EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Data Mining Group

Safe Step by Safe Step: Curriculum Learning in Constrained Reinforcement Learning

Master Thesis

Kelvin Toonen

Supervisor and committee members:
Dr. Thiago Dias Simão (supervisor)
Dr. Renata Medeiros de Carvalho (1st assessor)
MSc. Tristan Tomilin (2nd assessor)

Eindhoven, October 2024

Abstract

Deep reinforcement learning is a growing field that has been successfully applied to many domains. This growth gives rise to safety concerns about applications using reinforcement learning. Therefore it is crucial to investigate the safety aspect in this field, especially in the domain of robotics where agents can break surrounding objects or themselves. We believe that curriculum learning has the power to help with creating safer agents, because it helps the agent with learning faster and it allows for the agent to learn in safer and more controlled environments leading up to the target environment. More specifically, we change the environment only slightly as to make it easier to transfer knowledge from one environment to the next, while still influencing the exploration process of the agent. This project combines curriculum learning with constrained reinforcement learning, a specific form of incorporating safety, to create a framework that allows agents to learn safely, even during training. This framework is also extended to include automation of the curriculum.

Preface

Before you lies my master thesis "Safe Step by Safe Step: Curriculum Learning in Constrained Reinforcement Learning", which was researched and written from January to October 2024.

I have been interested in AI since I did my graduation project in high school on creating a neural network to distinguish apples from bananas, although this failed miserably back then. This interest is the main reason why I chose the master Data Science & AI, where I had to choose a thesis project. This project stood out to me, because I found the concept of curriculum learning very intriguing. I liked deep learning for the fact that it somewhat mimics the human mind and curriculum learning does so in a similar way, as we humans learn most tasks in an incremental fashion, by building on knowledge of previously performed tasks. Therefore, this was one of the reasons why you are reading this.

I would like to thank my supervisor, Dr. Thiago Dias Simão, for his help and guidance throughout the project. He is also one of the most important reasons why I chose this project, as he is kind, patient and very intellectual. I am especially glad for all of the ideas we came up with together during our brainstorming sessions, which helped move the project in a better direction.

I want to thank the HPC Lab at the TUe and Alain van Hoof in particular for allowing me to use the much-needed computing power and for always helping me on the multitude of occasions where I did not know how to get something working on the server.

I want to thank my friends and family for supporting me throughout this adventure. And finally, I want to thank you, the reader, for taking the time to read this thesis. I hope you enjoy the journey and at least learn something you find interesting.

Contents

Chapter 1

Introduction

Due to the growing popularity and use of AI, an important philosophical question arises: how safe is AI? This question is too broad to give a proper answer, but we can focus on a certain domain where a safety requirement is most evident, namely in the robotics domain. In the robotics domain, where interactions occur in the real world, it is clear there is a direct impact on the people and objects around a robot. If improperly trained, a robot may hurt bystanders, damage nearby objects or break down itself. An example is an autonomous helicopter that needs a lot of exploration due to the complexity of the task, but has the risk of crashing [Martín H. and de Lope, 2009]. Apart from physical risks, safety can also relate to environments with a lot of uncertainty that may cause poor performance for models that do not properly take this into account [Garcıa and Fernández, 2015]. An example would be that the quickest route to a goal has some uncertainty, where it is possible to fall of the path creating a large detour, while a slightly longer route may consistently reach the goal without any detours. As all of these issues are to be avoided, it is necessary to look at making AI models safer.

We try to mitigate these issues for a certain type of AI models, deep reinforcement learning models. These kinds of models are quite prevalent in, for example, the robotics domain and work by having an agent, the model, learn how to behave in the environment by exploring and earning a reward when it performs an action that we deem to be good. Here we want to optimize the parameters of the agent in such a way that the cumulative reward is maximized, which is the objective of the agent. Ultimately, we want to do this without having unwanted side effects or using unsafe exploration, which are more likely to occur in complex agents and environments [Amodei et al., 2016].

Motivation. Plenty of research has been done on safety in deep reinforcement learning [Kadota et al., 2006, Simão et al., 2021, Ramírez et al., 2022, Yang et al., 2023] and generally alters either the objective or the exploration process [Garcıa and Fernández, 2015]. Many of these methods still require a lot of training on top of the large amount needed for regular reinforcement learning. As this increases the number of environment interactions, there are more opportunities for the agent to make mistakes. For this research, we look at a method of creating safe agents in a faster way, namely through curriculum learning. Curriculum learning is based on the concept that, similar to humans, it is much easier for an agent to learn a task by starting from easy examples and progressively making the task more difficult until the target task is reached [Narvekar et al., 2020], see Figure 1.1 for an example. This can be especially useful to decrease the amount of environment interactions needed by many safety optimization techniques. On top of this, a shorter training time necessarily results in less opportunities for the agent to make mistakes during training. Previous attempts at using curriculum learning for safe reinforcement learning have focused on a strict definition of safety, where a state is safe as long as the agent can return to an initial state [Eysenbach et al., 2018, Turchetta et al., 2020]. This definition is limiting as many more forms of unsafe scenarios exist, where some small costs may be tolerated and therefore the initial state cannot be reached after incurring some costs. On top of this, both of these methods use two agents

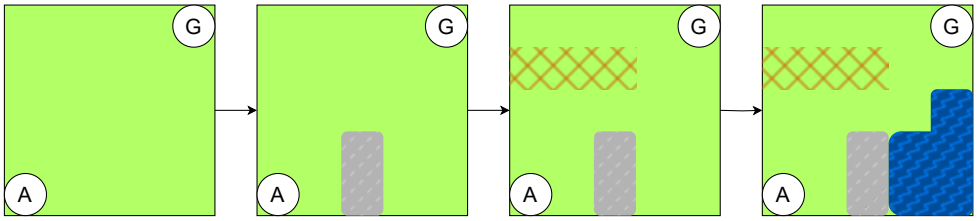


Figure 1.1: An example of a curriculum for an autonomous lawnmower A that needs to navigate to goal G. Ultimately we want the agent to do this in the right-most environment. To make it easier for the agent we first allow it to train in an environment without any obstacles. Then we gradually add more components of the final environment, like gravel, a fence and a pond, until we have reached our target environment.

to learn from each other, which induces a curriculum. However, using two agents to learn only increases the time and data needed to train a safe agent and the safety during training is often not considered.

We want to focus on an aspect that is often overlooked by similar discussions. Namely, in many safe reinforcement learning papers part of, or the whole, training is exempt from considerations of safety. An example is with [Turchetta et al., 2020], where the mistakes made by the students that are trained during the training of the teacher and by the reset controllers given to the teacher are not factored into the definition of safety. As in any real setting, these mistakes are also very important, we want to take the whole training process into account when discussing safety.

Approach. In this thesis, we are going to explore more in the combination of curriculum learning and safe reinforcement learning. The goal is to make a curriculum of tasks with slightly different environments that lead up to a target task where we want to stay within certain safety constraints as much as possible.

Firstly, we are going to look at a technique for creating a curriculum through small changes in the environment that go from a task with few and hard-to-reach hazards to a much harder target task where more hazards are in the way of reaching the goal. Because of this process, the agent should more quickly learn to reach the goal and to do so while avoiding hazards. Through these changes in the environment we can make it easier for the agent to behave safely, further improving the safety during training. As we keep these changes small, it should be easier for the agent to reuse knowledge from previous environments, while still allowing for flexibility in the creation of the curriculum.

Secondly, we want to broaden the notion of safety to go beyond avoiding unrecoverable states. To this end, we use safety constraints, where the agent should learn to reach the goal under these constraints [Altman, 1999]. This means that apart from a reward, actions can be associated with a cost, of which the total should be below a given threshold. This is necessarily an extension of the previously mentioned definition of safety, as when we give a cost greater than the threshold for reaching an unrecoverable state, no feasible solution can go through this state.
To achieve these goals, we take two steps that each address these goals while improving on the previous steps. We consider the scenario where the agent needs to reach a goal, while not moving over too many hazards.

First step. We first build a series of tasks by hand that will be the curriculum used to train the agents. These different tasks have the same observation space, thus the agent can be put on the next task with the same value functions. The only parts that change between tasks are, the starting positions of the agent and the position and number of hazards, where the first tasks have few hazards far away from the goal and the final tasks have more hazards close to the goal. After a fixed number of epochs, the agent will be put in the next task and has to adapt to a harder environment. It is only possible to change the environments in this way if some

assumptions are satisfied. One of the most important ones is that the observations of the agent should be invariant to the number of hazards. This can easily be satisfied by using a camera or, like we do in this project, by using a lidar that looks in all directions and returns an observation representing the distance to the closest hazard. We track both the training process as well as the intermediate evaluations, since we consider safety during training throughout the whole curriculum to be important. We compare some prominent safety algorithms to see which ones are best able to deal with a curriculum. We compare our approach with state of the art in safe RL such as PPOLag [Ray et al., 2019] and CPO [Achiam et al., 2017]. Since our setting is very different from state of the art that combines curriculum learning with safe RL [Eysenbach et al., 2018, Turchetta et al., 2020], we try to compare with them by transferring some of their key aspects, mainly by stopping the agent whenever it is unsafe, into our setting.

Second step. After this, we automate certain parts of our algorithm. This is done by automating the sequencing of the tasks and when it should switch to a different task. The reasoning is that finding the right order of tasks and knowing when to switch to a different task is very difficult. Many orders exist, especially when we can go back to previous tasks and there are even more points in time where a switch of tasks can occur. Even more so because all of this needs to be done before training, meaning that we cannot incorporate the current performance of the agent into the curriculum. This means that it is very likely that we will not find a sequence of tasks close to the optimal in a reasonable time if we do this by hand. Automating task sequencing greatly reduces the effort and time needed to create a curriculum and is much more likely to find better sequences of tasks, especially if sequencing is based on the current performance of the agent. To automate task sequencing we create a starting and target distribution over the tasks, where a distribution will be used by the algorithm to sample a task for the next epoch. The starting distribution will have a high probability for the easier tasks and vice versa for the target distribution. Then the algorithm will use the performance of the agent at each time step to determine when to shift the distribution towards the target distribution. Due to this distribution, the algorithm may decide that it is important to occasionally train on an easier task. When the target distribution is reached we train for some time until the agent has learned for a certain amount of time with this distribution. This is also tested together with some safety algorithms, after which this is also compared to state of the art.

Thus in the end a curriculum designer is created that automates the order of tasks. This algorithm does not need any pre-training or other agent to help train it, giving it an edge in this regard to state of the art. The algorithm adapts the curriculum based on the performance of the agent that is being trained, which results in an agent-specific curriculum. The fact that we create a curriculum through small changes in the environment gives us the freedom to create safer initial environments to train on, while teaching the agent valuable knowledge that it can use in the more unsafe, safety-critical environments. With the algorithm we also allow a more general definition of safety, allowing it to be used in many applications. The result is an algorithm that makes the agent safer during and after training with minimal reward decline compared to a similar agent trained without a curriculum.

1.1 Outline

The structure of the thesis will be as follows. Chapter 2 will discuss what the two components of this thesis are, namely safe RL and curriculum learning. On top of this, we will discuss some related work that has also combined these two components. In Chapter 3 we will formalize the problem and discuss some guiding questions that we will answer in the rest of the thesis. Then in Chapter 4 we will discuss what we will use for our approach and why. Chapter 5 explains how this will be used to create our framework. In Chapter 6 we will show experiments that analyze how well our approach works. Finally, Chapter 7 will give our main conclusions together with some limitations and future work.

Chapter 2

Literature Overview

In this section, we will give a general overview of the different parts that make up the problem domain of this thesis. First, we discuss reinforcement learning and how it is extended to deep reinforcement learning. Then, we discuss how safety can be incorporated into reinforcement learning and we discuss some state of the art that use curriculum learning. Thirdly, we discuss curriculum learning by exploring the components that make up a curriculum learning algorithm. Finally, we discuss some state of the art in more detail that use curriculum learning with safe RL.

2.1 Deep Reinforcement Learning

In reinforcement learning (RL) an agent is trained in some environment [Sutton and Barto, 2018]. When given the current state of the environment, the agent can take actions in this environment to reach different states in the environment. Each combination of state and action results in a reward, which is a numerical indication of how good it was to perform this action in this state. An example would be an autonomous vacuum cleaner that gets a reward for having cleaned a certain part of the house that was dusty. The idea is to reward behavior that we want the agent to have. As the agent does not know the dynamics of the environment, it has to learn this by exploring the environment. It often keeps track of its findings in a policy, which dictates what action should be taken in each state, together with a value function that gives the expected sum of rewards of following this policy. Then the goal of the agent is to optimize some criterion. This criterion is often the expected cumulative reward, also called the expected return, which is the sum of all of the rewards that we expect this agent to get when it follows its policy. The idea is that in this way the agent learns how to operate in the underlying process, more specifically the Markov decision process (MDP), of the environment, which we will introduce formally in Section 3.1.

When we replace one of the components of RL, such as the value function or the policy, with a deep neural network we call it deep reinforcement learning[1] [Li, 2017]. This is mainly where many of the recent successes in RL have come from, such as in robotics [Zhao et al., 2020], large language models [Ouyang et al., 2022] and games such as Go [Silver et al., 2016]. Despite the successes of deep reinforcement learning, it also has multiple challenges [Dulac-Arnold et al., 2021]. Two noteworthy challenges are learning on real-world systems from limited samples, i.e. learning should be done quickly, and reasoning about system constraints that should never or rarely be violated. Each of these challenges is addressed by two methods, which we discuss in the following subsections.

[1] In the rest of the thesis we will refer to deep reinforcement learning as RL, as the distinction between reinforcement learning and its deep version is not relevant for this research and we will only be focusing on deep reinforcement learning.

2.2 Safe Reinforcement Learning

Safe RL aims to solve the challenge for generic RL of reasoning about system constraints that should never or rarely be violated, which we can also interpret as reasoning about safety. For safe RL we want to incorporate safety in some form and when looking at the RL setup, we can logically change two parts to achieve this. We can alter the optimization criterion, for instance by maximizing the worst-case return, or we can alter the exploration process, for example by bootstrapping the learning algorithm with some external knowledge about the environment [Garcıa and Fernández, 2015].

There are a few ways to alter the optimization criterion [Garcıa and Fernández, 2015]. The first is to change it to the worst-case criterion [Tamar et al., 2013]. This means that we change the objective such that the worst return we expect to get is maximized. Another alteration is the risk-sensitive criterion, which adds a parameter to the optimization criterion to control the sensitivity to risk [Basu et al., 2008]. A simple example is when you subtract a numerical value for risk from the optimization criterion, such as the variability of the return. The constrained criterion is another way to change the objective [Kadota et al., 2006, Smith et al., 1997]. However, it does not replace the criterion, but adds constraints, indicating that some measures need to be above or below a threshold, which need to be satisfied when maximizing the return. An example would be to keep the expected return above some threshold. Adding constraints to an MDP results in a constrained MDP that describes the process with the constraints [Altman, 1999].

There are two main ways to alter the exploration process to mitigate risk [Garcıa and Fernández, 2015]. The first is incorporating external knowledge, which can be sub-categorized into three approaches. Initial knowledge can be provided by initializing the learning algorithm through bootstrapping, using this knowledge such that the algorithm does not need to explore possibly unsafe states, but can immediately follow the safe states [Martín H. and de Lope, 2009, Simão et al., 2021]. Another method is providing a set of demonstrations from which a model is learned to derive a policy instead of initializing an algorithm, which is done in the previous approach [Tang et al., 2010, Ramírez et al., 2022]. Lastly, it is possible to provide teacher advice by having a human or controller be able to advise the agent when it is necessary [Walsh et al., 2011, Yang et al., 2023]. Besides incorporating external knowledge it is also possible to incorporate risk-directed exploration by having some risk metric determine the probabilities of selecting different actions during the exploration process [Gehring and Precup, 2013].

For this thesis, we take special interest in the constrained criterion. We have chosen constraints as this is the most intuitive and flexible way to incorporate safety [Roy et al., 2022, Kamran et al., 2022]. When we compare it to the other ways to change the optimization criterion, it is the most expressive one due to the many constraints that can be added. The methods that change the exploration process are less intuitive and exploration is also changed implicitly by curriculum learning. On top of this, as we use curriculum learning, we already use transfer learning to bootstrap the agent for a new environment and the idea is that this already includes some safety guidelines. Thus in this sense, we already use external knowledge from previous tasks to alter the exploration process.

2.2.1 Metrics for safety

It is important to have a way to measure safety, as to be able to determine when something is safe. We look at two metrics that work well in the context of this thesis. Figure 2.1 shows the key difference between these two metrics.

Cost

A very intuitive way to measure safety is to look at the cost incurred. Costs are, similarly to the reward, associated with performing certain actions in certain states that we deem to be costly. This clearly shows how safe the agent is, as we want to minimize the costs. This is regardless of

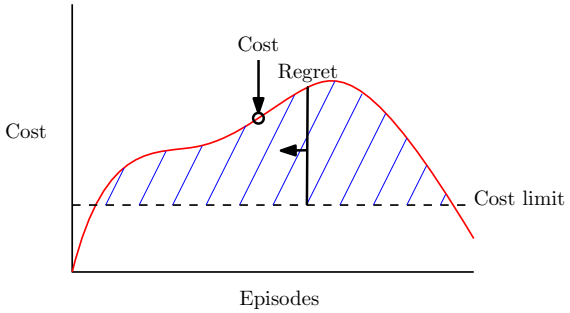


Figure 2.1: Illustration showing the core difference between cost (in red) and regret (in blue). Cost is evaluated at a single point, while regret takes all previous points into account.

return, meaning that it can be safe, while not performing well on the task. Thus cost is a way to evaluate how safe the policy is that has been learned at a certain point.

Regret

A very common metric for the safety of a learning agent is regret, which consists of two metrics: performance regret and constraint violation regret [Efroni et al., 2020]. Performance regret measures the sum of the differences between the optimal reward value function and the reward value function for the policy at each episode. Thus it measures how much reward was missed over the whole course of training. Constraint violation regret measures the sum of differences between the cost value function for the policy at each episode and the cost threshold. Therefore, it measures how much the constraints were violated. Constraint violation regret is then a way to evaluate how safe the agent is learning throughout training.

2.3 Curriculum Learning

In Section 2.1 we mentioned that one challenge of standard RL is that it needs a lot of samples, making the learning process slow. Curriculum learning may be a solution to this problem as it can speed up training [Narvekar et al., 2020].

The intuition behind curriculum learning is that it is easier to learn a task when the new concepts that need to be learned are not that far off from concepts that are already known. For example, when training a robot lawnmower, it is easier to first train it to navigate the grass patch and then train it to also avoid going over the gravel where it may break, then try to do both at the same time. If the robot already knows how to navigate the lawn, it just needs to adjust its path slightly to also avoid the gravel.

A curriculum is a sequence of tasks such that the order accelerates or improves learning and curriculum learning is then finding a curriculum such that the order of tasks is optimized [Narvekar et al., 2020]. The sequence need not be only on different tasks, but can be a sequence on multiple transition samples, i.e. a series of actions of an episode not necessarily generated by the agent itself, of a single task. This basically means that the agent learns from simple examples of how to act in the environment to harder examples, as opposed to learning how to act in simple environments to harder environments. For the thesis, we will focus on a linear ordering of tasks representation. This is because a simple representation that uses transition samples of the same task does not allow for changes in the environment, which is the focus of this thesis. As we also want to keep the curriculum relatively simple, we will be using a linear ordering for the curriculum.

Curriculum learning has three main elements: task generation, sequencing and transfer learning [Narvekar et al., 2020].

- Task generation is important as we need to generate the tasks to use in the sequence, which can also be pre-specified.
- Sequencing is used to find an ordering of these tasks, which also can be specified by a human beforehand.
- Transfer learning assures that the curriculum is useful, as with transfer learning the previously learned knowledge is transferred to the next task.

For this project, we take a special interest in sequencing, as it is the logical part to start automating. The reason for this is that it is much easier to create some tasks by hand than it is to determine both the order and the times at which a change of tasks should occur. On top of this, it is easier to automate sequencing than it is to automate task generation, as the latter requires a lot of domain-specific functionalities and to have different levels of difficulty between tasks, while for the former we only have to find the optimal ordering. Therefore we will be creating our own intermediate and target tasks.

Transfer learning can be complicated when the intermediate tasks are too dissimilar, such as when the observation spaces differ, meaning that a neural network from one task cannot be used for the other. In our case the tasks are similar enough such that we can directly transfer the agent to the new task and thus no special attention to this element is needed.

There are a few categories of task sequencing [Narvekar et al., 2020], namely:

1. *sample sequencing* that reorders transition samples of the target task and does not use any intermediate tasks.
2. *co-learning* that induces a curriculum through interaction between multiple agents.
3. *reward and initial/terminal state distribution changes* that allows some changes for intermediate tasks.
4. *no restrictions* that puts no restrictions on the type of intermediate tasks.
5. *human-in-the-loop curriculum generation* that makes curricula that are partially or fully created by humans.

As for this project we want to make slight changes in the environment, our method of sequencing will fall into category 3.

Automated methods for sequencing have only recently been developed, which create the order of the curriculum without human intervention. However, the definition of automation can be extended to also be adaptive, meaning that the distribution of intermediate tasks is adjusted during training such that it fits better with what the agent has already learned or not [Portelas et al., 2021].

2.4 Combining Curriculum Learning and Safe RL

There is not much research that combines curriculum learning and safe RL, but here we explain two approaches that do so. We also name some limitations and how we plan to address those in our own research.

Eysenbach et al. [2018] perform curriculum induction in the context of safe RL. Two agents are trained, one that learns the task and another that "resets" the former agent. This resetting is done by reversing actions to reach some initial state, which they call an early abort. Sometimes it is not possible to reverse certain actions and then a hard reset is needed, which is done by manually resetting the whole environment. The idea of their algorithm is to reduce the number of such hard resets, which are costly, by using early aborts. The algorithm induces a curriculum for

the resetting agent only, as the better both agents become, the later the early abort will occur. Therefore it learns to recover from more distant states as both agents learn.

A clear limitation of this approach is that a curriculum is only induced for the backward agent. This means that only for tasks where we actually want the resetting agent to have a curriculum, this approach can be used. In the domain that we are interested in, robotic navigation, this is already not possible anymore. We want to have the agent learn to avoid hazards and reach the goal through a curriculum and it is not so obvious how to do this using a resetting agent. An aspect of this approach that is not so much a limitation as it is just a slightly more involved approach to learning, is the fact that they train two agents. Although this works fine, we believe that a simpler method would be to train a single agent, which has the potential to be faster as only half of the number of agents needs to be trained.

Turchetta et al. [2020] have a similar algorithm, as it is also based on the idea of using two agents to learn from each other. However, the resetting agent is replaced by an agent, the teacher, that has access to a collection of resetting agents and learns when to use which of them. The agent that learns the task, the student, uses constraints in its objective, while the teacher learns to make a curriculum of the different resetting agents for the student. The idea is that the teacher starts with giving strict resetting agents that reset the student early and learns when to decrease the level of strictness. To do this, the teacher must be trained on several students such that it can adaptively choose the next resetting agent for any student.

This method has several assumptions that can be improved upon. One assumption they make is that we have access to a set of resetting agents that the teacher agent can choose from. Turchetta et al. [2020] mention that this can be done, for instance, by using the method of Eysenbach et al. [2018], which already has its own limitations. For this algorithm multiple such agents should be trained before the teacher can be trained, meaning that there is a lot of time needed to prepare for the training of the student. However, the assumption that we believe to be the most problematic is that only the costs during training of the final student matter. This means that they exclude the costs made during the training of the many students that were used to train the teacher and during training of all of the resetting agents. We believe that the mistakes made during training of all of these other agents should be accounted for when measuring safety. Thus, we will try to minimize costs throughout the training of our agents and in our case include the costs incurred during the intermediate tasks leading up to the target task.

A major limitation that both approaches have, is their definition of safety. They define any unrecoverable state as unsafe, which is a very restricting definition. Especially if we look at our selected domain, where this definition does not make much sense as we can recover from moving over some of hazards, but it is still costly to move in that way. Therefore we want to use a more general definition of safety, namely that of satisfying a number of constraints. This is an extension of their definition, as we can add the constraint that we cannot go into any unrecoverable states.

Chapter 3

Formalizing the Combination of Safe RL and Curriculum Learning

Within this chapter we will introduce the problem in a formal way. First the mathematical definition of the parts needed to understand the problem are explained, eventually building up to the final problem formulation. Then for this problem, we will list a few research questions that will guide our research to solving the problem in a step-wise manner.

3.1 Formal Definitions

In this section, we will make the problem explicit by giving the mathematical definitions of the different components needed to properly understand the problem we are trying to address.

3.1.1 Standard reinforcement learning

The main aspect of reinforcement learning is that we want the agent to learn how to behave optimally in a Markov decision process (MDP). An MDP in its simplest form is defined as the tuple $\langle S; A; p; r \rangle$ [Puterman, 1994] with:

- S is the state space,
- A is the action space,
- $p : S \times A \times S \to [0, 1]$ is the transition function that gives the probability $p(s' \mid s, a)$ of the next state being s' after taking action a in state s, and
- $r : S \times A \times S \to \mathbb{R}$ is the reward function.

Here, the transition function is stochastic, meaning that an action is chosen randomly with the appropriate probabilities $p(s' \mid s, a)$, but it can also be deterministic, meaning that the action with the highest probability is chosen.

The goal of RL is then to find a policy $\pi : S \to A$, which determines what action should be taken given the current state, out of all policies Π that maximizes the expected return with respect to that policy π. The return is defined as the cumulative discounted reward $R = \sum_{t=0}^{\infty} \gamma^t r_t$, where t is the time step, r_t represents a single real value used to evaluate the selection of an action in a particular state, i.e., the reward, and $\gamma \in [0, 1]$ is the discount factor that allows the influence of future rewards to be controlled [Puterman, 1994]. Thus, the objective of RL is:

$$\max_{\pi \in \Pi} \mathbb{E}_\pi(R) = \max_{\pi \in \Pi} \mathbb{E}_\pi(\sum_{t=0}^{\infty} \gamma^t r_t). \tag{3.1}$$

To approximate R, the value function $V_R^\pi : S \to \mathbb{R}$ is used. The value function is defined as:

$$V_R^\pi(s) = \mathbb{E}_\pi(R \mid s_0 = s), \tag{3.2}$$

where s_0 is the starting state. It shows the expected return when starting from a certain state s, following the policy π until the episode has finished [Sutton and Barto, 2018]. As this function can be approximated by observations, this is what most algorithms optimize, where the value function is taken over all of the starting states. However, the value function does not show which action to take in the current state. When learning the policy this is crucial, which is why the action-value function $Q_R^\pi : S \times A \to \mathbb{R}$ is used. The action-value function is defined as:

$$Q_R^\pi(s, a) = \mathbb{E}_\pi(R \mid s_0 = s, a_0 = a), \tag{3.3}$$

where s_0 is the starting state and a_0 is the action taken in the starting state. It shows the expected return when starting from a state s, performing action a and then following policy π until the episode is finished. In practice the action-value function and value function are estimated through trial and error by the agent, as it does not know the dynamics of the environment.

3.1.2 Constrained reinforcement learning

When moving from regular RL to safe RL, we introduce constraints to an MDP. A constrained MDP Altman [1999] is defined as $\langle S; A; p; r; c; \alpha \rangle$ with S, A, p and r the same as before, $c : S \times A \times S \to \mathbb{R}$ the cost function and α is the cost limit, where we adapted the notation from Garcıa and Fernández [2015]. The cost function is similar to the reward function in that it associates an action taken in a state together with the state reached with a real value. Moreover, there is also the notion of cumulative discounted cost $C = \sum_{t=0}^{\infty} \gamma^t c_t$, where c_t is the cost at time step t. The constrained criterion is then:

$$\max_{\pi \in \Pi} \mathbb{E}_\pi(R) \text{ subject to } \mathbb{E}_\pi(C) \leq \alpha. \tag{3.4}$$

There are other ways to choose the metrics that needs to be below the limit [Wachi et al., 2024], but we choose the expected cumulative cost because it is the simplest and most widely used. Similarly to the return, this is approximated through the cost value function V_C^π, which is the expected cumulative discounted cost following policy π starting from the starting states.

To measure safety within a constrained MDP, we mainly use an adapted definition of constraint violation regret from Efroni et al. [2020]. This is useful as it ignores the cost if it is below the allowed limit. Usually, it does not matter how much the cost is if it is below the limit, which means that this metric works better than simply taking the total cost incurred [Müller et al., 2024]. The constraint violation regret, which we will shorten to regret, is defined as:

$$\text{Reg}(K, c, \alpha) = \sum_{k=1}^{K} [V_c^{\pi_k}(s_0) - \alpha]_+, \tag{3.5}$$

where K is the number of episodes, π_k is the policy at episode k, s_0 is the starting state and $[x]_+ = \max(x, 0)$.

3.1.3 Curriculum learning

As we are combining safe RL with curriculum learning, we also need to address what makes a curriculum. A curriculum is defined as a directed acyclic graph $\langle \mathcal{V}, \mathcal{E}, \mathcal{T}, g \rangle$, where we adapted the definition from Narvekar et al. [2020]:

- $\mathcal{V}$ is the set of vertices,
- $\mathcal{E}$ is the set of directed edges $\{(v_i, v_{i+1}) \mid (v_i, v_{i+1}) \in \mathcal{V} \times \mathcal{V}\}$,

- $\mathcal{T}$ is the set of tasks $m_i = \langle S; A; p_i; r_i; c_i; \alpha \rangle$, where a task is thus a constrained MDP, and
- $g : \mathcal{V} \to \mathcal{T}$ is a function that associates each vertex with a task.

