IT UNIVERSITY OF COPENHAGEN

Testing and Symbolic Analysis For Reinforcement Learning

PhD Dissertation

Mohsen Ghaffari

Supervisors
Andrzej Wąsowski
Mahsa Varshosaz

Department of Computer Science
IT-University of Copenhagen

2024

Abstract

Reinforcement learning (RL) is a type of active learning whereby an agent learns to act in an environment by interacting with it. RL has applications in many domains, including robotics, gaming, electronics, healthcare, water management systems, etc. The majority of real-world applications of RL, such as those in robotics, necessitate a preliminary training phase in a simulation environment. It is otherwise either infeasible or prohibitively expensive to train the agent in a real-world setting. At the same time, there are clear advantages to be gained from the use of formal methods for the enhancement of software qualities. Given that RL and its applications are computer programs, the objective of this thesis is to employ formal methods, in particular specification, testing, and symbolic execution, in order to improve the reliability and explainability of reinforcement learning.

In this thesis, I first investigate the potential of reinforcement learning to address two real-world problems (applications): target search by Unmanned Aerial Vehicles (UAVs) and sewer overflow control. To this end, I present a collision avoidance mechanism between UAVs and train them to find the targets using a proximal policy optimization algorithm. I introduce a reinforcement learning algorithm combined with a model predictive controller that takes the weather prediction into account while synthesizing a policy for the wastewater management system in Copenhagen. Investigating these applications allows me to identify the inherent challenges of reinforcement learning in the development process of RL applications and algorithms, and in partitioning the state space of RL applications.

In order to address the difficulties with development process of RL applications and algorithms, I present a formal specification of the various elements involved, with a particular focus on the temporal difference methods and their definitions in backup diagrams. I further develop an associated testing harness which is reusable across a wide range of RL applications based on temporal difference learning, SARSA, Q-learning, and other similar techniques. I show that my test suite is effective in killing mutants (90% mutants killed for 75% of subject agents). More importantly, almost half of all mutants are killed by generic tests that apply to *any* reinforcement learning problem modeled using my library, without any additional effort from the programmer.

In theory, RL will converge to an optimal policy if it interacts with the environment an infinite number of times. However in practice the training should be completed at a specified point in time. Consequently, RL is only capable of observing the most probable scenarios and not the entirety of the environment, resulting in a limited understanding of the situation. One way to address this problem is to force the agent to explore rare but important situations, for example by initializing episodes with states that likely lead to a different outcomes. I propose to exploit a software analysis technique, symbolic execution, to analyze the environment dynamics. A set of initial states proposed by a symbolic executor enables the agent to explore the state space more thoroughly within shorter time. I show that this method always gets a higher reward and faster convergence than the baseline for any reinforcement learning algorithm.

Reinforcement learning is challenged by large state spaces, whether continuous or discrete. Deep Reinforcement Learning (DRL) has emerged as a prominent approach for addressing environments with continuous state spaces. Despite the research effort to make DRL explainable, the policies obtained by DRL are harder to explain than those obtained by the classic tabular methods. Partitioning the state space enables the use of tabular RL instead of DRL, thereby achieving the goal of explainability. To this end, I have developed a partitioning approach based on the analysis of environment dynamics with symbolic execution. The results of the experiment show an average improvement of 27% in the success rate in comparison to the baseline, while reducing the number of partitions for the majority of the examined cases by over 45%.

Resumé

Forstærkningslæring[1] (*Reinforcement Learning*, RL) er en type aktiv læring, hvor en agent lærer at handle i et miljø ved at interagere med det. RL har anvendelser inden for mange domæner, herunder robotteknologi, spil, elektronik, sundhedspleje, vandstyringssystemer osv. De fleste anvendelser af RL i den virkelige verden, såsom dem inden for robotteknologi, kræver en forudgående træningsfase i et simuleret miljø. Det er ellers enten uigennemførligt eller alt for omkostningstungt at træne agenten i en virkelig kontekst. Samtidig er der klare fordele ved at anvende formelle metoder for at forbedre softwarekvaliteterne. Da RL og dets anvendelser er computerprogrammer, er målet med denne afhandling at anvende formelle metoder, især specifikation, test og symbolsk eksekvering, for at forbedre pålideligheden og forklarligheden af forstærkningslæring.

I denne afhandling undersøger jeg først potentialet i forstærkningslæring til at løse to virkelige problemer (anvendelser): målsøgning med droner (*Unmanned Aerial Vehicle*, UAV) og kontrol af overløb i kloaksystemer. Til dette formål præsenterer jeg en kollisionsundgåelsesmekanisme mellem UAVer og træner dem til at finde målene ved hjælp af en proximal *policy optimization*-algoritme (PPO). Jeg introducerer en forstærkningslæringsalgoritme kombineret med en modelprædiktiv controller, der tager vejrudsigten i betragtning, mens den skaber en politik for spildevandsstyringssystemet i København. Ved at undersøge disse anvendelser kan jeg identificere de iboende udfordringer ved forstærkningslæring i udviklingsprocessen af RL-applikationer og algoritmer og i opdelingen af RL-applikationers tilstandsrum.

For at imødegå vanskelighederne med udviklingsprocessen af RL-applikationer og algoritmer præsenterer jeg en formel specifikation af de forskellige involverede elementer, med særligt fokus på temporal difference-metoderne og deres definitioner i backup-diagrammer. Jeg udvikler yderligere et tilhørende testmiljø, som er genanvendeligt på tværs af en bred vifte af RL-applikationer baseret på temporal difference learning, SARSA, Q-learning og andre lignende teknikker. Jeg viser, at min testsuite er effektiv til at eliminere fejl (90% af fejlene elimineres for 75% af de undersøgte agenter). Endnu vigtigere er det, at næsten halvdelen af alle fejl elimineres af generelle tests, som gælder for enhver forstærkningslæringsproblematik modelleret ved hjælp af mit bibliotek, uden nogen ekstra indsats fra programmørens side.

I teorien vil RL konvergere til en optimal politik, hvis den interagerer med miljøet et uendeligt antal gange. I praksis skal træningen imidlertid afsluttes på et specifikt

[1]This abstract has been produced with help of OpenAI's ChatGPT, but it was checked for factual correctness.

tidspunkt. Derfor kan RL kun observere de mest sandsynlige scenarier og ikke hele miljøet, hvilket resulterer i en begrænset forståelse af situationen. En måde at adressere dette problem på er at tvinge agenten til at udforske sjældne, men vigtige situationer, for eksempel ved at starte episoder i tilstande, der sandsynligvis fører til forskellige udfald. Jeg foreslår at udnytte en softwareanalyseteknik, symbolsk eksekvering (*Symbolic Execution*), til at analysere miljøets dynamik. Et sæt af initielle tilstande, foreslået af en symbolsk eksekveringsmetode, gør det muligt for agenten at udforske tilstandsrummet grundigere på kortere tid. Jeg viser, at denne metode altid opnår en højere belønning og hurtigere konvergens end baseline for enhver forstærkningslæringsalgoritme.

Forstærkningslæring udfordres af store tilstandsrums, uanset om de er kontinuerlige eller diskrete. Dyb forstærkningslæring (*Deep Reinforcement Learning*, DRL) er opstået som en fremtrædende metode til at tackle miljøer med kontinuerlige tilstandsrums. På trods af forskningsindsatsen for at gøre DRL forklarlig, er de politikker, der opnås af DRL, sværere at forklare end dem, der opnås med klassiske tabelmetoder. Opdeling af tilstandsrummet muliggør brugen af tabelbaseret RL i stedet for DRL og opnår dermed målet om forklarlighed. Til dette formål har jeg udviklet en opdelingsmetode baseret på analyse af miljøets dynamik med symbolsk eksekvering. Resultaterne af eksperimentet viser en gennemsnitlig forbedring af succesraten på 27% sammenlignet med baseline, mens antallet af opdelinger reduceres med over 45% for størstedelen af de undersøgte tilfælde.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisors, Andrzej Wąsowski and Mahsa Varshosaz, for their never-ending support, patience, and guidance throughout the course of this research. Their insightful advice and encouragement have been invaluable in shaping my understanding of the subject matter and driving me toward the successful completion of this thesis. I would also like to express my gratitude to Einar Broch Johnsen, who was not my officially designated supervisor but he was consistently available to provide invaluable guidance. I am also indebted to him for welcoming me as a visiting researcher at Oslo University.

I would like to express my gratitude to my colleagues at ITU, particularly the members of the SQUARE group, for providing an excellent collaborative environment that not only facilitated an educational research process but also made it enjoyable. The exchange of ideas and shared discussions proved invaluable in enabling the overcoming of the various challenges encountered during the course of this project.

I am also immensely grateful to my family for their unwavering support, love, and encouragement. Their belief in me provided the motivation to persevere, even in the most challenging times. To my parents, thank you for instilling in me the values of hard work and curiosity, without which this achievement would not have been possible. I would also like to acknowledge my friends for their moral support, companionship, and occasional distractions, which kept me grounded throughout this journey.

This research would not have been feasible without the financial support provided by DIREC. I am profoundly grateful for their contribution to my research and their investment in my academic development.

Finally, I am grateful to everyone who has, in one way or another, contributed to this work. Your encouragement and kindness have made this thesis possible.

To all of you, I extend my heartfelt thanks.

Mohsen Ghaffari
October 2024

CONTENTS

List of Figures

LIST OF TABLES

1 Introduction

1.1 Reinforcement Learning

Reinforcement learning (RL) is a form of active learning that addresses the question of how an agent can learn to make decisions in an environment in order to maximize a cumulative reward signal. In reinforcement learning, the agent interacts with the environment and takes actions based on the current state of the environment. Subsequently, the environment provides feedback to the agent in the form of a reward signal, which the agent utilizes to update its decision-making policy. In its most fundamental form, reinforcement learning provides a solution to decision-making problems in situations where prior knowledge is lacking or when it is difficult to ascertain analytical solutions [127].

1.1.1 Applications

Reinforcement learning has found applications in many domains, including robotics [66], gaming [128], electronics [40], healthcare [152], water management systems [131], etc. One of the most prevalent applications of RL is in the target search problem for unmanned aerial vehicles (UAVs) [18, 58, 107]. The objective is to identify the optimal solution in terms of various parameters, such as the shortest route, minimum threat, or maximum efficiency. Nevertheless, in the context of these types of problems, there are multiple agents present in the environment, which requires them to engage in either cooperative or competitive interactions [18, 58, 107]. One of the parameters that is absent in this domain is the ability to identify targets that are not at a fixed position.

Another challenging problem that requires substantial attention in real-world applications is the management of combined sewer system overflows. Given the high costs associated with infrastructure modifications, the identification of an effective controller for this system can be highly beneficial. Reinforcement learning offers a promising approach to water management through the synthesis of control strategies [15, 16, 157]. Although RL has already been applied to this class of problems, the controllers for many cities are still based on static rules. This is partly caused be the existence of specific local rules, which means that a new objective function is needed for each location. Nevertheless, all of these regulations are clearly delineated and accessible to water management companies. In addition, they possess expertise in the infrastructure and historical data of the system. Furthermore, it seems quite feasible to develop a model. Given the aforementioned knowledge is available, using a static controller does not seem to be a viable option. At the same time, there are techniques such as the

Model Predictive Controller (MPC), which is an advanced control strategy that uses a mathematical model of a system to predict its future behavior and optimize control actions. Accordingly, the combination of MPC with reinforcement learning is a potential avenue for further research.

1.1.2 Challenges

In the event of discrepancies between the anticipated and actual outcomes of reinforcement learning for each of these applications, what are the underlying causes? A preliminary response from the RL research community indicates that the modeling may be a contributing factor. While this can often be correct, modelling errors are not the only case. For many reasons such as novelty, specificity, flexibility, etc, RL researchers usually must implement their desired algorithms and applications themselves, which creates a significant risk of error during the implementation of either the algorithm or application. Currently, there is no systematic methods for implementing either RL algorithms or RL applications.

Although reinforcement learning can automatically synthesize controllers for many challenging control problems [127], it struggles with continuous state spaces, unless approximation techniques are used. While Deep Reinforcement Learning (DRL) achived spectacular success with continuous problems, the methods still suffer from low explainability, and the lack of convergence guarantees, typically amplified by sparse rewards and noisy environments. In contrast, discrete (tabular) learning methods have been shown to be more explainable [82, 106, 140, 154]. That also yields policies for which it is easier to assure safety [37, 55, 138]. Unfortunately the discrete learning methods require finding good state-space representations, so finding representations remains an active research area [4, 23, 53, 71, 85, 99, 145].

To adapt a continuous state space for discrete learning, one exploits partial observability, and merges regions of the state space into discrete partitions, each representing a subset of the states of the agent. Ideally, all states in a partition should capture meaningful aspects of the environment—best if they ensure the Markov property for the environment dynamics. Consequently, a good partitioning depends on the problem at hand. For instance, in safety critical environments, it is essential to identify small "singularities"—regions that require special handling—even if they are very small. Otherwise, if such regions are included in larger partitions, the control policy will not be able to distinguish them from the surrounding, leading to high variance at operation time and slow convergence of learning.

The trade-off between the size of the partitioning and the optimality and convergence of reinforcement learning remains a challenge [4, 23, 71, 85, 99, 145]. Policies obtained for coarse-grained partitionings are unreliable. Large fine-grained partitionings make reinforcement learning slow. The dominant methods are tiling and vector quantization [71, 85, 99, 145]; both are not adaptive to the structure of the state space. For a given problem, they ignore nonlinear dependencies between state components even though quadratic behaviors are common in control systems. So far, the shape of the state partitions has hardly been studied.

Exploration is a fundamental aspect of RL [35, 67, 130]. Common approaches include ε-greedy [127], count-based exploration [11, 80, 123], curiosity-based exploration [105], or methods specifically designed for exploring sparse reward contextual MDPs [109, 155]. All of these methods commence with the selection of initial states via a uniform distribution over a subset of the state space. One domain that remains largely uncharted territory for the RL community is the potential for augmenting the exploration process through alternative agent initialization methods.

Despite the difficulties currently being experienced by the RL community, there are a number of tools and techniques within the field of formal methods that have the potential to provide solutions to the challenges being faced.

1.2 Formal Methods

Formal methods represent a rigorous approach to software and system development. This approach encompasses a variety of techniques, including formal specifications, property-based testing, and symbolic execution, which serve to ensure the correctness and reliability of the software or system in question.

Formal specifications are precise, mathematical descriptions of software or system behavior that are utilized to define the intended functionality of a system. In contrast to informal specifications, which are typically composed in natural language and may be open to interpretation, formal specifications employ formal logic, set theory, algebra, or other mathematical notations to provide unambiguous descriptions of the system's properties, operations, and constraints.

In property-based testing, the tester defines general properties that should always hold true for any valid input, rather than specifying individual test cases with specific inputs and expected outputs. These properties delineate the anticipated behavior or invariants of the system undergoing examination. Subsequently, a testing framework automatically generates test cases based on predefined properties of the system, thereby ensuring that these properties hold across a wide range of inputs. In the event of a failure, the framework simplifies the input in order to identify the minimal case causing the failure, thus aiding in diagnosis [21].

Symbolic execution is a program analysis technique that systematically explores program behaviors by solving, using an SMT-solver, symbolic constraints obtained from conjoining the program's branch conditions [65]. Symbolic execution extends normal execution by running the basic operators of a language using symbolic inputs (variables) and producing symbolic formulas as output.

A satisfiability modulo theories (SMT) solver is a computational tool that determines whether a given logical formula can be satisfied. In other words, the question is whether there exists an assignment of values to variables that makes the formula true. In contrast to traditional satisfiability (SAT) solvers, which operate exclusively within the domain of Boolean logic, SMT solvers are capable of handling a more expansive range of complex theories, including those pertaining to arithmetic, bit-vectors, arrays, and beyond.

1.3 Thesis Statement

Objective. Given that reinforcement learning and its applications are computer programs, the objective of this thesis is to employ formal methods, specifically software verification and software analysis techniques, in order to improve the reliability of reinforcement learning.

Having established the principal conceptual framework, I now proceed to elucidate the central questions addressed by this study:

RQ1. *How can proximal policy optimization-based reinforcement learning enable a group of UAVs to cooperatively explore an unknown environment and locate a group of targets in an optimal manner?*
The problem of cooperative target search by UAVs where the location of the targets is not fixed represents a challenging problem that I have attempted to address through the application of reinforcement learning algorithms. Given the considerable size of the state and action spaces involved, I have proposed a PPO-based approach that has demonstrated efficacy in addressing problems with continuous or large discrete action spaces. Furthermore, I proposed a collision avoidance strategy to reduce the likelihood of damage to UAVs due to collisions (Chapter 4).

RQ2. *To what extent a simplified model of a wastewater management system could assist reinforcement learning to learn a useful policy for a semi-realistic environment?*
Combined sewer systems frequently result in combined sewer overflow during precipitation events, whereby sewage and stormwater are combined, thereby necessitating treatment prior to discharge [83]. The objective of my study was to investigate the potential of RL to enhance the mitigation of sewage overflows in pumping stations and gates of combined sewer system during storm events. To this end, a controller synthesis for the system that employs a combined method based on MPC and Q-learning is developed (Chapter 5).

RQ3. *How to facilitate the systematic development of reinforcement learning algorithms and applications?*
Given that reinforcement learning and its applications are computer programs, I present a formal specification of the various elements of RL problems and algorithms. I further develop an associated testing harness which is reusable across a wide range of RL applications based on temporal difference learning, SARSA, Q-learning, and other similar techniques (Chapter 6).

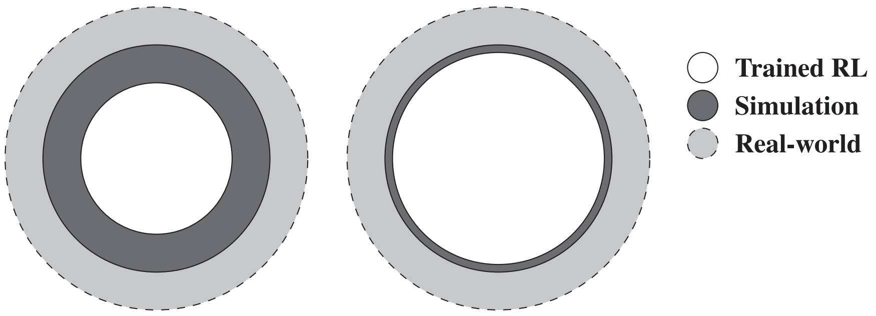
RQ4. *To what extent can pre-analysis of the environment simulator help to construct a comprehensive representation and achieve a quantitative coverage of the reinforcement learning state space?*
My investigation concerns the use of off-the-shelf tools for symbolic execution with the objective of extracting approximate adaptive partitionings that reflect the dynamics of the problem. Symbolic execution is a well-established technique for analyzing the behavior of computer programs. It generates a set of

constraints that must be satisfied for each possible execution path of the program. These conditions partition the state space of the program into groups that share the same execution path. My hypothesis is that the path conditions obtained by symbolic execution of an environment simulator provide a useful state space partition for reinforcement learning (Chapter 7).

RQ5. *How can pre-analysis of the environment simulator help the exploration of the reinforcement learning algorithms?*
The branches in the environment program likely reflect significant aspects of the problem dynamics that should be respected by an optimal policy. My hypothesis is that the path conditions obtained by symbolic execution may result in generating a set of initial states for any reinforcement learning algorithm, which could subsequently improve the exploration phase of RL and its overall performance. Additionally, a better initialization approach would facilitate the observation of states that are closer to the goal state, and subsequently it could help to handle better with the sparse rewards (Chapter 8).

In instances where reinforcement learning is employed to address a real-world problem, it is frequently necessary to conduct a preliminary training phase within a simulation environment. It is otherwise typically either infeasible or prohibitively expensive to train the agent in a real-world environment. For instance, training an RL agent to control wastewater treatment in a large urban area (Chapter 5) may result in flooding, leading to untreated wastewater being discharged into the sea. This could have a detrimental impact on the surrounding environment and human health. Also, each action takes minutes, and episodes would last years, not yielding enough data. Another potential scenario is the deployment of unmanned aerial vehicles (UAVs) to locate injured individuals (Chapter 4). There is a possibility that the search may not be completed in a timely manner, which could result in life-threatening situations for the individuals in need of assistance or the loss of the UAVs, potentially leading to the failure of the mission. It is also possible to generate crashing actions that could lead to a loss of each equipment.

At the same time, it is infeasible or exceedingly improbable to replicate the entirety of the environmental specifics within the simulation. This challenge is referred to as the Sim2Real gap (shown with gray color in Figure 1.1a). It should be noted that I am not attempting to resolve this issue. However, I am developing a testbed that enables experts to assess the functionality of the simulation and identify and address potential implementation-level issues (Chapter 6). Accordingly, I do not attempt to solve the Sim2Real gap, but to find bugs in the implementations (dark-gray region in Figure 1.1). In the following, it is assumed that a simulation, whether effective or ineffective, is available for use in conjunction with reinforcement learning algorithms for the purpose of agent training within that simulation environment. Since the training should be completed at a certain point in time, the RL agent is only able to observe the most likely scenarios and not the entire environment, resulting in a limited understanding of the situation Figure 1.1a. The objective is to enhance the efficacy of this phase by conducting a preliminary analysis of the simulation. Figure 1.1 illustrates that my approach (Chapters 7 and 8) aim to achieve the greatest possible approximation between



(a) Current situation. (b) Goal of the thesis.

Figure 1.1: Comparing the current situation of solving a real-world problem using RL with the goal of thesis.

the trained knowledge obtained through reinforcement learning (depicted in white) and the knowledge present in the simulation (shown in dark-gray).

I answer the aforementioned questions by advancing the following thesis:

THESIS

Formal specification and property-based testing facilitate the development of both algorithms and applications of reinforcement learning. Furthermore, symbolic execution can capture the dynamics of simulation environments, thereby improving the efficacy and performance of both tabular and deep reinforcement learning.

1.4 Thesis Contribution

The main contributions of this thesis are as follows:

Collision-Avoidance in Multi-Agent Reinforcement Learning for Search-and-Rescue. I suggest a collision avoidance strategy for multiple UAVs searching in an environment with limited range of motion. Furthermore, I evaluate the effectiveness of a Proximal Policy Optimization (PPO) algorithm in a search and rescue scenario, with varying degrees of complexity and uncertainty.

