent systems. However, future work could include the use of larger environments, environments with robotic simulations, or environments with moving obstacles. This environment is more realistic than the simplified version we used in this thesis. Specifically, Safety Gym is a great toolkit for studying safe reinforcement learning. So, integrating this algorithm into these large environments with different robotic simulations can make this experiment more realistic, and it could also help determine whether these challenges are based on the simplified environment or formulation.

Bibliography

[1] H. Wei, X. Liu, and L. Ying, "A provably-efficient model-free algorithm for infinite-horizon average-reward constrained markov decision processes," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 3868–3876.

[2] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," *Innovations in multi-agent systems and applications-1*, pp. 183–221, 2010, Available: ://link.springer.com/chapter/10.1007/978-3-642-14435-6$_7$.

[3] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. De Cote, "A survey of learning in multiagent environments: Dealing with non-stationarity," *arXiv preprint arXiv:1707.09183*, 2017.

[4] R. S. Sutton, A. G. Barto, et al., *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, Available: ://web.stanford.edu/class/psych209/Readings/SuttonBa

[5] M. Ono, M. Pavone, Y. Kuwata, and J. Balaram, "Chance-constrained dynamic programming with application to risk-aware robotic space exploration," *Autonomous Robots*, vol. 39, pp. 555–571, 2015, Available: https://web.stanford.edu/~pavone/papers/Ono.Pavone.ea.AUR014.pdf. DOI: 10.1007/s10514-015-9467-7.

[6] M. C. Y. F. Richter R. K. Orosco2, *Open-sourced reinforcement learning environments for surgical robotics*, Available: https://arxiv.org/abs/1903.02090, March 2019.

[7] L. Zheng and L. J. Ratliff, "Constrained upper confidence reinforcement learning with known dynamics," in *Proceedings of the 2nd Annual Conference on Learning for Dynamics and Control (L4DC)*, ser. Proceedings of Machine Learning Research, vol. 120, 2020, pp. 1–10.

[8] J. K. T. Quatmann, *Multi-objective optimization of long-run average and total rewards*, Available: https://rdcu.be/eTxlO, March 2021.

[9] P. Ashok, K. Chatterjee, P. Daca, J. Kretinsky, and T. Meggendorfer, *Value iteration for long-run average reward in markov decision processes*, Available: https://rdcu.be/eTxku, Jul. 2017.

[10] V. A. M. Agarwal Q. Bai, *Regret guarantees for model-based reinforcement learning with long-term average constraints*, Available: https://proceedings.mlr.press/v180/agarwal22b/agarwal22b.pdf, 2022.

[11] L. S. S V. Albrecht F. Christianos, *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2014, Available: https://www.marl-book.com/.

[12] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[13] D. Zhang, Q. Yuan, L. Meng, R. Xia, W. Liu, and C. Qin, "Reinforcement learning for single-agent to multi-agent systems: From basic theory to industrial application progress, a survey," *Artificial Intelligence Review*, 2025.

[14] P. Auer, T. Jaksch, and R. Ortner, "Near-optimal regret bounds for reinforcement learning," *Advances in neural information processing systems*, vol. 21, 2008.

[15] E. Altman, *Constrained Markov decision processes*. Routledge, 2021, Available: `https://doi.org/10.1201/9781315140223`.

[16] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002, Available: `https://arxiv.org/pdf/1301.3836`.

[17] C. Amato, "An introduction to centralized training for decentralized execution in cooperative multi-agent reinforcement learning," *arXiv preprint arXiv:2409.03052*, 2024.

[18] C. Amato, "An initial introduction to cooperative multi-agent reinforcement learning," *arXiv preprint arXiv:2405.06161*, 2024.

[19] M. Agarwal, Q. Bai, and V. Aggarwal, "Regret guarantees for model-based reinforcement learning with long-term average constraints," in *Uncertainty in Artificial Intelligence*, PMLR, 2022, pp. 22–31.

[20] C.-Y. Wei, M. J. Jahromi, H. Luo, H. Sharma, and R. Jain, "Model-free reinforcement learning in infinite-horizon average-reward markov decision processes," in *International conference on machine learning*, PMLR, 2020, pp. 10 170–10 180.

[21] P. Ashok, K. Chatterjee, P. Daca, J. Křetínskỳ, and T. Meggendorfer, "Value iteration for long-run average reward in markov decision processes," in *International Conference on Computer Aided Verification*, Springer, 2017, pp. 201–221.

[22] T. Brázdil, V. Brožek, K. Chatterjee, V. Forejt, and A. Kučera, "Markov decision processes with multiple long-run average objectives," *Logical Methods in Computer Science*, vol. 10, 2014.

[23] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Quantitative multi-objective verification for probabilistic systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2011, pp. 112–127.

[24] T. Quatmann and J.-P. Katoen, "Multi-objective optimization of long-run average and total rewards," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2021, pp. 230–249.