In this graph, before we can train on the task of node v_{i+1}, we need to train on the task of node v_i, since we use a linear ordering. There also is a single sink node v_t that contains the target task. On top of this in our setting the state space and action space should stay the same across all nodes. This is because we want to limit the changes in the environment to allow for transfer learning to work by using the same neural network.

It is important to note that multiple vertices can be associated to the same task, which allows the curriculum to revisit tasks. This must be the case as when only one vertex can be associated to a task, this would create a loop in the curriculum, making it ambiguous when this loop should be exited.

3.2 Problem statement

Using the definitions from the previous sections, the goal of our research is the following. We want to build a curriculum $\langle \mathcal{V}, \mathcal{E}, \mathcal{T}, g \rangle$ that helps an agent solve constrained MDP m_t according to Criterion 3.4, associated with vertex v_t, while minimizing the regret $\text{Reg}(K, c_i, \alpha)$ at each vertex.

To address the previously defined problem, we must address some questions that will guide the research. The goal is to use curriculum learning to have faster and safer training as opposed to not using a curriculum. The idea is to build towards a framework that has an automated curriculum designer that adaptively chooses the next task without being pretrained such that the agent being trained does so safely. To achieve this goal in this problem formulation we want to answer two main questions, where each question builds on the previous ones. Each question also has subquestions that help to answer its respective main question. The research questions are the following:

- **RQ1: Does curriculum learning through changes in the environment improve the safety of a safe RL agent?** Curriculum learning is known to speed up training, but it is still not well-researched whether this also improves safety in safe RL. Faster training results in fewer possibilities to make mistakes, but also may result in more mistakes being made due to the changing environments.
  - **RQ1a: Which safe RL algorithms work best with curriculum learning?** The particular safe RL algorithm already has impact on training without curriculum learning, thus it is crucial to investigate which ones are most suitable for curriculum learning.
  - **RQ1b: What are proper curriculum learning techniques for the context of safe RL?** As there are many techniques to implement curriculum learning, even in the subdomain of only changing some aspects of the environment, it is necessary to look at which ones are most suitable for safe RL.
  - **RQ1c: How does a framework that uses curriculum learning through changes in the environment compare to state of the art in terms of safety?** Apart from comparing to some baselines, comparing to the state of the art in safe RL is needed to conclude whether the proposed method is beneficial for safety. It is also interesting to see whether a curriculum allows the agent to learn to be safe in environments where it may be too difficult for a regular agent to learn to be safe.
- **RQ2: Can we create a curriculum learning algorithm with automated task sequencing that is safe?** Identifying a good curriculum by hand gives a lot of overhead and may be difficult to do. Therefore, making a curriculum automatically, reduces the effort needed to create an agent with curriculum learning. It also has the potential to be safer and faster if the curriculum is made adaptively.

- **RQ2a: What techniques exist that allow for automated task sequencing for curriculum learning in safe RL?** Apart from optimization algorithms, algorithms for automating sequencing exist. It is important to look at which of these algorithms align with our problem domain and assumptions.
- **RQ2b: Does a curriculum learning algorithm with automated task sequencing improve on safety compared to the static curriculum learning algorithm?** The main reasoning behind this extension on the static curriculum is that an adaptive sequencing method should increase the safety of the agent and thus this should be tested.

The rest of this thesis addresses these questions. In Chapter 4 we are going to address **RQ1a**, **RQ1b** and **RQ2a** by looking at important properties of different safe RL algorithms and different curriculum learning methods. Then in Chapter 5 we tackle one side of **RQ1c** and **RQ2b** by explaining how we implement our curriculum learning frameworks. Finally, in Chapter 6 we attend to all questions from an experimental point of view, with the main focus on **RQ1c** and **RQ2b** as we compare our framework with several state-of-the-art algorithms.

Chapter 4

Combining Curriculum Learning and Constrained RL

In this chapter, we will go deeper into the different components of the problem, namely safe RL and curriculum learning. Specific algorithms will be discussed with the end goal to answer the research questions **RQ1a** and **RQ1b**. In the end, we will conclude which algorithms we are going to use during our experiments, based on these answers.

4.1 Safe RL Algorithms

There are already many algorithms for RL that each have their own way of solving the RL objective of maximizing reward, like Q-learning [Watkins and Dayan, 1992], PPO [Schulman et al., 2017] and A3C [Mnih et al., 2016]. However, for safe RL we have changed the objective to maximize the reward subject to the constraints. Therefore we need specialized algorithms that can deal with constraints in the objective. To do this we will talk about a few types of algorithms that are specialized for constrained objectives and judge how suitable they are for curriculum learning.

Before we dive into specific algorithms, we must identify what makes a good safe RL algorithm when we are using curriculum learning:

Adapts quickly An important property is that the algorithm can adapt quickly when a new task of the curriculum is started. As then the environment has changed and previously unseen observations can occur. This means that a potential drop in performance due to these new observations is quickly addressed.

Generalizes well For the same reason good generalizability is also needed such that the drop in performance is small by keeping the information learned in previous tasks and applying it to new scenarios.

Converges quickly Another property that we think is crucial is fast convergence, because in a curriculum we do not want to have to train for too long on tasks that are not the target task such that we can get to the target task the fastest.

As such properties are quite difficult to fully satisfy, we would prefer the algorithm to be more careful, rather than more exploratory.

We will not be discussing the algorithms in detail as the thesis focuses mainly on the design of the curriculum, which apart from the previously discussed properties is agnostic to these details. We will also be looking only at on-policy algorithms. This is the case as we want to focus on similar algorithms and we do not foresee any large differences when looking at other types such as off-policy algorithms. A slight benefit to on-policy methods is that in general, they are more stable [Schulman et al., 2017]. Since a change in environments can lead to a drop in performance, being slightly more stable may help with limiting the drop.

4.1.1 Lagrangian approaches

One of the most common ways to deal with constraints in the objective is to use a Lagrangian approach. It works by adding the constraints to the objective as some sort of penalty [Altman, 1998]. Then the constraints are multiplied by the Lagrangian multiplier, λ, which acts as a penalty coefficient such that the optimal policy is one that satisfies the constraints [Kadota et al., 2006]. This transformation can be seen in Equation 4.1, where Criterion 3.4 is transformed into its dual, using the Lagrangian approach:

$$\max_{\pi \in \Pi} \mathbb{E}_\pi(R) \text{ subject to } \mathbb{E}_\pi(C) \leq \alpha \xrightarrow{\text{dual}} \max_{\pi \in \Pi} \min_{\lambda \geq 0} \mathbb{E}_\pi(R) + \lambda \cdot (\mathbb{E}_\pi(C) - \alpha). \tag{4.1}$$

Lagrangian methods can easily be applied to a non-safe RL algorithm, like PPO [Schulman et al., 2017] or TRPO [Schulman et al., 2015], which are then called PPOLag and TRPOLag respectively [Ray et al., 2019]. In this way, we get an algorithm that finds the optimal policy in almost the same way as the non-safe algorithm, but also keeps track of a Lagrangian multiplier and an adjusted objective. This allows the algorithm to consider safety such that it can eventually converge to a constraint-satisfying algorithm.

Constrained-controlled PPO (CPPO) is an extension of PPOLag that addresses a large drawback of the standard Lagrangian methods [Stooke et al., 2020]. These methods are very unstable, meaning that the Lagrangian multiplier and therefore the cost can oscillate [Platt and Barr, 1987]. The extension of the Lagrangian multiplier is called PID, proportional, integral and derivative. The main idea is that adding the constraint function, the proportional part, and its derivative to the Lagrangian multiplier gives the necessary information to dampen the oscillations. Therefore the Lagrangian multiplier oscillates less and the constraint function even less.

Constrained Update Projection (CUP) is a Lagrangian approach based on more precise surrogate functions for the return and cost that better approximate the value functions [Yang et al., 2022]. This can make it easier for the algorithm to have a somewhat monotonic increase in return, while staying under the cost limit. From their experiments, the biggest improvement compared to other algorithms is the speed at which the agents reach high return, while also converging reasonably fast to an expected cost under the cost limit.

Reward Constrained Policy Optimization (RCPO) is based on the idea that giving penalties for costly behavior should be adaptive [Tessler et al., 2018]. Instead of having the Lagrangian multiplier in the objective, this algorithm changes the reward function to give a penalty proportional to the cost incurred with the Lagrangian multiplier as the coefficient. This results in the value function of the critic reflecting this penalized reward and thus results in slightly different calculations for the gradient than we would have for PPOLag.

First Order Constrained Optimization in Policy Space (FOCOPS) has two stages in a single update [Zhang et al., 2020]. The first stage is to update the policy to a policy that is in the non-parameterized space, meaning that it does not rely on the size and weights of the neural networks. Then this policy is projected back into the parameterized space with a policy that is as close as possible to the non-parameterized one. The idea is that solving the problem in the non-parameterized space is easier, as it is almost a closed-form expression. The optimization is then done using the Lagrangian method.

4.1.2 Penalty function approaches

A similar transformation to the Lagrangian approach is proposed by Smith et al. [1997], namely penalty functions. Here the transformation is much simpler, where we add a function over the distance between the solution and the constraints, to the objective that we want to minimize. Examples of distance metrics for constraints are the number of constraints violated or the sum over each of the violations. A simple example of a penalty function transformation is:

$$\max_{\pi \in \Pi} \mathbb{E}_\pi(R) \text{ subject to } \mathbb{E}_\pi(C) \leq \alpha \xrightarrow{\text{penalty function}} \max_{\pi \in \Pi} \mathbb{E}_\pi(R) - I_{(\mathbb{E}_\pi(C) > \alpha)}, \tag{4.2}$$

where $I_{(\mathbb{E}_\pi(C)>\alpha)}$ is the indicator function indicating whether the constraint has been violated and is in this case also the distance metric. Different types of penalty functions exist, namely methods where no violations are accepted, methods where violations close to the boundary are slightly penalized and may be accepted and methods where all violations may be accepted, although they are still penalized.

Penalized Proximal Policy Optimization (P3O) instead uses penalty functions [Zhang et al., 2022]. In this case, the distance metric is a simple ReLU. To avoid making catastrophic updates when learning, they clip the gradients, such that they cannot be too large. This results in small iterative improvements to the policy that make sure that we have steady improvements with a very low chance of ending up in a catastrophic policy. Their method also uses a hyperparameter which is the coefficient for the distance metric and thus is similar to the Lagrangian multiplier. They show that this hyperparameter is very stable, meaning that different values do not change the performance much in both return and cost.

Interior-Point Policy Optimization (IPO) is another example that uses a penalty function and is a simple extension of PPO [Liu et al., 2020]. The only change is that a barrier function is added, which approximates an indicator function, indicating whether the constraints have been violated. The idea is that when maximizing the objective, a violation of the constraint should result in an additional infinite negative reward and when there is no violation there should be no negative additional reward. To do this the logarithm of the negative cost is used such that it approximates this idea and is still differential. However, to optimize under constraints IPO also uses a Lagrangian multiplier.

4.1.3 Trust region approaches

One of the first algorithms that optimize RL objectives under constraints is called Constrained Policy Optimization (CPO) [Achiam et al., 2017]. This method uses trust regions, which is a method of optimization not specific to safe RL that looks at a region of solutions around the current solution and chooses the best to be the next solution [Schulman et al., 2015]. This is different from gradient descent, as gradient descent does not pay attention to where the next solution ends up. As the next solution could be catastrophic, trust regions are a safer option. Achiam et al. [2017] show that their algorithm makes the expected cost quickly approximate the cost limit, but does not necessarily stay below it. They compare this to an algorithm that uses a Lagrangian approach, which often has a higher return and eventually stays below the threshold. However, this algorithm results in very unstable updates before staying below the threshold.

Projection-Based Constrained Policy Optimization (PCPO) is based on the idea that policy updates should be done in two stages [Yang et al., 2020]. The first stage optimizes the objective without constraints, while staying within the trust region. The second stage is to project the solution of the previous stage to the closest solution that satisfies the constraints. This means that at every update step, the policy should be very close to a safe policy. The distance is measured in either KL-divergence or L2-norm, where the KL-divergence results in very stable updates and the L2-norm in many cases has higher rewards and constraint satisfaction, but can be very unstable. They use approximations to make the calculations of both stages feasible for larger neural networks. For these approximations to be accurate, the trust region of the first step needs to be very small, in turn making updates much slower.

4.1.4 Heuristic approaches

Constrained-Rectified Policy Optimization (CRPO) is made to be an algorithm that does not use any dual variable, while competing with algorithms that do use them [Xu et al., 2021]. Lagrangian methods and similar ones add other variables to the optimization objective. However, a major drawback for using Lagrangian methods is that the algorithms are very sensitive to the Lagrangian hyperparameters [Chow et al., 2019]. As CRPO does not have any dual variables. it does not have these issues and gives good safety guarantees, making it much easier to use. It works by using approximations of the cost functions. If any of these approximations violates a constraint, then an

optimization step is taken to minimize the constraint. If no approximation violates a constraint, then an optimization step is taken to maximize the return. This means that optimization is prioritized for constraint satisfaction and when the safety constraints are met, the return can be optimized. Therefore, CRPO can respond much quicker to constraint violations, compared to Lagrangian methods that need a lot of iterations before the Lagrangian multiplier is large enough.

Early terminated algorithms are based on the very simple concept that an episode ends whenever the cost goes above the cost limit [Sun et al., 2021]. This means that during training the cost per episode usually remains around the cost limit. In certain environments, the agent may only need a small subset of the states to operate effectively [Ainsworth et al., 2019], thus limiting it to safe states can be useful. The method is very similar to the reset controllers from Eysenbach et al. [2018], where in the case of the early terminated algorithms the "reset controller" returns the agent to an initial position right after making a mistake instead of just before. These are combined with general RL algorithms, similar to the standard Lagrangian approach, to form PPOEarlyTerminated or TRPOEarlyTerminated.

4.2 Curriculum Learning

Similarly to safe RL algorithms, there is a wide range of curriculum learning approaches. To properly discuss these methods this section will be split up into four levels of automation, in an order that reflects the trend from previous works [Narvekar et al., 2020]. A diagram summarizing these four categories can be found in Figure 4.1. The first category will be about fully hand-made curricula, by which we mean that task generation and task sequencing are done by hand and that sequencing is not adaptive. The following sections progressively increase the amount of automation of the curricula. First, we discuss automating task sequencing, while still generating tasks by hand and without making the curriculum adaptive. Next, we discuss methods that extend the task sequencing to also be adaptive, meaning that it adapts the curriculum based on the performance of the agent. Finally, we describe possible ways to create a fully automatic curriculum designer, where besides an automatic and adaptive sequencing algorithm there is an algorithm for generating tasks.

4.2.1 Hand-made curriculum generation

Creating a curriculum by hand entails creating both the tasks and the ordering in which those tasks are presented to the agent. The moment at which a task change should occur is then also included in the curriculum. All of these steps are taken before training the agent.

A psychology-focused study was done on curricula designed by humans for a classification task [Khan et al., 2011]. This study found that one of the ways humans employ teaching strategies in this scenario is what the writers call the extreme strategy, where they first show easy objects far away from the decision boundary and progressively move to harder objects. This implies that curriculum learning is already quite intuitive to humans when trying to teach a difficult task.

A user study in the realm of curriculum learning in RL has shown that even non-expert humans can create good curricula [Peng et al., 2018]. This study gives several interesting findings of human-made curricula. Four principles that are applied by human curriculum designers are discussed, namely (1) isolating complexity, (2) selecting the simplest environments to introduce one complexity at a time, (3) choosing environments that are most similar to the target environment, and (4) introducing complexity by building on previous tasks rather than backtracking to introduce new types of complexity. These human-made curricula are shown to learn faster on the target task compared to learning from scratch, although the effort it takes to create the curriculum and to train on it can be more than starting on the target task directly.

There have also been attempts at human-made curricula with a graph structure instead of a sequence [MacAlpine and Stone, 2018]. In this approach a curriculum is created where there can be multiple source tasks for an intermediate task. The idea is that a complex task can be split up into multiple tasks that may be independent of each other. This allows for parallel learning

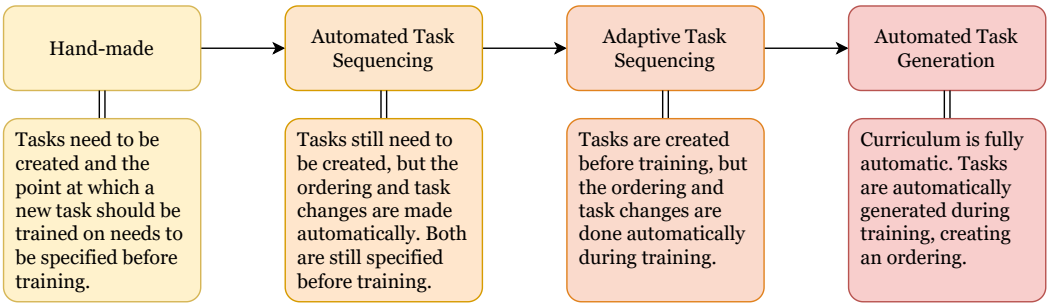


Figure 4.1: The different levels of automation when designing a curriculum.

of tasks, where their knowledge can then be combined to more easily solve a complex task. An example from MacAlpine and Stone [2018] is that a soccer robot can separately learn to kick the ball and to get up from a lying position. This is eventually combined into the behavior that the robot can kick the ball, fall down and get back up. A benefit of doing it in this way it is easier to combine the information, while in a sequence it could be the case that the information of the first task is overwritten by the second task.

4.2.2 Automated task sequencing

Creating the order in which tasks appear in the curriculum is called task sequencing. When it is not adaptive, this means that this ordering, together with the moments at which a new task should be trained on, is determined before training the agents, thus being independent of the behavior of the agent.

One study looked at the different ways to find the best sequence of a curriculum, given a set of tasks and the maximum length of the curriculum. To decide how task sequencing should be done, Foglino et al. [2019b] look at three scenarios: critical tasks, complex tasks and tasks that should be learned quickly. Here the first scenario is most related to this research, for which the metrics by which we determine how good a curriculum is are jumpstart and performance regret. The jumpstart metric measures the average performance in the first few episodes on the final task. Both metrics take into account that we want to explore the environment efficiently. Even though both metrics measure performance, they can also be used for the cost [Yang et al., 2023]. The authors show that for these metrics the most promising algorithms for task sequencing were Tabu Search and Beam Search. Tabu Search iteratively takes the best performing curriculum and evaluates all similar curricula. Similar, in this case, means that the other curricula can have only one task more or fewer and can swap only two tasks in the ordering. Beam Search takes a few of the best performing curricula and creates new curricula by appending a task to each of the curricula, via a Cartesian product with a set of tasks. However, curricula with repeating tasks are discarded. This means that both algorithms evaluate many curricula at each step by training agents on all of these curricula.

Besides general search algorithms, it is also possible to look at algorithms for specific problems, like for critical tasks. Foglino et al. [2019a] propose an algorithm that tries to limit suboptimal actions during the learning of a target task. Their algorithm first evaluates every pair of intermediate tasks, after which they iteratively add more tasks to the most promising pairs first and eventually to the less prominent pairs. Similar to the previously discussed paper by Foglino et al. [2019b], this results in many curricula needing to be evaluated as it eventually exhaustively searches the space of all curricula up to a maximum length. In their experiments, this algorithm can find the optimal curriculum faster than Tabu Search, as it needs to evaluate fewer curricula before the optimal is found.

Another way to automate sequencing, which is more similar to the generic search algorithms, is by using an integer linear program. Foglino et al. [2020] want to minimize performance regret

by solving a scheduling problem using linear programs. The goal is to find the right parameters such that the solution to the scheduling problem is a curriculum that minimizes the regret. The linear program needs good approximations of the "penalty" of not performing some preparatory task m_i before task m_j and of the performance regret that each of the tasks contributes. Then computing such estimates is the next problem, which requires computations of the regret for each curriculum of two tasks. Even though a linear program also searches for the best sequence, it does so for all tasks at the same time, unlike the search algorithms which build up a sequence by iteratively adding tasks.

4.2.3 Adaptive task sequencing

When making the sequencing adaptive, the behavior of the agent during training is taken into account when determining both the next task as well as the time at which this next task should be presented to the agent. This means that the curriculum depends on the performance of the agent so far and that the curriculum is much more flexible, even allowing revisiting previous tasks if needed.

An early example of adaptive task sequencing in curriculum learning tries to teach a vision-based robot to score a ball into a goal. Asada et al. [1996] use curriculum learning, here called learning from easy missions, to create some tasks by hand. They identify substates that together make up a state in the MDP, such as the size of the goal from the image and size of the ball, which are discretized into for example "small", "medium" and "large". Thus a state in the MDP could be a "small" ball with a "medium" goal. The curriculum starts with training on a set of states that all share one value of a particular substate. Thus for example all states, where the goal size is "large", regardless of the values of the other substates. Then the sequencing part is automated in some sense, despite the fact that the sequencing has the ordering implied by the discretized substates, by automating when a shift should be made to the next value of the particular substate. This is done adaptively by checking if the absolute difference between the Q-value of this time step and the previous time step is smaller than some threshold, thus checking if the agent has converged. If this is the case, the agent has learned enough on this set of states and we move to the next set of states in this substate, for example by including all states where the goal size is "medium", making the goal smaller in the images the agent receives. The algorithm only shifts between sets of states determined by one specific substate and it does not allow for shifting on multiple substates.

Another example of a vision-based model with adaptive sequencing is one that learns to play the video game Doom [Wu and Tian, 2016]. Similarly to Asada et al. [1996], this study uses hand-made tasks for the curriculum by varying some parameters that increasingly make the levels more difficult, such as enemy health, and by changing the reward function. The sequencing is made adaptively by creating a distribution over the tasks in the curriculum and when some performance metric of the agent reaches a threshold on the current distribution, they shift the distribution towards the more difficult levels. This also means that the sequencing is done automatically, because the order of tasks is not pre-defined as the distribution determines the next task.

It may also be beneficial to look at the curriculum design process as an MDP [Narvekar and Stone, 2019]. In this way, the states are all the policies that the agent can represent and actions are intermediate tasks that the agent can train on. Thus a new state would be all the policies after training on the task specified by the action. The reward in this case is defined as the negative of the time it takes to learn the task. This MDP is then called a curriculum MDP. After learning the curriculum MDP for a specific agent, the agent learns faster than without a curriculum and faster than some other curriculum learning methods. However, the main drawback is that optimizing this curriculum MDP is very expensive and may even take longer than learning the target task from scratch. Narvekar and Stone [2019] show that learning on intermediate tasks does not need to be done until some threshold is reached. Instead training for only a few episodes on an intermediate task can result in faster learning overall.

4.2.4 Automated task generation

A fully automated curriculum designer needs to generate its own tasks. Similarly to adaptive task generation, this is also tailored to the performance of the agent, making it even more flexible. This is also the most difficult part to do by hand, thus automating this relieves humans of putting a lot of time and effort into creating tasks.

One way to automate task generation is by using heuristics. Narvekar et al. [2016] mention several heuristics that take a task and a set of samples on that task as input, together with some potential parameters of the specific heuristic. Their method works by first training on the target task and after reaching some suboptimal policy, some samples using this policy are created. Then some heuristic, for which we will discuss a few options below, creates an intermediate task such that when the agent can perform well on this task, it will perform better on the target task than was done previously. However, this intermediate task may also result in a suboptimal policy and thus the same procedure is done to create an intermediate task that should be used before the other. This whole procedure thus builds a curriculum starting from the target task, such that essentially the agent does a backward and forward pass through this curriculum. It is also possible to compose heuristics to create a curriculum with tasks focusing on different aspects. Some of the heuristics that stand out are:

- "Promising Initializations", it takes a subset of all samples such that each sample has a reward within a percentile of the highest rewards. Then states near the origin state of the sample transition are added to a new task as initial states. In this way, the new task is only different in the starting states of the original task, such that the new starting states are closer to transitions that give a reward.

- "Action Simplification", its main goal is to reduce the set of actions such that mistakes are less likely. This method looks at all of the transitions that were mistakes and if an action occurred more often than a threshold, this action is removed for the next intermediate task. This is mainly useful for tasks with many actions that are correlated to mistakes or when used together with another heuristic that creates such a task.

- "Mistake Learning", this method also looks at mistakes. However, it does so by focusing on the mistakes as it changes the initial states for the new task to be some steps before the mistake was made. The idea is that in this way the agent is exposed more quickly, allowing it to correct itself faster.

- "Option Sub-goals", it tries to make a new task such that this is achieved before achieving the goal in the target task. For example, going to a state that has a high value with the value function. This teaches the agent to go towards the end goal in steps. To do this the new task has terminal states equal to such high-value states and a reward function that rewards this value to the agent upon reaching this sub-goal.

- "Link Subtask", it creates the link between two tasks. It starts from one of the given tasks, where its terminal states are changed to the initial states of the other given task, such that the first given task ends as soon as the second given task begins. Also, the reward function is changed to give a reward upon reaching the terminal states, which are the starting states of the second given task.

It is also possible to combine these heuristics with the concept of a curriculum MDP [Narvekar et al., 2017]. This is a very straightforward combination, where the algorithm for creating a curriculum MDP is changed such that the space of tasks is determined by the heuristics. However, as this paper precedes Narvekar and Stone [2019] that use a curriculum MDP, there are some changes between the algorithms. The biggest change is that this algorithm computes only a trajectory sample of the curriculum MDP, as opposed to the policy for this MDP. This would, however, likely result in a less expensive computation before training the agent. On the other hand, this would also not necessarily give the most optimal curriculum.

We can use a generative adversarial network (GAN) to separately learn to create tasks of increasing difficulty and learn to solve these tasks. Florensa et al. [2018] train a GAN together with the agent such that the generator of the GAN creates tasks that are in a range of difficulty that is appropriate for the agent. To do this, at each iteration, the GAN is used to return a generator and discriminator. The generator is used to create a set of tasks for the agent and the discriminator is used to evaluate whether the tasks are of an appropriate level. Then the agent is trained on all of these tasks, which gives a return for each task. These returns are used to determine if the goals were of appropriate level using a binary variable, which are given to the GAN to train with. The tasks generated by the generator all use a binary reward function that gives a reward of 1 upon reaching the goal state. The reward function is parameterized by the task, which determines the goal states. Therefore, the algorithm forces the agent to learn several tasks at the same time and thus may also have several target goals. All of this means that this algorithm is very much related to contextual MDPs.

Klink et al. [2020, 2021] used self-paced learning as a curriculum designer for RL. In this implementation automated curriculum generation is used. It is done through self-paced learning, where the idea is that we want to base the level of difficulty not on some objective measure, but determined by the abilities of the agent [Kumar et al., 2010]. Therefore we learn which tasks are hard and when the agent should be exposed to more difficult tasks. This is reflected by the fact that there is a distribution of tasks, which slowly shifts the probability mass towards higher probabilities for the more difficult tasks until a target distribution is reached. However, the algorithm needs to be given a starting distribution of tasks and an initial indication of the level of difficulty of the tasks.

Eimer et al. [2021] have a similar approach, which builds on the self-paced framework. It also uses automated curriculum learning, however, the algorithm does not need an indication of where the agents should start training nor which tasks are more difficult than others. Instead, an indication is calculated for a task of how much performance there is to gain from training on that task. This metric is used to select the next tasks in the curriculum.

4.3 Method Selection

In this section, we will discuss our answers to **RQ1a** and **RQ1b**. To make the research feasible we have to make some decisions regarding the options we want to use. This means that we have to take a subset of the algorithms that we will experiment with and we will choose at most one curriculum learning method per level of automation.

4.3.1 Algorithms

When deciding which safe RL algorithms to use to answer **RQ1a**, we must look for the properties discussed in Section 4.1 that are important when learning with a curriculum.

In Table 4.1 a summary is given to show which algorithms have which of the properties we look for, which we justify in this section. None of the algorithms have all the properties that we want, but there are still candidates that should perform well.

CUP, PCPO, CRPO all have two of the properties. CUP has shown to converge fast and there is no reason to think that it does not generalize well [Yang et al., 2022], but as it uses the Lagrangian multiplier we do not expect it to adapt quickly. PCPO has the issue that it uses trust regions, which makes generalizability more difficult as in a new environment the solution could be far outside of the trust region. However, it converges relatively quickly and should be adaptable since it can make quick adjustments based on the current performance of the agent [Yang et al., 2020]. CRPO is likely not to generalize well as it relies on approximations of the cost functions [Xu et al., 2021], which can differ significantly between environments. Nonetheless, it does prioritize safety above all, which should allow it to quickly adapt and it was shown by the authors to converge quickly.

Type	Algorithm	Adapts quickly	Generalizes well	Converges quickly
Lagrangian	PPOLag	No	Yes	No
	CPPO			No
	CUP			Yes
	RCPO			No
	FOCOPS			No
Penalty functions	P3O	No	Yes	No
	IPO			
Trust regions	CPO	No	No	No
	PCPO	Yes		Yes
Heuristic	CRPO	Yes	No	Yes
	PPOEarlyTerminated	No		No

Table 4.1: Table showing which algorithms have the properties important for combining safe RL with curriculum learning. All of the values are relative to the others, especially generalization as we have some algorithms that are expected to generalize poorly, while for the others we have no reason to believe they do not generalize well.