Optimizing Combined Sewer Overflows with Online Model-Predictive Reinforcement Learning. I study the feasibility of using reinforcement learning based controllers for realizing a storage-release controller, taking weather forecasts into account. I suggest a controller using Model Predictive Control in conjunction with Q-Learning for the area west of Copenhagen, the so-called Hvidovre pipeline.

Formal Specification and Testing for Reinforcement Learning. I provide a formal specification of the key elements of RL problems and algorithms, with a particular emphasis on temporal difference methods and their representation in backup diagrams. Additionally, I develop a related test harness that is reusable across a broad range of RL applications based on temporal difference learning, SARSA, Q-learning, and other similar techniques.

Partitioning the State Space of Reinforcement Learning Using Symbolic Execution. For the class of RL applications that the simulator of environment is available, I propose a state space partitioning approach based on the analysis of environment dynamics with symbolic execution.

Smarter Initialization gets Better Coverage for Reinforcement Learning. To enhance the exploration phase of any reinforcement learning algorithm and more effectively handle sparse rewards, I propose to use a symbolic executor to analyse the dynamics of the environment and generate a set of states that can be fed into the reinforcement learning algorithm. This approach ensures that the agent learns to act in the states where its transition leads to a different branch in the simulator.

1.5 List of Papers

PAPER I Danyal Yorulmaz, Tobias Gad Spoorendonk, Mohsen Ghaffari, Andrzej Wąsowski. Multi-Agent Reinforcement Learning for Search-and-Rescue with Cooperative Rotation Maneuver. Manuscript submitted for publication.

PAPER II Esther Hahyeon Kim, Mohsen Ghaffari, Martijn Goorden, Andreas Holck Høeg-Petersen, Thomas Dyhre Nielsen, Kim Guldstrand Larsen, Andrzej Wąsowski (2024). Minimizing Combined Sewer Overflows with Online Model-Predictive Reinforcement Learning. *Urban Water Journal (under review)*, 2024.

PAPER III Mahsa Varshosaz, Mohsen Ghaffari, Einar Broch Johnsen, and Andrzej Wąsowski. Formal specification and testing for reinforcement learning. *Proceedings of the ACM on Programming Languages*, 7 (ICFP), August 2023. DOI: 10.5281/zenodo.8083298.

PAPER IV Mohsen Ghaffari, Mahsa Varshosaz, Einar Broch Johnsen, and Andrzej Wąsowski. Symbolic State Partitioning for Reinforcement Learning. Manuscript submitted for publication.

PAPER V Mohsen Ghaffari, Cong Chen, Mahsa Varshosaz, Einar Broch Johnsen, and Andrzej Wąsowski. Symbolic State Seeding Improves Coverage of Reinforcement Learning. Manuscript submitted for publication.

1.6 Other Publications and Activities

- Mahsa Varshosaz, Mohsen Ghaffari, Einar Broch Johnsen, and Andrzej Wąsowski. Towards Formal Specification of Reinforcement Learning. Abstract from the 7th Workshop on Learning in Verification (LiVe), 2023.
- Mohsen Ghaffari, Mahsa Varshosaz, Einar Broch Johnsen, and Andrzej Wąsowski. Using Symbolic Execution to Discretize State Spaces for Reinforcement Learning. Proceedings of the 34th Nordic Workshop on Programming Theory (NWPT), 2023.

1.7 Overview of Thesis

This dissertation is organized as follows.

PART I. Chapter 2 delineates the preliminary knowledge requisite for the remainder of the thesis, including an introduction to the formal methods employed in this thesis (i.e., symbolic execution) and the fundamental concepts of reinforcement learning. Chapter 3 presents a comprehensive review of the current state of the art in the target search using unmanned aerial vehicles, wastewater management, testing and reinforcement learning, partitioning the state space of reinforcement learning, and seeding to reinforcement learning.

PART II. Chapter 4 suggests an enhanced collision avoidance strategy for multiple UAVs searching in an environment with restricted range of motion. Moreover, an assessment of the efficacy of a PPO algorithm in a search and rescue context, with varying degrees of complexity and uncertainty, is presented. Chapter 5 studies the potential for employing RL-based controllers in conjunction with an abstract model of the environment to develop a storage-release controller that incorporates weather forecasts.

PART III. Chapter 6 provides a formal specification of the principal elements of RL problems and algorithms, subsequently developing a test harness for RL applications and algorithms based on temporal difference learning.

PART IV. Chapter 7 proposes a state space partitioning approach based on the analysis of environment dynamics with symbolic execution. Chapter 8 suggests an initial state seeding methodology for a range of RL algorithms, including tabular and deep learning algorithms.

PART V. Chapter 9 presents the thesis's principal findings, offers a synthesis of these findings, and suggests avenues for future research.

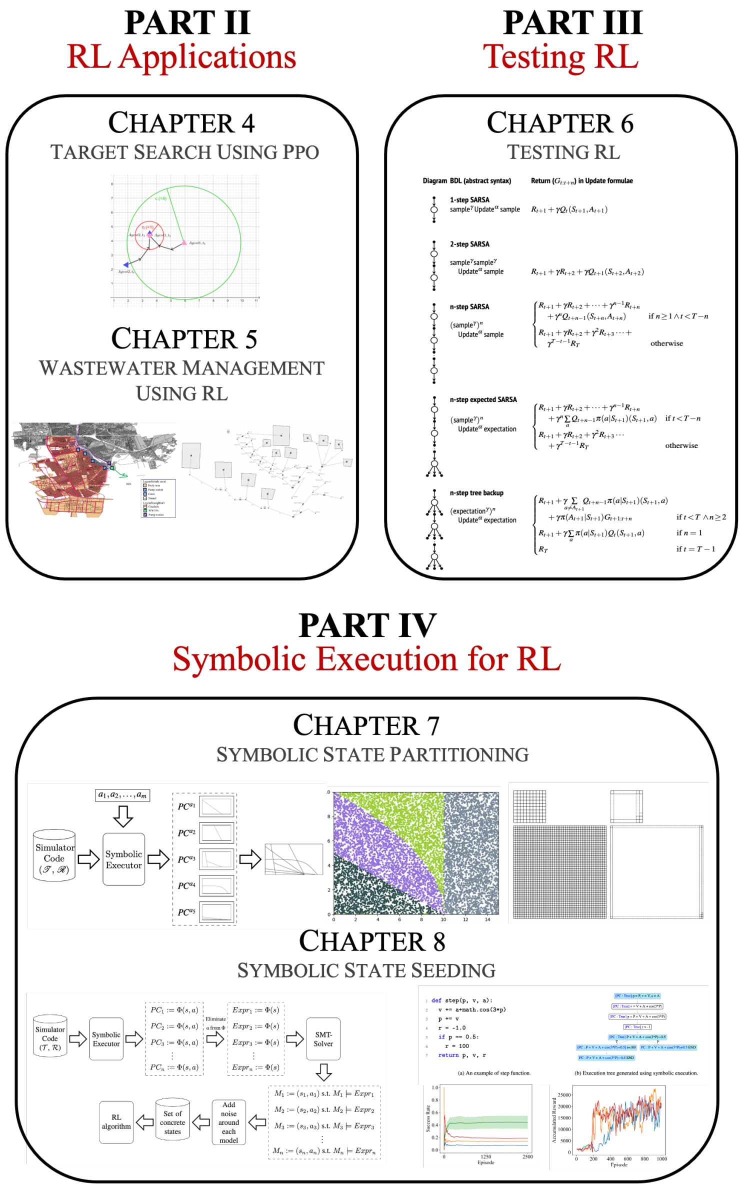


Figure 1.2: A visual overview of the dissertation.

2 Preliminaries

This chapter reviews the basic concepts used in the present thesis, including formal specifications, property-based testing, and symbolic execution. I also present the reinforcement learning and multi-agent systems.

2.1 Formal Specification

The first step behind any formal method is writing a precise specification of a system, using formal or mathematical notation. The notations is equipped with an unambiguous semantics, which is to say that a precise meaning is ascribed to each statement within the language [45].

A system specification may encompass one or more of a variety of elements, including functional behavior, structural or architectural aspects, and even non-functional characteristics such as timing or performance criteria. A precise specification of a system can be then used in a number of ways. First, it can be a means of articulating a comprehensive understanding of the system, thereby identifying any errors or instances of incompleteness. It can also be subjected to analysis or verified for correctness against relevant properties. Furthermore, a specification can guide the development process, either through the refinement of the specification toward code or through code generation. It is worth noting that a crucial element of the development process is testing, and a specification can also facilitate the testing process.

There exists a multitude of formal specification techniques, some of which are general-purpose, others tailored to specific application domains (e.g., concurrent systems). The majority of these techniques are supported by of tools. In what follows, I will survey some of the most prevalent notations as a preliminary investigation into their integration into the testing process.

The rationale behind the use of formal methods in engineering is based on the premise that the time invested in the initial specification and design phases will ultimately lead to a higher quality of product. This, in turn, contributes to the commercial rationale of attempting to reduce the cost of rework that may occur later on. It should be noted that formal methods do not guarantee the absolute correctness of a system. However, their use is intended to enhance our comprehension of the system by identifying errors or deficiencies in the design that could potentially be costly to rectify at a later stage.

2.2 Property-based Testing

Property-based testing (PBT) is a robust methodology for assessment of the software correctness. The process of PBT commences with a developer determining a formal specification that their code should satisfy and then encoding that specification as an executable property. An automated test harness then verifies the property against the code in question, using a number of randomly generated test inputs. In the event that the process identifies a counterexample to the property—an input value that causes the property to fail—the developer is notified [42].

Properties are executable specifications of programs. For example, suppose a developer is working on a new implementation of binary search trees (BSTs)—tree structures where each internal node is labeled with a data value that is greater than any labels in its left sub-tree and less than any in its right sub-tree. We know that all the operations on BSTs (insert, delete, etc.) must preserve this validity condition: given a valid BST, they should always produce a valid BST.

One can feed the properties in an off the shelf test harness and define a domain of random numbers that want to be generated for testing. Then, the framework generates hundreds or thousands of random numbers and tests the validity of condition. If the property ever fails during testing, the failing value is presented to the user. This value might be overly complex, with parts that are irrelevant to the failure of the property, so most PBT frameworks provide tools for test-case reduction [81], usually called shrinking [49] in the PBT literature.

2.3 Symbolic Execution

Symbolic Execution is a formally grounded program analysis technique that systematically explores program behaviors by solving symbolic constraints obtained from conjoining the program's branch conditions [12, 65]. Symbolic execution extends normal execution of a computer program by running the basic operators of a language using symbolic inputs (variables) and producing symbolic formulas as output. A symbolic execution of a program results in a set of *path conditions*—logical expressions that encode conditions on the input symbols to follow a particular path in the program.

For a program over input arguments $I = \{v_1, v_2, \ldots, v_k\}$, a path condition $\phi \in PC(I')$ is a quantifier free logical formula defined over symbolic variables $I' = \{\vartheta_1, \vartheta_2, \ldots, \vartheta_k\}$, where each symbolic variable ϑ_i corresponds to v_i.

I sketch a definition of symbolic execution for a minimal language, De Boer and Bonsangue [12] provide more details. Let V be the set of program variables and Ops be a set of arithmetic operations, $x \in V$, $k \in \mathbb{N}$ $n \in \mathbb{R}$, and $op \in$ Ops. I consider programs generated by the following grammar:

$$\begin{aligned} e &::= x \mid n \mid op(e_1, \ldots, e_n) \\ b &::= \text{True} \mid \text{False} \mid b_1 \text{ AND } b_2 \mid b_1 \text{ OR } b_2 \mid \neg b \mid b_1 \leq b_2 \mid e_1 < e_2 \\ s &::= x := e \mid x \sim \text{rnd} \mid s_1; s_2 \mid \text{if } b\ s_1 \text{ else } s_2 \mid \text{while } b\ s \mid \text{skip} \end{aligned} \tag{2.1}$$

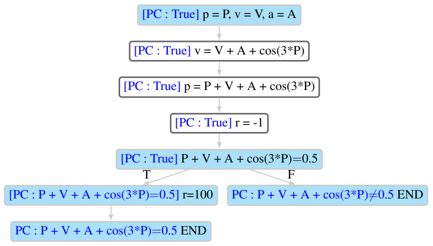
A symbolic store, denoted by σ maps input program variables $I \subseteq V$ to expressions, generated by productions e above. An update to a symbolic store is denoted $\sigma[x := e]$.

$$
\begin{array}{rrcl}
\text{S-ASSIGN} & (x := e, \sigma, k, \phi) & \rightarrow & (\text{skip}, \sigma[x \mapsto \sigma e], k, \phi) \\
\text{S-IF-T} & (\text{if } b \; s_1 \text{ else } \; s_2, \sigma, k, \phi) & \rightarrow & (s_1, \sigma, k, \phi \wedge \; \sigma b) \\
\text{S-IF-F} & (\text{if } b \; s_1 \text{ else } \; s_2, \sigma, k, \phi) & \rightarrow & (s_2, \sigma, k, \phi \wedge \; \sigma \neg b) \\
\text{S-WHILE-T} & (\text{while } b \; s, \sigma, k, \phi) & \rightarrow & (s \; ; \; \text{while } b \; s, \sigma, k, \phi \wedge \; \sigma b) \\
\text{S-WHILE-F} & (\text{while } b \; s, \sigma, k, \phi) & \rightarrow & (\text{skip} \; ; \; \sigma, k, \phi \wedge \; \sigma \neg b) \\
\text{S-SMLP} & (x \sim \text{rnd}, \sigma, k, \phi) & \rightarrow & (\text{skip}, \sigma[x \mapsto y_k], k+1, \phi)
\end{array}
$$

Figure 2.1: Symbolic execution rules for an idealized probabilistic language. Each judgement is a quadruple: the program, the symbolic store (σ), the sample index (k), the current path condition (ϕ).

```
def step(p, v, a):
  v += a + math.cos(3*p)
  p += v
  r = -1.0
  if p == 0.5:
    r = 100
  return p, v, r
```

(a) An example of step function.


[PC : True] p = P, v = V, a = A
[PC : True] v = V + A + cos(3*P)
[PC : True] p = P + V + A + cos(3*P)
[PC : True] r = -1
[PC : True] P + V + A + cos(3*P)=0.5
T
F
[PC : P + V + A + cos(3*P)=0.5] r=100
PC : P + V + A + cos(3*P)≠0.5 END
PC : P + V + A + cos(3*P)=0.5 END

(b) Execution tree generated using symbolic execution.

Figure 2.2: An example of step function, and its symbolic execution tree.

It replaces the entry for variable x with the expression e. An expression can be interpreted in a symbolic store by applying (substituting) its mapping to the expression syntax (written $e\sigma$).

Figure 2.1 gives the symbolic execution rules for the above language, in terms of traces (it computes a path condition ϕ for a terminating trace). In the reduction rules, ϕ stands for the path condition and k denotes the sampling index. The first rule defines the symbolic assignment; An assignment does not change the path conditions, but updates the symbolic store σ. When encountering conditional statements, the symbolic executor splits into two branches. For the true case (rule S-IF-T) the path condition is extended with the head condition of the branch, for the false case (S-IF-F), the path condition is extended with the negation of the branch condition. Similarly, for a *while* loop two branches are generated, with an analogous effect on path conditions. The last rule executes the randomized sampling statement. It simply allocates a new symbolic variable y_k for the unknown result of sampling, and advances the sampling index [139].

The above rules can be used to prove basic properties of the symbolic execution. For example, as each path condition contains conjunction of *different* branch conditions in a program, the path conditions of the same program are mutually exclusive [12].

Figure 2.2 demonstrates a simplistic example of step function (Figure 2.2a), and the execution tree of this function in a using a symbolic executor (Figure 2.2b). As illustrated, the execution tree carries a PC in each node and the leaf nodes are showing the logical expression that must be satisfied to reach the corresponding syntactic path in the program.

There exist practical symbolic executors for full scale programming languages. Even though I defined the concept at the level of syntax, the two most popular symbolic executors operate on compiled bytecode [17, 103]. In presence of loops and recursion, symbolic execution does not terminate. To halt symbolic execution, one can set a predefined timeout, an iteration limit, or a program statement limit. This produces an approximation of the set of path conditions.

2.4 Reinforcement Learning

Reinforcement learning is a form of active learning, where an agent learns to make decisions to maximize a reward signal [127]. The agent interacts with an environment and takes actions based on its current state. The environment rewards the agent, which uses the reward value to update its decision-making policy Figure 2.3. This method can automatically synthesize controllers for many challenging control problems [127], however dedicated approximation techniques, hereunder deep learning, are needed for continuous state spaces. Three of well known deep RL algorithms—DQN, A3C, and PPO—will be introduced later in this section.

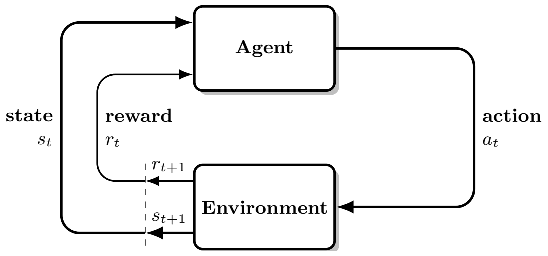


Figure 2.3: Reinforcement Learning Schematic.

An RL problem can be modeled as a Markov Decision Process (MDP). An MDP is a tuple $\mathcal{M} = (\overline{\mathcal{S}}, \overline{\mathcal{S}}_0, \mathcal{A}, \mathcal{S}, \mathcal{O}, \mathcal{T}, \mathcal{R}, \mathcal{F})$, where $\overline{\mathcal{S}}$ is a set of states, $\overline{\mathcal{S}}_0 \in \text{pdf}\,\overline{\mathcal{S}}$ is a probability density function for initial states, $\mathcal{A}$ is a finite set of actions, $\mathcal{S}$ is a finite set of observable states, $\mathcal{O} \in \overline{\mathcal{S}} \to \mathcal{S}$ is a total observation function, $\mathcal{T} \in \overline{\mathcal{S}} \times \mathcal{A} \to \text{pdf}\,\overline{\mathcal{S}}$ is the transition probability function, $\mathcal{R} \in \overline{\mathcal{S}} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\mathcal{F} \in \mathcal{S} \to \{0, 1\}$ is a predicate defining final states. The task is to find a policy $\pi : \mathcal{S} \to \text{Dist}(\mathcal{A})$ that maximizes the expected accumulated reward [127]. If $\overline{\mathcal{S}} = \mathcal{S}$ then the MDP is fully observable, but if this is not the case, then it is called Partially Observable Markov Decision Process (POMDP).

***Q*-Learning.** Q-Learning [144], one of the early breakthroughs in RL, is an off-policy Temporal Difference (TD) control that at each time step t, the controller receives a representation of the environment's state $s_t \in \mathcal{S}$, and takes an action $a_t \in \mathcal{A}$ based on a policy $\pi : \mathcal{S} \to \mathcal{A}$. The action changes the environment's state according to

$\mathcal{T}(s_t, a_t) = s_{t+1}$, possibly in a stochastic manner, and this transition results in a reward $r_t \in \mathcal{R}$. Then it updates its knowledge (Q-table) using the following equation:

$$Q_{t+1}(s_t, a_t) = (1-\alpha)Q_t(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)] \quad (2.2)$$

where, α is the learning rate, and γ is the discount factor. Once the learning process has converged to the optimal policy, the optimal policy for each state in the environment can be extracted from the Q-table.

Deep Q-Network (DQN) is introduced to overcome RL drawbacks that would raise by increasing the dimensions of state space [92]. The DQN value functions and policy are usually parameterized by the Deep Neural Networks (DNN) variables, rather than the Q-table in RL. The DNN represents the policy as a continuous function. In each episode, the DQN algorithm performs a gradient descent step with respect to the network parameter θ to minimize the loss $\mathcal{L}(\theta)$:

$$\mathcal{L}(\theta) = \left[\left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'; \theta') - Q(s_t, a_t; \theta)\right)^2\right] \quad (2.3)$$

where, θ is the parameters of the $Q-$network, $Q(s_t, a_t; \theta)$ is the current estimate of the $Q-$value for the state-action pair (s_t, a_t), $\max_{a'} Q(s_{t+1}, a'; \theta')$ is the maximum predicted $Q-$value for the next state s_{t+1}, using the target network with parameters θ'.

Asynchronous Advantage Actor-Critic (A3C) is a reinforcement learning algorithm where multiple agents (or copies of the environment) work in parallel to learn a task. These agents operate independently and asynchronously, meaning they collect experiences at different times [90]. Each agent uses an *actor* network to decide on actions and a *critic* network to evaluate how good those actions are, based on the current state. By working together, the agents share their learned experiences with a global model, allowing it to learn more efficiently and effectively.

Proximal Policy Optimization (PPO) is a policy gradient algorithm that combines ideas from Asynchronous Actor-Critic (A2C-having multiple workers) and Trust Region Policy Optimization (using a trust region to improve the actor) [117]. It iteratively learns a parameterized policy π_θ. In standard implementations, PPO regularizes the policy updates with clipped probability ratios, and parameterizes policies with either continuous Gaussian distributions or discrete Softmax distributions [116].

UPPAAL STRATEGO is a model checking and verification tool designed to model real-time systems [25]. For hybrid systems with control variables, the tool is capable of learning a near-optimal control strategy, using statistical model checking to give probabilistic guarantees for the properties of the system under the learned strategy. UPPAAL STRATEGO employs a non-deep reinforcement learning algorithm that differs from classical Q-learning, in that the continuous state space is discretized during learning via an *online refinement scheme* based on the variance in the expected reward in different regions of the state space [52].