As the remaining Lagrangian approaches are not expected to quickly adapt to new tasks, due to the Lagrangian multiplier having slow updates, and have not shown particularly fast convergence, they do not immediately seem interesting to test. However, PPOLag and FOCOPS are still interesting choices. PPOLag is used as comparison material for most of the other algorithms and is thus an important algorithm to also test. FOCOPS has potential since it has more generalizability than the other methods. FOCOPS was tested on generalizability and was shown by Zhang et al. [2020] to do much better than PPOLag.

The penalty functions use the Lagrangian multiplier, or something similar, to optimize, indicating that they are not likely to adapt quickly nor have they shown fast convergence. Therefore, we will not be using this type of algorithms with a curriculum.

CPO does not have any of the properties, due to trust regions making generalizability more difficult and having no clear indications that it adapts or converges quickly. Nevertheless, CPO is also a popular algorithm to compare with, thus we will use it without a curriculum as a state-of-the-art in safe RL.

Similarly to CPO, PPOEarlyTerminated does not generalize well as it learns only on states specific to the current environment and has no indications of having the other properties. Regardless, we will also include PPOEarlyTerminated both with and without a curriculum. The reason is that it works similarly to the reset controllers from Eysenbach et al. [2018] and Turchetta et al. [2020] in that they all train the agent by resetting it as soon as it performs an unsafe action. Also, in the literature, the reset controllers are trained together with the agent, but in the case of PPOEarlyTerminated, there is an oracle that stops the agent when the cost is too high. So in that sense, PPOEarlyTerminated is a single-agent version of the reset controller framework, where the reset controller itself is replaced by its best version, an oracle. In this way, we can in some form compare with state-of-the-art in combining safe RL with curriculum learning. Nonetheless, it should be noted that Eysenbach et al. [2018] and Turchetta et al. [2020] create their curricula in a very different way compared to ours. Where we create different environments, they use other agents that influence the way the primary agent can navigate the single environment.

4.3.2 Curriculum learning

Now we answer **RQ1b** based on the previously discussed theory. As we cannot test all of the methods we have to choose the ones that seem to be the best options for our problem. Therefore we will compare the methods per level of automation and decide which to use.

For the human-made curriculum generation methods, two of them are hard to apply to our problem. There is not a clear interpretation of a decision boundary in this case as used by Khan et al. [2011]. As we want to keep the task relatively simple, there is no need for a non-linear graph structure as presented by MacAlpine and Stone [2018]. Thus we will follow the guidelines presented by Peng et al. [2018].

When looking at the automation of task sequencing no method stands out. The most promising one by Foglino et al. [2019a] builds a curriculum iteratively by adding tasks to the most promising sequences. However, this algorithm clashes with the fact that we want to limit the amount of precomputations needed. This is also the case for the methods by Foglino et al. [2019b] and Foglino et al. [2020]. As these methods perform a search over the available tasks, they would require a lot of tasks to be designed such that there is a non-trivial sequence. Because of these issues, we decided to not implement any method that only automates task sequencing.

Adaptive curriculum generation has many interesting methods. The curriculum MDP by Narvekar and Stone [2019] appears to be a good option, however, this again has a lot of overhead before the actual training starts. The method by Asada et al. [1996] that uses substates is promising, as it is a simple way to add adaptivity to the curriculum. One disadvantage here is that the method rests on the idea that a task should be almost fully learned before moving on to the next one, while Narvekar and Stone [2019] show that this is not necessary. There is also the method by Wu and Tian [2016] that uses a probability distribution over the tasks. This does not force the agent to keep learning on tasks until it is good enough, but allows it to occasionally experience harder tasks or revisit previous tasks if necessary. Due to the simplicity and flexibility of this method, it appears to be the best option for this category. However, as we see a lot of overlap with Asada et al. [1996], we want to focus on first creating a simpler framework that incorporates the similarities such that it can be extended to either of these options, which we discuss in more depth in Section 5.2.

Implementing full automation is outside the context of this thesis. However, we do think that the best candidate is the method by Klink et al. [2020, 2021], as this method seems to be relatively close to our approach for adaptive sequencing. It would require minor adjustments to the adaptive method and to implement the task generation we could use some variables that can be sampled and from which a task can be created. This is an application of contextual MDPs, where these variables are called the context [Hallak et al., 2015].

Chapter 5

Automating Curriculum Learning in Safe RL

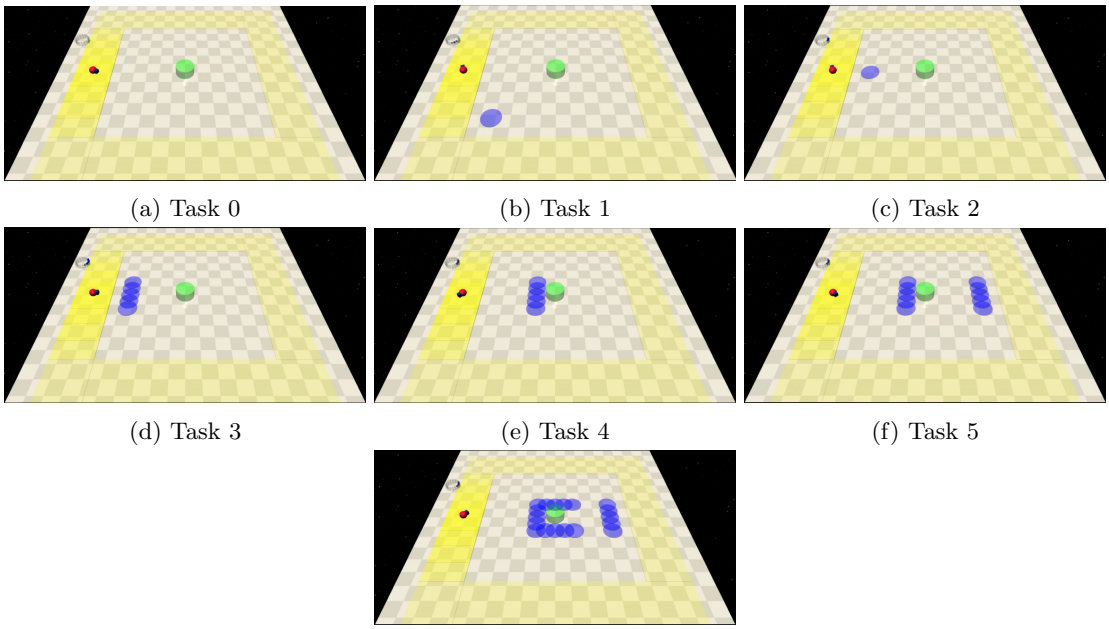
In this chapter, we describe how we will address our main problem of training an agent to be safe, with a focus on being safe throughout training. We do so by answering one part of **RQ1c** and **RQ2b**, namely how we implement a curriculum and how we combine it with safe RL. To do this, we will look at two strategies, namely building a curriculum fully by hand and extending this to automatically create a curriculum from a set of tasks, as discussed in Section 4.3.2.

5.1 Hand-made Curriculum

Here we will describe how we create a curriculum $\langle \mathcal{V}, \mathcal{E}, \mathcal{T}, g \rangle$. A simple first step to create a curriculum for safe RL is to do so by hand. As mentioned in Section 4.3.2, we will follow the guidelines of Peng et al. [2018]. This means that we create a curriculum of tasks with a focus on their four principles:

- We **isolate complexity** by creating tasks that try to teach the agent one new skill, such as not going in a straight line to the goal if there is a hazard in the way.
- This is done by **selecting the simplest environments to introduce one complexity at a time**, meaning that we create the simplest versions of the tasks that teach the agent a new skill.
- These tasks are chosen such that they are **most similar to the target environment**, such as by introducing more and more parts of the target environment into the tasks.
- This is also an example of **introducing complexity by building on previous tasks**, which in general can be done by requiring skills learned from previous tasks to complete the current task. In this way, a curriculum can be created with an implied ordering.

We put more restrictions on the intermediate tasks, such that we can keep the changes between tasks small enough to make transferring the knowledge from one task to the other easier. Only allowing small changes in the environment, means that we do not allow changes in the observation space or the action space. Therefore the only allowed changes are ones in the transition, reward or cost function. This implies that hazards can be placed in different parts and different numbers, or even the goal can be moved or repeated.



(a) Task 0 (b) Task 1 (c) Task 2

(d) Task 3 (e) Task 4 (f) Task 5

(g) Target Task

Figure 5.1: Curriculum tasks used for most experiments. Red is the agent, green is the goal, blue are the hazards and yellow are the possible starting locations, where a higher opacity relates to a higher probability of starting there.

The resulting tasks can be found in Figure 5.1 and the reasoning of each task is as follows:

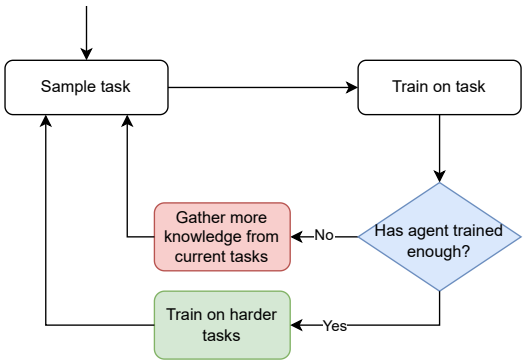
- Task 0 allows the agent to first learn that it must reach the goal.
- Task 1 allows the agent to only occasionally interact with a hazard, where the hazard is placed in one of the two corners on the left-hand side, such that it can carefully learn to avoid it.
- Task 2 puts a hazard in between the agent and the goal in many cases, forcing it to better deal with hazards in the way of a straight line to the goal.
- Task 3 extends this concept to a wall that is harder to avoid.
- Task 4 makes the strategy of moving away from hazards and moving towards the goal less useful, as the agent has to approach the hazards to reach the goal.
- Task 5 adds that the agent may need to move in between hazards to reach the goal.
- The target task forces the agent to move in tight spaces between hazards to reach the goal.

These tasks all share the same state space $\mathcal{S}$ and action space $\mathcal{A}$, so it therefore falls in our definition of only having small changes between the tasks. In Section 6.1.2 we discuss the more practical aspects of this curriculum, such as the reasoning behind a higher probability at the left-hand side.

Now the set of tasks $\mathcal{T}$ is clear, we need to create g to map vertices $\mathcal{V}$ to each of these tasks. This is simple as we can just map the ith vertex to the ith task:

$$g(v_i) = t_i, \forall i \in [0, 6], \tag{5.1}$$

where $v_i \in \mathcal{V}$, $t_i \in \mathcal{T}$ and $|\mathcal{V}| = |\mathcal{T}| = 7$.



Complex Metric	Complex Distribution
Has the policy converged?	Has the agent successfully completed the task?
Remain on current task	Shift distribution to easier tasks
Move on to the next task	Shift distribution to harder tasks

Figure 5.2: Flowchart depicting a general adaptive curriculum learning framework that illustrates the similarities between the approaches by Asada et al. [1996], referred to as Complex Metric, and Wu and Tian [2016], referred to as Complex Distribution.

The only remaining part of the curriculum is $\mathcal{E}$, which determines the relationship between tasks. As we want a linear ordering and we already have an implicit ordering we will create the edges as:

$$e_i = (v_i, v_{i+1}), \forall i \in [0, 5], \tag{5.2}$$

with $e_i \in \mathcal{E}$ and $|\mathcal{E}| = 6$.

Schedule. Although we have defined a curriculum, in practice we still need to define when to change tasks. There are too many options to try all of them, but we will follow some general heuristics. The first one is that later tasks, which are also the harder tasks as they require the knowledge of previous ones, need progressively more training time. The second is that we do not want to train until convergence for each intermediate task, because there might be a risk of overfitting on an easy task, resulting in possibly unsafe behavior in future tasks. More importantly, as was shown by Narvekar and Stone [2019], it may be faster to indeed not train until convergence. As for this type of curriculum, it needs to be decided beforehand when the task changes occur, we need to look at when the average agent converges and stop the task some episodes before that. After some small experiments, we decided to change tasks at epochs 10, 20, 40, 100, 300 and 700, where there is thus a shift from task 0 to 1 at epoch 10, from task 1 to 2 at epoch 20, etc. In this case, an epoch is 1000 steps and may consist of multiple episodes.

5.2 Adaptive curriculum

The main downside of the previous strategy is that all of it needs to be done by hand. To reduce effort we want to automate it, which since we make it adaptive, has the added benefit of creating agent-specific curricula. There are many ways to do this and even when looking at the strategy by Wu and Tian [2016] that changes the distribution of tasks, there are many ways to change a distribution. Or with the strategy by Asada et al. [1996] that changes tasks based on the difference in Q-values, there are many possible thresholds and there are several alternatives to using a metric like the Q-value.

To allow for flexibility, we first build a framework that generalizes these methods, with the option of extending it to either strategy. A comparison between these can be seen in Figure 5.2. Our framework contains two main components:

Probability distribution A probability distribution of tasks to sample from.

Algorithm 1 Learning algorithm with an adaptive curriculum for the basic instantiation

Number of training epochs n
Task distribution $p_c = (p_0, p_1, p_2, p_3, p_4, p_5, p_T)$ ▷ This is a Dirac distribution
Cost constraint `cost_limit`
Cost threshold factor β
Threshold of successful epochs needed κ

`successful_epochs` $\leftarrow 0$
for k = 0, 1, ..., n **do**
    Sample task $t_k \sim p$
    `task_completed`, `cost` $\leftarrow$ Train on task t_k
    if `task_completed` $= true$ **and** `cost` $\leq \beta \cdot$ `cost_limit` **then**
        `successful_epochs` $\leftarrow$ `successful_epochs` $+ 1$
    end if
    if `successful_epochs` $\geq \kappa$ **then**
        `update_distribution`(p) ▷ This shifts the Dirac distribution to the next task
        `successful_epochs` $\leftarrow 0$
    end if
end for

Competence metric A metric indicating whether a distribution change is warranted, based on the performance of the agent.

Then this can be used similarly to Wu and Tian [2016] by having a distribution that shifts based on the number of successful episodes. Or it can be used similarly to Asada et al. [1996] by making a Dirac distribution with probability 1 for the current task and 0 for all others and where the metric given is the Q-value, which after a certain amount of epochs moves the current task to the next. After each reset of the environment, we sample a new task from the distribution to use in the next epoch.

These changes do not necessarily result in changes in our curriculum $\langle \mathcal{V}, \mathcal{E}, \mathcal{T}, g \rangle$, defined in Section 5.1, apart from the schedule that is now determined by the competence metric instead of a list of epochs. In our version, we still have a linear ordering of the tasks, but if we extend it to the method by Wu and Tian [2016] every vertex is connected to every other vertex by an edge. This is due to the fact that the next task can be any task, depending on the distribution p_c. Where p_c is a new addition to the curriculum, which determines the probability of taking an edge e from each vertex v. Regardless, the tasks, vertices and g do not change.

5.2.1 Basic instantiation

Our framework does need a good probability distribution and competence metric to be able to perform experiments. To keep it simple, we use the distribution we would have for the strategy by Asada et al. [1996], meaning that we have only one task that we sample from and that this task changes to the next task as soon as some condition related to the competence metric is met. This metric should indicate that the agent can be considered to have learned enough on this task. What we want the agent to learn is to both complete the task and to do so safely. Therefore, it is crucial that we take both of these characteristics into account when determining the metrics necessary and the conditions associated with them. As we also want to keep this simple for now, we choose the competence metric to be the number of completed tasks where the cost is below the threshold. This directly tells us that the agent could behave safely and correctly on the current task and this gives us good reason that it will help with the next task. To add a little more flexibility to the competence metric, we divide it into two parts. One part is how many successful epochs need to be completed, which we will refer to as κ. The other part is related to when an epoch is successful, which we determine to be when at least one episode has been completed while

the expected cost is less than some factor of the cost limit, where we call this factor β.

Pseudocode for this algorithm can be found in Algorithm 1. At the start of each epoch, we sample a task from the distribution p_c (Line 8), which can be any distribution, but in the case of the basic instantiation it is a Dirac distribution, having probability 1 on task 0. After training on this task (Line 9), we look at the cost and whether an episode has been completed, and check if this resulted in a successful epoch (Lines 10-11). The condition in the if-statement can be changed depending on the strategy you want to use. For instance for the algorithm by Asada et al. [1996], we would check if the Q-value is less than some threshold, while for the algorithm by Wu and Tian [2016] we would do something similar to the basic version. After this if-statement, we check whether there were enough successful epochs on this distribution (Line 13). If this is the case we update the distribution to be a Dirac one over the next task and reset the number of successful epochs (Line 14). The condition in the if-statement and the exact functionality of updating the distribution can be changed depending on the strategy you want to apply. For Asada et al. [1996] this would require only one successful epoch and the distribution would change in the same way, while for Wu and Tian [2016] the condition would be the same, but the distribution would instead shift towards more difficult tasks. Also for the approach by Wu and Tian [2016] there would be an else-clause after the if-statement to shift the distribution to easier tasks.

Chapter 6

Experiments

In this chapter, we discuss the experiments needed to empirically address the research questions. We first discuss the details of the experiments that almost all of them share, which mainly relate to the environment setup. After this, we go over each experiment that we performed.

- Section 6.2 contains some initial curriculum experiments with the goal of finding good hyperparameters for the algorithms we used in the experiments. The experiments also give insights into some of the differences between agents with and without a curriculum.
- Section 6.3 takes this further by comparing several baselines that do not use a curriculum with several agents using a curriculum to find the effect of a static curriculum on the safety of the agents during training. We also try a different curriculum to see if this effect generalizes to other curricula.
- Section 6.4 builds on this by extending the curriculum with adaptive task sequencing and compares the effects that this addition has on safety with both the baselines that do not use a curriculum and the static curriculum.

A brief discussion on the code for these experiments can be found in Appendix A.

6.1 Setup

In this section, we explain several parts of the setup that are similar for most experiments. First, we discuss the necessary details of the environment that are the same for each experiment. Secondly, we discuss some of the practical details of the curriculum shown in Figure 5.1. Then, we briefly mention how we evaluated the agents. Finally, we discuss what methods we used to judge the performance of the agents in the experiments.

6.1.1 Environment details

The environment we used for the experiments is based on Safety Gymnasium [Ji et al., 2023a], an example of an environment in Safety Gymnasium can be seen in Figure 6.1. In Safety Gymnasium the agent moves around in a continuous space, where there are different types of hazards, such as the dark blue circles or the light blue box in Figure 6.1. For the navigation task, the goal is for the agent to reach the green cylinder. Besides Safety Gymnasium, we also used Omnisafe [Ji et al., 2023b], which is a framework that combines the environments from Safety Gymnasium with the infrastructure to train agents in those environments, such as CPO Achiam et al. [2017] and PPOLag Ray et al. [2019].

The main components of the environment are the reward and cost functions and the action and observation space. Even though the first two can change depending on where the goal and hazards are placed, there is a general formulation regardless of the positions.

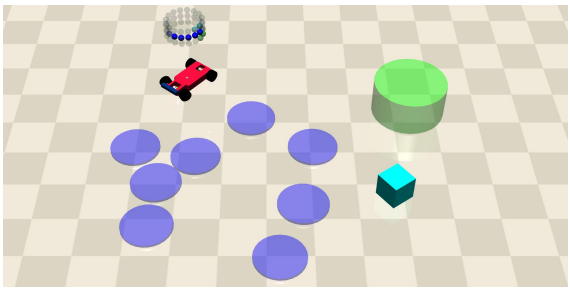


Figure 6.1: An example environment from Safety Gymnasium where the car needs to reach the goal without touching the hazards.

The reward function is defined as:

$$r_t = d^g_{t-1} - d^g_t + 10 \cdot 1_{d^g_t \leq \sigma_g}, \tag{6.1}$$

where d^g_t is the current distance to the goal, d^g_{t-1} is the distance in the last step, σ_g is the size or radius of the goal and $1_{d^g_t <= s_g}$ is the indicator function that is 1 when the agent has reached the goal and 0 otherwise. This results in a positive reward if the agent moves closer to the goal and a negative reward if the agent moves further away. For the cost function we use:

$$c_t = \sum_{h \in H} [\sigma_h - d^h_t]_+, \tag{6.2}$$

where h is a hazard from the set of all hazards H, σ_h is the size or radius of the hazard h, d^h_t is the distance from the agent to the center of the hazard and $[x]_+ = \max(x, 0)$. This means that the agent incurs a larger cost if it is closer to the center and incurs zero cost outside of the hazard. The cost limit we use is 5.

The observations of the agent, apart from the basic ones like velocity, are represented through a lidar, which means that there are a number of bins, 16 by default, that each correspond to a sector of the circle around the agent, so 1/16th part of the circle by default. Each bin indicates how close the closest object is to the agent in the direction of that bin. The agent has a lidar for each group of objects, in our case one for the goal and one for the hazards and in the case of Figure 6.1 one for the goal, one for all of the dark blue hazards and one for the light blue hazard. The lidar also has a limited range, which means that when the agent starts close to the outermost corners of the starting positions, the agent cannot see where the goal is. The actions of the agent are limited to rotation and forward or backward movement.

6.1.2 Environment design of the curriculum

In this section, we discuss some of the practical aspects of the curriculum shown in Figure 5.1. Instead of explaining the reasoning behind designing each task in the curriculum as we did in Section 5.1, we discuss how the implementation of the tasks was influenced by looking at the behavior of the agents using a curriculum. We will refer to agents that learn using a curriculum as curriculum agents.

We designed the target task such that it should be of relatively high difficulty, to see if there is a task where either the baseline or the curriculum agents outperform the others. We came up with the task shown in Figure 6.2, which requires the agent to move in between hazards and make a relatively sharp turn, and there is no position where the agent can go in a straight line without going over the cost limit. Although changes have been made during the hyperparameter tuning experiments detailed in Section 6.2, the final six intermediate tasks we devised that build up to this target task can be seen in Figure 5.1. In all of the tasks shown, the red sphere is the agent, where the two lidars are hovering above it. The goal is indicated as a green cylinder and the hazards are the blue circles. Lastly, the initial positions of the agent are shown using the yellow

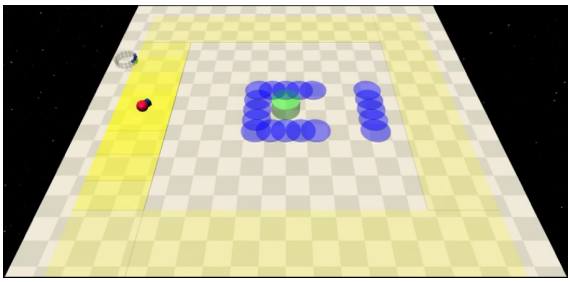


Figure 6.2: The target task from Figure 5.1

areas. A higher opacity indicates that there is a larger probability of starting there. This is done in such a way that in roughly 50% of the cases, the agent starts on the left side of the square, which is also considered to be the more difficult side.

The starting position bias to the left-hand side is important, because the agent uses the expected cost in its calculations for staying below the threshold. This means that in a uniform distribution, it is possible in many of the intermediate tasks to use the strategy of going directly to the goal regardless of the costs. As we want the agent to learn more sophisticated strategies, we decided to increase the likelihood of starting in the most difficult position for most environments to about 50%. We also do not want to always start at the left side, since it is good for generalizability if it can start anywhere, which also is the case when this would be applied in a practical setting.

Another important addition to the environments is the fact that the hazards overlap. We noticed from the early results that, when the hazards do not overlap, it is a good strategy to try to move in between the hazards and incur only a small cost. This is not our goal and this strategy is hard to consistently pull off, making it occasionally a very costly strategy. Therefore, we decided to put hazards in between the other hazards for the final curriculum, such that it is not a feasible solution to try to go in between and to encourage the agents to find a better strategy.

6.1.3 Metrics for performance

We have already put emphasis on the regret for measuring the safety of the agents, but this is not the only measure we use in these experiments. We use the return and cost to get an idea of how the agents are performing throughout training. However, we do not use the return and cost of each episode or epoch, but we use a moving average over the last 100 episodes. This means that results of the first 100 episodes of training are a lot more sensitive to each new result than those later on in training, as it is then the average over only a few episodes. The benefit of this moving average is that it more accurately shows the expected return and cost at each epoch, although this is most accurate when the agent has converged. An epoch consists of 1000 steps and can consist of several episodes, depending on whether the agent completes an episode in less than 1000 steps. Finally, we calculate the regret over the moving average of the cost.

For our experiments, we used different seeds to repeat the experiments and get a good estimate of the general performance of each setting. The results then contain the average return, cost and regret over these seeds. Since the return, cost and regret themselves are moving averages, the average over the moving averages is taken for the results.

6.1.4 Evaluation

Even though the performance during training is very important when it comes to safety, it is still important to also look at the performance on evaluation. Unlike during training, the evaluation is not stochastic and therefore demonstrates how the policy would actually perform. On top of this, specifically for PPOEarlyTerminated, during evaluation the agent is not stopped anymore, showing how good the policy is when there is no outside help.

To evaluate the agents, we saved the model during training such that we could run some evaluation episodes after training. The evaluations are always on the target task of the experiment, therefore if we take a subset of the experiment, we evaluate on the last task of that subset. The agents were evaluated for 5 episodes every 10 epochs, apart from the hyperparameter tuning experiments in Section 6.2.1 and Section 6.2.2, to make the results more reliable. We do this for every seed and take the average over all of these seeds for each performance metric.

6.2 Hyperparameter Tuning

After some initial experiments with some of the algorithms, we saw that they performed quite poorly. Therefore we decided to tune several important hyperparameters. We have chosen to do so for the PPOLag algorithm, as in many of the papers discussed in Section 4.1, PPOLag was used to compare with the respective algorithms. Therefore, it is useful for us to have a good version of the PPOLag agent so that we can make proper comparisons.

Therefore, in this section, we investigate which hyperparameter values work best for a PPOLag agent without a curriculum and whether these values are the same for a PPOLag agent with a curriculum. We specifically tune for a PPOLag agent that does not use a curriculum, to ensure fairer comparisons with curriculum agents, reducing bias in favor of those agents. We first tune four of the most impactful hyperparameters for learning and compare the agents with and without a curriculum to see if there are any differences between optimal values. Then we change the cost limit, to see if the agents perform safer with a lower cost limit. Finally, we train the best hyperparameter combinations for longer to find the best hyperparameters for the baseline and to see how the curriculum agents perform on them.

6.2.1 Most significant parameters for learning

The goal of this experiment is to find which parameter values give the PPOLag agent without a curriculum the best performance and whether these are different from the best parameters for PPOLag with a curriculum. We have chosen four parameters to tune, which were the Lagrangian parameter initialization and learning rate, the size of the neural networks and the number of update iterations. The reasoning is that the Lagrangian parameters are known for needing tuning for each specific problem [Chow et al., 2019], the neural network size impacts the performance significantly as this determines the learning capacity of the agent and the number of update iterations affects how much the model changes after each epoch.

Setup

To accomplish our goal, we designed the following grid search experiment. Two types of PPOLag agents were used, one with and one without the curriculum. For the four hyperparameters, we chose the following values to be investigated:

- Lagrangian multiplier initialization: $[0.001, 0.01, 0.1]$
- Lagrangian multiplier learning rate: $[0.01, 0.035, 0.05, 0.1]$
- Number of neurons per hidden layer: $[64, 256]$
- Number of update iterations: $[1, 10, 50]$

This leads to 72 unique combinations, where we trained both a baseline agent without a curriculum and an agent with a curriculum for each combination. We trained these agents with the target task being task 4. This was to see how well each of the algorithms performed on a task of medium difficulty. Each of the agents was tested with 3 different seeds. Due to the large number of agents we trained each for 100 epochs and they had only 3 evaluations per 20 episodes.

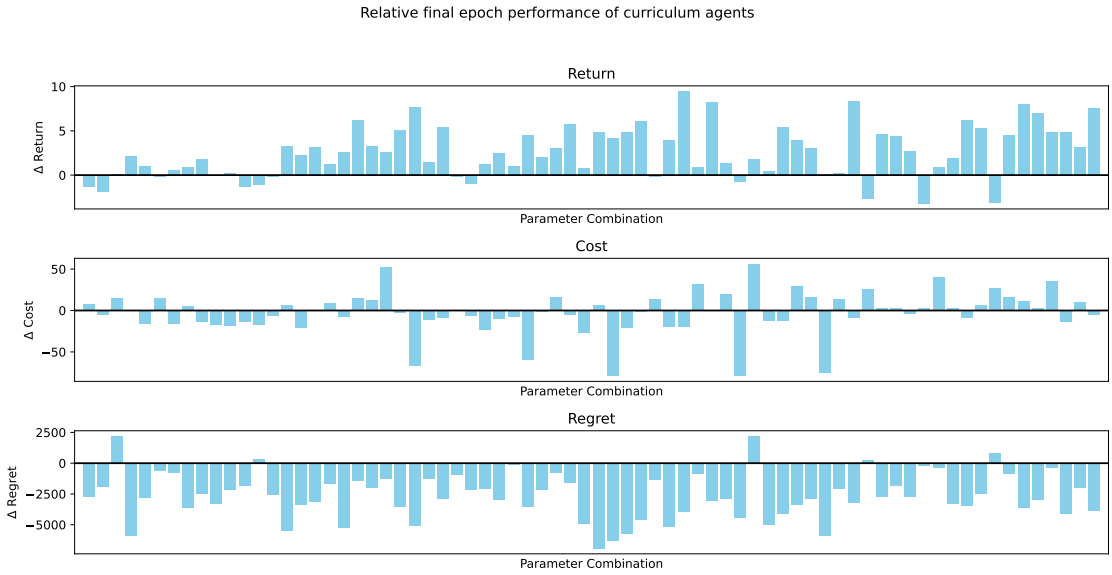


Figure 6.3: Relative performance of the curriculum agents to the baseline agents, sorted from highest evaluation return to lowest of the baseline agents.