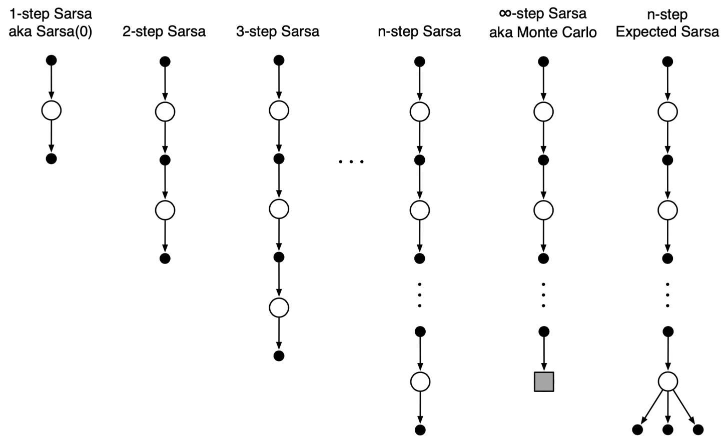


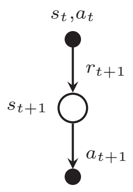
Figure 2.4: The backup diagrams for the spectrum of n-step methods for state-action values.

MPC *Q*-Learning vs *Q*-Learning. Q-learning entails training a model utilizing either a simulator or a physical environment, followed by its application to real-world problem-solving (the evaluation step). During this process, the agent learns the optimal Q-function, which determines the best actions to maximize rewards given specific states and actions. In evaluation, the agent uses the learned Q-function to choose optimal actions and solve problems accordingly. Q-learning is effective when the given simulator or data comprehensively covers all potential scenarios. It requires an accurate model, focusing initially on handling all possible situations. Conversely, MPC Q-learning introduces interaction between the agent and the environment during both training and evaluation phases. Even after the initial training phase, the model continues to improve and optimize based on real-time feedback from the environment. In the evaluation phase, the agent interacts with the environment in real-time to make decisions. This real-time interaction allows the agent to adapt more quickly and choose optimal actions in response to dynamic changes in the environment. MPC Q-learning is particularly effective in addressing uncertainties and variability that may occur in real-world environments. It allows continuous refinement of the model based on actual feedback.

Multi-agent System (MAS). In many practical scenarios, the environment contains multiple agents, which are collectively referred to as a multi-agent system (MAS). In these systems, agents may have the same goal (cooperative) or be adversarial to each other (non-cooperative). When solving a problem using RL, one may consider either a centralized control, where the central entity has access to the information of all agents and decides their action, or a decentralized control, where each agent must update its own knowledge and make decision independently.

Backup Diagrams. Backup diagrams in reinforcement learning are a visual tool, introduced by Sutton and Barto [127], which serves to illustrate the manner in which value updates are conducted in various reinforcement learning algorithms. In these diagrams, each node represents either a state or a state-action pair, while arrows indicate the direction of information flow for updating estimates based on rewards or subsequent values. The backup patterns of different algorithms, such as Monte Carlo, Temporal-Difference (TD), and n-step methods, are distinct, demonstrating how they utilize experience over time. Figure 2.4 demonstrates examples of backup diagrams from [127] for SARSA algorithm. I explain the concept of backup diagrams for one-step SARSA algorithm in the following example.

Example 2.4.1. The popular SARSA algorithm is a TD algorithm. Given a Q-table and the learning rate α, it performs the following steps for a prescribed number of episodes. The second but last line, performs the update. The term $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$ defines the return G_t in this case, where $\gamma \in [0, 1]$ is the so-called discount factor, weighing immediate rewards vs future rewards. ■



3 State of the Art

Reinforcement learning [127] is a technique for decision-making in uncertain environments. It provides ways of learning controllers that tend to perform well either in presence of a model of environment or without it. RL has applications in many domains, including robotics [66], gaming [128], electronics [40], healthcare [152], water management systems [131], etc. This thesis examines two significant and complex applications of RL: target search using multiple UAVs and wastewater management in an urban area. The objective is twofold: firstly, to address real-world issues, and secondly, to identify challenges inherent to RL and the process of resolving these issues through RL.

The following section presents a concise overview of the current state of research in these two areas, along with an explanation of the gap in knowledge that this study aims to address.

3.1 Exploring by UAVs

Cooperative target search is a classic problem that has received a lot of attention in recent years [120, 143]. Search operations are inherently complex, due to their unpredictable nature [22]. Many works use biologically inspired meta-heuristics [29, 56, 104]. As multiple agents can collaborate effectively, *multi-agent reinforcement learning* (MARL) has the potential to improve the execution of search-and-rescue operations [18, 46, 143, 153]. In MARL, agents can adjust their search strategy in response to new information from the environment obtained by any other agent [28, 101]. The target search requires use of technologies such as communication, trajectory optimization, obstacle avoidance, and cooperative control [143]. It is important to respect the physical limitations of the agents.

Proximal policy optimization (PPO) shows good potential for the search-and-rescue problem [18, 142, 150, 153]. Yue et al. [153] use PPO to learn a strategy to suppress the enemy's aerial defense. Cai et al. [18] address the problem of cooperative navigation, modeling the navigation policy as a combination of dynamic target selection and collision avoidance. Xia et al. [150] focus on multi-target search by unmanned surface vehicles using a distributed partially observable multi-target hunting PPO algorithm. However, the collision avoidance issue has not been addressed adequately in the above works. To address the issue, Gandhi et al. [38] penalize the agent for not following a collision-free path to the target. This unfortunately requires that the agents collide many times before they learn to avoid collisions. Wang and Fang [143] propose to directly force the agents to move in the opposite direction when in vicinity of another agent. This requires that UAVs are able to rotate $180°$ instantly, which is not realistic for many types of drones, especially underactuated drones that are used in long-distance operations. For this reason, Wang and Fang [143] limit the angle of rotation that the agent can perform in a single time-step. Moreover, the above papers assume that the target locations are fixed during training and execution [79, 143].

This thesis proposes an enhanced collision avoidance strategy for multiple unmanned aerial vehicles (UAVs) operating in an environment with constrained range of motion. Moreover, the efficacy of a PPO algorithm is assessed in a search and rescue scenario, with varying degrees of complexity and environmental uncertainty Chapter 4.

3.2 Wastewater Management

Real-Time Control of Urban Water Management. Real-time control is a crucial strategic tool in urban water management [39, 63]. It leverages real-time sensor data such as rainfall and water levels to efficiently operate systems and preemptively address potential issues. In this field, real-time control is primarily used to minimize combined sewer overflows, manage stormwater quality, and optimize water supply and demand. We distinguish rule-based control and optimization-based control methods to implement real-time control. Rule-based control relies on the expertise of operational staff or offline optimization processes to determine actuator set-points. However, its fixed configuration constrains its capacity to adapt to varying rainfall-runoff scenarios [72]. model predictive control extensively researched as an optimization-based method, integrates system models with rainfall forecasts and optimization algorithms to recursively calculate optimal control actions [31, 77, 131]. Although model predictive control has been extensively studied, practical implementations are limited. Challenges include hardware instability, significant computational resource consumption in real-time optimization, and uncertainty in rainfall forecasting [76, 77].

Reinforcement Learning Based Control. Control synthesis based on reinforcement learning offers an alternative approach to water management. This methodology focuses on modeling complex environments and training agents to make optimal real-time control decisions [127]. In reinforcement learning, two main approaches are commonly used: direct and indirect learning [43]. Direct learning involves the agent interacting with the environment to learn a policy that maximizes rewards [14, 15, 157]. The agent observes states, selects actions to maximize rewards based on these observations, and learns iteratively through trial and error. This approach is particularly useful when the environment is complex or not fully known beforehand. Finding the path that maximizes rewards requires many trials and experiments, and it does not guarantee safety. In contrast, indirect learning utilizes a model of the environment to predict states and rewards, improving policies based on these predictions. This method can enhance learning efficiency when the model accurately represents the environment [78, 113, 131, 141]. For instance, in water resource management, modeling the stormwater system allows for simulation-based optimization of control strategies before implementation in the real-world, showcasing the applicability of indirect learning in complex real-world scenarios. The accuracy of the environmental model significantly affects the performance of learning. If the model is inaccurate, it can limit policy improvement. The choice between these approaches depends on the specific characteristics and requirements of the problem at hand, playing a crucial role in various applications of RL.

Control Based on Deep Reinforcement Learning. As is typical in such cases, the large state space inherent to this problem necessitates the use of deep reinforcement learning (DRL). It has emerged as a leading technique, demonstrating enhanced efficacy in flood mitigation. One of the earliest efforts to apply DRL for managing stormwater systems was investigated by Mullapudi et al. [97], who used Deep Q-learning [91] to control a system during heavy rainfall. Later they evaluated how the RL agent controls the multiple stormwater basins. Due to the risks associated with using a trial-and-error approach like RL in real-world applications, the usage of this work is limited to simulations of an actual system. The study conducted by Saliba et al. [114] demonstrates how another DRL algorithm, specifically Deep Deterministic Policy Gradients, enhances flood mitigation compared to a passive control system considering data uncertainty of both state and forecast data. Yin et al. present a method that enables a single neural network for both combined sewer overflows prediction and optimization tasks [151]. Tian et al. [131] train five distinct DRL models, including PPO, with varied architectures, with control actions selected by a voting mechanism. Negm et al. provide a comprehensive review of DRL applications in water systems [98]. Despite being a recent survey, it only references a few papers related to stormwater system control. None of the mentioned studies address the comparison between offline learning and model predictive control reinforcement learning in this context. Consistently, Fu et al. note that the application of deep learning techniques is still in its early stages, with most studies relying on benchmark networks, synthetic data, and laboratory or pilot systems to test performance [36].

This thesis examines the potential of employing reinforcement learning-based controllers for the development of a storage-release controller, with the integration of weather forecasts. It is proposed to implement a controller using model predictive control in conjunction with Q-learning. It is then demonstrated for the area west of Copenhagen Chapter 5.

Dealing with the aforementioned problems have highlighted significant shortcomings within the RL community. These include the handling of buggy implementations, the management of large state spaces in the context of tabular RL, and the sensitivity to initial states. The following sections of this chapter will provide an overview of the current state of the art in these areas.

3.3 Testing and Reinforcement Learning

One of the most significant shortcomings within the reinforcement learning community is that when an error is identified in the results, the primary focus is on the manner in which the problem was modeled with reinforcement learning, rather than on the underlying cause of the error. However, it is important to consider that programmers are prone to error, and it is therefore possible that we have a buggy implementation and lack the necessary testbed to identify and address implementation bugs. The following section provides a review of the state of the art with regard to the use of testing for reinforcement learning and vice versa.

In the field of reinforcement learning, a key topic of prior research has been the assessment of the reliability of a trained agent. The field of adversarial machine learning is concerned with understanding the behaviour of models and algorithms in contexts and situations that induce failure. Huang et al. investigate impact of the effectiveness of adversarial examples on a deep RL algorithm [47]. Lin et al. introduce strategically timed attacks on reinforcement learning agents [73]. Amirloo et al. propose to guide adversarial sampling by a predictor trained along with the agent to predict the probability of failure [2]. Vardhan and Sztipanovits use a generative model to find the failure scenarios [137]. Ruderman et al. study the worst-case analysis to detect the directions in which agents may have failed to generalize while learning the policy [112]. To overcome the small adversarial perturbations on the agent's inputs, Oikarinen et al. propose to train RL agents with improved robustness against l_p-norm bounded adversarial attacks [100]. All these works focus on optimality and generality of the obtained policies. However, they side step the problem of correctness of the reinforcement learning implementations used to learn the policies. In contrast, I follow a modular testing strategy, not unlike unit testing, for low-level properties of individual elements in RL applications, hoping that this exposes problems early and close to their origins. Furthermore, this approach also serves to encourage the development of formal verification techniques for reinforcement learning, as properties align with the conventional style employed in verification.

A large body of recent work studies the use of RL and deep RL to improve testing processes. Such techniques [74, 111, 124, 133, 134, 156, 158] are applied for testing a variety of systems (e.g., video games, web applications, and cyber physical systems). In contrast, this thesis is concerned with the opposite problem—applying testing to reinforcement learning.

Many authors focus on testing machine learning algorithms broadly. For example, optimizing stochastic regression tests in machine learning projects [30], augmenting a deep learning test set to increase its mutation score [110], testing bias in machine learning software [19], pointwise robustness in deep neural networks [148], concolic testing for deep neural networks [125], formally verifying safety properties of deep reinforcement learning system [51]. The present thesis does not contribute to testing neural networks (even if they are a representation of value functions used in reinforcement learning) but addresses testing the correctness of reinforcement learning problems and algorithm implementations by providing specifications for their basic blocks.

Other software engineering methods have been applied to test and verify reinforcement learning agents, including black-box fuzzing [102], search-based testing [129], mutation testing [75], deductive reasoning [26], using machine learning models and genetic algorithms to test policies [159]. Alur et al. have studied the formal specifications of reinforcement learning tasks [59] and of multi-agent reinforcement learning problems [62], transforming task specifications in RL [7], RL algorithms in abstract decision processes [60], and compositional reinforcement learning from logical specifications [61] are other cases that software engineering to reinforcement learning. In contrtast, my work is concerned with providing a direct formal specification of correctness for RL problems and algorithms themselves, as opposed to the policies that they output. I develop a property-based test harness for all elements of a reinforcement learning problem and algorithm Chapter 6. I am not aware of similar formal definitions of RL problems that are precise and self-contained and nor of prior uses of property-based testing for reinforcement learning.

3.4 Partitioning the State Space of Reinforcement Learning

An examination of the RL applications referenced in Sections 3.1 and 3.2 revealed limitations associated with the use of DRL algorithms, particularly in terms of explainability and convergence guarantee. These shortcomings are not inherent to tabular RL. Nevertheless, tabular approaches necessitate a good representation of the state space, which is a challenging task. This section presents a review of the existing literature on this topic.

Reinforcement learning can automatically synthesize controllers for many challenging control problems [127], however dedicated approximation techniques, hereunder deep learning, are needed for continuous state spaces. Unfortunately, despite many spectacular success with continuous problems, deep reinforcement learning suffers from low explainability and lack of convergence guarantees. At the same time discrete (tabular) learning methods have been shown to be more explainable [82, 106, 140, 154] and to yield policies for which it is easier to assure safety [37, 55, 138]; for instance using formal verification [3, 57, 132]. Thus, finding a good state-space representation for discrete learning remains an active research area [4, 23, 53, 71, 85, 99, 145]. This representation is called partitioning of the state space of reinforcement learning which obtains by mapping from a continuous state space to a discrete one or by aggregating discrete states.

The trade-off between the size of the partitioning and the optimality and convergence of reinforcement learning remains a challenge [4, 23, 71, 85, 99, 145]. Policies obtained for coarse-grained partitionings are unreliable. Large fine-grained partitionings make reinforcement learning slow.

To the best of my knowledge the earliest use of partitioning was the BOXES systems [88]. The Parti-game algorithm [95] automatically partitions state spaces but applies only to tasks with known goal regions and requires a greedy local controller. While tile coding is a classic method for partitioning [6], it often demands extensive engineering efforts to avoid misleading the agent with suboptimal partitions. [70] extended learning classifier systems to use tile coding. Techniques such as vector quantization [71, 85, 99, 145] and decision trees [118, 136, 147] lack adaptability to the properties of the state space and may overlook non-linear dependencies among state components. Techniques that gradually refine a coarse partitioning during the learning process [4, 23, 53, 85, 145] are time-intensive, and require generating numerous partitions to achieve better approximations near the boundaries of nonlinear functions.

The concept of bisimulation metrics [33, 34] defines two states as being behaviorally similar if they satisfy two criteria: (1) yield comparable immediate rewards and (2) transition to states that are behaviorally aligned. Bisimulation metrics have been employed to reduce the dimensionality of state spaces through the aggregation of states. Nevertheless, they have not been extensively investigated due to the high computational costs associated with their implementation. Furthermore, it should be noted that bisimulation-minimization-based state-space-abstraction is too fine-grained to be applicable to the problem at hand. The requirement that any states lumped together exhibit the same behavior is an unnecessary constraint from the perspective of reinforcement learning. This field does not favor any particular behavior, provided that it leads to the same long-term reward. As long as the same long-term reward estimate is expected for the same (best) local action in two states, it is sufficient for the two states to be lumped together. This suggests that it may be beneficial to explore weaker principles than bisimulation metrics for reducing dimensionality.

In Chapter 7, I develop a novel symbolic execution-based partitioning approach that incurs no learning costs (offline), requires no engineering effort (automated), and is not problem specific in contrast to some of the existing techniques (general). It produces a partitioning that effectively captures non-linear dependencies as well as narrow partitions, without incurring additional costs or increasing the number of partitions at the boundaries.

3.5 Initializing the RL algorithms Using Analysis of Simulation

It is widely acknowledged that the initial states of a trained model using reinforcement learning play an instrumental role in its performance [54]. Nevertheless, this topic has not been subjected to the same degree of investigation as that of initializing the policy [1, 10, 68, 135]. This is primarily due to the significance of attaining an optimal policy through the utilization of RL algorithms. However, the initialization of the policy would direct the agent to learn a policy that is roughly analogous to the initial one. This approach does not allow for the exploration of the environment in a more comprehensive manner, resulting in the limitation of the state space to a specific range. This algorithm gets stuck in local optima, which could impede the ability to explore the state space effectively. Furthermore, it may result in states that are less likely to be observed being entirely bypassed, which presents a significant risk for safety-critical systems. Another domain of research that has evidence on importance of initial states, partitions the state space into “slices”, and optimizes an ensemble of policies, each on a different slice [41].

Some research has been conducted on mapping the trained initial states at the simulator level to those in the real world. This is exemplified by studies in robotics [94, 122]. The aforementioned works do not address the initial states that an agent would take at the training level in a simulator. Instead, they focus on the initial states that an agent, such as a robot, should adopt when retrained or tested in the real world. In robotics, another branch of study attempts to learn the reset function or initial states [64]. Similarly, [87] selects states from past experiences and uses them to initialize the agent in the environment, thereby guiding it toward a more informative state. This work demonstrates the significance of maintaining a sufficiently limited initial state to facilitate the repetition of previously initiated states. Moreover, the initial state should be sufficiently expansive to guarantee that the agent has adequately explored it, which is not considered by any of the above mentioned works.

4 Target Search Using Deep Reinforcement Learning

This chapter addresses the target search problem using a group of Unmanned Aerial Vehicles (UAVs). We introduce a collision avoidance mechanism, called *Cooperative Rotating Maneuver* (CRM), a strategy for multiple UAVs searching in an environment with limited range of motion. Furthermore, we evaluate the effectiveness of a Proximal Policy Optimization (PPO) reinforcement learning algorithm in a search and rescue scenario, with varying degrees of complexity and uncertainty.

4.1 Introduction

The objective is to identify the optimal search strategy in terms of various parameters, such as the shortest path, minimum threat, or maximum efficiency [153]. In particular, this study focuses on the domain of search and rescue, in which emergency responders strive to locate and assist civilians in distress. In light of the inherently unpredictable nature of the well-being of the civilians in this situation, it is of the utmost importance that emergency responders attempt to locate and assist them as expeditiously as possible. The deployment of multiple UAVs to cover the search area is more efficient, reliable, and has faster run time than a single UAV search [120]. However, it introduces an additional layer of complexity, as the UAVs must coordinate with each other to optimally leverage their respective advantages while avoiding collisions, which could potentially result in the destruction of the UAVs and failure in finding the targets. This is referred to as cooperative target search, a topic of significant interest to numerous parties, including this study [58, 107, 149].

The objective of this chapter is to develop a PPO-based learning which enables a group of UAVs to cooperatively explore an unknown environment and locate a group of targets in an optimal manner, while avoiding collision with other UAVs (**RQ1**).

4.2 Method

The environment is represented as a rectangular area, with dimensions $W_B \times H_B$, which encompasses both targets and obstacles. The search is conducted by a team of UAVs that engage in collaborative operations. Our model is based on the environmental framework proposed by Wang and Fang [143], which implements collision avoidance by the generation of a repelling force between agents. Although this approach effectively prevents collisions, it is based on the assumption that agents are able to rotate with complete freedom of movement. However, this assumption is not compatible with reality, as UAVs are only able to rotate a limited degree in one step. Interestingly, this fact is considered in the motion model by Wang and Fang [143] and the action space utilized in the present study, wherein the agents' heading angle alterations are constrained.

To overcome this issue, we propose an alternative collision avoidance mechanism, called *Cooperative Rotating Maneuver* (CRM), that operates as follows. Let $i \in I$ be an agent, and let E be the set of agents that are closer than c to i, so i shall attempt to avoid collisions with them. Then define a set $V = \{[x_i - x_e, y_i - y_e]^\intercal \mid e \in E\}$ and $\vec{v}_f = \sum_{\vec{v} \in V} \hat{v}$, where $\hat{v}$ is $\vec{v}$ normalized. Vector $\vec{v}_f$ is the desired new heading vector for agent i. The desired heading angle φ^θ is then atan2$(\vec{v}_f)$. Finally, the new heading angle is calculated by, ensuring that any change in the heading angle θ^i is limited to $\Delta\theta_{\max}$.

$$\theta^i_{t+1} = \begin{cases} (\theta^i_t - \Delta\theta_{\max}) & \varphi^\theta_t \leq (\theta^i_t - \Delta\theta_{\max}) \\ (\theta^i_t + \Delta\theta_{\max}) & \varphi^\theta_t \geq (\theta^i_t + \Delta\theta_{\max}) \\ \varphi^\theta_t & (\theta^i_t - \Delta\theta_{\max}) < \varphi^\theta_t < (\theta^i_t + \Delta\theta_{\max}) \end{cases} \tag{4.1}$$

However this method increases the likelihood of collisions compared to the one proposed by Wang and Fang [143] in a simulation environment, it is feasible to execute it in a real-world scenario. Furthermore, we evaluate the performance of the PPO algorithm in a search and rescue scenario, accounting for varying levels of complexity and uncertainty, which were not previously explored by Wang and Fang [143].

4.3 Results

To evaluate the proposed method presented in this work, the following questions will be addressed.

To what extent do the different spawn modes of target and obstacle affect the average performance of a PPO algorithm in cooperative target search? To answer this question, we assess the impact of uncertainty on the rate of completion of a search. To this end, the average reward and time to completion will be collected in four scenarios: **a)** obstacles and targets are fixed in position, **b)** obstacles are randomly placed in each episode but targets are fixed, **c)** obstacles are fixed but targets are randomly placed in each episode, and **d)** obstacles are randomly placed and targets are randomly placed. Subsequently, an analysis of the resulting data facilitate the comprehension of the strategies employed by the agents. Additionally, the values of target and obstacle spawn modes will be modified. The specific configuration of these parameters will be delineated for the data in question.