We decided that the return was a very important metric to focus on when determining the best hyperparameters. This is the case as when the agent cannot even solve the problem with a high enough return, it does not matter how safe it is.

Results

Figure 6.3 and Figure 6.4 show a summary of the results and the full results can be found in Appendix B.1.

Figure 6.3 shows three bar charts, one for each performance metric. The y-axis shows the difference between the performance at the end of training of the curriculum agent and that of the baseline agent. The x-axis represents all 72 hyperparameter combinations, sorted from highest evaluation return to lowest for the baseline agents. This means that the first bar in in the return plot has the same hyperparameter values as the first bar in the cost and regret plots and the same holds for the ith bar. If a bar is above 0, this means that the value for that metric of the curriculum agent was higher for that parameter combination and worse if it is below 0. Therefore a bar above 0 indicates for the return that the curriculum agent was better, but for the cost and return this means that the baseline was better as we want to limit those.

Figure 6.4 shows three heatmaps, one for each performance metric. The y-axis represents whether it belongs to the baseline or curriculum agent results. The x-axis represents each of the hyperparameter combinations, independently ordered from best to worst value for that type of agent. Therefore, the first hyperparameter combination for the baseline row is not necessarily the same as the first hyperparameter combination for the curriculum row, even for the same performance metric. The heatmap thus shows how the curriculum agents perform with their best hyperparameters for each metric, compared to the baseline agents with their own best hyperparameters for that metric. The cells contain the value at the end of training of the respective metric for the type of agent and parameter combination. The color of the cells is relative to the combined values of the baseline and curriculum agents for that metric.

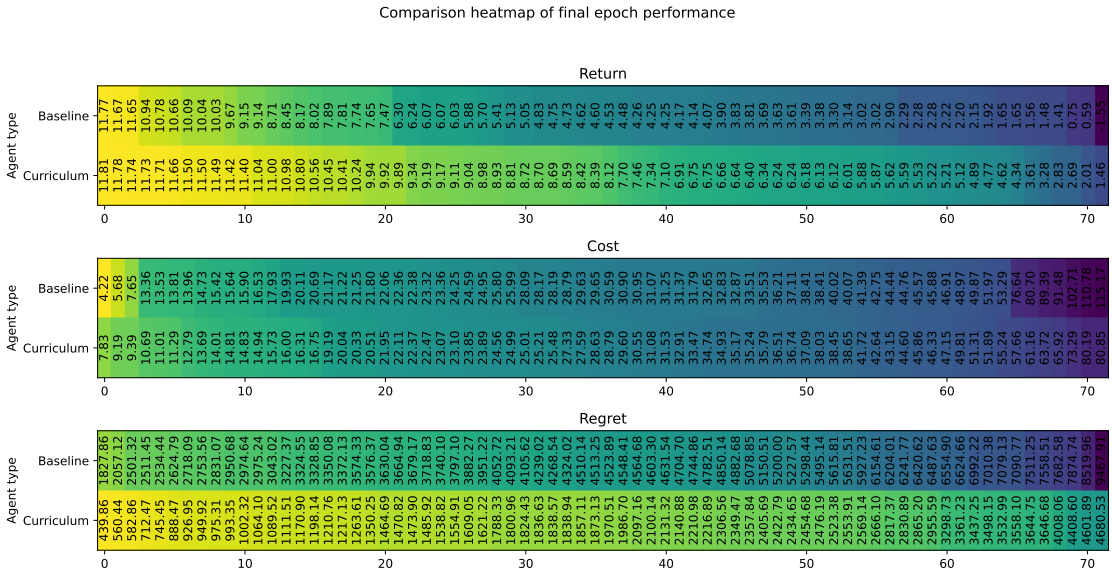


Figure 6.4: Comparison between the curriculum and baseline agents on different hyperparameters. Each of the six rows is ordered independently from best to worst performance on that metric.

Analysis

The best parameters are not all the same for the baseline as for the curriculum agent, but there is some overlap. Out of the nine combinations that are the best for the baseline agents, three of them are also the best for the curriculum agents. The best nine parameter combinations for the baseline agent were chosen by looking at the ones with a high return and a cost that was not more than 35. The full results in Appendix B.1 show what these nine parameter combinations are.

The curriculum agent had a much lower regret in most of the settings. This can be seen in Figure 6.3, where in only 5 out of the 72 hyperparameter combinations, the baseline agents have a lower regret. This shows that the safety increase is close to invariant to the hyperparameter. The return is also often larger or very similar to the baselines. The cost shows that, unlike these two other metrics, the end cost of the baselines is much more similar to the curriculum agents.

The curriculum agents are more stable across the different hyperparameters. This can be seen in Figure 6.4, where the difference between the best and worst performance in each metric is much smaller than that of the baseline agent. This is especially evident for the regrets, where the difference between the best and worst regret for curriculum agents is about 4241 and for the baseline agents it is about 7640.

6.2.2 Different cost limits

The goal of this experiment is to see the effect of a different cost limit on the baseline and curriculum agents with the hyperparameter combinations that we found in Section 6.2.1.

Setup

To do this, we designed the following experiment. We used baseline and curriculum versions of the PPOLag algorithm. We then tested our standard cost limit of 5 against a lower cost limit of 1. We trained both baseline and curriculum agents with each of these cost limits and with each of the nine parameter combinations from Section 6.2.1, resulting in eighteen baseline agents and eighteen curriculum agents. We trained these agents with the target task being task 4. We trained

				Cost Baseline		Cost Curiculum	
LMI	LMLR	UI	NN	$\alpha = 1$	$\alpha = 5$	$\alpha = 1$	$\alpha = 5$
0.001	0.01	1	64	39.89	55.29	39.96	27.10
0.001	0.01	1	256	34.68	30.02	98.06	35.34
0.001	0.01	10	64	40.48	37.56	24.76	30.80
0.01	0.01	1	64	35.23	25.45	22.14	17.20
0.01	0.01	10	64	43.34	25.42	57.40	41.35
0.1	0.01	1	64	32.51	25.10	27.22	33.22
0.1	0.01	1	256	44.68	26.76	47.62	24.21
0.1	0.01	10	64	65.95	35.75	29.56	136.78
0.1	0.01	10	256	34.68	67.76	72.88	44.66

Table 6.1: Table of average training cost of the PPOLag agents. LMI is the Lagrangian multiplier initial value, LMLR is the Lagrangian multiplier learning rate, UI is the number of update iterations and NN is the neural network size and α is the cost limit. The last two slanted rows use the same parameters as the two above, but the former were trained for 800 epochs instead of 500.

each of the agents for 100 epochs and with 5 different seeds. We still only made 3 evaluations per 20 episodes.

Results

The results can be found in Table 6.1, where the first four columns represent the values of the hyperparameters in the order: the Lagrangian multiplier initial value, the Lagrangian multiplier learning rate, the number of update iterations and the neural network size. The remaining columns give the average cost at the end of training of the respective agents for both a cost limit of 1 and a cost limit of 5. Appendix B.1 contains the full results.

Analysis

Using a lower cost limit results in larger constraint violations. In Table 6.1, of the eighteen times we compare a cost limit of 1 with a cost limit of 5, there are only five hyperparameter combinations where a cost limit of 5 has a higher cost. We hypothesize that due to it being more difficult for the agents to satisfy, the entire performance suffers. This also implies that changing the cost limit could be part of a curriculum, where it is gradually lowered to the desired limit, which may be useful in scenarios where there is a total cost that should not be violated.

6.2.3 Best hyperparameters

The goal of this experiment is to find the best hyperparameters that have a significant impact on learning and to see how well curriculum agents perform compared to the best version of the baseline agents.

Setup

For this purpose, we have designed the following experiment. We used the PPOLag algorithm for baseline and curriculum agents. As we had more samples for the nine combinations of parameters, we decided to pick six of them that performed the best for this experiment, based on their learning curves and their behavior after learning. These six combinations were trained for 500 epochs such that convergence is more likely, with the target task being task 4. After having seen the results, we retrained two of the combinations again for 800 epochs, as they were the only ones that had not converged. We increased the number of repetitions to 10 and starting from this experiment, we changed the evaluations to 5 every 10 epochs.

					Return		Cost		Regret	
Epochs	LMI	LMLR	UI	NN	Base	Curr	Base	Curr	Base	Curr
500	0.001	0.01	1	64	12.46	12.48	10.38	8.83	9558	2949
500	0.001	0.01	1	256	9.90	11.16	11.84	12.18	12048	7427
500	0.01	0.01	1	64	11.42	12.46	9.89	9.39	7448	4086
500	0.1	0.01	1	64	10.66	11.86	8.67	9.01	9011	4165
500	0.1	0.01	1	256	9.12	8.86	9.26	7.62	10470	8149
500	0.1	0.01	10	64	11.55	9.28	8.57	12.51	10646	13264
800	*0.1*	*0.01*	*1*	*256*	*12.47*	*10.18*	*8.02*	*4.13*	*9333*	*9964*
800	*0.1*	*0.01*	*10*	*64*	*10.71*	*11.82*	*18.54*	*9.13*	*14870*	*5824*

Table 6.2: Table of average PPOLag agent training performance. Epochs is the number of epochs trained on, LMI is the Lagrangian multiplier initial value, LMLR is the Lagrangian multiplier learning rate, UI is the number of update iterations and NN is the neural network size, Base indicates the baseline agents and Curr indicates the curriculum agents. The last two slanted rows use the same parameters as the two above, but the former were trained for 800 epochs instead of 500.

We saw during the evaluations of previous experiments that some agents seemed to perform better than others, which turned out to be because they had lucky initial positions that allowed them to go straight to the goal and thus had a low cost during evaluation. As this made comparisons more difficult we decided to temporarily add an evaluation environment for this experiment. This evaluation environment placed the agent in the hardest position, the center-left. Here it needs to make the biggest turn to go around either side of the wall of hazards to reach the goal. It was removed after this experiment, as the agent optimizes for the expected cost, which is different if instead of a random starting position, it always starts at the most difficult position.

Results

Table 6.2 shows the results of this experiment. The first column indicates for how many epochs the agents of each row were trained, where we have separated and italicized the ones that were trained for longer, although the versions where they were trained for 500 epochs are still included. The next four columns represent the values of the hyperparameters in the order: the Lagrangian multiplier initial value, the Lagrangian multiplier learning rate, the number of update iterations and the neural network size. The remaining columns give the average value at the end of training of the respective agents for both the baseline and curriculum versions and for each performance metric. A similar table containing the evaluation results can be found in Appendix B.1.

Analysis

There are no combinations that ended with a cost below the cost limit within 500 epochs, but there is one that does so within 800 epochs. From the results in Table 6.2, we find that no combinations can stay below the cost limit within 500 epochs. The two that had not converged and which we retrained for 800 epochs still show unstable cost curves. However, one of them is the only one with an average end cost of below 5 for the curriculum agent, while the baseline with the same parameters is slightly above the limit. In addition, this is also the combination with the highest return, lowest cost and one of the lowest regret for the baseline agent. Therefore, despite its unstable cost, this was chosen to be the best parameter combination for the baseline.

There is a significant performance gap at epoch 500 between agents trained up to that point and those trained for a longer duration. Especially with the return, there is a large difference when comparing the same hyperparameters with a different number of total

training epochs at the same epoch. This happens because the algorithms use a learning rate decay, which means that the total number of training epochs can have a large impact on performance throughout training. In our case, the return of the curriculum agents is much larger at epoch 500 when the total is 800 epochs. For the baseline, this is only the case for one of the hyperparameter combinations, while the other combination see a decrease in return.

6.3 Static Curriculum

In this section, we investigate whether curriculum learning improves the safety of an RL agent, such that we can answer research question **RQ1**. Firstly, we compare all of the algorithms that we selected in Section 4.3.1 to find if they indeed work well with a curriculum, such that we can address **RQ1a**. Then, using the algorithms that showed to work well with a curriculum, we compare their performance on each task with baselines that do not use a curriculum. This is to study the effect of curriculum learning compared to state of the art in safe RL, thus addressing **RQ1c**. We further address this research question by investigating agents that were trained using a different curriculum to see if the effect of a curriculum persists with different curricula. This is followed by two ablation studies. The first discusses the benefit of resetting the learning rate after changing tasks and the second discusses the benefit of not resetting the Lagrangian multiplier after changing tasks. These ablation studies give insights into what properties are important for curriculum learning in the context of safe RL and thus address **RQ1b**.

6.3.1 Comparing safe RL algorithms

The goal of this experiment is to see which of the safe RL algorithms selected in Section 4.3.1 work well with a curriculum, as to address **RQ1a**.

Setup

To do this we designed the following experiment. The agents used in this experiment consist of a curriculum version and a baseline version, which does not use a curriculum, of the algorithms CRPO [Xu et al., 2021], CUP [Yang et al., 2022], FOCOPS [Zhang et al., 2020], PCPO [Yang et al., 2020], PPOEarlyTerminated [Sun et al., 2021] and PPOLag [Ray et al., 2019]. We included two more agents with only a baseline version, namely PPO and CPO. PPO gives perspective on the behavior of an agent that does not prioritize safety and CPO is to show another popular algorithm. These two were also only used without a curriculum, all of the others were tested both with and without a curriculum. The reason that PPO was not tested with a curriculum is that nothing changes for the PPO agent with a curriculum, as only the hazards change, which PPO ignores. CPO was also only used without a curriculum as it does not have any of the three properties we found to be important for curriculum learning.

We trained these agents with the target task being task 4. This was to see how well each of the algorithms performed on a task of medium difficulty. The hyperparameters found in Section 6.2 were used for all of these agents. That means that all agents use 1 update iteration and a neural network with two hidden layers of size 256. All agents with a Lagrangian approach use a Lagrangian multiplier initialization of 0.1 and a Lagrangian multiplier learning rate of 0.01. We trained the agents with 15 seeds each for 1000 epochs.

Results

Figure 6.5 shows the results of this experiment. It shows the average return, cost and regret at the end of training over the seeds for each agent. The x-axis shows the performance on each metric at the end of training for the baseline versions of the agents and the y-axis does so for the curriculum versions. The diagonal is drawn to indicate more clearly whether the performance of the baseline version or the curriculum version was better for that metric and for each algorithm. If an algorithm is below the diagonal with respect to the return, that means that the return of the

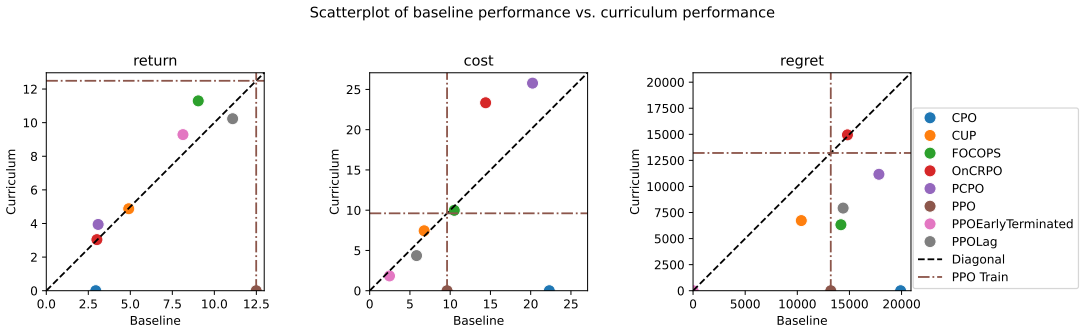


Figure 6.5: Plots showing how the algorithms performed with and without the curriculum, with the target task being task 4. The return and cost are the average of the metric in the final epoch of each repetition and the regret is the average of each repetition. Since PPO and CPO have only been used for the baseline, their respective curriculum values are considered 0.

curriculum version is worse than the baseline version and vice verse. For the cost and regret is the opposite as a lower cost or regret is better. On top of this, lines are drawn for both the x and y-axis indicating the performance of the PPO agent on that metric. This is to indicate how the safe RL algorithms performed with respect to a general RL algorithm. A similar figure with the evaluation performance can be found in Appendix B.2, where we also provide the learning curves of the return and regret of all agents.

Analysis

The regret of the curriculum agents is lower during both training and evaluation. From the results in Figure 6.5 and the evaluation results in Appendix B.2, we can see that for most algorithms the regret is always lower than that of the PPO agents for the curriculum versions, with the exception of CRPO. However, for the baseline versions only two of the algorithms have a regret lower than the PPO agents, namely PPOEarlyTerminated and CUP. An interesting case is PPOEarlyTerminated, which is the only one that is close to 0 during training. However, during evaluation, where the agent is not stopped when reaching the cost limit, the regret is by far the largest. Still, the curriculum variant has a much lower regret than the baseline during evaluation.

There is a clear distinction between algorithms that have a high return and ones that have a low return. The PPO agent has the highest return of around 12.5. The others that perform well on return are FOCOPS, PPOLag and PPOEarlyTerminated, which all have an average return of at least 8. All of the other algorithms have an average below 6, which have in common that they start to fall off starting from task 3 and do not recover that well. This is all quite similar when looking at the evaluation returns. On top of this, the returns do not differ that much between baseline and curriculum agents, although it seems that most algorithms perform slightly better under a curriculum. The costs show something similar, but the costs of the curriculum agents seem to be higher in some extreme cases. In these cases, the baselines also have high costs.

Curriculum agents have a large return advantage at the start of training. The learning curves for return and regret, which can be seen in Appendix B.2, show that the curriculum algorithms have a larger return than the baselines at the start. This is due to the easier tasks and seems to allow the agent to keep a relatively high return from the point that it switches to the more difficult tasks. The cost curves of the baselines show large spikes near the beginning of training. These spikes are dampened when looking at the cost curves of the curriculum agents. This is also visible in the regret curves, where the difference is very big at the start, but stays relatively constant after a while.

We can conclude from the results that curriculum learning is quite helpful in lowering the regret when optimizing under safety constraints. It is also clear that not all algorithms are able to consistently solve the target task, as an average return of less than 5 is quite poor. As the PPO agent does not have the lowest regret, the algorithms that have a higher regret can be deemed unfit for this task, as an agent that does not care about safety is safer than those. It does seem that these algorithms have learned something that does not work starting from task 3 and are not able to properly adjust afterward.

The PPOEarlyTerminated agent seemed promising during training as it has the lowest regret and a high return, but during evaluation this turned out to be misleading. The evaluation performance is significantly worse with regards to safety than all other algorithms, meaning that it has not learned to be safe and it fully relies on the early stop mechanism to show a safe behavior during training.

In summary, the best algorithms do seem to be FOCOPS and PPOLag, as they both have high return and relatively low regret. Still, CUP also has a very low regret and has the best return of the ones that have lower return. This suggests that the properties from Section 4.1 were not indicative of good performance, as these CUP only had two of them,while PPOLag and FOCOPS only had one. However, this would potentially indicate that generalization is extremely important for curriculum learning.

6.3.2 Learning on each task

The goal of this experiment is to make the analysis of our static curriculum more comprehensive by further investigating the algorithms that worked well with a curriculum and to find the differences between agents that use a curriculum and agents that do not, such that we can address **RQ1c**.

Setup

To accomplish this we designed the following experiment. We used the best algorithms for curriculum learning found in Section 6.3.1, which were CUP, FOCOPS and PPOLag. As PPOEarlyTerminated is similar to state of the art, we also included this algorithm in this experiment. These four agents were trained with a baseline and curriculum version. For the same reasons as before, we train only a baseline version for the PPO and CPO algorithms.

As the previous experiment only compared the baselines and the curriculum agents on task 4, we wanted to compare them on all tasks to see if there is any point at which large differences occur. Therefore we trained agents with each task in the curriculum being the target task for one of the subexperiments, with the exception of task 0, as then there is no difference between using a curriculum or not, as there is only one task. We decided to train the agents for 2000 epochs for tasks 3 through 5 to allow for convergence. Tasks 1 and 2 were trained for 500 epochs, as they need less time to converge. The target task was trained for 3000 epochs as initial experiments showed few cases that had converged by epoch 2000. All of the settings were repeated with 10 different seeds.

We also made a change to the way that curriculum agents learn in this experiment. As we saw that the agents did not seem to adapt all that well after curriculum changes, we found that resetting the learning rate may have a positive impact. This is because the learning rate has a linear decay to allow convergence of the algorithms. However, we believe that when changing to a different task, the learning rate should increase to allow the agent to quickly adjust to the new environment. We decided to do this when a change of tasks occurs, allowing the agent to more quickly adapt to the new environment and hopefully be more safe. We performed an ablation study in Section 6.3.4 to show the effectiveness of this strategy.

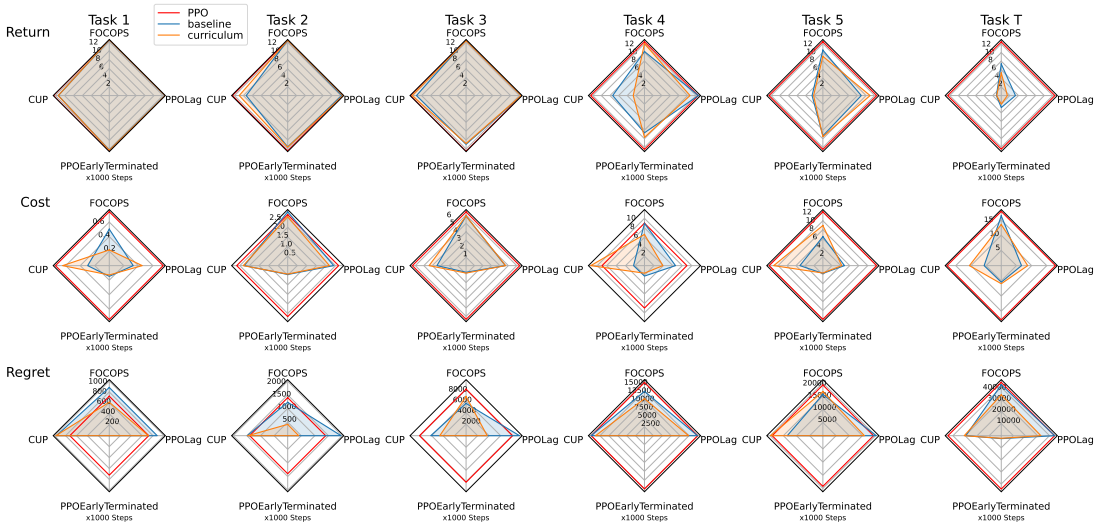


Figure 6.6: Radar charts showing for each task other than 0 what the difference in training performance is between the baseline and the curriculum agents. The performance of the PPO agent is included as a reference point of what is the maximum value allowed for each of the metrics. For return we consider it better to have a larger area in the spider plot and for cost and regret we consider it better to have a smaller area.

Results

Figure 6.6 and Figure 6.7 show the results of this experiment.

Figure 6.6 is a grid of radar charts, where the columns represent the target tasks and the rows represent the performance metrics. Each radar chart consists of four axes representing the four algorithms that were tested with and without a curriculum. The values on each axis show the average performance at the end of training on that metric for that algorithm on that target task. These values are also connected to give an impression of the performance of the curriculum and baseline agents over all algorithms. This means that when the area of the baseline agents is fully within the area of the curriculum agents for the return, the curriculum agents in general outperform the baseline agents with regards to return on that target task and vice versa. For the cost and regret, this is reversed as higher values represent worse performance. The average performance of the PPO agents is also included to give perspective on whether the algorithms performed better or worse than PPO. A similar figure for the evaluation results can be found in Appendix B.2.

Figure 6.7 shows the learning curves of the two types PPOLag agents. The columns represent the performance metrics and the rows represent the target tasks. Each learning curve contains the value for the respective metric on the y-axis and the epochs on the x-axis. The dashed line indicates the cost limit and the vertical grey lines indicate the epochs where a task change occurs. A similar figure for the evaluation results can be found in Appendix B.2, where we also show the learning curves for return and regret for all agents on the target task.

Analysis

The curriculum reduces the regret in many of the tasks and algorithms, while the return is about as good or even better. This can be seen from Figure 6.6. For PPOLag the difference between baselines and curriculum agents is the largest, in favor of the curriculum agents. For PPOEarlyTerminated this difference is the smallest, where the curriculum does not seem to add much to the performance. CUP seems to be the worst algorithm in general, as it has the lowest returns and still has high costs and regrets. However, this algorithm seems to also not

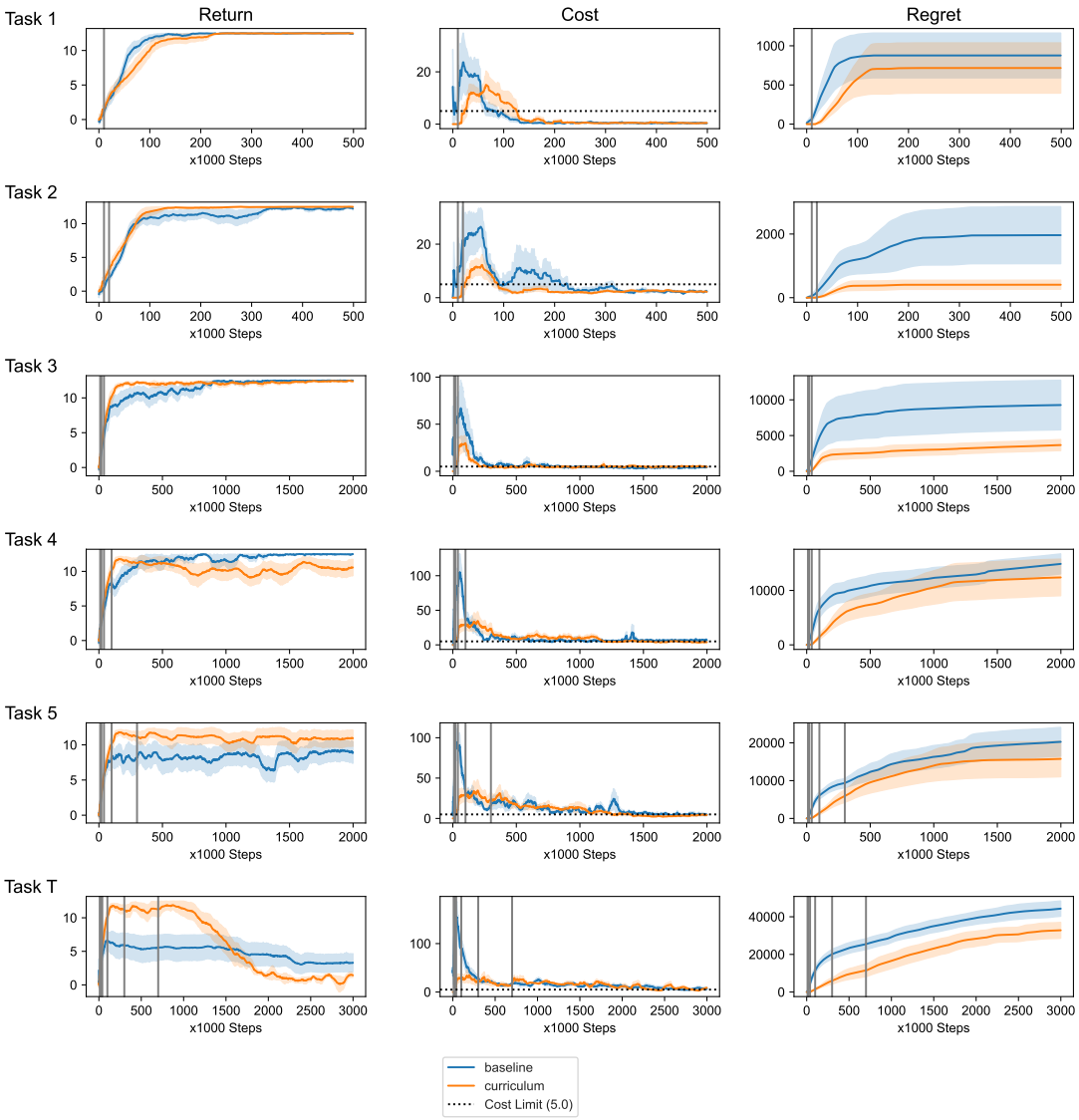
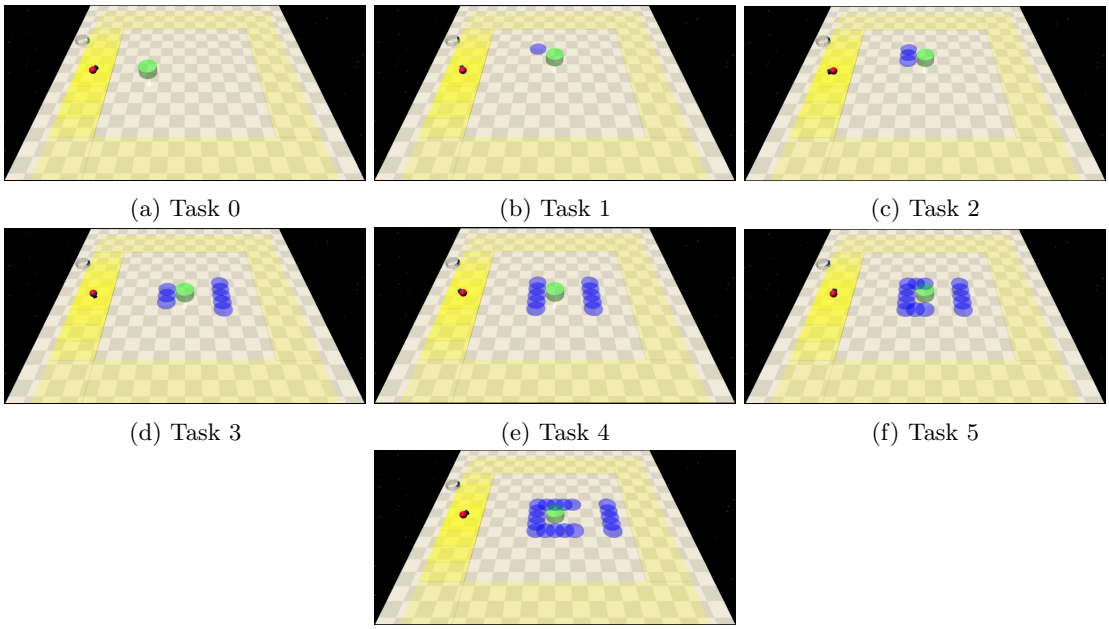


Figure 6.7: Training performance curves of the PPOLag agents, comparing baseline with curriculum progression. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents.

benefit as much from a curriculum and in some cases is significantly worse off if we look at the costs and returns. For FOCOPS the differences between the baselines and the curriculum agents are not always as large as with PPOLag, but it comes quite close. For some tasks, the baselines still have a lower regret, but in general it seems as though the curriculum helps with both safety and return. When comparing to the PPO agent, both types of agents seem to perform better with regard to safety on all algorithms, which reinforces that all of these algorithms behave safer than the minimum safety required when using a curriculum. In Appendix B.2, we show the return and regret curves of all algorithms on the target task. Here CPO consistently has the worst return and one of the worst regrets. Although this is for the very difficult target task, the performance of CPO on all previous tasks is very similar, even for task 1. So it seems that CPO without any tuning cannot properly find the optimal policy in these environments.