Table 4.1 (Right) depicts the influence of uncertainty in target and obstacle spawn mode on the mean number of steps requisite to complete a mission. There is a discernible discrepancy in the time required to complete a mission when the targets remain at known locations versus when they are randomly generated (Table 4.1). The former scenario exhibits a completion time of approximately 15.5 steps, while the latter requires approximately 29.3 steps. This is to be expected, given that the change from

Reward		**Obstacles**	
		const.	**rand.**
Targets	**const.**	0.5	0.46
	rand.	0.4	0.26

Time		**Obstacles**	
		const.	**rand.**
Targets	**const.**	15	16
	rand.	27	32

Table 4.1: Mean reward per episode with different modes of targets and obstacles (Left), Mean steps to completion with different modes of targets and obstacles (Right).

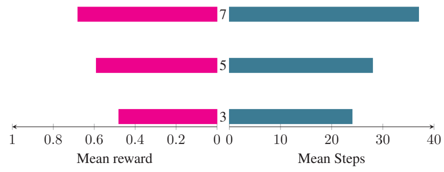
constant to random target spawn alters the nature of the problem from pathfinding to area coverage.

When the targets are fixed at known locations, the obstacle spawn mode has only minimal impact. This is due to the fact that, the probability of an obstacle randomly appearing on path between an agent and a target is relatively low. In the event that an obstacle does spawn on the agent's path, it is not problematic for the agent, as it can simply fly higher to pass the obstacle at the cost of some negative reward. This is illustrated in Table 4.1 (Left), in which the average episodic return is shown to be more significantly affected by the random obstacles with constant targets than the average steps to completion would suggest. However, when the targets are random, the change in obstacle spawn mode is more significant. This is because the projected path of the agent is much longer for the case of an area-coverage problem. Consequently, it is more probable that an obstacle will spawn on the path.

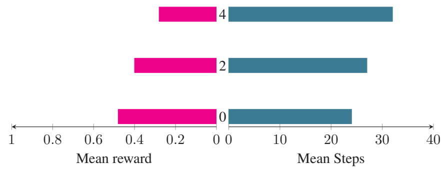
To what extent do number of targets and obstacles affect the average performance of a PPO algorithm in cooperative target search? To answer this question, we gather the mean number of steps required to complete the mission while incrementally increasing the number of obstacles and targets. We then undertake a comparative analysis of the results in order to ascertain the impact of this on mission success. It is our intention to use these results in order to gain insight into the significance of complexity and to establish a benchmark for the complexity of a scenario that our simulation is capable of addressing. In order to achieve this, we vary the number of targets and obstacles.

Figure 4.1a shows a significant correlation between an increase in the number of targets and the time required to locate all targets. This is an inherent consequence of the revised objective, which now includes the additional goal of locating a greater number of targets. As illustrated in Figure 4.1a, an increase in the number of targets is accompanied by an increase in the mean reward. This is expected, as locating a target results in a positive reward, and the average number of additional steps required per new target is less than the reward for locating a target.

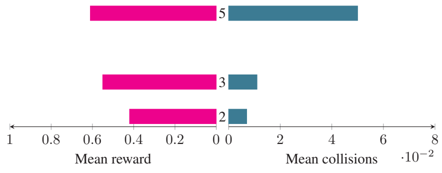
Figure 4.1b shows a negative correlation between the number of obstacles and the average steps to completion. This is a logical consequence of the fact that obstacles impede the optimal routes, compelling the agents to traverse alternative paths to achieve coverage. Furthermore, Figure 4.2a illustrates the correlation between collisions and the number of obstacles. Given that the episodic return incorporates both search time and collision frequency into account, it is unsurprising that Figure 4.1b demonstrates a negative correlation between the number of obstacles and average episodic reward.



(a) An analysis of the impact of the number of targets on search reward and the required number of steps to complete a task.

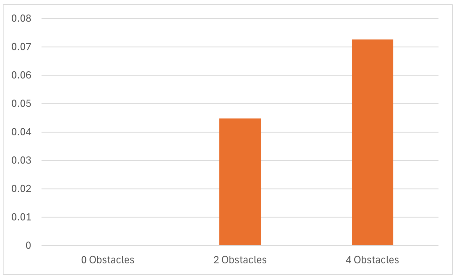


(b) An analysis of the impact of the number of obstacles on search reward and the required number of steps to complete a task.

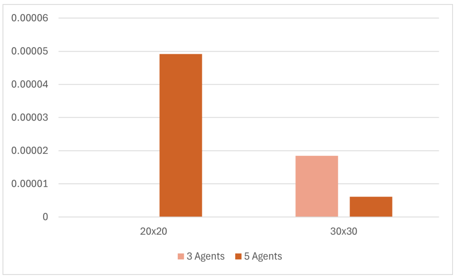


(c) An analysis of the amount of collisions and mean reward with no collision avoidance and different number of agents.

Figure 4.1: Analyses of the impact of the number of targets, obstacles, and collision distance.



(a) Mean collisions per step while varying the number of obstacles.



(b) Mean collisions per step while varying the size of search area and number of agents.

Figure 4.2: Analysis of collisions with varying number of obstacles and size of the search area.

How does our collision avoidance mechanism, CRM, perform, and how is it affected by the number of agents? To answer this question, we quantify the number of collisions that occur between agents while varying the values of number of obstacles, number of UAVs, and collision avoidance distance between experiments. Subsequently, we examine how the collision frequency varies contingent on the scenario and contrast these variations to gain insight into the overall performance. Additionally, we evaluate CRM in comparison to the mechanism proposed by Wang and Fand [143] and the model devoid of a collision avoidance mechanism, employing the same metrics.

Table 4.2 and Figure 4.3 report the average collision frequencies, and rewards for both collision avoidance mechanisms, while varying number of UAVs and collision

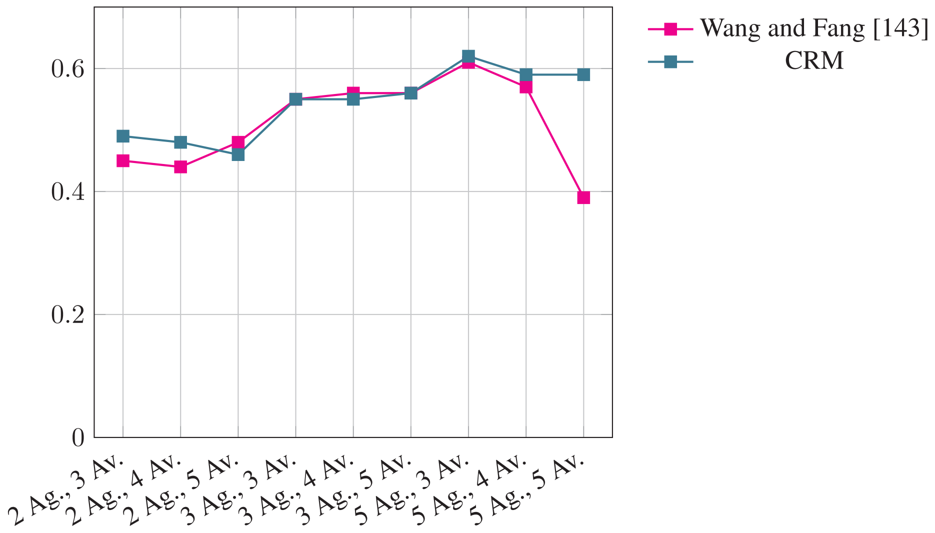


Figure 4.3: Mean reward per episode while varying the number of agents and collision distance for presented collision avoidance mechanism in this work in compare with the presented mechanism by Wang and Fang [143].

avoidance distance. As you can see in Table 4.2, the collision frequency of CRM is slightly higher than the collision avoidance mechanism presented by Wang and Fang [143]. This is to be expected, since they introduce an infeasible collision avoidance by repelling force between agents, while CRM guides agents to rotate as much as they can but cooperatively.

	2 Ag., 3 Av.	2 Ag., 4 Av.	2 Ag., 5 Av.	3 Ag., 3 Av.	3 Ag., 4 Av.	3 Ag., 5 Av.	5 Ag., 3 Av.	5 Ag., 4 Av.	5 Ag., 5 Av.
CRM	6.2	0.0	0.0	0.9	0.0	0.0	3.3	0.5	0.1
Wang and Fang [143]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 4.2: Mean collisions per step while varying the number of agents and collision distance for presented collision avoidance mechanism in this work in compare with the presented mechanism by Wang and Fang [143]. All reported values are scaled by 10^{-4}.

Table 4.2 shows that the frequency of collisions for three agents is less than that observed for two or five agents. This phenomenon can be attributed directly to the underlying cause of collisions. In the case of a limited number of agents, the primary cause of collisions is when two agents enter the collision avoidance distance while facing each other directly. In such instances, the projected corrected courses for both agents intersect as a consequence of the restricted turning range, thereby precipitating a collision. In the case of two agents, the agents spawn far from each other. Consequently, by the time the agents meet, there is a greater degree of randomness in the potential directions they could be facing. In contrast, when five agents are present, they encounter one another with greater frequency, reducing the probability that they will be facing directly toward one another. However, due to the increased number of agents, the probability of more complex and risky scenarios, such as three or more agents interacting, increases. Three agents represents an optimal equilibrium given the scale of the environment. It should be noted that these probabilities are subject to change as the size of the area increases Figure 4.2b. For the same reason we explained above, finve agents has fewer collisions, in compare to three agents.

Figure 4.1c shows a significant correlation between the number of agents and the number of collisions when collision avoidance is not in effect. Additionally, the mean reward value increases with the number of agents. This can be attributed to the discrepancy between the penalty for a collision and the reward for identifying all targets. A comparison of the number of collisions per step, as illustrated in Table 4.2 and Figure 4.1c, reveals that the CRM mechanism has a significant impact on reducing collisions. However, a comparison of Figures 4.1c and 4.3 indicates that there is no notable change in the mean reward. Consequently, it can be concluded that the CRM fulfills its intended purpose.

4.4 Conclusion

These findings indicate that the enhanced collision avoidance mechanism is applicable in scenarios with constrained range of motion, effectively reducing the incidence of collisions while maintaining the overall operational success rate. Furthermore, the results of simulation experiments demonstrate that augmenting the intricacy and ambiguity of the cooperative target search model leads to an elevated mean target search duration. Furthermore, the following observation were derived from the research.

- Designing a neural network for PPO, as a DRL algorithm, is not a straightforward,
- PPO could find a good policy (near optimal) but the policy is not easily explainable,
- Training an agent in a real-world scenario is not always feasible due to the costs, dangers, time constraints, and other factors.

It should be noted that this work serves as a guide for the remainder of the dissertation. In addition to the the aforementioned observations of the current work, which led us to develop a method for partitioning the state space to be able to use tabular reinforcement learning, it also revealed significant yet previously unidentified challenge in RL. The testing of the optimal collision avoidance for the proposed mechanism by Wang and Fang [143] demonstrates that their approach is effective. Nevertheless, if one were to attempt to apply this methodology in a real-world scenario, it would most likely fail and may result in an expensive damage. This is due to the fact that the model has not been tested independently. One of the properties that will fail if the test is conducted is as follows. *As they are using centralized MARL, the action space is a Cartesian product of the actions of all agents, which is the pose rotation of each agent. As the UAVs are capable of rotating within a limited range, the action selected by each* *agent, represented by* a^i *for* i*th agent, must satisfy the constraint* $-30° \leq a^i \leq 30°$. *Consequently, the action generated by the collision avoidance mechanism must also adhere to this constraint. However, testing this property will fail because the collision avoidance mechanism is designed to choose* $180°$. This observation motivated us to investigate the potential for developing a framework for property-based testing for reinforcement learning applications.

5

Wastewater Management Using Reinforcement Learning

This chapter addresses the challenges posed by urban stormwater runoff in light of increased urbanization and climate change, which strain traditional stormwater infrastructure. We focus on mitigating Combined Sewer Overflows by maximizing urban runoff storage in stormwater tunnels during Wastewater Treatment Plant capacity overloads. Unlike passive rule-based control, this research explores adaptive control systems that utilize weather forecasts and dynamic strategies. We introduce a novel control synthesis approach, combining Model Predictive Control (MPC) and Q-learning, to optimize combined sewer overflow management based on real-time weather predictions. We evaluate our models on the Hvidovre stormwater tunnel in Copenhagen—Denmark, simulated in EPA-SWMM.

5.1 Introduction

Increasing urbanization and impervious surfaces in cities contribute to rising wastewater runoff and the release of pollutants into the surrounding ecosystems [32, 146]. Stormwater systems manage runoff to prevent flooding and reduce environmental impact, but climate change strains their effectiveness [5, 119]. Urban stormwater infrastructure can be separated or combined with sewers. Combined sewer systems often generate combined sewer overflow during storms, mixing sewage and stormwater, necessitating treatment before discharge [83]. Various infrastructural solutions, collectively termed Best Management Practices, aim to mitigate the impacts of urban runoff [84]. These solutions generally aim to reduce stormwater inflow, increase combined stormwater and sewage storage, and treat overflows. This paper focuses on minimizing combined sewer overflow by storing it before treatment, as retrofitting existing infrastructure for changing rainfall is costly. Consequently, water utility companies seek better methods to utilize their current infrastructure.

This chapter is concerned with the combined sewer overflow storage system for the area west of Copenhagen, the so-called Hvidovre pipeline. The current rule-based controller in the Hvidovre tunnel is reactive. It responds to the present conditions (water level of the pump storage and gates) rather than proactively considering future events, such as a forecast of heavy rain. It keeps all tunnel gates completely open during dry

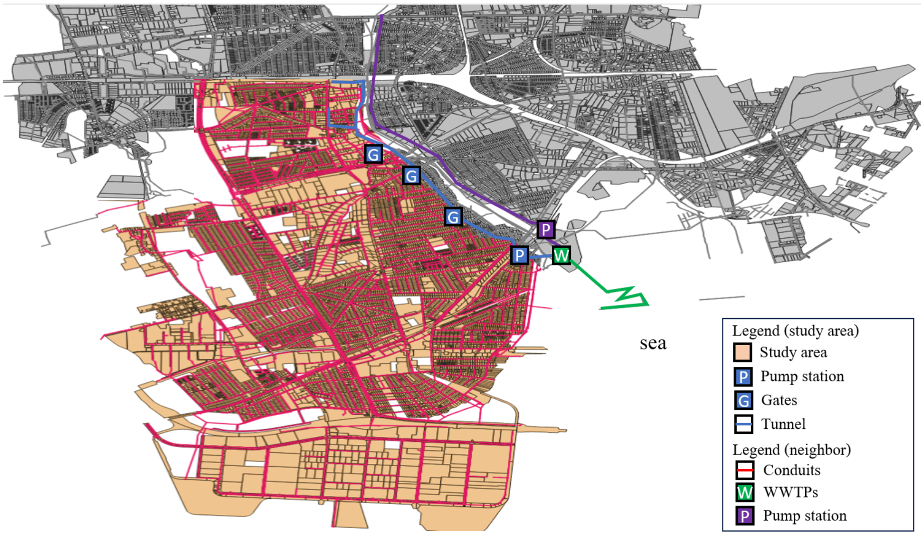


Figure 5.1: The essential elements in our case study, including the Hvidovre tunnel

weather and light rain. We study the feasibility of using RL-based controllers [127] for realizing a storage-release controller, taking weather forecasts into account.

The objective of the combined sewer overflow storage system, as illustrated in our case study, can be described as twofold. The initial objective is to eliminate overflow at the pumping station. This phenomenon occurs when underground pipes in residential areas become inundated with water, which may result in the overflow of sewage. The second objective is to reduce the occurrence of bypasses at the treatment plant. Bypasses occur when untreated sewage overflows into nearby rivers. The objective of this research is to investigate the possibility of achieving the aforementioned goals through the utilisation of a reinforcement learning method that benefits from a simplified model of the system for decision-making purposes. This enables us to respond to **RQ2** of the dissertation.

5.2 Method

Case Study. The case study area concerns the stormwater tunnel in Hvidovre municipality of Copenhagen, Denmark (Figure 5.1). The wastewater from this urban region, along with rainwater, is collected in the Hvidovre tunnel, depicted by blue solid lines in Figure 5.1, which constitutes the combined sewer system. The tunnel is segmented into four sections. Each section is equipped with three gates (indicated by blue 'G') and AMN (Åmarken Nord) pump station (indicated by blue 'P'). There are two pumps: the default pump and the emergency pump, each with different performance characteristics. The system measures the water levels at the gates, the AMN pump station, and the flow rate at the wastewater treatment plant (indicated by green 'W'). It controls the gates and the AMN pump operation to regulate the

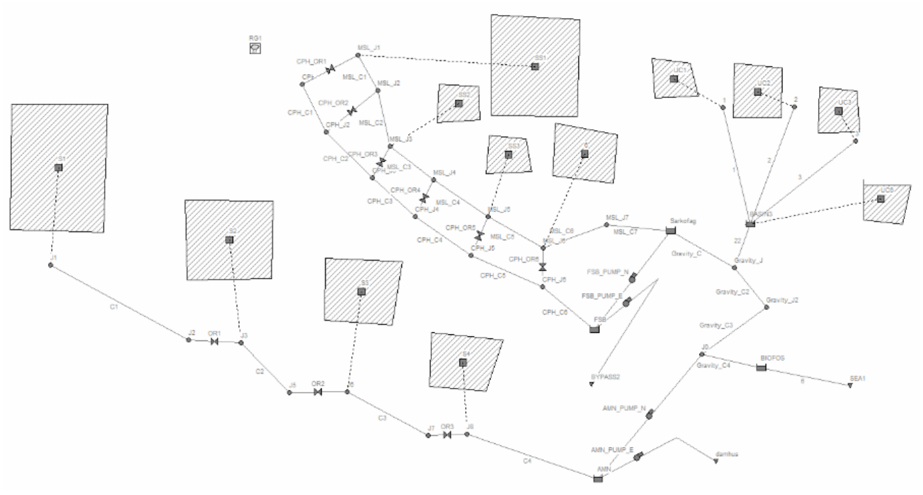


Figure 5.2: SWMM model of the system.

inflow rate into the wastewater treatment plant, ultimately minimizing combined sewer overflows. The wastewater treatment plant consists of two stages: a physical treatment stage and a biological treatment stage. The purified wastewater, having passed through both stages, is discharged into the sea.

5.2.1 Modeling

We first model the combined sewer system, as illustrated in Figure 5.2, along with the case study and neighboring infrastructures, reflecting the real-world system under investigation. Then, we train a controller using Model Predictive Control (MPC) in conjunction with Q-Learning [144] against an *abstract model* of the water environment and weather within the tool UPPAAL STRATEGO [25]. We also train an alternative controller using standard Q-Learning directly using a SWMM model [48, 86] as an oracle, without an abstract model. We compare the models against a baseline existing rule-based controller using a SWMM model.

5.3 Results

We evaluate the effectiveness of four control approaches: (i) MPC Q-learning, (ii) classic Q-learning, (iii) Q-learning using UPPAAL model, and (iv) Rule-based control. Our assessment focuses on the control objectives of removing overflow at the pump station and reducing bypasses into wastewater treatment plant. We compare the performance of different controllers synthesized using various methods by applying them to the PySWMM model [86].

Table 5.1 presents the experimental results of aforementioned approaches. We assess the performance of each control method using a rain scenario that the system had not previously encountered, employing k-fold cross-validation. Each experiment was

	MPC Q-learning (UPPAAL)		**Q-learning (Classic)**		**Q-learning (UPPAAL)**		**Rule-based**	
	O_T(m^3)	B_T(m^3)	O_T(m^3)	B_T(m^3)	O_T(m^3)	B_T(m^3)	O_T(m^3)	B_T(m^3)
Rain 1	0	30,460 ± 180	0	37,860 ± 160	0	38,236 ± 80	16,200	24,840
Rain 2	0	2,195 ± 55	0	2,520 ± 51	0	3,155 ± 37	0	5,330
Rain 3	0	4,701 ± 78	0	5,580 ± 90	0	5,774 ± 53	0	6,280
Rain 4	0	5,160 ± 70	0	6,059 ± 90	0	6,923 ± 30	0	7,200
Rain 5	0	31,510 ± 240	0	37,668 ± 300	0	44,041 ± 190	16,920	27,300

Table 5.1: The evaluation results over rain events in prioritizing overflow O_T than the bypass of pump station B_T.

repeated five times, and the mean values and standard errors, with a 95% confidence interval. The rule-based controller result shows overflow in two extreme rain scenarios, while both the learning-based methods have no overflow in any case. Across all rain scenarios, when considering bypass, MPC Q-learning consistently outperforms classic Q-learning by an average of 17%, Q-learning using UPPAAL by 34%, and rule-based controller by 72%. These findings confirm the effectiveness of MPC Q-learning and Q-learning-based controllers in achieving primary operational goals.

Figure 5.3 presents qualitative outcomes of all methods only for the heaviest rain scenario. Due to similarity of the behavior of gates, we demonstrate the control behavior of gate 3. In the case of MPC Q-learning, it is observed that the gate remains closed until the water level is high to minimize gate operations and maximize water retention, compared to classic Q-learning and UPPAAL model based Q-learning. Compared to rule-based controller, it is noted that the gate opens more quickly to release water after detecting the end of the heavy rain. The last three plots in the figure show information about the pump station. Notably, the emergency pump only operates in the rule-based controller scenario. MPC Q-learning and Q-learning trigger a bypass earlier than rule-based controller, preventing the emergency pump from operating by maintaining a lower water level at the pump station. This decision is guided by incorporating weather forecasts into the control synthesis to better prepare for heavy rain. In the case of MPC Q-learning control, it can be observed that the control behavior frequency is lower compared to Q-learning based control, and the amount of bypass is also less.

5.4 Conclusion

The primary findings of this chapter indicate that MPC Q-learning, even when employing an abstract model of the environment, can facilitate the learning of a better controller for the real-world problem. The key contributions of this study are summarized as follows.

- RL-based control strategies can more effectively mitigate combined sewer overflows in both the pumps and gates of combined sewer systems compared to the rule-based controller used in real systems,

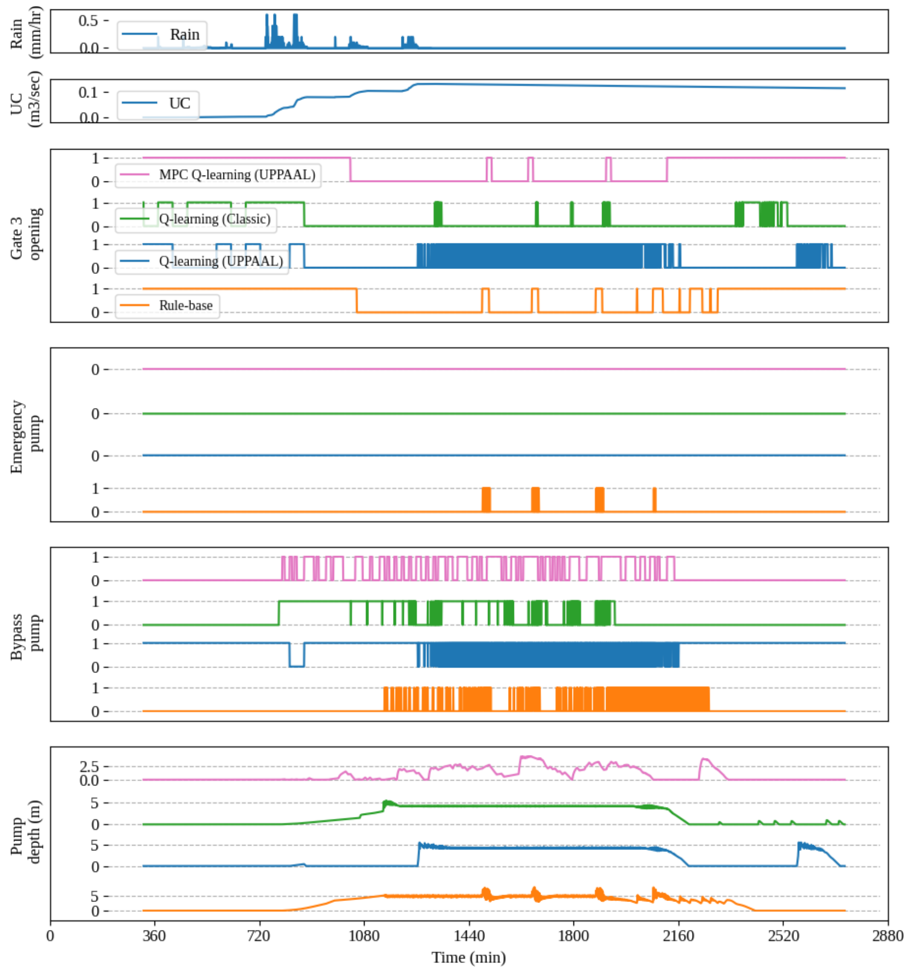


Figure 5.3: Results of MPC Q-learning using UPPAAL model, Q-learning, Q-learning using UPPAAL model, and rule-based controller applied control strategies in response to Rain Scenario 1.

- Among different RL-based controllers, MPC Q-learning using the UPPAAL model proved to be the most suitable controller for adapting to unknown environmental conditions.

Furthermore, the following observation were derived from the research.

- The implementation of an RL algorithm and application would result in the generation of numerous bugs, for which there is currently no tool for testing and finding,
- Training an agent in a real-world scenario is not always feasible due to the costs, dangers, time constraints, and other factors,
- Finding an appropriate discretization for a continuous state space, to allow us using tabular RL, is a challenging task that requires a significant effort and a detailed understanding of the environment.

6 Testing Reinforcement Learning

This chapter presents a systematic approach to testing development of reinforcement learning applications. We present a formal specification of reinforcement learning problems and algorithms, with a particular focus on temporal difference methods and their definitions in backup diagrams. Furthermore, a testing harness for a substantial category of reinforcement learning applications based on temporal difference learning, including SARSA and Q-learning, has been developed.

6.1 Introduction

A typical development process for a reinforcement learning application is exploratory rather than systematic, which has the effect of reducing the potential for reuse between applications and increasing the probability of introducing errors into the implementation. This, in turn, has the consequence of reducing the overall trustworthiness and effectiveness of the applications. Unfortunately, techniques and tools for systematic quality assurance of reinforcement learning applications are few and rare.

The research field of reinforcement learning tends to focus on the evaluation of the quality of the obtained policies that are obtained. Nevertheless, the findings of even the most precisely designed evaluation experiments are of limited value unless the assessed methodologies are implemented correctly. Our long-term objective is to facilitate the systematic development and testing of RL applications (to answer **RQ3** in this dissertation), thereby enhancing the reliability of their outcomes through the provision of convenient, automated testing. In this chapter, we propose a methodology for directly formalizing the specifications of correctness for RL problems and the learning algorithms themselves, without considering the policies that they output.

6.2 Method

To address an optimization problem using reinforcement learning, one should consider three primary elements: the agent, the environment, and the algorithm. As reinforcement learning algorithms are concerned solely with changes in the environment that affect the agent (state), it is possible to incorporate these changes into the agent component, thereby eliminating the need for the environment component. Therefore, it is sufficient to formally specify only the agent and the RL algorithms. We present a formal specification of each element of the agent, including state, action, observation, reward, transition, and so forth. Then the reinforcement learning algorithm needs to be specified. In this research, we focus on specifying temporal difference reinforcement learning methods [126], which update an estimate of a state-action value function using a number of sampling steps and a prior estimate. As our objective is to ensure the correctness of the algorithm and the agent, we have settled on simple representations of value estimation—discrete tables. However, our specification method appears to be robust also for approximating representations, such as neural networks used in deep reinforcement learning.

We perform a domain analysis of the temporal difference algorithms and produce a formal specification. We identify commonalities and differences between the different reinforcement learning problems and between the various temporal difference algorithms. Particular attention is paid to the update step in the algorithms, commonly described using so-called *backup diagrams* [126]. Our domain analysis formally defines a *back-up diagram language*—a compositional, domain-specific language for describing updates in reinforcement learning. An interpreter for back-up diagram language, formalized in a denotational style, serves as a specification of correctness for updates in individual temporal difference algorithms. The ability to characterize many temporal difference algorithms at once is enabled by a compositional denotational definition of this interpreter. The grammar of the backup diagrams language is as follows.

$$\begin{aligned} \mathit{est} &::= \mathsf{sample}^{\gamma} \mid \mathsf{expectation}^{\gamma} \\ \mathit{bdl} &::= \mathit{est}^{+}\ \mathsf{Update}^{\alpha}(\mathsf{sample} \mid \mathsf{expectation})\ . \end{aligned} \tag{6.1}$$

Example 6.2.1. Figure 6.1 lists backup diagrams, their *bdl* abstract syntax, and Sutton&Barto-style return calculations for five examples of temporal difference algorithms. The *bdl* syntax for 1-step SARSA is $\mathsf{sample}^{\gamma}\ \mathsf{Update}^{\alpha}\ \mathsf{sample}$. The update step includes updating the Q-value of a state-action pair (s_t, a_t) using the reward and the Q-value of the state-action pair (s_{t+1}, a_{t+1}) resulting from one time policy sampling. Similarly, for the 2-step SARSA, the diagrams represent two sampling steps composed with a sampling update: $\mathsf{sample}^{\gamma}\mathsf{sample}^{\gamma}\ \mathsf{Update}^{\alpha}\ \mathsf{sample}$. In contrast, the update in the last step of n-step Expected SARSA is calculated by taking an expectation of values for all possible actions instead of using the value for the sampled action. See Figure 6.1 for this and further examples. ■

In the following, I demonstrate an example of the aforementioned semantics in Paper 3 (for further details, please refer to the original paper). Figure 6.2 shows the semantic function for a sampling estimation step. First, the step function is applied to the environment state $\overline{s}_t$ and action a_t. This results in a distribution over successor states, and we bind a successor to $\overline{s}_{t+1}$. The reward is computed deterministically for a pair of successor state $\overline{s}_{t+1}$ and the performed action a_t, the result bound to r_{t+1}. The observable target state s_{t+1} is obtained using a deterministic function $\mathcal{O}$, and the next action is selected on policy π obtaining a non-trivial distribution again. At this point, we compute the return by adding the prior return $G_{:t}$ with the discounted reward. Finally, the discount factor is updated by this step's discount rate γ. In the last line

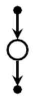
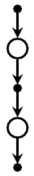
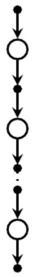
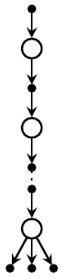
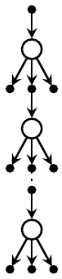
Diagram	BDL (abstract syntax)	Return ($G_{t:t+n}$) in Update formulae
	**1-step SARSA** sample$^{\gamma}$ Update$^{\alpha}$ sample	$r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1})$
	**2-step SARSA** sample$^{\gamma}$sample$^{\gamma}$ Update$^{\alpha}$ sample	$r_{t+1} + \gamma r_{t+2} + \gamma Q_{t+1}(s_{t+2}, a_{t+2})$
	**n-step SARSA** (sample$^{\gamma}$)^{n} Update$^{\alpha}$ sample	$\begin{cases} r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} \\ \quad + \gamma^n Q_{t+n-1}(s_{t+n}, a_{t+n}) & \text{if } n \geq 1 \wedge t < T-n \\ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \cdots + \\ \quad \gamma^{T-t-1} r_T & \text{otherwise} \end{cases}$
	**n-step expected SARSA** (sample$^{\gamma}$)^{n} Update$^{\alpha}$ expectation	$\begin{cases} r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} \\ \quad + \gamma^n \sum_a Q_{t+n-1}\pi(a\|s_{t+1})(s_{t+1}, a) & \text{if } t < T-n \\ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \cdots \\ \quad + \gamma^{T-t-1} r_T & \text{otherwise} \end{cases}$
	**n-step tree backup** (expectation$^{\gamma}$)^{n} Update$^{\alpha}$ expectation	$\begin{cases} r_{t+1} + \gamma \sum_{a \neq a_{t+1}} Q_{t+n-1}\pi(a\|s_{t+1})(s_{t+1}, a) \\ \quad + \gamma\pi(a_{t+1}\|s_{t+1}) G_{t+1:t+n} & \text{if } t < T \ \wedge n \geq 2 \\ r_{t+1} + \gamma \sum_a \pi(a\|s_{t+1}) Q_t(s_{t+1}, a) & \text{if } n = 1 \\ r_T & \text{if } t = T-1 \end{cases}$

Figure 6.1: Examples: A representation of updates in temporal difference learning as (a) a backup diagram, (2) a *bdl* term, and (3) a return calculation after Sutton & Barto [127]. T is the final time step in an episode.

$$\begin{aligned}
[\![\mathsf{sample}^{\gamma}]\!]_{est}\, Q_t \;=\; & \lambda(\bar{s}_t, a_t, G_{:t}, \gamma_t).\, \mathcal{T}\ \bar{s}_t\ a_t \ \gg= && \text{(run the system in state } \bar{s}_t \text{ executing action } a_t\text{)}\\
& \lambda\bar{s}_{t+1}.\, \mathrm{Det}\,(\mathcal{R}\ \bar{s}_{t+1}\ a_t) \ \gg= && \text{(reward } r_{t+1} \text{ for the action and the resulting state)}\\
& \lambda r_{t+1}.\, \mathrm{Det}\,(\mathcal{O}\ \bar{s}_{t+1}) \gg= && \text{(observe discrete state } s_{t+1} \text{ reached)}\\
& \lambda s_{t+1}.\, \pi\, Q_t\, s_{t+1} \gg= && \text{(select the next action } a_{t+1} \text{ following policy } \pi\text{)}\\
& \lambda a_{t+1}.\, \mathrm{Det}\,(G_{:t} + \gamma_t r_{t+1}) \ \gg= && \text{(discount accumulated return by } \gamma_t \text{ and add } r_{t+1}\text{)}\\
& \lambda G_{:t+1}.\, \mathrm{Det}\left(\gamma_t \cdot \gamma\right) \ \gg= && \text{(accumulate the discount factor)}\\
& \lambda\gamma_{t+1}.\, \mathrm{Det}\,(\bar{s}_{t+1}, a_{t+1}, G_{:t+1}, \gamma_{t+1})
\end{aligned}$$

Figure 6.2: Semantic function for a sampling estimation step.

all the four elements, i.e., next state $\overline{s}_{t+1}$, next action a_{t+1}, return $G_{:t+1}$, and discount factor γ_{t+1} to be used in the next step are returned.
The obtained specification may be employed for of testing and verification. We translate the aforementioned specification into a test harness using the property-based testing paradigm. The test harness comprises a general interpreter of back-up diagram language terms—formal models of updates for temporal difference learning algorithms. It is used as an oracle. The testing harness can be imposed on different algorithms and can be extended with properties specific for an agent.

In our test harness, we have categorised the tests for each problem and algorithm of reinforcement learning into two distinct groups: generic and specific. The following examples illustrate the characteristics of these categories.

Generic Problem Properties. These properties should hold for all implementations of RL problems. We begin with the totality of the observation function $\mathcal{O}$. The observation function $\mathcal{O}$ links the environment and the agent with the state space of the learning algorithm—as the algorithm only 'sees' and 'learns' about the observable states. The function $\mathcal{O}$ should be total in the sense that every system state should have a translation to an observable state; otherwise some system trajectories will lead to crashes or unexpected runs of the learning algorithm.

$$\forall \overline{s} \in \overline{\mathcal{S}}.\, \mathcal{O}\ \overline{s} \in \mathcal{S} \tag{6.2}$$

Specific Problem Properties. These tests capture idiosyncratic properties of the problem domain; they cannot be formulated without a concrete problem. I include a test for braking car problem. *Braking Car* describes a car moving towards an obstacle with a given velocity and distance. The goal is to stop the car to avoid a crash with minimum braking pressure. Equation (6.3) states that a forward-moving car cannot move backward by braking.

$$\forall s_1, s_2 \in \overline{\mathcal{S}}. \forall a \in \mathcal{A}.\ [s_1.v > 0 \wedge (\mathcal{T}\ s_1\ a)\, s_2 > 0] \rightarrow s_2.p \geq s_1.p \tag{6.3}$$

Generic Algorithm Properties. These tests apply widely to TD learning methods. Let Q_0 be the initial value of a Q-table. We test whether all entries of Q_0 are initialized to zero—a common choice—in Equation (6.4).

$$\forall s_0 \in \mathcal{S}.\ \forall a_0 \in \mathcal{A}.\ Q_0\,(s_0,\ a_0) = 0.0\ . \tag{6.4}$$

Specific Algorithm Properties. These tests require a concrete reinforcement learning algorithm. In an on-policy ε-greedy learning algorithm, the policy selects a random action with probability ε and otherwise it greedily follows the highest-value action. This requirement can be cast as a probability distribution test. For every state s_t, the selected action a_t should be distributed according to the distribution $\pi\, Q_t\, s_t$. In Equation (6.5), we derive a Boolean random variable that tracks selecting the highest value action. We check whether this random variable is distributed according to a Bernoulli distribution.

$$\forall Q_t \in \mathbf{Q}.\ \forall s_t \in \mathcal{S}.\ \left[(\pi\, Q_t\, s_t) \ggcurly (\lambda a_t.\, a_t \neq \arg\max_a Q_t\,(s_t, a))\right] \sim \mathrm{Bern}\left(\varepsilon \cdot \frac{|\mathcal{A}|-1}{|\mathcal{A}|}\right) \quad (6.5)$$

In practice during testing, the policy is not available as a symbolic representation of a distribution, but as a sampling algorithm. Therefore, testing the above law requires a statistical test. In our test harness for reinforcement learning, we perform a Bayesian test here. We use a weak prior (a Beta distribution) which encodes that the actual parameter of the Bernoulli distribution is essentially unknown. We collect a sample of executions of the policy and estimate the posterior belief in this parameter given the outcomes of these executions, whether the maximum value action or another action has been selected. This can be calculated analytically for a Beta prior using the conjugate update rule for a Bernoulli likelihood [69]. We check whether in the obtained posterior distribution over values concentrates 0.95 of the probability mass in a small *credible interval* containing $\varepsilon \cdot (1 - |\mathcal{A}|^{-1}|)$ (also known as a *high density interval* [69]).

In PBT, program properties are written as executable predicates over input data. Testing a property involves generating inputs automatically, evaluating a predicate on the inputs, and checking whether it holds on all the inputs. For each input data variable we need a test case generator. These are typically associated with the types in PBT, at least in strongly typed languages. The PBT testing libraries provide generators for standard types, and since generators are compositional, it is relatively cheap to add custom ones, as we also show below. PBT testing libraries are available for most mainstream programming languages.

6.3 Experimental Evaluation

The present study assesses the applicability of the specification and evaluates the effectiveness of the resulting tests harness. We examine the potential for reusing these tests, which could lead to a reduction in the cost of testing in such setups.

Is the specification sufficiently general to accommodate diverse reinforcement learning problems? In order to respond to this question, a series of case studies were implemented (first 5 columns in Table 6.1). It is evident that the types that implement the concepts of our formal specification are capable of validating reinforcement learning problems of varying scales. The framework allows the learning of policies through the execution of different learning algorithms and the test of applications. Each application has specific tests, and generic tests can be reused across applications. In total,

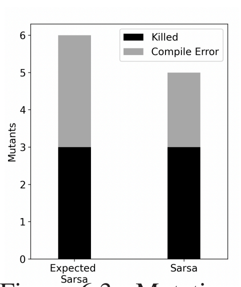
	state space size		episodic	gen. size	test cases	mutants			time	mutation
agent	continuous	observable		[LOC]	[#]	kill.	surv.	inv.	[s]	score [%]
Pumping	$\mathbb{R}\times\mathbb{R}\times\mathbb{R}\times\mathbb{R}$	92160	✓	17	31	43	16	0	1787	73
Mountain Car	$\mathbb{R}\times\mathbb{R}$	121	×	11	20	24	7	1	34	77
Braking Car	$\mathbb{R}\times\mathbb{R}$	12	✓	9	17	25	0	2	43	100
Windy Grid	-	70	✓	6	17	7	0	1	31	100
Cliff Walking	-	38	✓	6	13	31	1	0	5	97
Simple Maze	-	12	✓	7	15	26	2	0	8	93
Golf	-	10	✓	6	12	9	1	0	7	90
K-arm Bandit*	-	2	✓	3	5	2	0	4	7	100
Unit Agent	-	1	✓	3	-	-	-	-	-	-

Table 6.1: Experiment results. *K-arm bandit is the class of stateless randomized problems. The Unit agent was used in testing learning algorithms but it has no tests itself and was not mutated as it represents an artificial problem.

the case studies collectively necessitated as little as 68 lines of code for generator implementations. Consequently, the generators are not difficult or costly to develop. This highlights the benefit of using the types provided by the specification, which facilitates reuse in the implementation of reinforcement learning problems.

How effective is the test harness in finding bugs in reinforcement learning problems? We evaluate the adequacy and effectiveness of the test suite, employing mutation testing techniques [27, 44]. In the context of mutation testing, variants of a program, called *mutants*, are generated by applying syntactic changes, a class of fault injections. The objective of mutation testing is to evaluate the capacity of a test suite to differentiate between the output of the original program and its mutants. The results of evaluation show that in 75% of the cases the mutation score is above 90%. One of the common reasons for surviving mutants is lack of tests for extreme values which is due to the limitation of test data generators. Additionally, there are mutants, for example in the pump case, that are result of applying changes in a function in which the outcome is conditionally selected from overlapping intervals. Hence, writing tests that distinguish changes in the conditions is not feasible.

To what extent can generic problem properties be used to reduce the cost of testing reinforcement learning problems? To answer this question, we use only generic tests. The results of mutation testing using generic problem tests are shown in Figure 6.3. This figure shows the number of mutants killed and survived. As shown in the right plot Figure 6.3, the mutation rate is above 48% for the majority of cases. So, these tests can effectively identify a sizeable subset of bugs and provide developers with the advantage of not having to rewrite the tests, thereby reducing the cost of testing. The left plot Figure 6.3 illustrates the results of performing mutation testing on the SARSA and Expected SARSA algorithms. Each of the algorithms was subjected to seven tests. In the SARSA algorithm, five faults are introduced. In the Expected SARSA algorithm, six faults are introduced. It should be noted that mutants which swap division for multiplication are stillborn and are caught by the Scala type checker (Scala is used for implementing the framework) due to a type mismatch introduced by the change. Consequently, when considering the valid mutants, the tests designed for



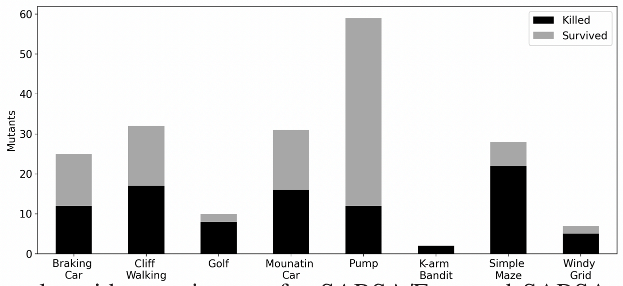


Figure 6.3: Mutation results with generic tests for SARSA/Expected SARSA algorithms (left), for the case studies (right).

temporal difference algorithms can effectively detect all bugs. These test cases can be parameterized and reused between algorithms, thereby reducing the cost of testing the RL setups.

6.4 Conclusion

We have presented a formal specification of the various components of reinforcement learning, with a particular focus on temporal difference methods. The formalization allows us to derive a test harness that can be reused across a large class of reinforcement learning applications based on Q-learning, SARSA, and other similar techniques. The back-up diagram language allows for the straightforward development of a novel RL algorithm and the execution of generic tests, which have been provided for this purpose. This enables the identification of potential errors at an early stage. At the same time, the generic tests provided in the framework for RL problems can be reused, and implementing problem-specific tests is easily possible.