(a) Task 0 (b) Task 1 (c) Task 2

(d) Task 3 (e) Task 4 (f) Task 5

(g) Target Task

Figure 6.8: Curriculum tasks created to test the difference in performance when the curriculum is focused on reward instead of cost. Red is the agent, green is the goal, blue are the hazards and yellow are the possible starting locations, where a higher opacity relates to a higher probability of starting there.

Curriculum agents dampen the cost spike in early episodes. Figure 6.7 shows that both types of agents have a spike in their costs near the start of training, but in each task this spike is much lower for the curriculum. This is most likely the main reason that the regret is lower for the curriculum agents, as after the start, the regrets seem to have similar slopes. The evaluation data can be found in Appendix B.2.

The return of the curriculum agents is about the same as that of the baseline agents. There is an interesting case with the target task, where the return stays high at first, but eventually stays below the baseline. This is most likely due to the fact that the Lagrangian multiplier is also reset when changing tasks, meaning that it takes a while to adapt to the target task. As it seems a promising extension to the current method to keep the multiplier from a previous task, we perform a small ablation study using this new method and compare it to the one used for these experiments in Section 6.3.5. The fact that it goes below the baseline demonstrates that it knows when it is too difficult to attempt to get a high return. When looking at the individual results of the baselines we can see that the ones with a higher return also have large cost limit violations. The curriculum agents instead do not reach these high costs and drive circles around the hazards.

6.3.3 Comparing with a different curriculum

The idea behind this experiment is to see if the effects of a curriculum from the previous experiments persist with a different curriculum and thus give more support for our answers to **RQ1c**. Our main curriculum is focused on improving the safety of the RL agent during the learning process. We could also have taken the approach of getting a good return, while not neglecting the safety aspect. Therefore we created a reward-based curriculum, which can be seen in Figure 6.8. The reasoning behind the curriculum in Figure 6.8 is as follows:

- Task 0 allows the agent to first learn that it must reach the goal and is quite easy most of the time.
- Task 1, where the hazard and the goal can be mirrored horizontally, allows the agent to learn to go around a hazard and can go in a straight line to the goal.
- Task 2, where the hazard and the goal can be mirrored horizontally, forces the agent to almost always at least make a slight turn to reach the goal.
- Task 3 is much closer to the target task and forces the agent to make an even larger turn to get to the goal.
- Task 4 takes the concept of task 3 and makes it slightly more difficult.
- Task 5 adds on top of this that the agent should make a U-turn to reach the goal in a way that is very close to the target task.
- The target task is the same as before and requires the agent to make a larger turn and move in between hazards for longer.

Setup

To test whether this curriculum also shows safety benefits, we designed the following experiment. We used PPOLag to train curriculum and baseline agents on this curriculum. We used the same approach as in Section 6.3.2 and thus train both types of agents on each task as a target task. We again decided to train the agents for 500 epochs on tasks 1 and 2, 2000 epochs for tasks 3 through 5 and 3000 epochs for the target task. All of the settings were repeated with 10 different seeds.

We also wanted to compare it to our main curriculum. As there is no clear way to do this, we compared the performance of the PPOLag agent using the main curriculum on each of its own tasks. This gives somewhat of a misleading view, as you cannot compare the performance on each task in a fair way due to the tasks being different, but the main goal is to compare the progression throughout their respective curricula.

Results

Figure 6.9 shows the results of this experiment. The columns represent the performance metrics and the rows represent the target tasks. Each learning curve contains the value for the respective metric on the y-axis and the epochs on the x-axis. The dashed line indicates the cost limit and the vertical grey lines indicate the epochs where a task change occurs. Each cell in this grid contains the performance of the two types of agents we discussed and the curriculum agent trained on our original curriculum. The performance of the latter cannot directly be compared to the other two, apart from on the target task, as they do not share the intermediate tasks. However, it can be used to compare the progression throughout the respective curricula of the two types of curriculum agents. This means that when the return of the reward curriculum agent is higher on a task, then this shows that the curriculum up to that task was easier to learn than that of the standard curriculum agent and vice versa. Something similar can be said for the cost and regret curves, where a lower cost or regret shows that the respective curriculum is easier to learn from. A similar figure for the evaluation results can be found in Appendix B.2.

Analysis

Even with a differently focused curriculum, the curriculum agents are much safer, with minimal loss to the return. When looking at Figure 6.9, we see that the return is slightly higher for the reward curriculum agent than for the reward baseline. On top of this, the regret is also lower, which indicates that a decent curriculum that tries to keep the return high, while paying less attention to the safety aspect, can still result in a safer learning trajectory than when not using a curriculum. The difference in return between the reward curriculum and

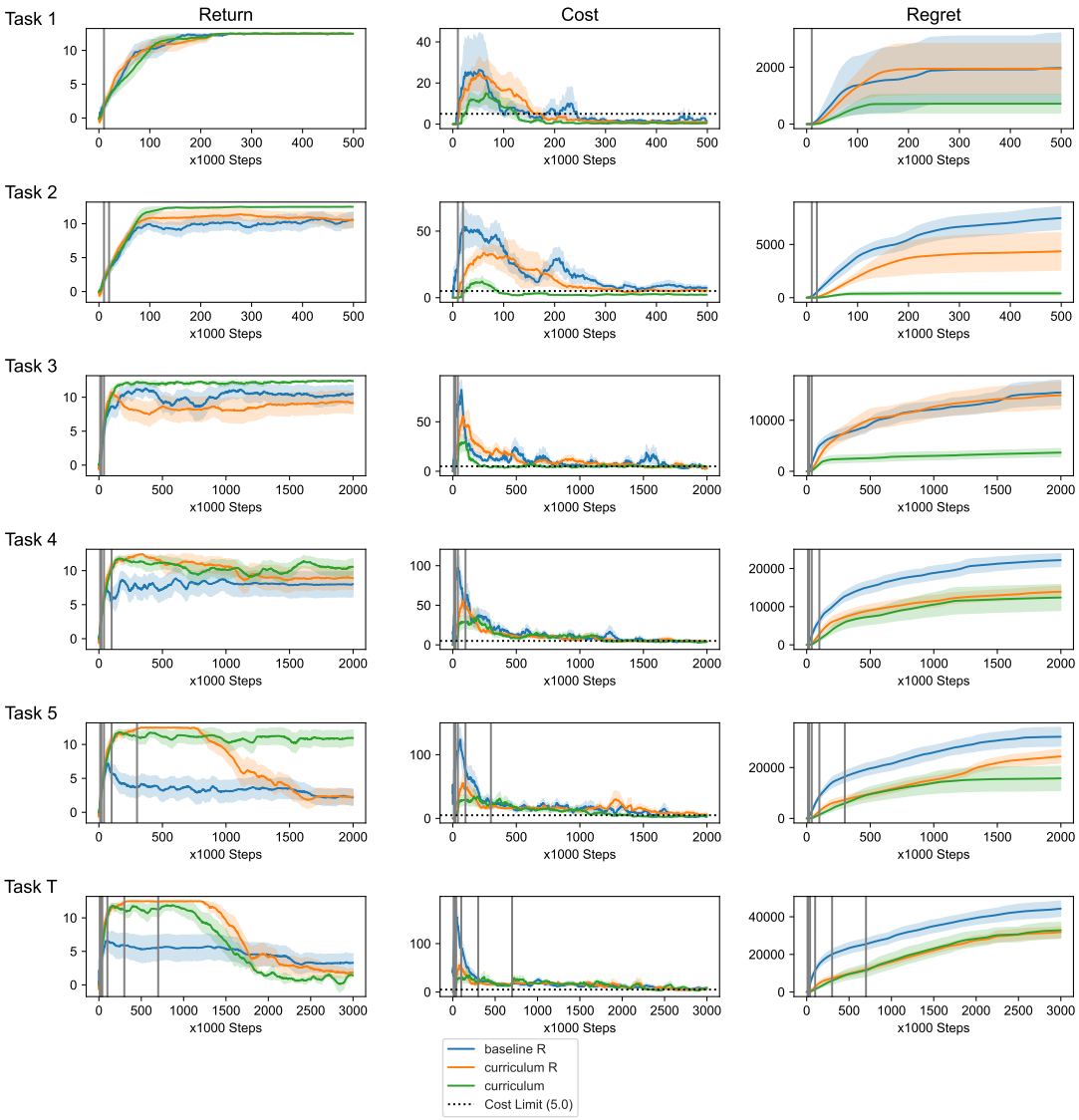


Figure 6.9: Training performance curves of the PPOLag agents, comparing baseline with curriculum progression on the reward-based curriculum. The shaded area indicates the standard error. To also compare to the progression of our main curriculum, the performance of this method is also included. However, the only task that they share with the reward-based curriculum agents is the target task. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents.

reward baseline also becomes larger in later tasks, suggesting that it indeed helps with keeping a high return. In the target task, the return is even higher than the standard curriculum. The performance during evaluation can be found in Appendix B.2.

The specific intermediate tasks may only have a small effect on final performance. When comparing the progression between the different curricula, we can see that in terms of regret the main curriculum is much better at the start and this gap starts to close until at the target task the regret is almost the same. However, this is mainly because of the fact that the starting tasks for the main curriculum are easier than the reward curriculum. In general, the progression through the curriculum is quite similar for both types and it is interesting how similar their performance

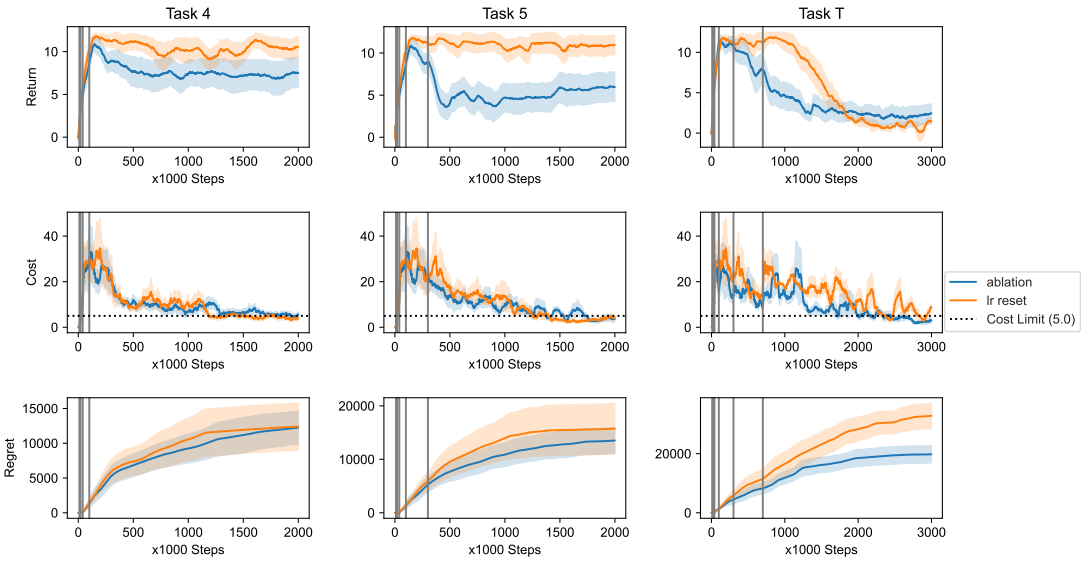


Figure 6.10: Results showing the effect of resetting the learning rate when changing a task. Ablation refers to a curriculum agent that does not reset the learning rate and lr reset refers to a curriculum agent that does reset the learning rate. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents.

is when looking at the target task. This shows that perhaps even the specific intermediate tasks do not matter all that much and that it is better than not using a curriculum.

6.3.4 Learning rate reset ablation

The goal of this experiment is to show the benefit of resetting the learning rate after a task change in the curriculum, such that we can further address **RQ1b**.

Setup

For this purpose, we designed the following experiment. We trained two types of PPOLag agents, one that resets the learning rate after a task change and one that does not. We did this with the PPOLag algorithm as this is one of the best-performing algorithms. These agents were trained with three different target tasks, namely task 4, task 5 and the target task. Each agent had 10 seeds and was trained for 2000 epochs for tasks 4 and 5, and 3000 epochs for the target task.

Results

Figure 6.10 shows the learning curves of the two types of agents. The columns represent the performance metrics and the rows represent the target tasks. Each learning curve contains the value for the respective metric on the y-axis and the epochs on the x-axis. The dashed line indicates the cost limit and the vertical grey lines indicate the epochs where a task change occurs. A similar figure for the evaluation results can be found in Appendix B.2.

Analysis

Resetting the learning rate gives a boost in return at the cost of a slightly larger regret. Figure 6.10 shows that for tasks 4 and 5 the return is higher when the learning rate is reset and for the target task the return is only slightly lower. However, neither seems to be

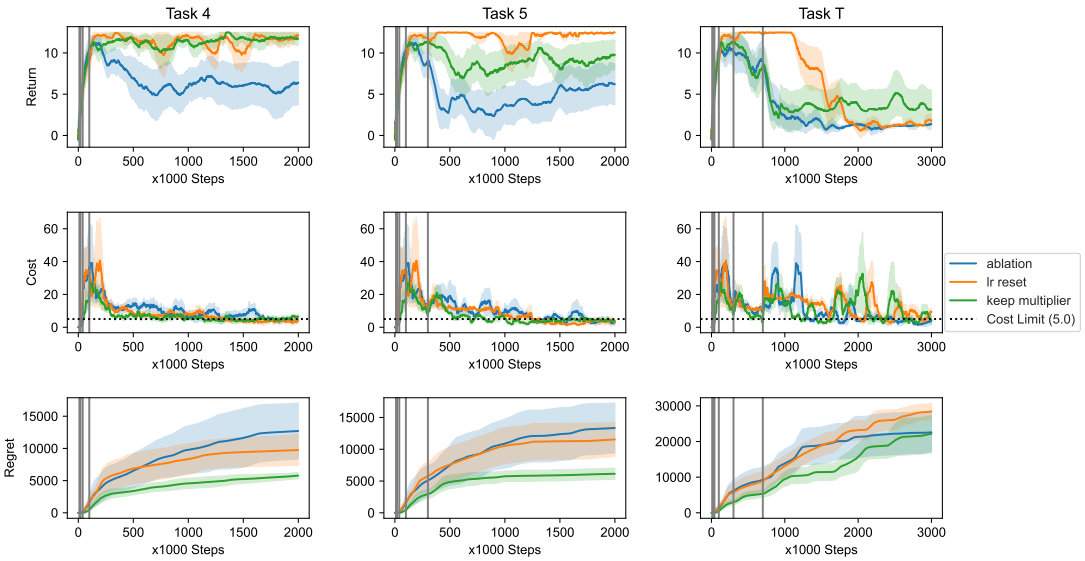


Figure 6.11: Results showing the effect of resetting the learning rate and keeping the Lagrangian multiplier when changing a task. Ablation refers to a curriculum agent that does not reset the learning rate or Lagrangian multiplier, lr reset refers to a curriculum agent that resets the learning rate and the Lagrangian multiplier and keep multiplier refers to a curriculum agent that resets the learning rate and not the Lagrangian multiplier. The shaded area indicates the standard error. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents.

able to solve the target task successfully. The regrets for tasks 4 and 5 are only slightly larger compared to the ablation, but more unstable as the standard error is larger. For the target task, the regret is much larger, which can be explained by the fact that also the Lagrangian multiplier is reset. Therefore even though the learning rate is still high, the Lagrangian multiplier is still low. Therefore the regret is only significantly larger when the agent is not able to complete the task due to its difficulty.

Due to the shown benefits, we conclude that resetting the learning rate is beneficial for the performance of the curriculum agents.

6.3.5 Lagrangian multiplier initialization ablation

The goal of this experiment is to show the benefit of not resetting the Lagrangian multiplier when we reset the learning rate, as to extend our answer to **RQ1b**. We saw from previous results that, especially in the target task, the algorithms using a curriculum seemed to have an unnecessarily high cost when changing tasks. This is explained by the fact that the Lagrangian multiplier is reset, along with the learning rate. Therefore we decided to try keeping the multiplier throughout the tasks to see whether this would indeed solve this issue.

Setup

To find out the effectiveness of this approach we designed the following experiment. We trained three types of PPOLag agents, one that resets the learning rate and Lagrangian multiplier after a task change, another that does neither and one that resets the learning rate, but not the Lagrangian multiplier. These agents were trained with the PPOLag algorithm on three different target tasks, namely task 4, task 5 and the target task. Each agent had 5 seeds and was trained for 2000 epochs for tasks 4 and 5, and 3000 epochs for the target task.

Results

Figure 6.11 shows the learning curves of the three types of agents. The columns represent the performance metrics and the rows represent the target tasks. Each learning curve contains the value for the respective metric on the y-axis and the epochs on the x-axis. The dashed line indicates the cost limit and the vertical grey lines indicate the epochs where a task change occurs. A similar figure for the evaluation results can be found in Appendix B.2.

Analysis

Keeping the Lagrangian multiplier significantly decreases the regret, while having only a slightly lower return. Figure 6.11 shows that the ablation agents have both a worse or about the same return and regret as the other two and thus demonstrate the benefit of using these two extensions. Keeping the Lagrangian multiplier also has some benefits over resetting it, where the main advantage is that the regret is much lower. Still, the return is about as high, or even better as resetting the learning rate and multiplier. On top of this, the difference in return on the target task can be explained by the fact that for the first time, there was a seed that managed to solve the target task safely.

With all of these benefits and with a special eye on the fact that keeping the Lagrangian multiplier resulted in a single case that solved the target task, we conclude that this is a very helpful extension.

6.4 Adaptive Curriculum

These experiments are focused on answering **RQ2** and thus seeing how safe an adaptive curriculum is. We first perform some hyperparameter tuning on the β and κ parameters that we discussed in Section 5.2. Afterward, we compare their performance on each task with agents that use a static curriculum and baselines that do not use a curriculum. This is to study the effectiveness of adaptive curriculum learning compared to the static curriculum and state of the art in safe RL, thus addressing **RQ2b**.

6.4.1 Tuning adaptive parameters

The goal of this experiment is to have a good version of our adaptive curriculum framework such that we can properly compare it with the static curriculum and baselines.

Setup

To find good hyperparameters for our adaptive curriculum framework, we designed the following experiment. We trained several PPOLag agents using our adaptive curriculum with different combinations of β and κ. For β we chose the values $0.5, 1.0$ and 1.5. As β is the factor of the cost limit that should be satisfied for κ successful epochs before going to the next task, a value of 0.5 is very strict as it requires a very low cost to be maintained and a value of 1.5 is thus very lenient. We did something similar for the number of successful epochs needed κ, which took the values 5, 10 and 20, where 5 is very lenient as it requires few such epochs and 20 is much more strict. We define a successful epoch to be an epoch where at least one episode has been completed by reaching the goal while the expected cost is less than a factor β of the cost limit. This results in nine unique combinations for the agents. We trained them for 3000 epochs with the full curriculum. All of the agents were tested with 10 seeds.

Since the ablation studies showed that resetting the learning rate and keeping the Lagrangian multiplier on a task change is beneficial, we used this method for all of the adaptive experiments.

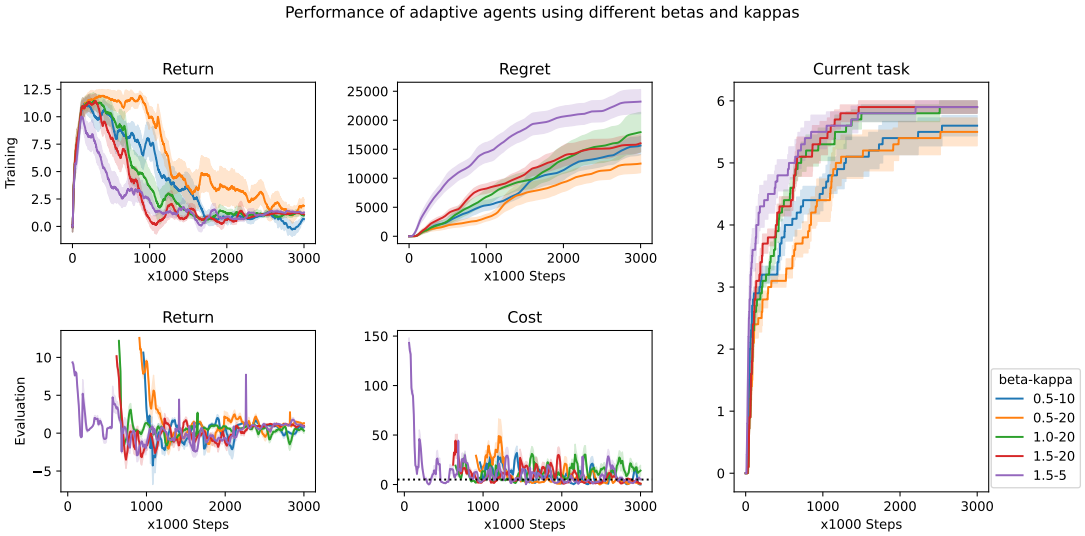


Figure 6.12: Performance of the best 5 combinations of β and κ with their average task progression. The shaded area indicates the standard error. The evaluation metrics have been smoothed by taking the moving average over the last 25 evaluations.

Results

Figure 6.12 shows the results of the five best combinations from this experiment. We have chosen the five parameter combinations to be the best, based mainly on their regret and task progression. It consists of five learning curves, which all have the epochs on the x-axis and their respective metric on the y-axis. The learning curves in the first column show the return throughout training for training and evaluation. As we only include the evaluations of the agents when they reach the target task, the evaluation results are different from before. Here the amount of seeds per epoch can differ as different seeds reach the target task at different epochs. The curves in the second column show the training regret and evaluation cost. The cost is used instead of the regret for the evaluation as the evaluation regret curves behave differently. This is due to the different amounts of seeds per epoch and can thus cause the regret curve to go down when a new seed with low costs reaches the target task. The final column consists of only the average task progression per combination and thus shows how quickly each of the combinations moved through the curriculum. A figure showing the task progression of the individual seeds can be found in Appendix B.3.

Analysis

The performances of the different combinations are very similar. The returns, regrets and task progressions of these five combinations in Figure 6.12 are all very close to one another. This small variation in results is accentuated by the fact that the two extremes are both in the top 5, namely the most lenient combination, $\beta = 1.5$, $\kappa = 5$, and the most strict combination, $\beta = 0.5$, $\kappa = 20$. It can also be seen in their task progressions that the lenient combination switches very quickly between tasks, while the stricter version switches much more slowly. Still, the lenient combination at some point is surpassed by another combination, meaning that the most lenient combination is not necessarily the fastest in task progression. Thus it may eventually be faster to stay on the earlier tasks a little longer such that future tasks are slightly easier. The contrast between the most lenient and strict combinations can also be seen in the return and regret curves. The lenient combination is the first to fall off in return and has one of the largest regrets, but always reaches the target task, while the strictest combination is the last to fall off in return and has one of the lowest regrets, but struggles to reach the target task.

The combination with the highest return and lowest regret does not often reach the target task. The combination that stands out the most is $\beta = 0.5$, $\kappa = 20$, since its return during training is consistently the highest. Moreover, it has the lowest regret out of all of the parameter combinations. However, this is misleading, as it does not indicate its performance on the target task. As can be seen from the task progression curves in Figure 6.12 and in Appendix B.3, there are several cases in which the agent does not even reach the target task. These cases skew the curves to look better and we can see this in the evaluation results, where there is no clear performance advantage over the other hyperparameter combinations. A similar case is that of $\beta = 0.5$, $\kappa = 10$, which also cannot reach the target task consistently and is almost as strict.

Choosing the best combination is difficult as there is not one that stands out as being generally better than the rest. However, the most important property is that the combination is able to reach the target task consistently. If agents do not reach the target task, then the curriculum is not useful. Since the ones with $\beta = 0.5$ do not have this, we only consider the other three combinations. Since $\beta = 1.5$, $\kappa = 5$ has the worst regret and a very bad return, we also do not consider this combination. The last two combinations are very close in their performance on the target task and when looking at their performance on the intermediate tasks, $\beta = 1.0$, $\kappa = 20$ has a better return. Therefore we chose $\beta = 1.0$, $\kappa = 20$ to use in our adaptive experiments, as it regularly reaches the target task, had a good regret on the target task and its performance on the intermediate tasks is better than many of the other parameter combinations.

6.4.2 Comparing to baseline and static curriculum

The focus of this experiment is to find out how well an adaptive curriculum agent performs compared to a static curriculum agent, as to answer **RQ2b**.

Setup

To achieve this goal we set up the following experiment, which is very similar to the one from Section 6.3.2. We used the algorithms CUP, FOCOPS and PPOLag as they performed well in the previous experiments. PPOEarlyTerminated was included as it is similar to the state of the art. For these four algorithms, we trained adaptive curriculum learning agents and curriculum learning agents. Both types were trained with resetting the learning rate and keeping the Lagrangian multiplier on a task change.

Similarly to Section 6.3.2, we trained the agents up to each task in the curriculum as if they were the target task. We decided to train on tasks 1 and 2 for 500 epochs, on tasks 3 and 4 for 2000 epochs, and on task 5 and the target task for 3000 epochs. All of the settings were repeated with 10 seeds. We trained on task 5 for 3000 epochs, since we saw during tuning that the agents did not always reach task 5 within 2000 epochs.

We reused the data from Section 6.3.2 for the baseline agents to give more perspective on the performance of the adaptive curriculum agents. We could not directly reuse the data for task 5, as those baseline agents were trained for 2000 epochs. Therefore, we retrained the baseline agents using CUP, FOCOPS, PPOLag, PPOEarlyTerminated, PPO and CPO with task 5 as the target task. These were trained for 3000 epochs and repeated with 10 seeds.

Results

Figure 6.13, Figure 6.14 and Figure 6.15 show the results of this experiment.

Figure 6.13 is a grid of radar charts, where the columns represent the target tasks and the rows represent the performance metrics. Each radar chart consists of four axes representing the four algorithms that were tested with a static curriculum, with an adaptive curriculum and without a curriculum. The values on each axis show the average performance at the end of training on that metric for that algorithm on that target task. These values are also connected to give an impression of the performance of the three types of agents over all algorithms. This means that when the area of the curriculum agents is fully within the area of the adaptive curriculum agents

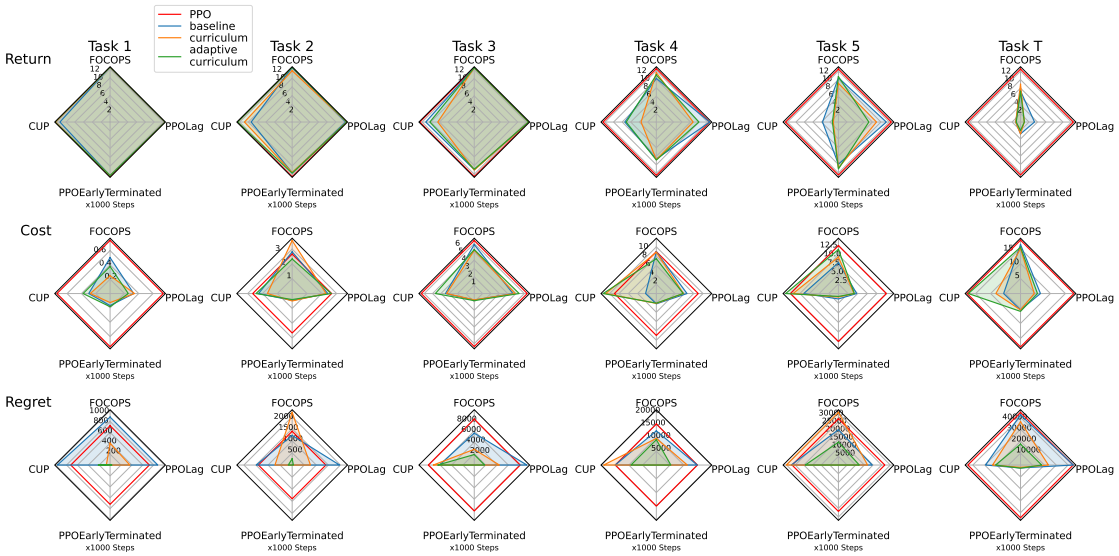


Figure 6.13: Radar charts showing for each task other than 0 what the difference in training performance is between the baseline, the curriculum and adaptive curriculum agents. The performance of the PPO agent is included as a reference point of what is the maximum value allowed for each of the metrics. For return we consider it better to have a larger area in the spider plot and for cost and regret we consider it better to have a smaller area.

for the return, the adaptive curriculum agents in general outperform the curriculum agents with regards to return on that target task and vice versa. For the cost and regret, this is reversed as higher values represent worse performance. The average performance of the PPO agents is also included to give perspective on whether the algorithms performed better or worse than PPO. A similar figure for the evaluation results can be found in Appendix B.3.