7 Symbolic State Partitioning

This chapter addresses the challenges that tabular reinforcement learning algorithms encounter when the state space is continuous. One potential solution to this problem is to partition the state space into a finite number of discrete partitions. In this chapter we use symbolic execution to extract partitions reflecting the environment dynamics. The results demonstrate that symbolic partitioning enhances state space coverage with respect to environmental behavior, thereby enabling reinforcement learning to perform more effectively in the presence of sparse rewards.

7.1 Introduction

Reinforcement learning promises to automatically learn controllers for a multitude of challenging control problems. However, training from scratch in the real world is infeasible for many problems, necessitating a pre-training phase in a simulated environment. However, even for simulated environments, one needs to use approximation techniques, such as deep learning, to succeed in continuous state spaces. Deep reinforcement learning is unfortunately distinguished by a dearth of explainability and the absence of convergence guarantees. In contrast, tabular learning methods have been shown to support better explainability and facilitate the assurance of safety for policies generated by such methods.

The available methodologies for partitioning the state space are not sufficiently adaptive to the inherent structure of the state space. These methods fail to account for nonlinear dependencies between state components, despite the prevalence of such nonlinear behaviors in control systems. Furthermore, they necessitate engineering a good partitioning for each problem or generating the partitioning while training, which is a time-consuming process.

As we are considering simulated environments, a computer program, serving as the environment, is accessible for analysis and partitioning of the state space. This chapter investigates the use of a software analysis tool as a means of extracting approximate adaptive partitioning if the state space that reflect the dynamics of the problem (to answer **RQ4**). The objective is to create a partitioning of the state based on the syntactic similarity of their behavior, with the intention of achieving a semantically good partitioning. Additionally, the method should be general, offline (not during the learning process), and capable of capturing non-linear dependencies between the state components, and narrow partitions.

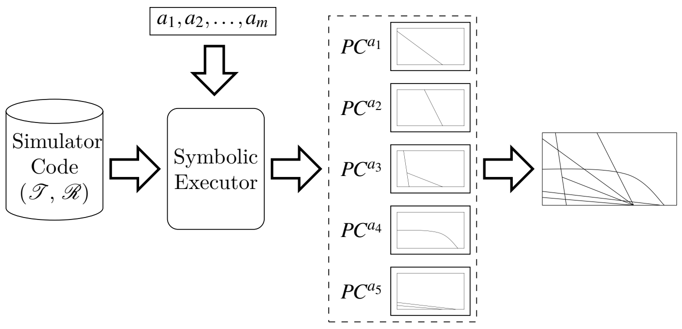


Figure 7.1: Overview of SymPar.

7.2 Method: SymPar Design

Symbolic execution is a well-established technique for program analysis, with frequent application in testing and verification. A symbolic executor generates a set of path conditions, which are sufficient pre-conditions for each execution path of the program to be taken. The path conditions partition the input space of the executed program into groups that share the same execution path. Accordingly, we develop a method, called SymPar (Figure 7.1), that employs symbolic execution to analyze the environment program, which takes the state and action as inputs, then partition the state space using the obtained path conditions.

The environment program comprises a single-step or a transition function, and a reward function of the reinforcement learning problem, which accept states and action as input. SymPar executes these two functions symbolically, using symbols to represent states and concrete values for actions. So, we generate a set of path conditions for each action. Each partitioning relation can be dually seen as an equivalence relation on states, so we have n equivalence relations, where n is the (finite) number of actions. A unique greatest (weakest) equivalence relation that is finer (stronger) than any of these equivalence relations exists, their greatest lower bound [24]. We compute this unique relation, or rather the corresponding partitioning, by taking an n-way intersection of all the partitions. Consequently, the expression will constitute a unique partition. Further details regarding the algorithm are provided in Paper IV.

7.3 Results

This section presents a theoretical analysis and a practical evaluation of the partitioning obtained by SymPar.

Theorem 7.3.1. *The obtained partitioning $\mathcal{P}$ is total for loop-free programs: $\forall \overline{s} \in \overline{\mathcal{S}}\ \exists!\, \mathcal{P}_0 \in \mathcal{P} \cdot\ \overline{s} \in \mathcal{P}_0$.*

In practice, symbolic execution is not complete, as most interesting programs with loops have infinitely many symbolic paths. Computing these infinite partitionings by enumeration is obviously infeasible. In practice, this is easily overcome, by stopping the symbolic execution early and adding one extra path condition, the complement of the union of the computed subset, to cover for the unexplored paths.

The cost of SymPar amounts to exploring all paths in the program symbolically and then computing the coarsest partitioning. The symbolic execution involves generating a number of paths exponential in the number of branch points in the program (and at each branching point one needs to solve an SMT problem—which is in principle undecidable, but works well for many practical problems). Computing the coarsest partitioning requires solving $|\mathcal{P}|^{|\mathcal{A}|}$ number of SMT problems where $|\mathcal{P}|$ is the upper bound on the number of partitions (symbolic paths) and $|\mathcal{A}|$ is the number of actions. The other operations involved in this process such as computing and storing the path conditions in the required syntax are polynomial and efficient in practice.

Theorem 7.3.2. *Let PC^a be the set of path conditions produced by SymPar for each of the actions $a \in \mathcal{A}$. The size of the final partitioning $\mathcal{P}$ returned by SymPar is bounded from below by each $|PC^a|$ and from above by $\prod_{a\in\mathcal{A}} |PC^a|$.*

The theorem follows from the fact that $\mathcal{P}$ is finer than any of the PC^as and the algorithm for computing the coarsest partitioning finer then a set of partitionings can in the worst case intersect each partition in each set PC^a with all the partitions in the partitionings of the other actions.

To evaluate SymPar empirically, we investigate the trade-off between partitioning granularity and combinatorial explosion, and conduct an empirical evaluation of the performance of our implementation in SymPar. A comparison is presented between SymPar and CAT-RL [23] an (online) partitioning method and tile coding techniques (offline) partitioning for different case studies [127]. Tile coding is a classic partitioning technique. It divides each feature of the state into rectangular tiles of equal size. While new problem-specific versions have been developed, we have elected to utilize the classic version due to its broad applicability. Finally, we compare the accumulated reward for SymPar with known algorithms DQN [92], A2C [90], PPO [116], using the Stable-Baselines3 [1] implementations [108]. We attempt to assess the performance of SymPar partitioning by answering the following research questions.

To what extent does SymPar produce smaller partitionings than other methods and how do these partitionings affect the performance of learning? To answer this question, we measure the *size of partitioning*, the *failure and success rates* and the *accumulated reward*. Tables 7.1 and 7.2 show that SymPar consistently outperforms both tile coding and CAT-RL on discrete state space in terms of success and failure rates, and reduced number of timeouts ($\mathbf{T_{out}}$) during learning. Also, the agents using SymPar partitionings obtain maximum reward more often than agents using tiling (**Opt**), cf. Table 7.1. Note that in Table 7.1, the size of partitionings is substantially biased in favour of tiling. Still, SymPar enables better learning. In Table 7.2, CAT-RL obtains smaller partitionings in the first and third cases in the same amount of time as SymPar

[1] https://github.com/DLR-RM/stable-baselines3

	SymPar					**Tile Coding**				
	$\lvert\mathcal{S}\rvert$	**Succ**	**Fail**	$\mathbf{T_{out}}$	**Opt**	$\lvert\mathcal{S}\rvert$	**Succ**	**Fail**	$\mathbf{T_{out}}$	**Opt**
	(#)	(%)	(%)	(%)	(%)	(#)	(%)	(%)	(%)	(%)
Simple Maze	33	74.9	<0.1	25.0	5.0	10^4	6.0	7.1	86.9	0.0
MA Navigation	130	5.8	82.6	11.6	0.0	10^4	0.0	99.6	0.4	0.0
Wumpus World 1	73	18.4	0.0	81.6	2.1	8^4	9.6	0.0	90.4	0.0
Wumpus World 2	52	37.3	22.9	39.8	4.2	64	19.1	33.2	47.7	0.0
Navigation	51	13.2	4.8	82.0	<0.1	64	0.0	0.0	100.0	0.0
Braking Car	81	89.1	10.9	0.0	29.8	81	82.0	18.0	0.0	14.9
Mountain Car	70	82.2	0.0	17.8	61.3	81	59.4	0.0	40.6	14.7
Random Walk	184	61.2	11.1	27.7	44.0	196	6.5	5.1	88.4	<0.1

Table 7.1: Partitioning size and learning performance. Discrete cases above bar, continuous below.

	SymPar				**CAT-RL**			
	$\lvert\mathcal{S}\rvert$	**Succ**	**Fail**	$\mathbf{T_{out}}$	$\lvert\mathcal{S}\rvert$	**Succ**	**Fail**	$\mathbf{T_{out}}$
	(#)	(%)	(%)	(%)	(#)	(%)	(%)	(%)
Mountain Car	70	82.2	0.0	17.8	16	78.7	0.0	21.3
Wumpus World 1	73	18.4	0.0	81.6	157	2.7	0.0	97.3
Wumpus World 2	52	37.3	22.9	39.8	22	14.5	30.2	55.3
Braking Car	81	89.1	10.9	0.0	127	34.0	66.0	0.0

Table 7.2: Partitioning size and learning performance with SymPar and CAT-RL online partitioning.

but the quality of learning is worse. For the other test problems, SymPar is better than CAT-RL in both the size and learning performance.

We compare the accumulated rewards for two complementary cases: (1) randomly selected states and (2) states that are less likely to be chosen by random selection. The latter are identified by SymPar's partitioning. For randomly selected states, the three top plots in Figure 7.2, show that the agents trained by SymPar obtain a better normalized cumulative reward and subsequently converge faster to a better policy than the best competing approaches (more results in Paper IV). The three bottom plots in the figure show the accumulated reward when starting from unlikely states (small partitions) for the best competing approaches (more in Paper IV) Here, we expect to observe a good policy from algorithms that capture the dynamics of environment. Interestingly, the online technique CAT-RL struggles when dealing with large sets of initial states. This can be seen in, e.g., the training for Braking Car, where each episode introduces new positions and velocities.

How does the granularity of the partitioning affect the learning performance? To answer this question, we create different learning problems with various partitioning granularity by changing the search depth for the symbolic execution. If we stop the symbolic executor closer to the entry point, we obtain shorter symbolic path and a smaller set of weaker path conditions. We then compare the maximum accumulated reward of the learned policy to gain an understanding of the learning performance

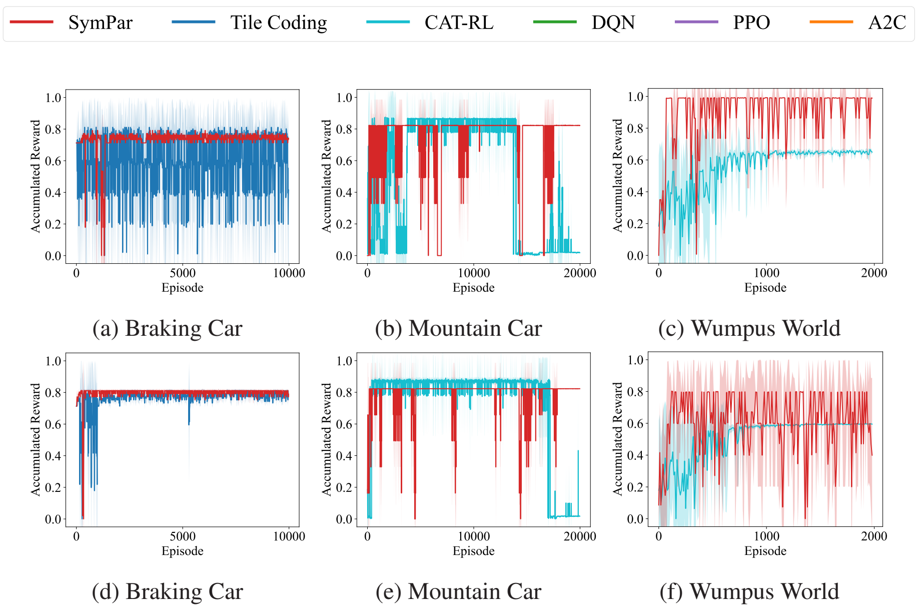
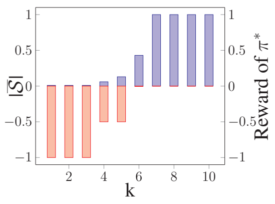


Figure 7.2: Normalized cumulative reward per episode while evaluating ten random states (Top), and less likely states (Bottom). The best approach for each case is shown; see Appendix A for the rest.

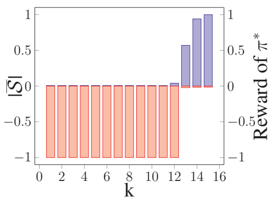
for the given abstraction. The four leftmost plots in Figure 7.3 show that a higher granularity of partitionings yields a higher accumulated reward achieved with the optimal policy.

The two rightmost plots in Figure 7.3 show the shapes of partitions obtained by SymPar for Braking Car and Simple Maze. The first plot represents different partitions with different colors. Notably, the green and purple partitions depict partitioning expressions that contain a non-linear relation between the components of the state space (position and velocity). Close to the x-axis, narrow partitions are discernible, marked in yellow and pink. To illustrate the partitions obtained for Simple Maze, the expressions are translated into a 10×10 grid (Figure 7.3f). These two visualizations shed light on the intricacies of state space partitioning and hint at the logical explainability of the partitionings obtained by SymPar.

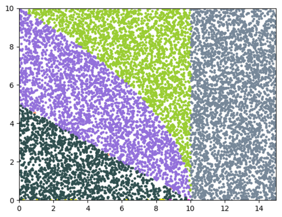
How does SymPar scale with increasing state space sizes? To address this question, we compare the number of partitions when increasing the state space of problems. Table 7.3 shows that the number of SymPar partitions is independent of the size of the state space for simple maze, wumpus world and navigation problems. However, this does not imply the universal applicability of the same partitioning across different sizes, as the conditions specified within the partitions may be size-dependent. Consequently, when analyzing environments with different sizes for a given problem, running SymPar is necessary to ensure the appropriate partitioning, even though the total number of

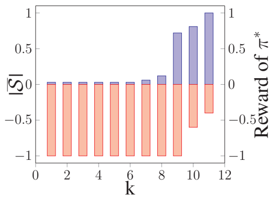


(a) Braking Car

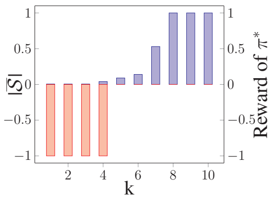


(b) MA Navigation


(c) Braking Car



(d) Simple Maze



(e) Random Walk

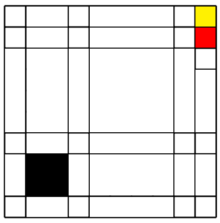

(f) Simple Maze

Figure 7.3: Normalized granularity of states and its performance for symbolic execution with search depth k (left and mid). Partitionings with SymPar for the Braking Car and Simple Maze (right).

	$\lvert\overline{\mathcal{S}}\rvert$	$\lvert\mathcal{S}\rvert$		$\lvert\overline{\mathcal{S}}\rvert$	$\lvert\mathcal{S}\rvert$		$\lvert\overline{\mathcal{S}}\rvert$	$\lvert\mathcal{S}\rvert$
Simple Maze	10×10	33	Wumpus World	64×64	73	Navigation	10×10	51
	$10^2 \times 10^2$	33		$10^2 \times 10^2$	73		$10^2 \times 10^2$	51
	$10^3 \times 10^3$	33		$10^3 \times 10^3$	73		$10^3 \times 10^3$	51

Table 7.3: Size of state space and partitioning for test problems.

partitions remains the same.

To what extent does SymPar partition the states that are behaviourally similar together? To answer this question, we select five random partitions from the partitioning obtained by SymPar, and five arbitrary concrete states from each partition. Then, we feed the concrete states as initial states to RL, and compute the accumulated reward using the optimal policy, or more precisely the policy obtained by RL for this partitionings. Table 7.4 presents the variance in accumulated rewards for concrete states across various partitions. Note that consistency in accumulated rewards among states within the same partition, indicating minimal divergence. This is particularly evident when the mean and normalized standard deviation are compared, which demonstrates that the standard deviation is considerably smaller in relation to the mean accumulated reward.

7.4 Conclusion

The principal conclusion of this study is that the pre-analysis of the environment simulator facilitates the construction of a comprehensive representation and the

	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$
Braking Car	$-0.05 \pm 0.0\%$	$-0.01 \pm 0.0\%$	$-0.5 \pm 0.0\%$	$-10.0 \pm 0.0\%$	$-10.01 \pm 0.0\%$
Mountain Car	$996.1 \pm 0.1\%$	$975.5 \pm 0.2\%$	$979.04 \pm 0.2\%$	$986.7 \pm 0.2\%$	$981.6 \pm 0.2\%$
Wumpus World	$486.8 \pm 0.3\%$	$490.0 \pm 0.2\%$	$477.6 \pm 0.2\%$	$475.0 \pm 0.0\%$	$495.8 \pm 0.1\%$

Table 7.4: Assessment of similarity of concrete states within partitions.

attainment of a quantitative coverage of the reinforcement learning state space. The contribution of this work is a method for partitioning the state space of reinforcement learning, which is:

- Generic and offline,
- Operates automatically,
- Effectively capture the semantics of the environment dynamics,
- Accommodates non-linear environmental behaviors by using adaptive partition shapes, instead of rectangular tiles,
- Improves state space coverage with respect to environmental behavior.

8 A Method for the Initialization of Reinforcement Learning

This chapter addresses the challenge of more effectively exploring the reinforcement learning environment. The initialization approach proposed is designed for the class of RL applications for which the simulation environment is available, either in a black-box or white-box manner. The proposed approach is evaluated on a group of well-known RL algorithms, including DQN, PPO, A3C, and CAT-RL.

8.1 Introduction

The RL agent's training is often constrained by limited time, resulting in exposure to only the most likely scenarios. Limited exploration narrows the agent's understanding of the full environment and can lead to several challenges such as disastrous results for safety-critical systems where a safe policy is needed for every possible state, the agent may get stuck in local optima, policies may become overly sensitive to initial and observed states, dealing with sparse rewards becomes more difficult.

Exploration is a fundamental aspect of RL [35, 67, 130]. Common approaches include ε-greedy [127], count-based exploration [11, 80, 123], curiosity-based exploration [105], or methods specifically designed for exploring sparse reward contextual MDPs [109, 155]. All of these methods begin by seeding the initial states using a uniform distribution over a subset of the state space. The issues with uniformly selecting the initial state are threefold. Firstly, the agent may spend a considerable amount of time exploring areas of the state space that are free of immediate reward. Secondly, it could result in the agent overfitting to regions it encounters frequently initially, while neglecting others that are critical in later stages of training. Thirdly, it may lead the agent to focus on suboptimal areas, thereby reducing the utility of early policies. In contrast, our objective is to enhance the exploration process by leveraging prior knowledge about the dynamics of the environments.

Are there advantages to be gained by having insights into a set of states that are likely to vary significantly? Additionally, how can expert knowledge help address states that are less likely to be sampled during the training process? We hypothesize that insights into the structure of state space will enable more efficient exploration. Specifically, initializing the RL agent in expert-provided states ensures that these states are explored, allowing the agent to learn appropriate policies for critical situations. This

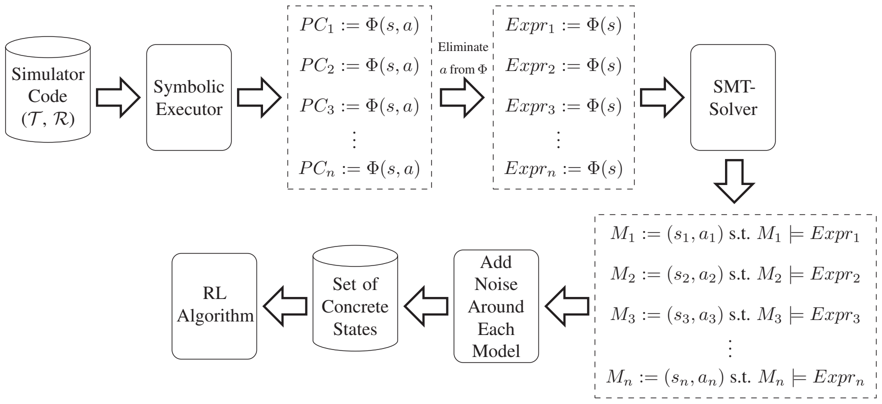

Simulator Code $(\mathcal{T}, \mathcal{R})$
Symbolic Executor
$PC_1 := \Phi(s,a)$
$PC_2 := \Phi(s,a)$
$PC_3 := \Phi(s,a)$
$PC_n := \Phi(s,a)$
Eliminate a from Φ
$Expr_1 := \Phi(s)$
$Expr_2 := \Phi(s)$
$Expr_3 := \Phi(s)$
$Expr_n := \Phi(s)$
SMT-Solver
$M_1 := (s_1, a_1)$ s.t. $M_1 \models Expr_1$
$M_2 := (s_2, a_2)$ s.t. $M_2 \models Expr_2$
$M_3 := (s_3, a_3)$ s.t. $M_3 \models Expr_3$
$M_n := (s_n, a_n)$ s.t. $M_n \models Expr_n$
Add Noise Around Each Model
Set of Concrete States
RL Algorithm

Figure 8.1: Overview of the approach.

approach can also help the agent reach a goal state more quickly, effectively distributing reward values to intermediary states, which is particularly useful in scenarios with sparse rewards.

Who can provide RL with such knowledge? To answer this question, we make an assumption that the simulation of the environment of RL is available. We can use software analysis tools that extract such knowledge. In particular, symbolic execution meets our needs. Symbolic execution, a widely used technique in program analysis, can be used to automatically generate test inputs for a program or to detect hidden problems in the implementation [8, 9, 17, 20], guaranteeing high coverage of the simulator definition.

The objective of this study is to investigate the impact of seeding reinforcement learning algorithms with values derived from symbolic execution of the environment simulation, on the exploration and convergence in the learning process (to answer **RQ4**).

8.2 Method: SymSeed Design

We present a methodology for generating a set of initial states for reinforcement learning algorithms through the analysis of environment, assumed to be a computer program, dynamics. To this end, the environment is executed symbolically using an off-the-shelf tool, and the path conditions (PCs) are extracted. Each PC is a logical expression over the input variables of the program, in this case state and action. Subsequently, we can solve each PC using an SMT solver, resulting in a concrete state and action that satisfies the given expression. Finally, we introduce noise around the obtained states with the objective of increasing the number of samples, and feed all of them as initial states for an RL algorithm Figure 8.1.