Figure 6.14 shows the learning curves of the three types of PPOLag agents. The columns represent the performance metrics and the rows represent the target tasks. Each learning curve contains the value for the respective metric on the y-axis and the epochs on the x-axis. The dashed line indicates the cost limit and the vertical grey lines indicate the epochs where a task change occurs. A similar figure for the evaluation results can be found in Appendix B.3.

Figure 6.15 shows the task progression of the individual seeds for the four algorithms that were trained with an adaptive curriculum. The x-axis indicates the epochs and the y-axis indicates the task the agents are training on during each epoch. As the lines of the seeds are transparent, a more opaque line indicates more seeds at that task and epoch.

Analysis

The regret of the adaptive curriculum agents is generally lower than that of the baseline and curriculum agents. From the results in Figure 6.13 we can see that the area of the adaptive curriculum in all of the radar charts is almost completely encapsulated in the areas of the other two types and the PPO agents. This is however not reflected in the end costs, as the costs of the adaptive curriculum agents are about the same as the others, except when the algorithm is CUP. In the case of CUP, it has a relatively high cost compared to the baseline and curriculum agents. PPOEarlyTerminated behaves consistently safe for all tasks during training, but just as with the previous experiments, its performance during evaluation is considerably worse as it has the largest regret on the more difficult tasks.

Adaptive curriculum agents have a slightly lower return than the other two types. The return of the adaptive curriculum agents is about the same or marginally smaller than that of the baseline and curriculum agents. Out of all of the algorithms only FOCOPS manages to get

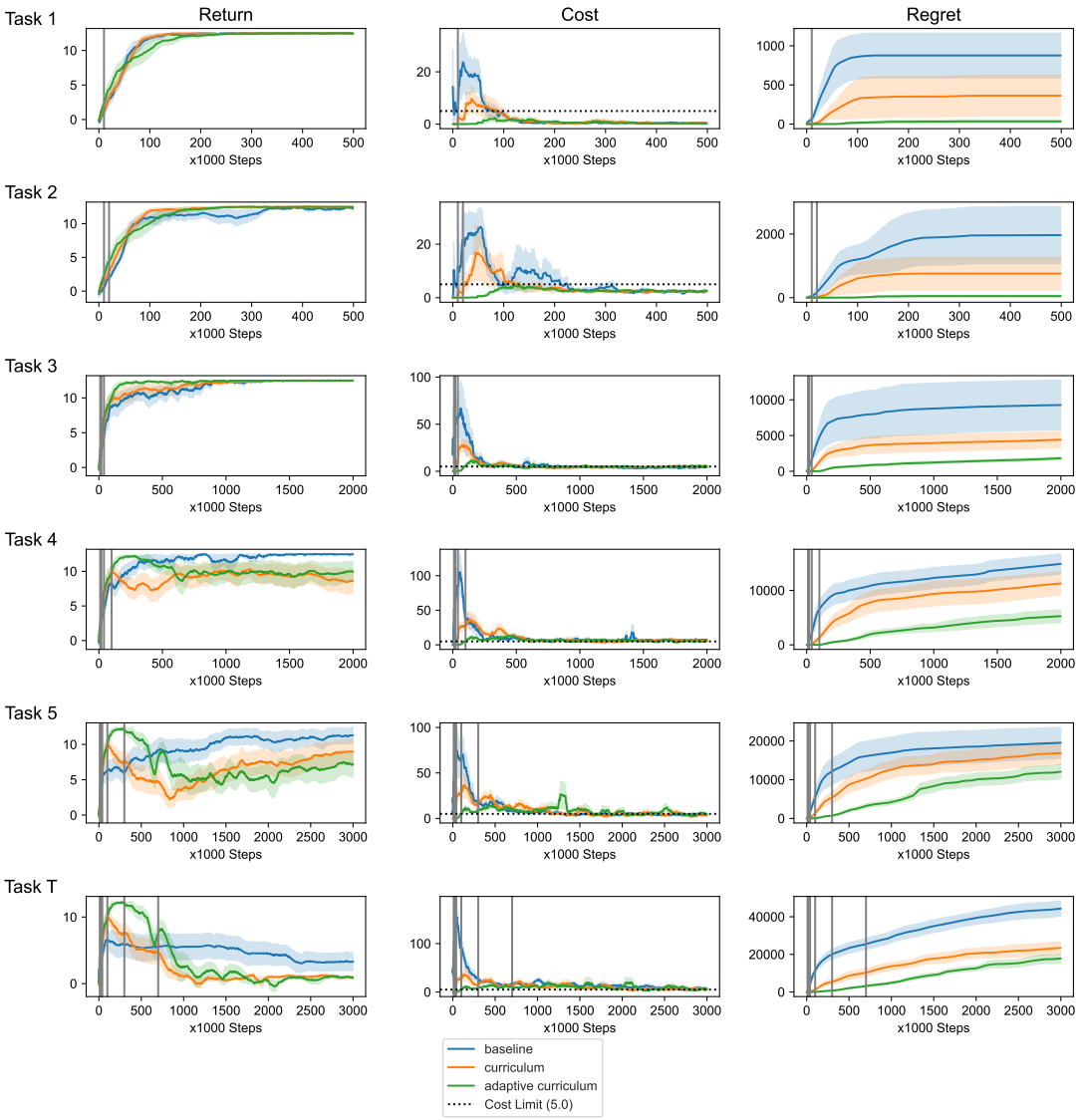


Figure 6.14: Training performance curves of the PPOLag agents, comparing baseline, curriculum and adaptive curriculum progression. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents.

a high return on the target task. However, as can be seen from its cost and regret values, the FOCOPS agent is not safe when it has high return on the final task. After some investigation, we found that indeed the strategy of the agents with a high return was to go straight to the goal and incur large costs. This lack of learning is most likely due to the fact that the FOCOPS algorithm, just like the CUP algorithm, puts an upper limit on the Lagrangian multiplier such that it cannot become too large.

For PPOLag the regret is lower for the adaptive agents as it further decreases spikes in the cost. In Figure 6.14 the difference between regrets is very clear and the adaptive curriculum is consistently the lowest. Especially in the earlier tasks we can see that the adaptive curriculum has the largest impact. In more difficult tasks, this difference gets smaller, but the regret of the adaptive curriculum still remains the lowest. Waiting for the agent to behave safely on the earlier tasks before going to the next task is thus a good strategy to limit regret later

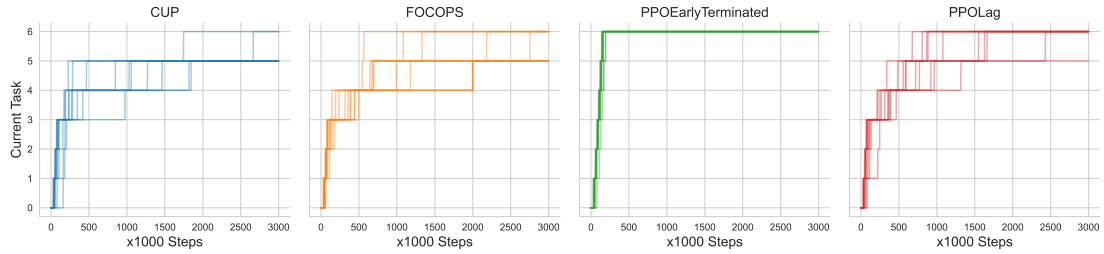


Figure 6.15: Task progression of the different safe RL algorithms used with an adaptive curriculum.

on. Still, this lower regret has a trade-off in these later tasks, where the return of the adaptive curriculum agents is slightly lower compared to the other two versions. The difference in regrets is very much visible in the cost curves. We have already seen that a curriculum dampens the spike that a baseline agent has, but the adaptive curriculum does the same with the curriculum agents. The adaptive agents dampen the spike in costs even more and thus get a headstart in regret right at the beginning.

Only PPOLag and PPOEarlyTerminated consistently reach the target task. It is important to note that the results of the adaptive curriculum agents show an average over the seeds that were trained to reach that task. Thus it is important to also look at how many seeds reached each task and at what times. In Figure 6.15 we can see that not all algorithms managed to consistently reach the target task, namely CUP and FOCOPS. CUP has eight seeds that stayed at task 5 and FOCOPS has five such seeds. Thus, the results for these algorithms on the target task consist of only a few seeds that reached the target task and many that were still on task 5. On the other hand, we have PPOLag, which had all but one seed reach the target task. This is consistent with the tuning that we did and this difference in task progression between algorithms may indicate that the parameters should be tuned for the specific algorithm. PPOEarlyTerminated also consistently reaches the target task and it uses close to the minimum amount of epochs needed for each task. This is because it does not go over the cost limit by definition and thus only needs 20 epochs before it can advance.

Tasks 0 through 3 switch to the next task at roughly the same epochs across all four algorithms. The start of the task progression curves of all four algorithms are almost identical, implying that the first 3 tasks need about the same time regardless of agent before going to the next task. This is less true for the change from task 3 to task 4, but also here there seems to be a general area, where this task change occurs, namely around 350 epochs. This is already a stark difference between our static curriculum, where this change occurred at epoch 100. After task 3, we can see that there are large differences between when the seeds go to the next task, although the ones from PPOLag are much closer together than those from CUP and FOCOPS.

Chapter 7

Final Remarks

In this chapter, we are going to list the main conclusions of our research. After this we will list some potential extensions of our method that could be interesting for future research.

7.1 Conclusions

To draw our main conclusions we will again look at our research questions to see what the contributions are of our research.

7.1.1 Using a static curriculum to teach a safe RL agent

Here we will try to answer **RQ1** and its subquestions.

We have experimented with different types of safe RL algorithms to see which ones work best with curriculum learning. We ruled out many possible algorithms based on theory, as they did not possess the characteristics that we found important, namely adapting quickly, generalizing well and converging quickly. From the resulting candidates, we saw during the experiments that CRPO, PCPO and PPOEarlyTerminated did not work well with our curriculum. It is difficult to determine why they did not work well, but it may have to do with the fact that these algorithms struggle with this specific environment. Still, these were outperformed by the algorithms that worked better with both our curriculum and without. Upon further investigation, our tuned PPOLag agents performed the best with a curriculum and in many cases outshined their baseline counterparts, while FOCOPS came close and CUP was less effective. Therefore to address **RQ1a**, the best safe RL algorithms for curriculum learning seem to be PPOLag and FOCOPS.

There are many curriculum learning techniques and we have investigated a few of them with different levels of automation. We argued that the guidelines for creating a curriculum by Peng et al. [2018] were most useful for us and that a combination of the methods by Asada et al. [1996] and by Wu and Tian [2016] results in a general framework for adaptive task sequencing. Our experiments showed that our chosen methods worked well with our best algorithms. It was even the case that when choosing a different curriculum the performance was about the same, indicating that the chosen curriculum learning methods are proper methods to use for safe RL, which answers **RQ1b**.

We compared the performance of multiple agents with and without a curriculum. The overall result was that a curriculum was beneficial to the performance of the agent. Although it is usually better only in terms of regret, it is still on par with the baselines in terms of return. Still, there were cases where also the return was much higher. In many of the learning curves, it was visible that the cost spikes from the baselines were dampened by a curriculum, often being a major reason for why the regret was lower. On top of all of this, the agents using a curriculum were the only ones who had a success on the difficult target task. We also compared to PPO as a limit on the cost and regret, as it does not take safety into account. The curriculum agents often stayed below

this limit, while the baselines where much closer or even above this limit. In a similar fashion, we included CPO only as a baseline, but the performance of these agents was very poor and did not give many insights. We also included an algorithm that is inspired by the methods from Eysenbach et al. [2018] and Turchetta et al. [2020], namely PPOEarlyTerminated. PPOEarlyTerminated is also based on the idea that we want to stop the agent before going over the cost limit and making it learn in this way. On top of this the reset controller from the literature, which is trained together with the agent, is in this case replaced by a perfect oracle that always stops the agent right after going over the cost limit. Although this method seemed to perform reasonably well during training, during evaluation it was clear that the regret was the largest of all of the algorithms. Therefore, we conclude for **RQ1c** that our framework of making a curriculum based on small changes in the environment is safer than state of the art, with similar if not better return.

After discussing all of this evidence we believe that a proper static curriculum through changes in the environment helps with learning a safe RL agent.

7.1.2 Creating an adaptive curriculum

In this section, we address **RQ2** and its subquestions.

Similarly to our response to **RQ1b**, we looked at several methods for adaptive task sequencing in safe RL. As the two methods by Asada et al. [1996] and by Wu and Tian [2016] seemed quite similar, we decided to make a simple combination of them that can be extended to represent either method. This method is flexible in the way that the condition for updating the distribution is computed and in the type of distribution that is used. For our experiments we chose to count successful safe epochs and use a degenerate distribution that changes to the next task in the sequence upon an update. We also tuned some parameters to reach a technique that allowed for adaptive task sequencing for curriculum learning in safe RL and thus represents one possible class of techniques for the answer of **RQ2a**

We performed experiments to show the effect of an adaptive curriculum, compared to a static curriculum. The results showed that in general, the regret is slightly smaller when using an adaptive curriculum, at the cost of a slightly lower return for the more difficult tasks. Nonetheless, this strategy seemed to perform extremely well for the easier tasks, where the regret was significantly smaller and the return was the same. As the static curriculum was already an improvement on the baseline that does not use a curriculum, the adaptive curriculum is also an improvement on the baseline. When comparing to Eysenbach et al. [2018] and Turchetta et al. [2020], we can say the same that we did for our answer to **RQ1c**, namely that our adaptation of these methods performed well during training, but very poor during evaluation. Therefore, we conclude for **RQ2b** that extending our framework to include adaptive task sequencing does indeed result in safer learning.

All in all, we have shown evidence that we can indeed create a curriculum learning algorithm with adaptive task sequencing that is also safe.

7.2 Limitations and Future Work

A major limitation to our approach is that it is non-trivial to create the tasks for a curriculum and it can take quite some time for more difficult target tasks. One solution to this problem would be to automate task generation. Extending our approach to also generate the tasks to have a fully automatic curriculum has promising prospects. We already saw that changing from a static curriculum to an adaptive one improves on the safety. We have already listed some options in Section 4.2.4 for automating our method that should not be too difficult to add on top of our approach. In particular, we consider the method by Klink et al. [2020, 2021], as their approach can easily be added on top of our framework.

Another limitation is that we have not fully figured out what properties are important for an algorithm to make the most use of a curriculum. The properties we came up with, were not indicative of their performance, so perhaps there are more complex properties that influence the

performance of an algorithm using a curriculum. These insights could perhaps lead to a safe RL algorithm that is specifically built for a curriculum and could have much better performance.

One potential limitation is that we focus our curriculum on changing the hazards. This could be disadvantageous for the agent as tactics that used to work with less or more easily avoidable hazards could become hard to pull off without a large cost. We have already seen in Section 6.3.2 that most strategies learned for task 5 do not work well for the target task and adjusting them seems to be difficult for the agents. Therefore it would be insightful to compare our approach with curricula that have a different focus. It would be interesting to see the effects of using, for example, a curriculum of goals that guide it around the hazards.

An interesting idea for future work would be to change the cost limit throughout the curriculum, making it part of the curriculum, as we saw in the results in Section 6.2.2. This could allow the agents to more gradually approach a safe policy instead of giving up on the more difficult tasks. As this also has the benefit of being closer to real applications, due to the fact that in those cases there is usually a total budget, this would allow for easier integration into such applications. On top of the cost limit, other variables could be included in the curriculum, like the reward function, to allow for more flexibility, especially if they are used with an adaptive curriculum.

Bibliography

J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 22–31. Proceedings of Machine Learning Research, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/achiam17a.html. 3, 15, 28

S. Ainsworth, M. Barnes, and S. Srinivasa. Mo'states mo'problems: Emergency stop mechanisms from observation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/966eaa9527eb956f0dc8788132986707-Paper.pdf. 16

E. Altman. Constrained Markov decision processes with total cost criteria: Lagrangian approach and dual linear program. *Mathematical methods of operations research*, 48:387–417, 1998. 14

E. Altman. *Constrained Markov Decision Processes*. Routledge, 1999. 2, 5, 10

D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016. 1

M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine learning*, 23:279–303, 1996. 18, 22, 25, 26, 27, 52, 53

A. Basu, T. Bhattacharyya, and V. S. Borkar. A learning algorithm for risk-sensitive cost. *Mathematics of Operations Research*, 33(4):880–898, 2008. doi: 10.1287/moor.1080.0324. URL https://doi.org/10.1287/moor.1080.0324. 5

L. Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com. 60

Y. Chow, O. Nachum, A. Faust, E. Duenez-Guzman, and M. Ghavamzadeh. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*, 2019. 15, 31

G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021. 4

Y. Efroni, S. Mannor, and M. Pirotta. Exploration-exploitation in constrained MDPs. *arXiv preprint arXiv:2003.02189*, 2020. 6, 10

T. Eimer, A. Biedenkapp, F. Hutter, and M. Lindauer. Self-paced context evaluation for contextual reinforcement learning. In *International Conference on Machine Learning*, pages 2948–2958. Proceedings of Machine Learning Research, 2021. 20

B. Eysenbach, S. Gu, J. Ibarz, and S. Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. In *International Conference on Learning Representations*, 2018. 1, 3, 7, 8, 16, 21, 53

C. Florensa, D. Held, X. Geng, and P. Abbeel. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning*, pages 1515–1528. Proceedings of Machine Learning Research, 2018. 20

F. Foglino, C. C. Christakou, R. L. Gutierrez, and M. Leonetti. Curriculum learning for cumulative return maximization. In *IJCAI*, pages 2308–2314. ijcai.org, 2019a. 17, 22

F. Foglino, C. C. Christakou, and M. Leonetti. An optimization framework for task sequencing in curriculum learning. In *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 207–214, Aug 2019b. doi: 10.1109/DEVLRN.2019.8850690. 17, 22

F. Foglino, M. Leonetti, S. Sagratella, and R. Seccia. A gray-box approach for curriculum learning. In H. A. Le Thi, H. M. Le, and T. Pham Dinh, editors, *Optimization of Complex Systems: Theory, Models, Algorithms and Applications*, pages 720–729, Cham, 2020. Springer International Publishing. ISBN 978-3-030-21803-4. 17, 22

J. Garcıa and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015. 1, 5, 10

C. Gehring and D. Precup. Smart exploration in reinforcement learning using absolute temporal difference errors. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '13, page 1037–1044, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450319935. 5

A. Hallak, D. Di Castro, and S. Mannor. Contextual Markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015. 22

J. Ji, B. Zhang, J. Zhou, X. Pan, W. Huang, R. Sun, Y. Geng, Y. Zhong, J. Dai, and Y. Yang. Safety Gymnasium: A unified safe reinforcement learning benchmark. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023a. URL https://openreview.net/forum?id=WZmlxIuIGR. 28, 60

J. Ji, J. Zhou, B. Zhang, J. Dai, X. Pan, R. Sun, W. Huang, Y. Geng, M. Liu, and Y. Yang. OmniSafe: An infrastructure for accelerating safe reinforcement learning research. *arXiv preprint arXiv:2305.09304*, 2023b. 28, 60

Y. Kadota, M. Kurano, and M. Yasuda. Discounted Markov decision processes with utility constraints. *Comput. Math. Appl.*, 51(2):279–284, jan 2006. ISSN 0898-1221. doi: 10.1016/j.camwa.2005.11.013. URL https://doi.org/10.1016/j.camwa.2005.11.013. 1, 5, 14

D. Kamran, T. D. Simão, Q. Yang, C. T. Ponnambalam, J. Fischer, M. T. Spaan, and M. Lauer. A modern perspective on safe automated driving for different traffic dynamics using constrained reinforcement learning. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 4017–4023, 2022. doi: 10.1109/ITSC55140.2022.9921907. 5

F. Khan, B. Mutlu, and J. Zhu. How do humans teach: On curriculum learning and teaching dimension. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper_files/paper/2011/file/f9028faec74be6ec9b852b0a542e2f39-Paper.pdf. 16, 22

P. Klink, H. Abdulsamad, B. Belousov, and J. Peters. Self-paced contextual reinforcement learning. In *Conference on Robot Learning*, pages 513–529. Proceedings of Machine Learning Research, 2020. 20, 22, 53

P. Klink, H. Abdulsamad, B. Belousov, C. D'Eramo, J. Peters, and J. Pajarinen. A probabilistic interpretation of self-paced learning with applications to reinforcement learning. *The Journal of Machine Learning Research*, 22(1):8201–8252, 2021. 20, 22, 53

M. Kumar, B. Packer, and D. Koller. Self-paced learning for latent variable models. *Advances in neural information processing systems*, 23, 2010. 20

Y. Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017. 4

Y. Liu, J. Ding, and X. Liu. Ipo: Interior-point policy optimization under constraints. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):4940–4947, Apr. 2020. doi: 10.1609/aaai.v34i04.5932. URL https://ojs.aaai.org/index.php/AAAI/article/view/5932. 15

P. MacAlpine and P. Stone. Overlapping layered learning. *Artificial Intelligence*, 254:21–43, 2018. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2017.09.001. URL https://www.sciencedirect.com/science/article/pii/S0004370217301066. 16, 17, 22

J. A. Martín H. and J. de Lope. Learning autonomous helicopter flight with evolutionary reinforcement learning. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Computer Aided Systems Theory - EUROCAST 2009*, pages 75–82, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-04772-5. 1, 5

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. Proceedings of Machine Learning Research. URL https://proceedings.mlr.press/v48/mniha16.html. 13

A. Müller, P. Alatur, V. Cevher, G. Ramponi, and N. He. Truly no-regret learning in constrained MDPs. In *ICML 2024 Workshop: Foundations of Reinforcement Learning and Control – Connections and Perspectives*, 2024. 10

S. Narvekar and P. Stone. Learning curriculum policies for reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, page 25–33, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450363099. 18, 19, 22, 25

S. Narvekar, J. Sinapov, M. Leonetti, and P. Stone. Source task creation for curriculum learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, AAMAS '16, page 566–574, Richland, SC, 2016. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450342391. 19

S. Narvekar, J. Sinapov, and P. Stone. Autonomous task sequencing for customized curriculum design in reinforcement learning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, page 2536–2542. AAAI Press, 2017. ISBN 9780999241103. 19

S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *The Journal of Machine Learning Research*, 21(1):7382–7431, 2020. 1, 6, 7, 10, 16

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf. 4

B. Peng, J. MacGlashan, R. Loftin, M. L. Littman, D. L. Roberts, and M. E. Taylor. Curriculum design for machine learners in sequential decision tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(4):268–277, 2018. doi: 10.1109/TETCI.2018.2829980. 16, 22, 23, 52

J. Platt and A. Barr. Constrained differential optimization. In D. Anderson, editor, *Neural Information Processing Systems*, volume 0. American Institute of Physics, 1987. URL https://proceedings.neurips.cc/paper_files/paper/1987/file/a87ff679a2f3e71d9181a67b7542122c-Paper.pdf. 14

R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer. Automatic curriculum learning for deep RL: a short survey. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20, 2021. ISBN 9780999241165. 7

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994. 9

J. Ramírez, W. Yu, and A. Perrusquía. Model-free reinforcement learning from expert demonstrations: a survey. *Artificial Intelligence Review*, 55(4):3213–3241, 2022. 1, 5

A. Ray, J. Achiam, and D. Amodei. Benchmarking safe exploration in deep reinforcement learning, 2019. URL https://cdn.openai.com/safexp-short.pdf. 3, 14, 28, 36

J. Roy, R. Girgis, J. Romoff, P. Bacon, and C. J. Pal. Direct behavior specification via constrained reinforcement learning. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 18828–18843. PMLR, 2022. 5

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. Proceedings of Machine Learning Research. URL https://proceedings.mlr.press/v37/schulman15.html. 14, 15

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 13, 14

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. 4

T. D. Simão, N. Jansen, and M. T. J. Spaan. Alwayssafe: Reinforcement learning without safety constraint violations during training. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, page 1226–1235, Richland, SC, 2021. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450383073. 1, 5

A. E. Smith, D. W. Coit, T. Baeck, D. Fogel, and Z. Michalewicz. Penalty functions. *Handbook of evolutionary computation*, 97(1):C5, 1997. 5, 14

A. Stooke, J. Achiam, and P. Abbeel. Responsive safety in reinforcement learning by PID lagrangian methods. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9133–9143. Proceedings of Machine Learning Research, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/stooke20a.html. 14

H. Sun, Z. Xu, M. Fang, Z. Peng, J. Guo, B. Dai, and B. Zhou. Safe exploration by solving early terminated MDP. *arXiv preprint arXiv:2107.04200*, 2021. 16, 36

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 4, 10

A. Tamar, H. Xu, and S. Mannor. Scaling up robust mdps by reinforcement learning. *arXiv preprint arXiv:1306.6189*, 2013. 5

J. Tang, A. Singh, N. Goehausen, and P. Abbeel. Parameterized maneuver learning for autonomous helicopter flight. In *2010 IEEE International Conference on Robotics and Automation*, pages 1142–1148, 2010. doi: 10.1109/ROBOT.2010.5509832. 5

C. Tessler, D. Mankowitz, and S. Mannor. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074*, 05 2018. 14

M. Turchetta, A. Kolobov, S. Shah, A. Krause, and A. Agarwal. Safe reinforcement learning via curriculum induction. *Advances in Neural Information Processing Systems*, 33:12151–12162, 2020. 1, 2, 3, 8, 21, 53

A. Wachi, X. Shen, and Y. Sui. A survey of constraint formulations in safe reinforcement learning. *arXiv preprint arXiv:2402.02025*, 2024. 10

T. Walsh, D. Hewlett, and C. Morrison. Blending autonomous exploration and apprenticeship learning. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper_files/paper/2011/file/e034fb6b66aacc1d48f445ddfb08da98-Paper.pdf. 5

C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992. 13

Y. Wu and Y. Tian. Training agent for first-person shooter game with actor-critic curriculum learning. In *International Conference on Learning Representations*, 2016. 18, 22, 25, 26, 27, 52, 53

T. Xu, Y. Liang, and G. Lan. CRPO: A new approach for safe reinforcement learning with convergence guarantee. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11480–11491. Proceedings of Machine Learning Research, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/xu21a.html. 15, 20, 36

L. Yang, J. Ji, J. Dai, L. Zhang, B. Zhou, P. Li, Y. Yang, and G. Pan. Constrained update projection approach to safe policy optimization. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 9111–9124. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/3ba7560b4c3e66d760fbdd472cf4a5a9-Paper-Conference.pdf. 14, 20, 36

Q. Yang, T. D. Simão, N. Jansen, S. H. Tindemans, and M. T. Spaan. Reinforcement learning by guided safe exploration. In *ECAI 2023*, pages 2858–2865. IOS Press, 2023. 1, 5, 17

T. Yang, J. Rosca, K. Narasimhan, and P. J. Ramadge. Projection-based constrained policy optimization. In *ICLR*. OpenReview.net, 2020. 15, 20, 36

L. Zhang, L. Shen, L. Yang, S. Chen, X. Wang, B. Yuan, and D. Tao. Penalized proximal policy optimization for safe reinforcement learning. In *IJCAI*, pages 3744–3750. ijcai.org, 2022. 15

Y. Zhang, Q. Vuong, and K. Ross. First order constrained optimization in policy space. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15338–15349. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/af5d5ef24881f3c3049a7b9bfe74d58b-Paper.pdf. 14, 21, 36

W. Zhao, J. P. Queralta, and T. Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE, 2020. 4

Appendix A

Code Management

As RL does not need data in the conventional sense due to the fact that data is generated and used during training, there is no need for a dataset. Instead we use the framework Omnisafe Ji et al. [2023b] for setting up the environments and agents. The environments are in turn taken from the Safety Gymnasium Ji et al. [2023a] API, where it is possible to create custom environments that are needed to make curricula. However, as both of these libraries had to be adjusted throughout the project, we have forked both of the GitHub repositories to create our own versions. The GitHub repository of our main code and these forks are made public and can be found here: the main code, the Omnisafe fork and the Safety Gymnasium fork. We also use Weights & Biases Biewald [2020] to log our experiments and use some of the automatically generated graphs to acquire knowledge about the results also during training.