The simulator of the environment must be a program that takes the current state of the agent and its action, then computes the next state that the agent reaches and the immediate reward that obtains for the current transition. This program is known as a *step function* in the literature on reinforcement learning. In other words, this program has been designed to implement a single-step transition of MDP for the given problem, the dynamics of the environment. This is why we chose to analyze the simulator program. The Simulator does not have to be faithful and detailed. In a pre-training simulation abstract environment models can be used.

Symbolic execution of a program explores feasible execution paths of the program and generates a set of PCs, each corresponding to a specific execution path within the program. To execute a program symbolically, it is necessary to have two key elements: the program itself, or the byte code of the program, and the variables within the program that are to be treated as symbolic variables. Accordingly, the symbolic execution of a given step function captures the significant dynamics of the environment in a set of PCs. Each PC should be a logical expression over the state and action components. Subsequently, an SMT-solver can be employed to resolve each of the PCs. The specific techniques employed to solve the PCs may vary depending on the SMT-solver utilized. In any case, upon requesting the solution to an expression, the solver returns concrete values for the variables of the formula that can satisfy the entire formula. These can be used as initial states for learning.

SMT solvers typically yield the same solution for a given formula, even if multiple solutions are feasible. Moreover, requesting numerous solutions from an SMT solver is a time-consuming. Alleviate this, we request a single solution from the SMT solver and then add small noise to it. It is highly probable (heuristically) that this increases the number of samples of each PC.

The output of solving each PC using the SMT-Solver is a concrete valuation of states and actions that can satisfy the given PC. However, in reinforcement learning, actions are not initialised. Consequently, we just ignore the action components the answers, yielding a set of concrete values for the state components. After adding noise, this set (both original set solutions and small perturbed samples) is fed into RL algorithms, both tabular and deep-learning-based.

8.3 Results

We evaluate SymSeed, with well-known reinforcement learning algorithms: DQN [92], A3C [90], PPO [116], and CAT-RL [23], in a series of experiments on a set of classic case studies. The training of each agent is conducted using three distinct initialization strategies: (a) a uniform distribution over the entire state space, (b) a uniform distribution over the states generated by SymSeed, and (c) a combination of (a) and (b), controlling the percentage of mixing.

Test Problems. The **Office World 1** problem is a grid map comprising four distinct rooms at its corners. The objective is to collect and deliver mail and coffee to the designated office location, resulting in a positive reward; otherwise, no reward is given. The **Office World 2** is analogous to Office World 1, with the exception that the location of the goal is situated at the farthest distance from the start position of the agent. The **Office World 3** represents a combination of the first and second office worlds, incorporating two distinct goal states. The agent only succeeds in one of the two possible outcomes, with the other office acting as a local optimum. The **Safari**

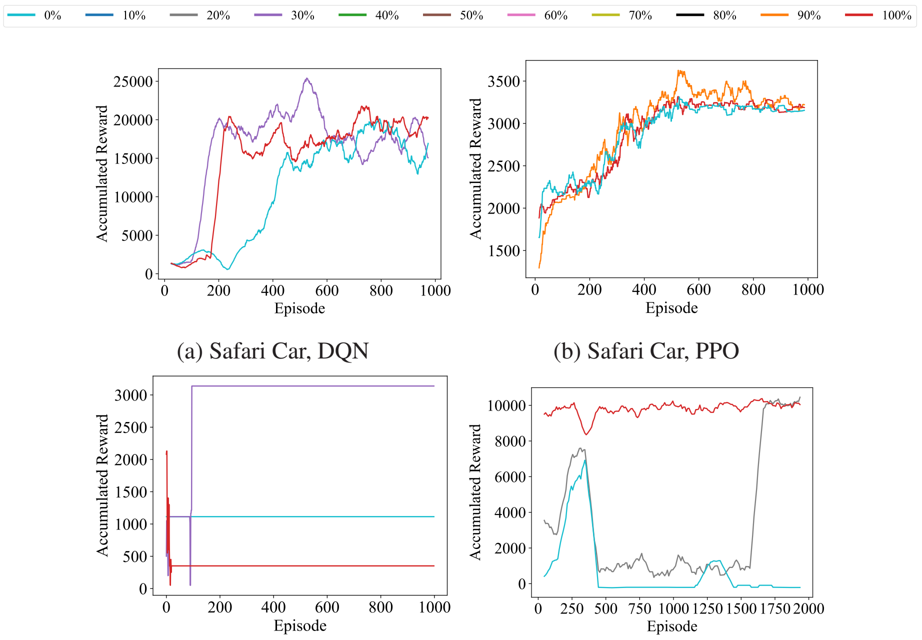
	Num New States	Max Frequency	Mean Frequency	Std
0%	159028	35662	5.64	127.03
10%	176573	29131	10.82	112.74
20%	194564	20766	11.15	80.69
30%	153059	27072	12.84	249.43
40%	178155	14351	11.93	87.79
50%	177352	21108	10.82	117.84
60%	149587	21543	4.99	118.35
70%	109722	47648	11.7	307.09
80%	160338	40829	11.72	264.9
90%	138977	45349	11.74	329.14
100%	145279	40533	9.31	267.8

Table 8.1: Visited states for Safari Car using CAT-RL, with sampling probabilities from SymSeed data ranging from 0% to 100% in 10% increments.

Car requires to learn how to obtain enough momentum to move up two steep slopes, similar dynamics to the mountain car [96], but only with two steep slopes.

To what extent SymSeed decreases the number of visited states? Does SymSeed help reaching a higher reward? To answer this question, we count the visited states during training for SymSeed and the baseline. We also measure the mean of the accumulated reward of all episodes. Figure 8.2 illustrates the best mixing strategy of initializing the RL algorithms with SymSeed data, complete initialization with SymSeed (100%), and not using it at all (0%) which serves as the baseline. A comparison of the accumulated reward achieved by each of these strategies, as illustrated in Figure 8.2, demonstrates that SymSeed has enhanced the performance of all the studied RL algorithms, resulting in higher rewards. Furthermore, it attains the higher reward more rapidly than other initializations, which are generated at random across the entire state space. For the same experiments, an analysis of the visited states is presented in Table 8.1. The results demonstrate that the initial states provided by SymSeed allow RL algorithms (in this experiment, we present the results of CAT-RL) to explore a smaller number of new states compared to the baseline (0%), while achieving a higher reward. Additionally, Figure 8.3 demonstrates the heatmap plot of visited states for each strategy while training in the safari car environment using CAT-RL. The weight of points in the plot refers to the frequency with which a given state has been visited. As the training for all strategies is 100,000 episodes, a higher number in this plot indicates a lower number of new states visited, but the same states are visited multiple times. More interestingly, this plot shows how increasing the percentage of using SymSeed effects the visited states during training.

Does SymSeed help the learning algorithms to avoid local optima? To answer this question, we have designed a series of examples, i.e. the Safari Car test case, which demonstrate the potential for local optima to impede the agent's progress if the environment is not adequately explored. The agent will be deemed to have succeeded if it is able to identify the global optimum. Subsequently, to answer this question, the success rate was measured during training for each strategy. The success rate for



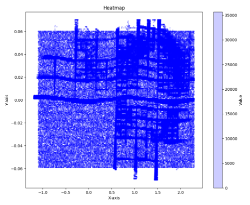
(a) Safari Car, DQN (b) Safari Car, PPO

(c) Safari Car, A3C (d) Safari Car, CAT-RL

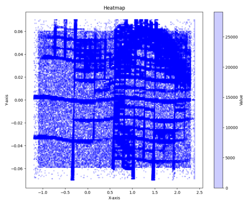
Figure 8.2: RL algorithms evaluated for ten uniformly random initial states, every 100 episodes. The baseline (0%), the full data generated by SymSeed (100%), and the best mixed method are selected.

each of the aforementioned strategies is illustrated in Figure 8.4, which illustrates that each of those algorithms exhibits a superior success rate in the presence of SymSeed-provided initial states than in their absence. However, there is no consistency in the observed outcomes when the percentage of SymSeed data is increased. In some cases, including more SymSeed data led to improved performance, while in other cases, the opposite was true. However, a statistical analysis of the results indicates that mixing in SymSeed data consistently enhances performance over the baseline.

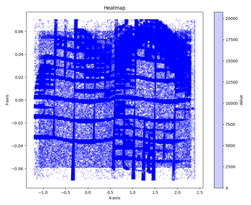
Does SymSeed improve the performance in the presence of sparse reward? To answer this question, we have designed a series of examples based on Office World, in which the agent obtains the reward only in the final states (no intermediate rewards). The goal is to show how SymSeed helps the agent to find goal states at the early episodes and to expand awareness of them to the rest of states. We measure the accumulated reward for evaluating trained policy every ten episodes for ten randomly selected states. Figure 8.6 illustrates that CAT-RL initialized with SumIn data, converges to an optimal policy faster than with other initialization. This phenomenon can be attributed to the fact that the agent may commence from states that are in close proximity to the final states (given that in many cases, the final states are conditionally defined, and SymSeed is capable of acquiring this knowledge through symbolic



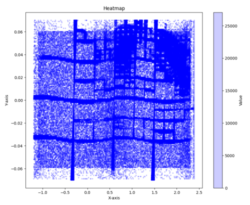
(a) visited states in 0%



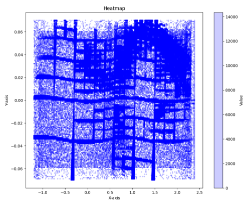
(b) visited states in 10%



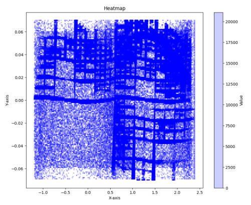
(c) visited states in 20%



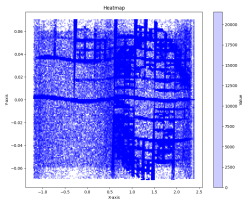
(d) visited states in 30%



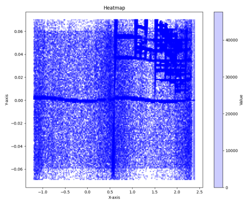
(e) visited states in 40%



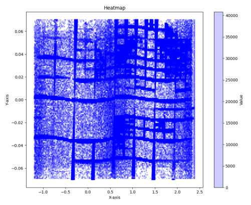
(f) visited states in 50%



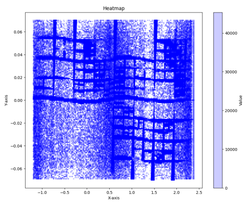
(g) visited states in 60%



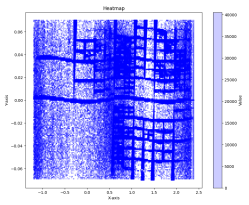
(h) visited states in 70%



(i) visited states in 80%



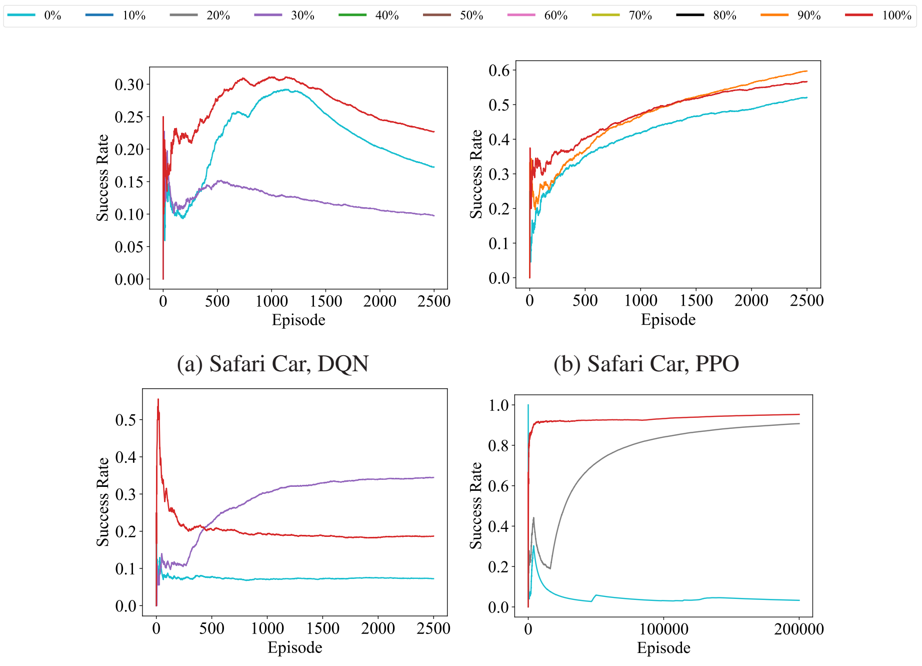
(j) visited states in 90%



(k) visited states in 100%

Figure 8.3: Visited states of Safari car during 100 K episodes training of CAT-RL, with sampling probabilities from SymSeed data ranging from 0% to 100% in 10% increments.

execution). This allows the agent to observe the final state at an early stage, thereby enabling the distribution of the reward across the neighboring states of the final state. Consequently, the impact of the reward is distributed rapidly, which is equivalent to an environment with a non-sparse reward.



(a) Safari Car, DQN (b) Safari Car, PPO

(c) Safari Car, A3C (d) Safari Car, CAT-RL

Figure 8.4: The success rate is calculated during the learning by counting success for all training episodes. The baseline (0%), the full data generated by SymSeed (100%), and the best mixed method are selected.

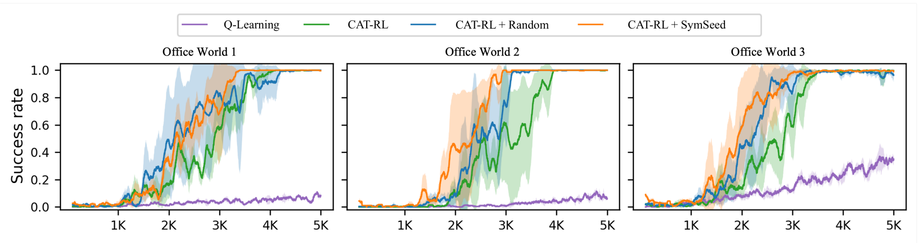


Figure 8.5: Success rate for 3 runs averaged over the last 100 training episodes of 4 methods on 3 Office World domains with 5K episodes.

8.4 Conclusion

The findings of this study indicate that reinforcement learning algorithms are sensitive to initial states. The acquisition of additional knowledge about the environment can facilitate more optimal initialization. The enhanced initialization of SymSeed facilitates more effective exploration and performance of RL algorithms. In this context, the pre-analysis of the environment simulator allows for the generation of a

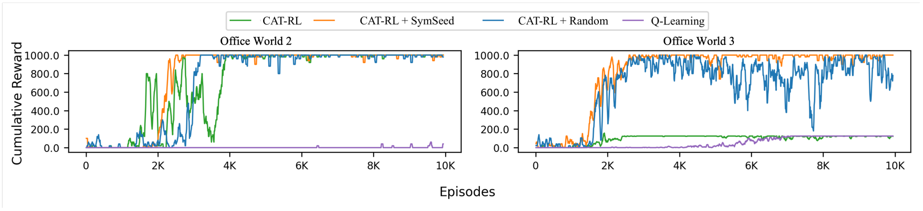


Figure 8.6: Cumulative reward for a single run of running 4 methods on Office World-modified map and Office World-variant map with 10K episodes.

set of initial states for RL algorithms that can enhance exploration and facilitate more effective response to sparse rewards.

9 Conclusion and Future Works

This chapter presents the conclusions drawn from the thesis, discusses the limitations of the work, and outlines future directions for research in this field.

9.1 Conclusion

The thesis statement of the current work is as follows (Chapter 1):

$\underbrace{\text{Formal specification and property-based testing}}_{c_1}$ $\underbrace{\text{facilitate the development}}_{c_2}$ of both $\underbrace{\text{algorithms}}_{c_3}$ and $\underbrace{\text{applications}}_{c_4}$ of RL. Furthermore, $\underbrace{\text{symbolic execution can}}_{c_5}$ $\underbrace{\text{capture the dynamics of simulation environments}}_{c_5}$, thereby $\underbrace{\text{improving the efficacy and}}_{c_6}$ $\underbrace{\text{performance}}_{c_6}$ of both $\underbrace{\text{tabular}}_{c_7}$ and $\underbrace{\text{deep RL}}_{c_8}$.

In this thesis, I first investigated two real-world problems in Chapters 4 and 5 (c_4) using reinforcement learning (c_3) to gain insight into the difficulties and challenges that a researcher would encounter in this domain. I found the development process to be exploratory, which increases the possibility of an error-prone implementation (c_2). Furthermore, I encountered significant obstacles when attempting to partition the state space of RL through techniques such as tiling. It is crucial to note that while employing PPO (c_3) for the application in Chapter 4 (c_4), I noted challenges with convergence and the explainability of the policies. Consequently, I gravitated towards the use of tabular RL (c_7) in preference over DRL. Accordingly, I undertook a comprehensive examination of the systematic development process of reinforcement learning algorithms and applications in Chapter 6 (c_1-c_4). Additionally, I have investigated the impact of analysing the simulation of reinforcement learning environments using symbolic execution (c_5) on the efficacy and performance (c_6) of both tabular (c_7) and DRL (c_8) in Chapters 7 and 8.

Now, I will break the statement into smaller statements (link to the research questions in the Chapter 1) and discuss how the material presented supports it.

Statement 1. *A PPO-based algorithm, when combined with a collision avoidance strategy, enables a group of UAVs to cooperatively explore an unknown environment and locate a group of targets in an optimal manner, even in the case of targets that* *are not in a fixed position. Furthermore, this approach reduces the likelihood of taking unsafe actions.*

As detailed in Section 4.2, a new collision avoidance strategy was devised, based on an angle of motion that deviates from the trajectory of other UAVs within the visible zone. As demonstrated in the results presented in Section 4.3, this technique can effectively minimize the number of collisions during the learning process, while implausible control decisions, as an instant 180° rotation, are avoided. We also demonstrated how to use PPO to train UAVs a policy capable of locating targets even when their positions were not fixed.

Statement 2. *An abstract imprecise model of a wastewater management system helps RL to learn a better policy than training alone.*

Chapter 5 studied the use of RL to mitigate sewage overflows in a combined sewer systems during storm events. We trained a controller for sewer systems in the Hvidovre municipality of Copenhagen using a new approach called MPC Q-learning, leveraging an imprecise abstract model. As shown in Section 5.3, the controller obtained with MPC Q-learning outperforms other controllers, including those synthesized with Q-learning against the abstract model, classic Q-learning without a model, and a baseline rule-based controller. To assess the efficacy of the policies generated by each real-time control system in a more realistic setting, we developed a configuration of the SWMM model that accurately represents the actual infrastructure system. The key contributions of this study are as follows.

- RL-based control strategies can more effectively mitigate combined sewer overflows in both the pumps and gates of combined sewer systems compared to the rule-based controller used in real systems,
- Among different RL-based controllers, MPC Q-learning using the abstract model proved to be the most suitable controller for adapting to unknown environmental conditions.

Statement 3. *The formal specification of RL components helps developers to avoid errors in the implementation and to identify any such errors if they occur.*

To overcome the exploratory nature of the development process for RL applications and to make it more systematic, Section 6.2 provided a formal specification of the key elements of RL problems and algorithms. This specification places a particular emphasis on temporal difference methods and their representation in backup diagrams. Additionally, a test harness was developed that is reusable across a wide range of RL applications based on temporal difference learning, SARSA, Q-learning, and other similar techniques.

Statement 4. *Testing RL setups should not be limited to the optimality of policies, but also ascertain the correctness of the implementation of both RL algorithms and applications.*

The results of mutation testing experiments on applications studied in our framework (Section 6.3) show that in 75% of the cases the mutation score is above 90%.

This proofs the effectiveness of the test harness in finding bugs in RL problem definitions. Additionally, the results of mutation testing using generic problem tests show for the majority of cases the mutation rate is above 48%. These suggests that the tests can effectively identify about half of the bugs without the cost of any problem-specific tests.

Statement 5. *A pre-analysis of the environment simulator enables the construction of an efficient representation of the reinforcement learning state space.*

The approach presented in Section 7.2 analyzes the dynamics of simulation-based environments using a symbolic executor, thereby generating a partitioning of the state space. This representation enables the more effective use of the tabular RL algorithms, which are more comprehensible than DRL. This can be useful if the obtained policy needs to be analyzed, for instance, for functional correctness.

Statement 6. *A pre-analysis of the state space of the environment simulator can help the exploration of both deep and tabular reinforcement learning algorithms.*

Chapter 8 developed an approach to generate diverse states of the agent based on analysis of the environment simulation. These states can then be used as initial states for RL algorithms. The results presented in Section 8.3 demonstrate that this initialization method enhances the performance of both tabular RL and deep RL algorithms, facilitating more efficient exploration of the state space. The results of the experiment show an average improvement of 27% in the success rate in comparison to the baseline, while reducing the number of partitions for the majority of the examined cases by over 45%.

9.2 Discussion

Target Search for UAVs Using DRL. In our UAV target search model, we assume a rectangular search area to generalize the environment, although the solution is less efficient when applied to other shapes. Furthermore, we assume that targets remain stationary, which may not always be the case, as evidenced by studies on missing persons cases [93]. However, this assumption is more likely to hold true in scenarios such as natural disasters, where civilians are often unable to move freely [13]. Furthermore, we assume constant communication between agents, which simplifies the problem and reflects technological advances [121]. Finally, we discretize the action space for clarity, although the core argument remains valid even with a continuous action space.

Wastewater Management Using RL. The results demonstrated that all RL-based controllers, including MPC Q-learning with the UPPAAL model, Q-learning with the UPPAAL model, and classic Q-learning, consistently outperformed static controllers. The RL-based controller that integrates weather forecasts demonstrated an effective mitigation of overflow risks, particularly during periods of heavy rainfall. Among these, MPC Q-learning exhibited the most optimal performance, underscoring its superior adaptability in wastewater management systems, which are highly sensitive to rain conditions. The findings illustrate that RL, when employed with an online methodology, produces enhanced outcomes in dynamic contexts. The selection of the cost function is of great consequence with respect to the shaping of the learning process, and the performance of the learning algorithm may vary based on the priorities embedded in the cost structure. The efficacy of the RL-based controller is contingent upon the availability of real-time sensor data from the urban water system, including information regarding water levels and flow rates. In the event of sensor failure or the provision of inaccurate data, the controller's decision-making capabilities may be adversely affected.