Appendix B

Figures

B.1 Hyperparameter Tuning

Here we give the full results of our hyperparameter tuning experiments on PPOLag, as discussed in Section 6.2.

Figure B.1 and Figure B.2 show two heatmaps that contain the full results of the experiment in Section 6.2.1, one for the baseline agents and one for the curriculum agents. On the x-axis we have the performance metrics for both training and evaluation. On the y-axis we have the 72 hyperparameter combinations, where the colors indicate which type of agent they were best for. The cells contain the value at the end of training for the combination of metric and combination and are colored relative to the other values in the column. The combinations are ordered on evaluation return from bottom to top.

The nine best parameters were chosen for the other tuning experiments based mainly on how high the evaluation return was, as a low cost or regret does not mean much if the agent still cannot perform the task. Still, cost and regret were the secondary criteria for deciding the best parameters.

Figure B.3 and Figure B.4 show two heatmaps from the experiment in Section 6.2.2, the first for the baselines and the second for the curriculum agents. On the x-axis we have the performance metrics for both training and evaluation. On the y-axis we have the hyperparameter combinations, where the colors indicate whether the last four parameters were only part of the nine best baseline parameters or whether they were also part of the best curriculum learning parameters from the experiment in Section 6.2.1. The cells contain the value at the end of training for the combination of metric and combination and are colored relative to the other values in the column. The combinations are ordered on evaluation return from bottom to top.

Table B.1 shows the evaluation results of the experiments from Section 6.2.3. The first column indicates for how many epochs the agents of each row were trained, where we have separated and italicized the ones that were trained for longer, although the versions where they were trained for 500 epochs are still included. The next four columns represent the values of the hyperparameters in the order: the Lagrangian multiplier initial value, the Lagrangian multiplier learning rate, the number of update iterations and the neural network size. The remaining columns give the average value at the end of training of the respective agents for both the baseline and curriculum versions and for each performance metric.

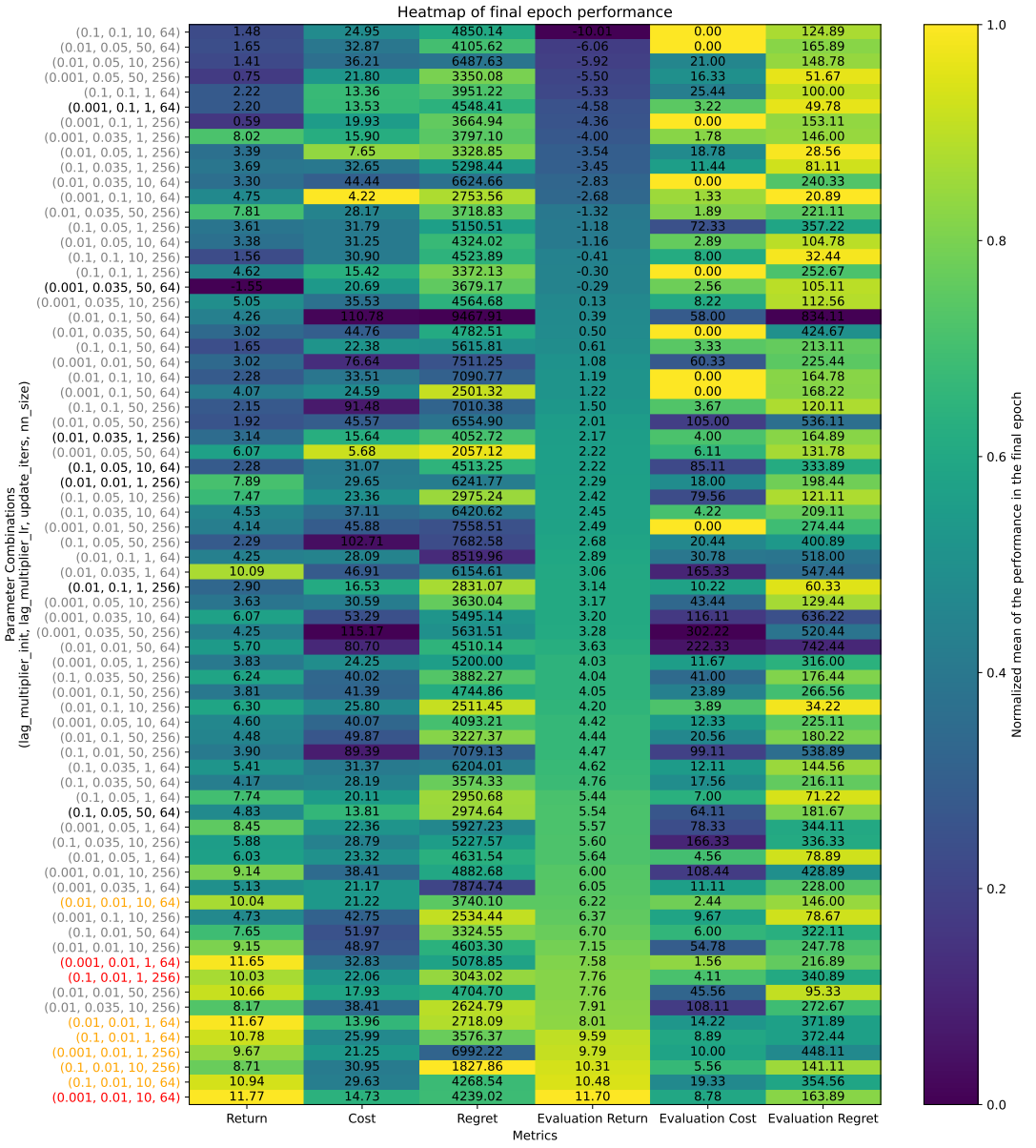


Figure B.1: Heatmap of the grid search experiments from section 6.2.1 of the baseline agents. As there is a large disparity in costs, the colors for these use the log values. The parameters are the Lagrangian multiplier initialization, Lagrangian multiplier learning rate, the number of update iterations and the amount of nodes in each of the two hidden layers of the actor and critic networks. For the ticks on the y-axis indicates that this was among the best for both the baseline and curriculum agents, indicates only for baseline agents and indicates only for curriculum agents.

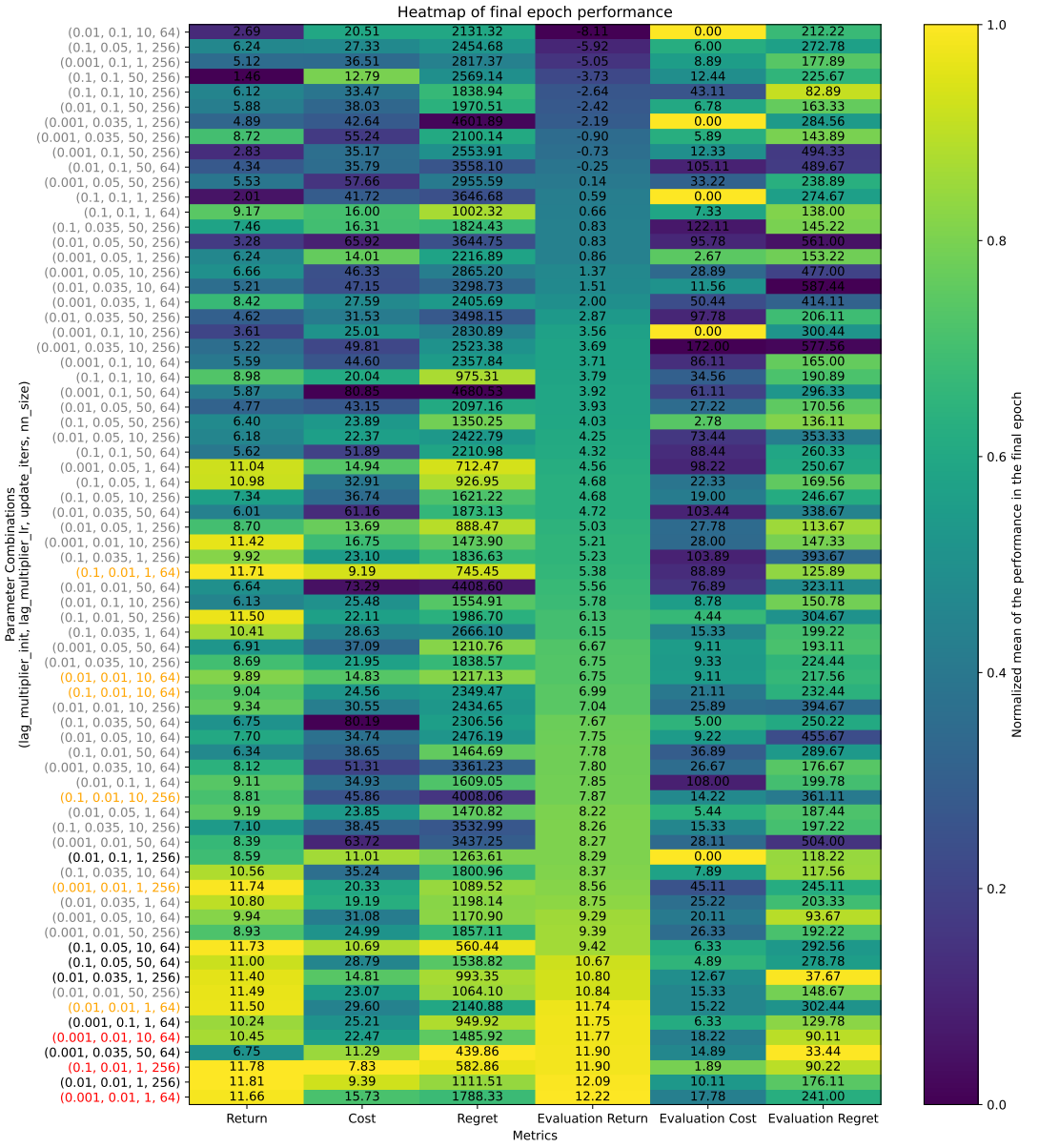


Figure B.2: Heatmap of the grid search experiments from section 6.2.1 of the curriculum agents. As there is a large disparity in costs, the colors for these use the log values. The parameters are the Lagrangian multiplier initialization, Lagrangian multiplier learning rate, the number of update iterations and the amount of nodes in each of the two hidden layers of the actor and critic networks. For the ticks on the y-axis ■ indicates that this was among the best for both the baseline and curriculum agents, ■ indicates only for baseline agents and ■ indicates only for curriculum agents.

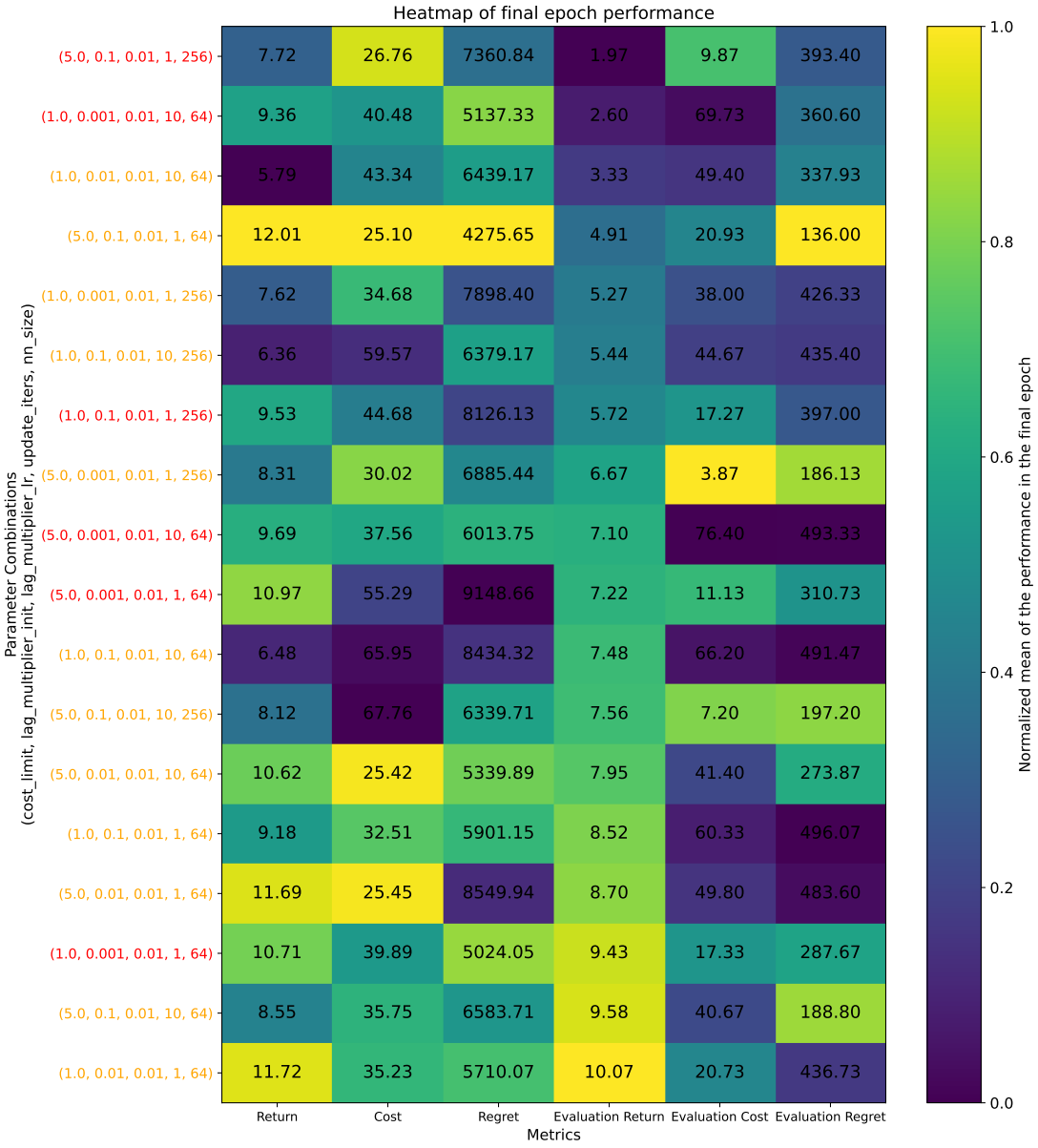


Figure B.3: Heatmap of the grid search experiments from section 6.2.2 of the baseline agents, focusing on different cost limits. As there is a large disparity in costs, the colors for these use the log values. The parameters are the Lagrangian multiplier initialization, Lagrangian multiplier learning rate, the number of update iterations and the amount of nodes in each of the two hidden layers of the actor and critic networks. For the ticks on the y-axis ■ indicates that this was among the best for both the baseline and curriculum agents and ■ indicates only for baseline agents.

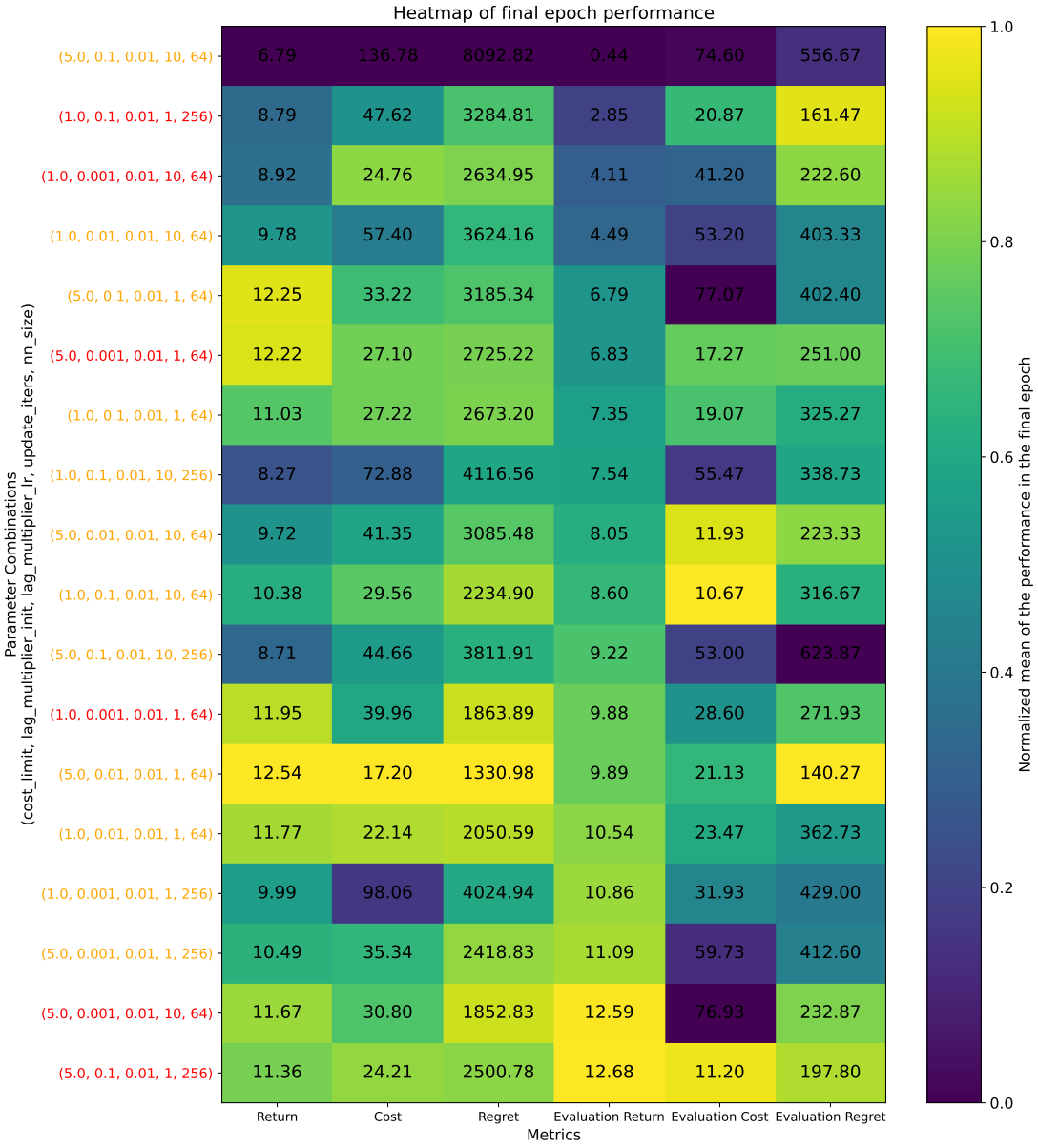


Figure B.4: Heatmap of the grid search experiments from section 6.2.2 of the curriculum agents, focusing on different cost limits. As there is a large disparity in costs, the colors for these use the log values. The parameters are the Lagrangian multiplier initialization, Lagrangian multiplier learning rate, the number of update iterations and the amount of nodes in each of the two hidden layers of the actor and critic networks. For the ticks on the y-axis ■ indicates that this was among the best for both the baseline and curriculum agents and ■ indicates only for baseline agents.

				Return		Cost		Regret	
LMI	LMLR	UI	NN	Base	Curr	Base	Curr	Base	Curr
0.001	0.01	1	64	12.24	12.24	17.12	16.82	1292	852
0.001	0.01	1	256	9.51	10.44	13.68	11.90	1668	1265
0.01	0.01	1	64	11.13	13.24	14.90	16.64	899	885
0.1	0.01	1	64	7.16	11.50	13.56	15.94	1084	1000
0.1	0.01	1	256	7.64	9.55	9.58	12.84	1351	1226
0.1	0.01	10	64	9.47	6.87	8.76	9.56	1179	1904
0.1	*0.01*	*1*	*256*	*11.83*	*10.24*	*11.14*	*10.7*	*1623*	*1493*
0.1	*0.01*	*10*	*64*	*9.92*	*11.73*	*25.02*	*16.5*	*2536*	*1197*

Table B.1: Table of average PPOLag agent evaluation performance. LMI is the Lagrangian multiplier initial value, LMLR is the Lagrangian multiplier learning rate, UI is the number of update iterations and NN is the neural network size, Base is the baseline agents and Curr is the curriculum agents. The last two rows use the same parameters as the two above, but the former were trained for 800 epochs instead of 500.

B.2 Safe RL Algorithms

Now we will present some of the figures relating to Section 6.3, which revolve around comparing the different algorithms that we reasoned to work well with curriculum learning.

Figure B.5 shows the evaluation results of the experiment from Section 6.3.1. It shows the average return, cost and regret at the end of training over the seeds for each agent. The x-axis shows the performance on each metric at the end of training for the baseline versions of the agents and the y-axis does so for the curriculum versions. The diagonal is drawn to indicate more clearly whether the performance of the baseline version or the curriculum version was better for that metric and for each algorithm. If an algorithm is below the diagonal with respect to the return, that means that the return of the curriculum version is worse than the baseline version and vice verse. For the cost and regret is the opposite as a lower cost or regret is better. On top of this, lines are drawn for both the x and y-axis indicating the performance of the PPO agent on that metric. This is to indicate how the safe RL algorithms performed with respect to a general RL algorithm.

Here it is clear how much of an outlier PPOEarlyTerminated is during evaluation, as the distance to the other algorithms is very large for the cost and regret. Even though regret makes less sense in the context of evaluation, it shows that the cost is consistently much higher during training.

Figure B.6 and Figure B.7 show the learning curves of the return and regret, respectively, during training of all agents from the experiment of Section 6.3.1. The learning curves have the metric on the y-axis and the epoch on the x-axis. They show the progression of each metric throughout training for all agents.

Here there is a clear divide between the algorithms that achieve a return higher than 7 and a return lower than 6. Also, the regrets show that the lowest regrets are mostly those of the curriculum agents.

Figure B.8 is a grid of radar charts of the experiment from Section 6.3.2, where the columns represent the target tasks and the rows represent the performance metrics. Each radar chart consists of four axes representing the four algorithms that were tested with and without a curriculum. The values on each axis show the average performance at the end of training during evaluation on that metric for that algorithm on that target task. These values are also connected to give an impression of the performance of the curriculum and baseline agents over all algorithms. This means that when the area of the baseline agents is fully within the area of the curriculum agents for the return, the curriculum agents in general outperform the baseline agents with regards to return on that target task and vice versa. For the cost and regret, this is reversed as higher values represent worse performance. The average performance of the PPO agents is also included to give

perspective on whether the algorithms performed better or worse than PPO.

Here it is clear that PPOEarlyTerminated performs differently during training than it does during evaluation.

Figure B.9 shows the evaluation learning curves of the two types PPOLag agents of the experiment from Section 6.3.2. The columns represent the performance metrics and the rows represent the target tasks. Each learning curve contains the value for the respective metric on the y-axis and the epochs on the x-axis. The dashed line indicates the cost limit and the vertical grey lines indicate the epochs where a task change occurs.

It should be noted that there is a gap between the first epoch and the start of the respective task for the curriculum agents. This is because the agents are only evaluated on the current task. However, the first epoch does then not show how well it does on the first epoch, but instead shows how well the agent performs on the current task while only having trained on previous tasks. In this way, it shows how well the agent transfers knowledge to the new task.

Figure B.10 and Figure B.11 show the learning curves during training of the algorithms on the target task of the experiment in Section 6.3.2. Each learning curve contains the value for the respective metric on the y-axis and the epochs on the x-axis. The vertical grey lines indicate the epochs where a task change occurs.

They show that the returns of the baselines end up slightly higher than those of the curriculum agents, but the curriculum agents in general have a lower regret.

Figure B.12 shows the evaluation results of the experiment from Section 6.3.3. The columns represent the performance metrics and the rows represent the target tasks. Each learning curve contains the value for the respective metric on the y-axis and the epochs on the x-axis. The dashed line indicates the cost limit and the vertical grey lines indicate the epochs where a task change occurs. Each cell in this grid contains the performance of the two types of agents we discussed and the curriculum agent trained on our original curriculum. The performance of the latter cannot directly be compared to the other two, apart from on the target task, as they do not share the intermediate tasks. However, it can be used to compare the progression throughout the respective curricula of the two types of curriculum agents. This means that when the return of the reward curriculum agent is higher on a task, then this shows that the curriculum up to that task was easier to learn than that of the standard curriculum agent and vice versa. Something similar can be said for the cost and regret curves, where a lower cost or regret shows that the respective curriculum is easier to learn from. The first epoch indicates how well the agent performs when only training on the preceding tasks.

Figure B.13 shows the evaluation learning curves of the two types of agents from the experiment in Section 6.3.4. The columns represent the performance metrics and the rows represent the target tasks. Each learning curve contains the value for the respective metric on the y-axis and the epochs on the x-axis. The dashed line indicates the cost limit and the vertical grey lines indicate the epochs where a task change occurs. The gap in results has the same reasoning as before and thus the first epoch is without training on the respective task, but only on all of its previous tasks.

Figure B.14 shows the evaluation learning curves of the three types of agents of the experiment from Section 6.3.5. The columns represent the performance metrics and the rows represent the target tasks. Each learning curve contains the value for the respective metric on the y-axis and the epochs on the x-axis. The dashed line indicates the cost limit and the vertical grey lines indicate the epochs where a task change occurs. The gap in results has the same reasoning as before and thus the first epoch is without training on the respective task, but only on all of its previous tasks.

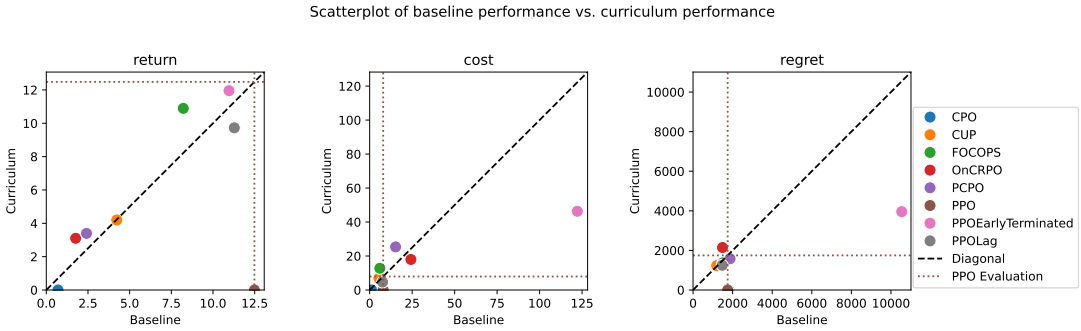


Figure B.5: Plots showing how the algorithms performed with and without the curriculum. The return and cost are the average of the metric in the final epoch of each repetition and the regret is the average of each repetition. Since PPO and CPO have only been used for the baseline, their respective curriculum values are considered 0.

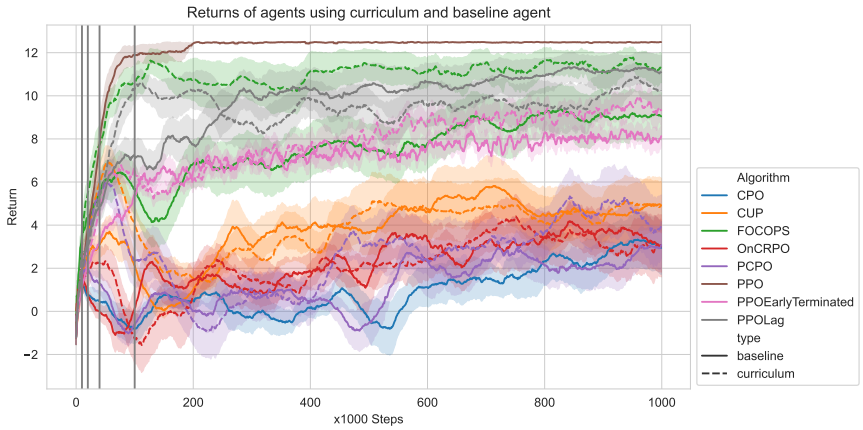


Figure B.6: Return curves during training of several algorithms, comparing baselines with curriculum agents, where the target task is task 4. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents.

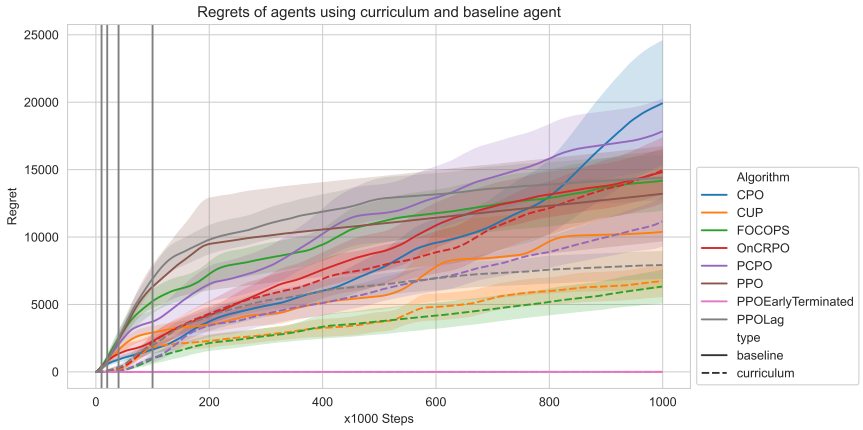


Figure B.7: Regret curves during training of several algorithms, comparing baselines with curriculum agents, where the target task is task 4. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents.

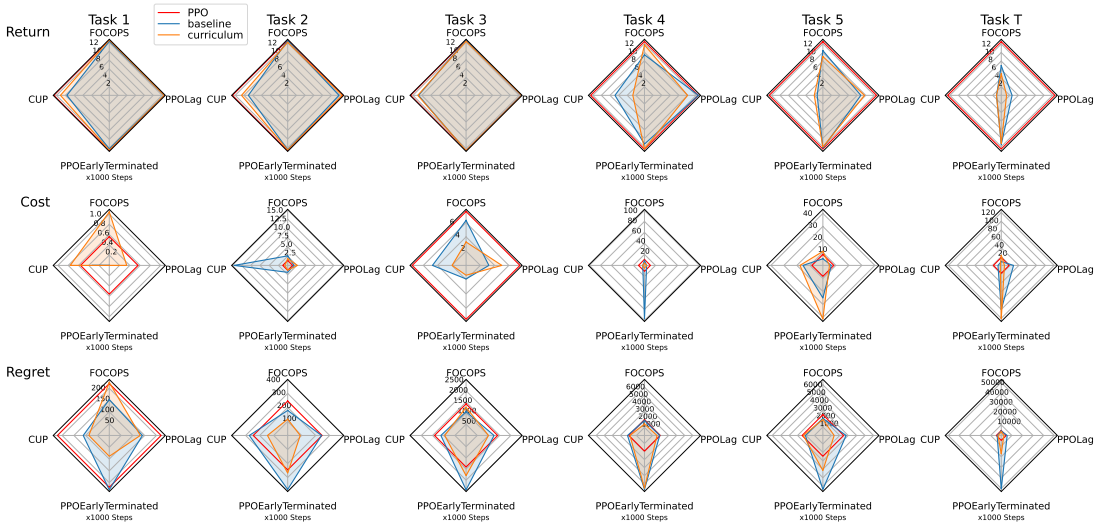


Figure B.8: Radar charts showing for each task other than 0 what the difference in evaluation performance is between the baseline and the curriculum agents. The performance of the PPO agent is included as a reference point of what is the maximum value allowed for each of the metrics. For return we consider it better to have a larger area in the spider plot and for cost and regret we consider it better to have a smaller area.

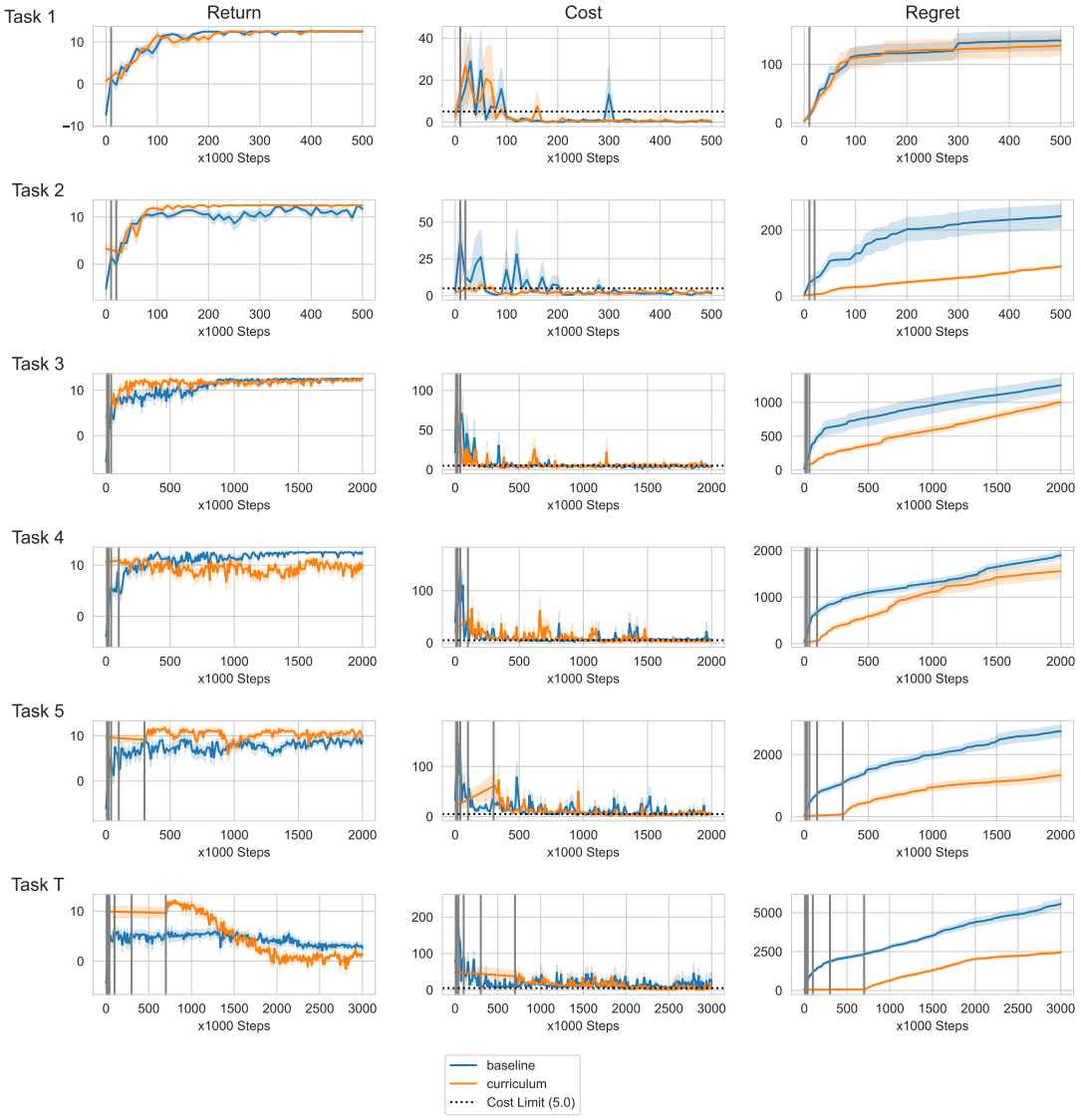


Figure B.9: Evaluation performance curves of the PPOLag agents, comparing baseline with curriculum progression. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents.

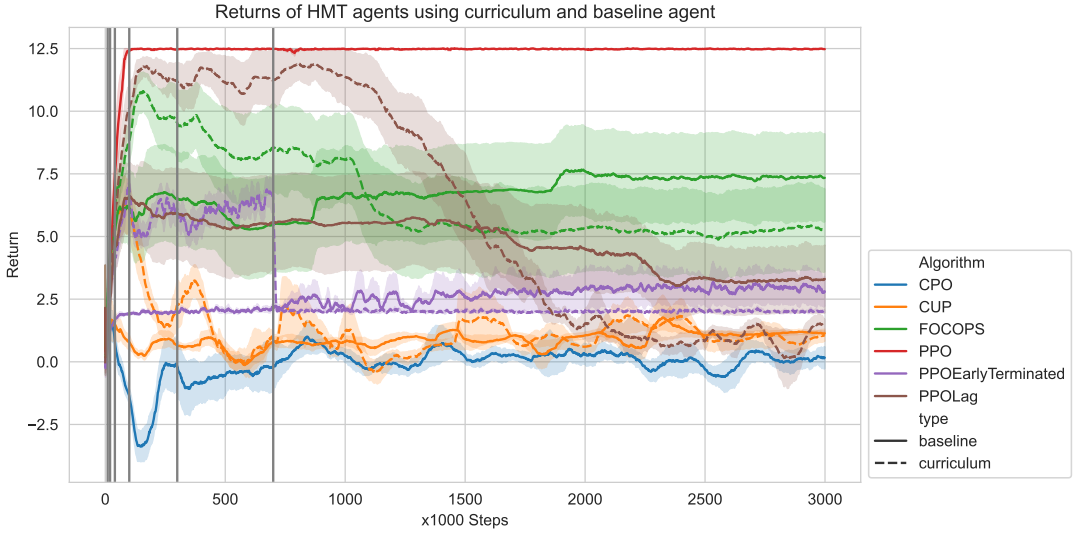


Figure B.10: Return curves during training of several algorithms, comparing baselines with curriculum agents on the target task. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents.

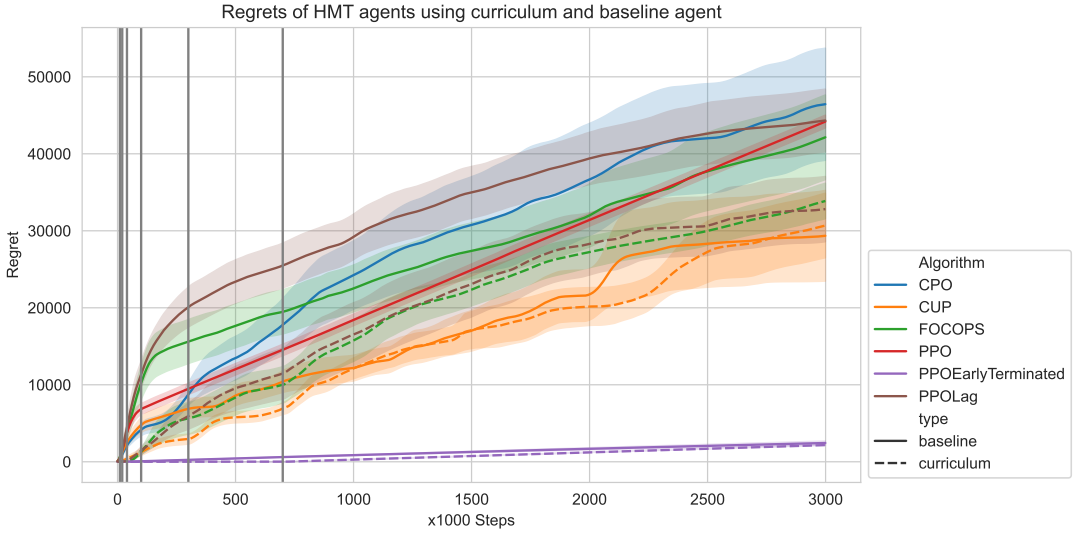


Figure B.11: Regret curves during training of several algorithms, comparing baselines with curriculum agents on the target task. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents.

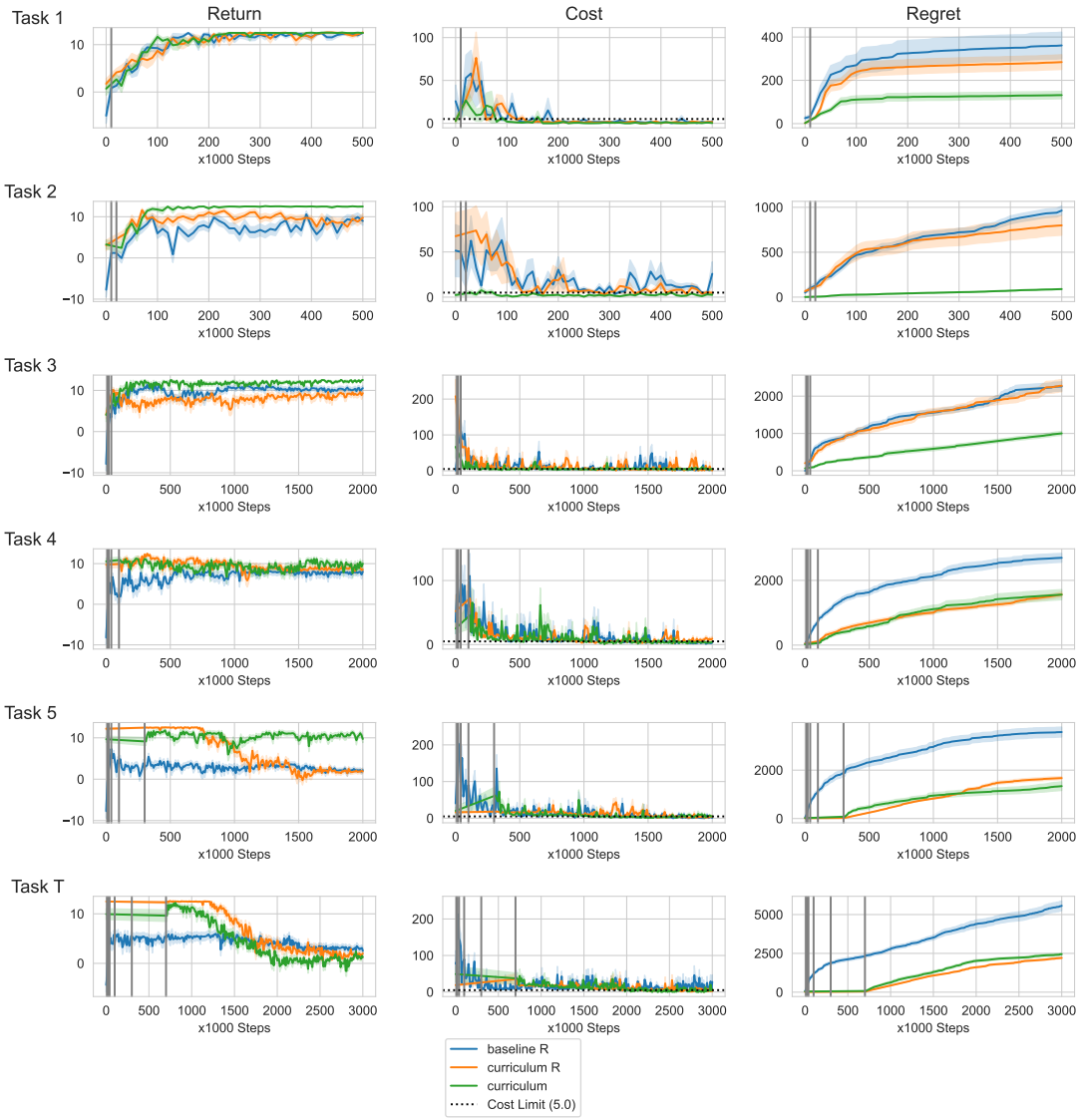


Figure B.12: Evaluation performance curves of the PPOLag agents, comparing baseline with curriculum progression on the reward-based curriculum. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents. To also compare to the progression of our main curriculum, the performance of this method is also included. However, the only task that they share with the reward-based curriculum agents is the target task.

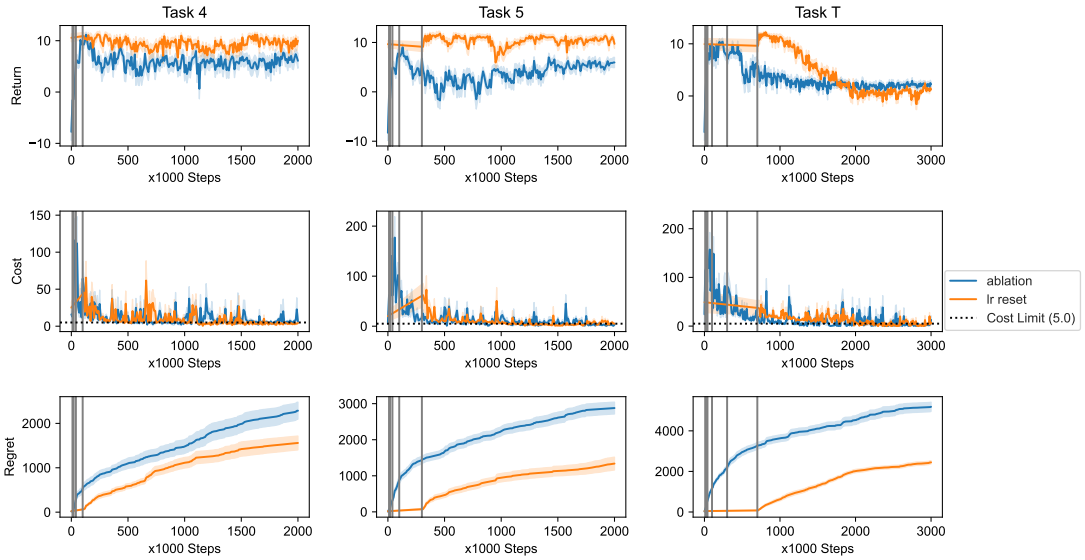


Figure B.13: Results showing the effect of resetting the learning rate and the Lagrangian multiplier when changing a task during evaluation. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents.

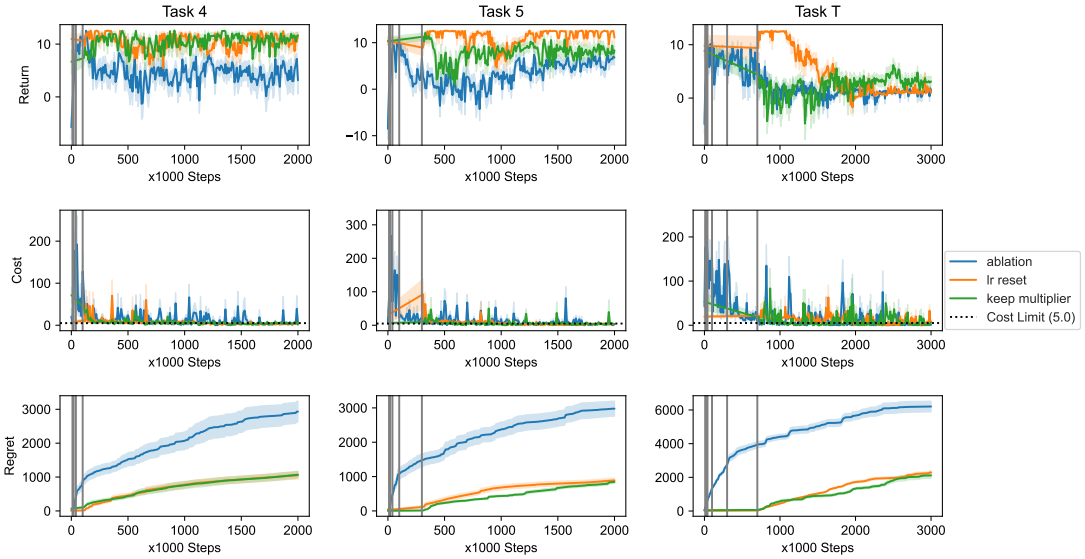


Figure B.14: Results showing the effect of resetting the learning rate and keeping the Lagrangian multiplier when changing a task during evaluation. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents.

B.3 Adaptive Curriculum

Here we present some extra figures from the experiments in Section 6.4, which focus on extending our framework to include adaptive task sequencing.

Figure B.15 shows the task progression during the tuning experiments of Section 6.4.1 for the five best parameter combinations. The x-axis indicates the epochs and the y-axis indicates the task the agents are training on during each epoch. As the lines of the seeds are transparent, a more opaque line indicates more seeds at that task and epoch.

Figure B.16 shows the spider plots of the evaluation performance from the experiments in Section 6.4.2. Also here it is clear that PPOEarlyTerminated performs differently during training than during evaluation.

Figure B.17 shows the learning curves of the PPOLag agents on the same experiments. It should be noted that there is a gap between the first epoch and the start of the respective task for the curriculum agents. This is because the agents are only evaluated on the current task. However, the first epoch does then not show how well it does on the first epoch, but instead shows how well the agent performs on the current task while only having trained on previous tasks. In this way, it shows how well the agent transfers knowledge to the new task. Also, the regret for the adaptive curriculum has some spikes and is not always monotonically increasing. These are both caused by the different amount of seeds having evaluations at those points as not all seeds have reached the target task at the same epoch and the graphs show the average.

Figure B.16 is a grid of radar charts from the experiment in Section 6.4.2, where the columns represent the target tasks and the rows represent the performance metrics. Each radar chart consists of four axes representing the four algorithms that were tested with a static curriculum, with an adaptive curriculum and without a curriculum. The values on each axis show the average performance at the end of training during evaluation on that metric for that algorithm on that target task. These values are also connected to give an impression of the performance of the three types of agents over all algorithms. This means that when the area of the curriculum agents is fully within the area of the adaptive curriculum agents for the return, the adaptive curriculum agents in general outperform the curriculum agents with regards to return on that target task and vice versa. For the cost and regret, this is reversed as higher values represent worse performance. The average performance of the PPO agents is also included to give perspective on whether the algorithms performed better or worse than PPO.

Figure B.17 shows the evaluation learning curves of the three types of PPOLag agents of the experiment from Section 6.4.2. The columns represent the performance metrics and the rows represent the target tasks. Each learning curve contains the value for the respective metric on the y-axis and the epochs on the x-axis. The dashed line indicates the cost limit and the vertical grey lines indicate the epochs where a task change occurs.

It should be noted that there is a gap between the first epoch and the start of the respective task for the curriculum agents. This is because the agents are only evaluated on the current task. However, the first epoch does then not show how well it does on the first epoch, but instead shows how well the agent performs on the current task while only having trained on previous tasks. In this way, it shows how well the agent transfers knowledge to the new task. Also, the regret for the adaptive curriculum has some spikes and is not always monotonically increasing. These are both caused by the different amount of seeds having evaluations at those points as not all seeds have reached the target task at the same epoch and the graphs show the average.

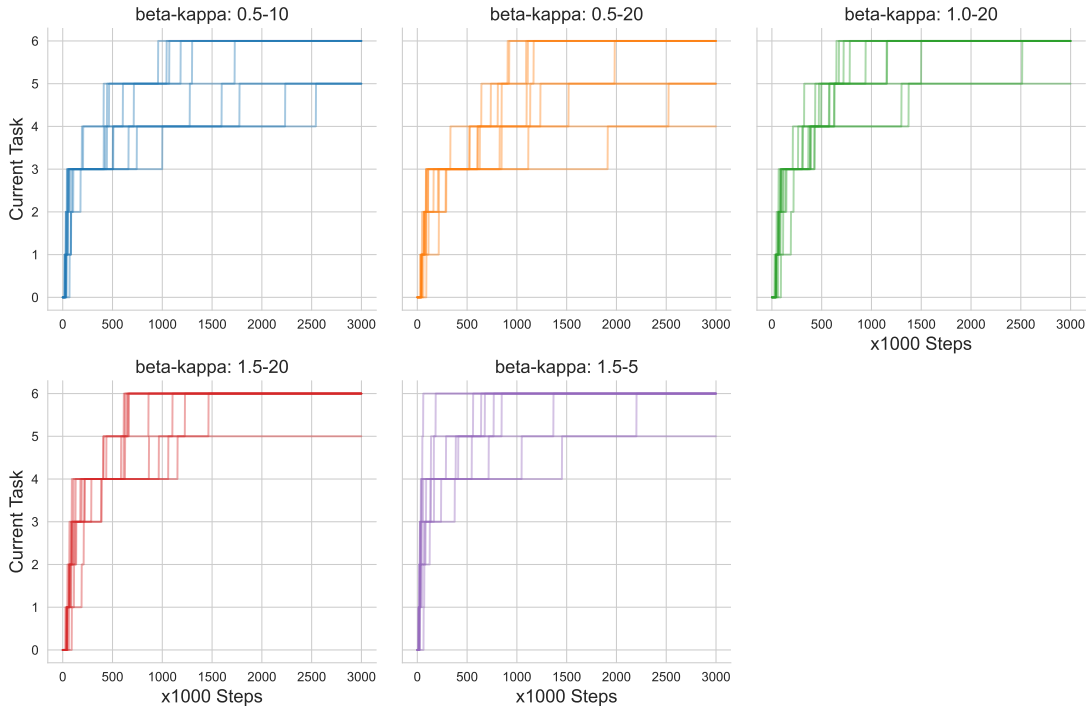


Figure B.15: Task progression of the individual seeds of the 5 best combinations for β and κ.

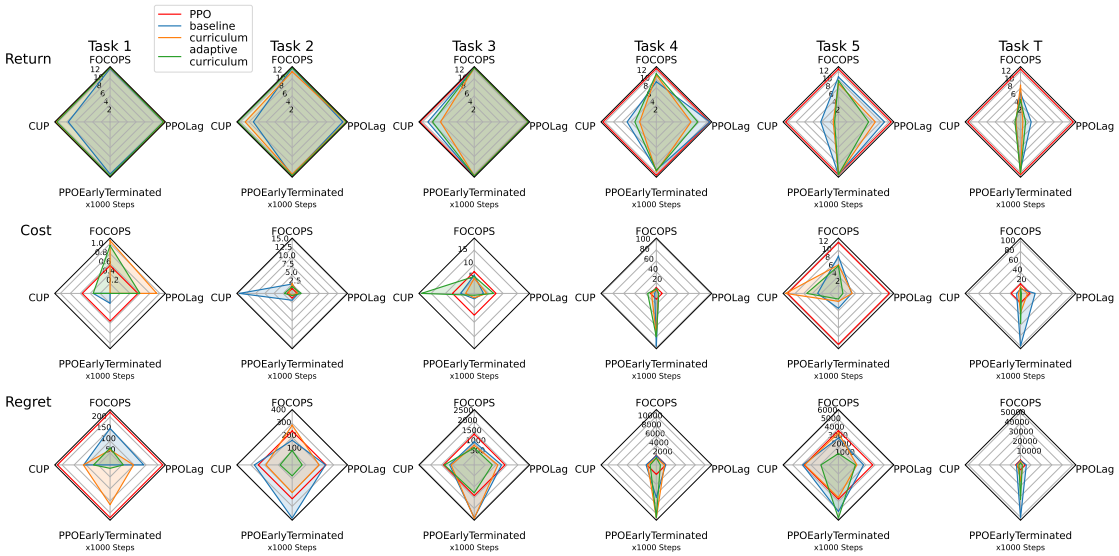


Figure B.16: Radar charts showing for each task other than 0 what the difference in evaluation performance is between the baseline and the curriculum agents. The performance of the PPO agent is included as a reference point of what is the maximum value allowed for each of the metrics.

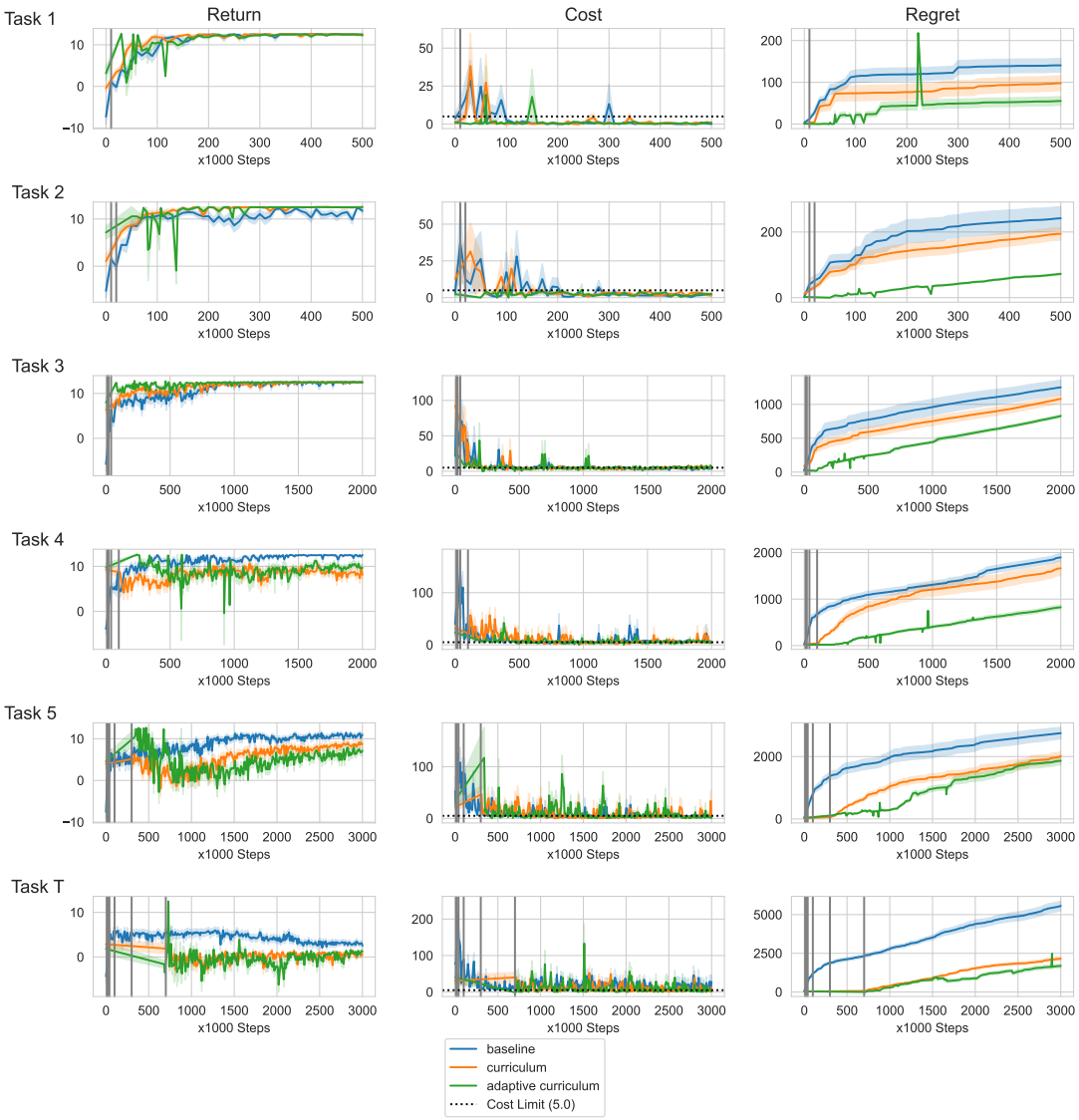


Figure B.17: Evaluation performance curves of the PPOLag agents, comparing baseline with curriculum progression. The shaded area indicates the standard error. The vertical grey lines indicate the epochs at which a task change occurs for the curriculum agents and not for the adaptive curiculum agents.