Testing Reinforcement Learning. In this work, we focused on testing on-policy TD learning, and exploited an equivalence of updates to also test Q-learning, which is an off-policy algorithm. In general, specifying off-policy learning requires a refinement of our semantics of the backup diagram language to support two policies simultaneously. There are no technical obstacles to this, other than maintaining the simplicity of the representation. It is also interesting to generalize to different value function representations, to allow approximate learning with neural networks. This appears within reach for methods similar to Q-learning, like deep Q-learning [89], but harder for popular newer policy iteration methods like PPO [115]. Additionally, statistical tests are inherently flaky, and finding a balance between flakiness and effectiveness is challenging. We adjusted test thresholds through trial and error, but more systematic methods are needed.

Symbolic State Partitioning. It should be noted that SymPar is only capable of handling environments that have been implemented as programs. Furthermore, it will perform poorly in environments with minimal branching, as observed in the Cart Pole problem [127]. The simulation of Cart Pole only branches on final states. The path conditions identified through symbolic execution are of limited utility in this context. Moreover, SymPar employs a satisfiability check step that is computationally expensive and may result in combinatorial explosion for large systems. Finally, SymPar employs a range of off-the-shelf tools, including a symbolic executor and an SMT solver. Each of these tools has specific limitations, and unfortunately, SymPar has inherited these shortcomings.

Symbolic State Seeding. Similar to SymPar, SymSeed also requires the environment be a computer program and it inherits the limitations of symbolic execution and SMT-Solver. Furthermore, we introduce noise around the single solution of each path condition's expression. However, the question of how large the distance of noise from the solution should be remains unanswered in this work.

9.3 Future Directions of Research

This section addresses the limitations of the studies in Chapters 4 to 8 and outlines a clear path for future studies in each domain.

Search in a Concrete Environment With Dynamic Targets. We assumed a simplified search area (Chapter 4), modeled as rectangle, which can be an approximation of any area. This may result in reduced efficacy for different shapes due to searching outside the defined search area. Our other assumption was having stationary targets, which is the case for instance in natural disasters where civilians are trapped or injured [13], but is incorrect for other cases such as missing people [93]. Furthermore, research could be conducted to investigate the impact of simplifying the shape of the area and develop a model for finding dynamic targets. In particular, the state space should be enriched with features that can be used to assess the probability of finding targets in different parts of the map.

Continuous Action Space With Limited Communication. In Chapter 4, the action space is discretized to simplify understanding of the trained behavior, without affecting the main argument. While a continuous action space might require more training time, the general conclusions should hold. Additionally, Chapter 4 assumes that UAVs are equipped with real-time communication, which may not apply in scenarios with limited communication. Handling limited communication requires either partial observability, or introducing explicit communication actions.

MPC Q-Learning for Other Domains. Using an abstract model during the training phase of RL appears to be a viable approach across a range of domains. We applied this method to wastewater management (Chapter 5). However, it can also be applied to energy management in smart grids, where adapting to fluctuating demand is essential; autonomous navigation, where dynamic environments require adaptable planning; traffic signal control, to optimize flow in real time; and financial portfolio management, where continuous learning from market trends improves decision-making.

MPC Combined With DRL. Chapter 5 presented an MPC Q-learning method. Our method showed good performance compared to the baseline controllers. However, given the recent successes of DRL across many domains, it can be combined with MPC to enhance performance in scenarios that involve high-dimensional state spaces, non-linear dynamics, or uncertain environments.

Testing Off-Policy Algorithms. We focused on testing on-policy TD learning, and exploited an equivalence of updates to test also Q-learning, which is an off-policy algorithm. In general, specifying off-policy learning requires a refinement of *bdl* semantics to support two policies simultaneously. There are no technical obstacles to it, besides maintaining the simplicity of exposition. One needs to distinguish the policy π used for estimation in the final update from the one that is used to select the next action for execution. Presently, the same policy is used for both.

Extending Symsim to Test DRL Algorithms. At this time, our testbed (Chapter 6) is only capable of evaluating tabular reinforcement learning algorithms. A rational future development is to extend our testbed to handle popular deep learning algorithms in the future.

Extending SymPar to Analyze More Than One Step. SymPar introduces a partitioning by analyzing one step of the RL environment. However, it is recognized that this approach may not fully capture the intricacies of the environment. In some cases, the underlying dynamics may not be entirely obscured by a multitude of conditions. Consequently, it is postulated that a more comprehensive analysis encompassing multiple steps may yield a more nuanced understanding and a finer partitioning. Ultimately this could lead to new method symbolic reinforcement learning.

Handling Larger Action Sets While Using SymPar. The brute-force step of the SymPar algorithm Chapter 7 is particularly time-consuming when the number of actions available in the action set is high. To address this challenge, a viable approach would be to randomly select only a few actions for partitioning. Nevertheless, further investigation is required to assess the potential efficiency of this approach.

Partitioning More Complex Environments Using SymPar. Symbolic execution may not terminate in the presence of loops or the number of path conditions of the program may explode. This problem can be controlled by limiting the depth of search or employing other heuristics as already shown (Chapter 7). Still, it may be beneficial to investigate more complex heuristics to facilitate the partitioning process.

Using the Symbolic Execution Outcome for Shielding. Path conditions, which can be obtained from symbolic execution, typically contain valuable insights into the environment (Chapters 7 and 8). Additionally, they encompass information about the critical states that an agent may encounter. Consequently, it is interesting to investigate the potential of utilizing this knowledge to develop shields that encourage an agent to select safer actions during training and subsequently in test mode.

Symbolic Execution to Automate Generating Reward Machines. Using reward machines for high-level task specification in reinforcement learning has been demonstrated to be advantageous [50]. Reward machines can expose structure in the reward function and, in so doing, can speed up learning as demonstrated in their experiments. However, the construction of a reward machine remains a topic that is not widely discussed within this community. One potential approach is the construction of the machine based on the insights derived from analysis of the reward function using a symbolic executor.

References

[1] Abel, D. et al. “Policy and value transfer in lifelong reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 20–29.

[2] Abolfathi, E. A. et al. “CoachNet: An Adversarial Sampling Approach for Reinforcement Learning”. In: *NeurIPS2019 Workshop on Safety and Robustness in Decision Making*. 2021.

[3] Adelt, J. et al. “Towards Safe and Resilient Hybrid Systems in the Presence of Learning and Uncertainty”. In: *Proc. 11th Intl. Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles (ISoLA 2022)*. Vol. 13701. Lecture Notes in Computer Science. Springer, 2022, pp. 299–319.

[4] Akrour, R. et al. “Regularizing reinforcement learning with state abstraction”. In: *Proc. Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 534–539.

[5] Alamdari, N. et al. “Evaluating the Impact of Climate Change on Water Quality and Quantity in an Urban Watershed Using an Ensemble Approach”. In: *Estuaries and Coasts* vol. 43 (2020), pp. 56–72.

[6] Albus, J. S. *Brains, behavior, and robotics*. BYTE Books, 1981.

[7] Alur, R. et al. “A Framework for Transforming Specifications in Reinforcement Learning”. In: *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*. Ed. by Raskin, J.-F. et al. Cham: Springer Nature Switzerland, 2022, pp. 604–624.

[8] Baldoni, R. et al. “A survey of symbolic execution techniques”. In: *ACM Computing Surveys (CSUR)* vol. 51, no. 3 (2018), pp. 1–39.

[9] BALL, T., DANIEL, J., and Ball, T. *Deconstructing Dynamic Symbolic Execution*. Tech. rep. MSR-TR-2015-95. Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence, Boston, MA. Jan. 2015.

[10] Barreto, A. et al. “Fast reinforcement learning with generalized policy updates”. In: *Proceedings of the National Academy of Sciences* vol. 117, no. 48 (2020), pp. 30079–30087.

[11] Bellemare, M. et al. “Unifying count-based exploration and intrinsic motivation”. In: *Advances in neural information processing systems* vol. 29 (2016).

[12] Boer, F. S. de and Bonsangue, M. M. “Symbolic execution formally explained”. In: *Formal Aspects Comput.* vol. 33, no. 4-5 (2021), pp. 617–636.

[13] Bortolin, M. “55 - Urban Search and Rescue”. In: *Ciottone’s Disaster Medicine (Third Edition)*. Ed. by Ciottone, G. Third Edition. New Delhi: Elsevier, 2024, pp. 359–363.

[14] Bowes, B. D. et al. “Flood Mitigation in Coastal Urban Catchments Using Real-Time Stormwater Infrastructure Control and Reinforcement Learning”. In: *Journal of Hydroinformatics* vol. 23 (Oct. 2020), pp. 529–547.

[15] Bowes, B. D. et al. “Reinforcement Learning-Based Real-Time Control of Coastal Urban Stormwater Systems to Mitigate Flooding and Improve Water Quality”. In: *Environmental Science: Water Research & Technology* vol. 8 (2022), pp. 2065–2086.

[16] Bowes, B. D. et al. “Reinforcement learning-based real-time control of coastal urban stormwater systems to mitigate flooding and improve water quality”. In: *Environmental Science: Water Research & Technology*, no. 10 (2022), pp. 2065–2086.

[17] Cadar, C., Dunbar, D., and Engler, D. R. “KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs”. In: *Proc. 8th Symposium on Operating Systems Design and Implementation (OSDI 2008)*. USENIX Association, 2008, pp. 209–224.

[18] Cai, Y., Yang, S. X., and Xu, X. “A combined hierarchical reinforcement learning based approach for multi-robot cooperative target searching in complex unknown environments”. In: *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. 2013, pp. 52–59.

[19] Chakraborty, J., Majumder, S., and Menzies, T. “Bias in Machine Learning Software: Why? How? What to Do?” In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2021. Athens, Greece: ACM Press, 2021, pp. 429–440.

[20] Chen, Y.-F. et al. “PyCT: A Python Concolic Tester”. In: *Programming Languages and Systems: 19th Asian Symposium, APLAS 2021, Chicago, IL, USA, October 17–18, 2021, Proceedings 19*. Springer. 2021, pp. 38–46.

[21] Claessen, K. and Hughes, J. “QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs”. In: *5th ACM SIGPLAN International Conference on Functional Programming (ICFP’00)*. ACM Press, 2000, pp. 268–279.

[22] Cooper, D. C. *Fundamentals of search and rescue*. Jones & Bartlett Learning, 2005.

[23] Dadvar, M., Nayyar, R. K., and Srivastava, S. “Conditional abstraction trees for sample-efficient reinforcement learning”. en. In: *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*. ISSN: 2640-3498. PMLR, July 2023, pp. 485–495.

[24] Davey, B. A. and Priestley, H. A. *Introduction to lattices and order*. Cambridge: Cambridge University Press, 1990.

[25] David, A. et al. “Uppaal Stratego”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science. 2015, pp. 206–211.

[26] Déletang, G. et al. “Causal Analysis of Agent Behavior for AI Safety”. In: arXiv, 2021.

[27] DeMillo, R. A., Lipton, R. J., and Sayward, F. G. “Hints on test data selection: Help for the practicing programmer”. In: *Computer* vol. 11, no. 4 (1978), pp. 34–41.

[28] Ding, Z. et al. “Introduction to Reinforcement Learning”. In: *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Ed. by Dong, H., Ding, Z., and Zhang, S. Singapore: Springer Singapore, 2020, pp. 49–50.

[29] Dorigo, M., Caro, G. D., and Gambardella, L. M. “Ant Algorithms for Discrete Optimization”. In: *Artificial Life* vol. 5, no. 2 (1999), pp. 137–172.

[30] Dutta, S. et al. “TERA: Optimizing Stochastic Regression Tests in Machine Learning Projects”. In: *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2021. Virtual, Denmark: ACM Press, 2021, pp. 413–426.

[31] El Ghazouli, K. et al. “Model Predictive Control Based on Artificial Intelligence and EPA-SWMM Model to Reduce CSOs Impacts in Sewer Systems”. In: *Water Science and Technology* vol. 85 (Nov. 2021), pp. 398–408.

[32] Ellis, J. and Marsalek, J. “Overview of urban drainage: environmental impacts and concerns, means of mitigation and implementation policies”. In: *Journal of Hydraulic Research* vol. 34 (1996), pp. 723–732.

[33] Ferns, N., Panangaden, P., and Precup, D. “Bisimulation metrics for continuous Markov decision processes”. In: *SIAM Journal on Computing* vol. 40, no. 6 (2011), pp. 1662–1714.

[34] Ferns, N., Panangaden, P., and Precup, D. “Metrics for Finite Markov Decision Processes.” In: *UAI*. Vol. 4. 2004, pp. 162–169.

[35] Fruit, R. and Lazaric, A. “Exploration-exploitation in mdps with options”. In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 576–584.

[36] Fu, G. et al. “The Role of Deep Learning in Urban Water Management: A Critical Review”. In: *Water Research* vol. 223 (Sept. 2022), p. 118973.

[37] Fulton, N. and Platzer, A. “Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning”. In: *Proc. 32nd Conf. on Artificial Intelligence (AAAI-18)*. AAAI Press, 2018, pp. 6485–6492.

[38] Gandhi, D., Pinto, L., and Gupta, A. “Learning to fly by crashing”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3948–3955.

[39] García, L. et al. “Modeling and real-time control of urban drainage systems: A review”. In: *Advances in Water Resources* vol. 85 (2015), pp. 120–132.

[40] Ghaffari, M. and Afsharchi, M. “Learning to shift load under uncertain production in the smart grid”. In: *International Transactions on Electrical Energy Systems* vol. 31, no. 2 (2021), e12748.

[41] Ghosh, D. et al. “Divide-and-conquer reinforcement learning”. In: *arXiv preprint arXiv:1711.09874* (2017).

[42] Goldstein, H. et al. “Property-Based Testing in Practice”. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 2024, pp. 1–13.

[43] Guan, Y. et al. “Direct and indirect reinforcement learning”. In: *International Journal of Intelligent Systems* vol. 36, no. 8 (2021), pp. 4439–4467.

[44] Hamlet, R. G. “Testing Programs with the Aid of a Compiler”. In: *IEEE Transactions on Software Engineering* vol. 3, no. 4 (1977), pp. 279–290.

[45] Hierons, R. M. et al. “Using formal specifications to support testing”. In: *ACM Computing Surveys (CSUR)* vol. 41, no. 2 (2009), pp. 1–76.

[46] Hou, Y. et al. “UAV Swarm Cooperative Target Search: A Multi-Agent Reinforcement Learning Approach”. In: *IEEE Transactions on Intelligent Vehicles* vol. 9, no. 1 (2024), pp. 568–578.

[47] Huang, S. et al. *Adversarial Attacks on Neural Network Policies*. 2017.

[48] Huber, W. C., Rossman, L. A., and Dickinson, R. E. “EPA storm water management model, SWMM5”. In: *Watershed models* vol. 338 (2005), p. 359.

[49] Hughes, J. “QuickCheck testing for fun and profit”. In: *International Symposium on Practical Aspects of Declarative Languages*. Springer. 2007, pp. 1–32.

[50] Icarte, R. T. et al. “Using reward machines for high-level task specification and decomposition in reinforcement learning”. In: *Intl. Conf. on Machine Learning*. PMLR. 2018, pp. 2107–2116.

[51] Ivanov, R. et al. “Case study: verifying the safety of an autonomous racing car with a neural network controller”. In: *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. 2020, pp. 1–7.

[52] Jaeger, M. et al. “Teaching Stratego to Play Ball: Optimal Synthesis for Continuous Space MDPs”. In: *Automated Technology for Verification and Analysis*. Ed. by Chen, Y.-F., Cheng, C.-H., and Esparza, J. Cham: Springer International Publishing, 2019, pp. 81–97.

[53] Jaeger, M. et al. “Teaching Stratego to Play Ball: Optimal Synthesis for Continuous Space MDPs”. In: *Proc. 17th Intl. Symposium on Automated Technology for Verification and Analysis (ATVA 2019)*. Vol. 11781. Lecture Notes in Computer Science. Springer, 2019, pp. 81–97.

[54] Jang, S. and Kim, H.-I. “Entropy-aware model initialization for effective exploration in deep reinforcement learning”. In: *Sensors* vol. 22, no. 15 (2022), p. 5845.

[55] Jansson, A. D. “Discretization and Representation of a Complex Environment for On-Policy Reinforcement Learning for Obstacle Avoidance for Simulated Autonomous Mobile Agents”. In: *Proc. 7th Intl. Congress on Information and Communication Technology*. Vol. 464. Lecture Notes in Networks and Systems. Springer, 2023, pp. 461–476.

[56] Jevti, A. et al. “Building a swarm of robotic bees”. In: Oct. 2010, pp. 1–6.

[57] Jin, P. et al. “Trainify: A CEGAR-Driven Training and Verification Framework for Safe Deep Reinforcement Learning”. In: *Proc. 34th Intl. Conf. on Computer Aided Verification (CAV 2022)*. Vol. 13371. Lecture Notes in Computer Science. Springer, 2022, pp. 193–218.

[58] Jin, Y. et al. “Efficient Multi-agent Cooperative Navigation in Unknown Environments with Interlaced Deep Reinforcement Learning”. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 2897–2901.

[59] Jothimurugan, K., Alur, R., and Bastani, O. “A Composable Specification Language for Reinforcement Learning Tasks”. In: *Advances in Neural Information Processing Systems*. Ed. by Wallach, H. et al. Vol. 32. Curran Associates, Inc., 2019.

[60] Jothimurugan, K., Bastani, O., and Alur, R. “Abstract Value Iteration for Hierarchical Reinforcement Learning”. In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by Banerjee, A. and Fukumizu, K. Vol. 130. Proceedings of Machine Learning Research. PMLR, 2021, pp. 1162–1170.

[61] Jothimurugan, K. et al. “Compositional Reinforcement Learning from Logical Specifications”. In: *Advances in Neural Information Processing Systems*. Ed. by Ranzato, M. et al. Vol. 34. Curran Associates, Inc., 2021, pp. 10026–10039.

[62] Jothimurugan, K. et al. “Specification-Guided Learning ofăNash Equilibria withăHigh Social Welfare”. In: *Computer Aided Verification*. Ed. by Shoham, S. and Vizel, Y. Cham: Springer International Publishing, 2022, pp. 343–363.

[63] Kerkez, B. et al. *Smarter stormwater systems*. 2016.

[64] Kim, J. et al. “Automating reinforcement learning with example-based resets”. In: *IEEE Robotics and Automation Letters* vol. 7, no. 3 (2022), pp. 6606–6613.

[65] King, J. C. “Symbolic execution and program testing”. In: *Communications of the ACM* vol. 19, no. 7 (1976), pp. 385–394.

[66] Kober, J., Bagnell, J. A., and Peters, J. “Reinforcement learning in robotics: A survey”. In: *The Intl. Journal of Robotics Research* vol. 32, no. 11 (2013), pp. 1238–1274.

[67] Kolter, J. Z. and Ng, A. Y. “Near-Bayesian exploration in polynomial time”. In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 513–520.

[68] Kraemer, L. and Banerjee, B. “Reinforcement learning of informed initial policies for decentralized planning”. In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* vol. 9, no. 4 (2014), pp. 1–32.

[69] Kruschke, J. *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press, 2014.

[70] Lanzi, P. L. et al. “Classifier prediction based on tile coding”. In: *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2006)*. ACM, 2006, pp. 1497–1504.

[71] Lee, I. S. and Lau, H. Y. “Adaptive state space partitioning for reinforcement learning”. In: *Engineering applications of artificial intelligence* vol. 17, no. 6 (2004), pp. 577–588.

[72] Lin, P. et al. “Rule-based object-oriented water resource system simulation model for water allocation”. In: *Water Resources Management* vol. 34 (2020), pp. 3183–3197.

[73] Lin, Y.-C. et al. “Tactics of Adversarial Attack on Deep Reinforcement Learning Agents”. In: *Proc. 26th International Joint Conference on Artificial Intelligence*. IJCAI’17. AAAI Press, 2017, pp. 3756–3762.

[74] Liu, J. et al. “Learning Contract Invariants Using Reinforcement Learning”. In: *Proc. 37th IEEE/ACM International Conference on Automated Software Engineering, (ASE 2022)*. ACM Press, 2022, 63:1–63:11.

[75] Lu, Y., Sun, W., and Sun, M. “Mutation Testing of Reinforcement Learning Systems”. In: *Proc. 7th International Symposium on Dependable Software Engineering: Theories, Tools, and Applications (SETTA 2021)*. Ed. by Qin, S., Woodcock, J., and Zhang, W. Vol. 13071. Lecture Notes in Computer Science. Springer, 2021, pp. 143–160.

[76] Lund, N. S. V. et al. “CSO reduction by integrated model predictive control of stormwater inflows: a simulated proof of concept using linear surrogate models”. In: *Water Resources Research* vol. 56, no. 8 (2020), e2019WR026272.

[77] Lund, N. S. V. et al. “Model predictive control of urban drainage systems: A review and perspective towards smart real-time water management”. In: *Critical Reviews in Environmental Science and Technology* vol. 48 (2018), pp. 279–339.

[78] Luo, X. et al. “Machine Learning-Based Surrogate Model Assisting Stochastic Model Predictive Control of Urban Drainage Systems”. In: *Journal of Environmental Management* vol. 346 (Nov. 2023), p. 118974.

[79] Lygouras, E. et al. “Unsupervised human detection with an embedded vision system on a fully autonomous UAV for search and rescue operations”. In: *Sensors* vol. 19, no. 16 (2019), p. 3542.

[80] Machado, M. C., Bellemare, M. G., and Bowling, M. “Count-based exploration with the successor representation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 5125–5133.

[81] MacIver, D. R. and Donaldson, A. F. “Test-case reduction via test-case generation: Insights from the hypothesis reducer (tool insights paper)”. In: *34th European Conference on Object-Oriented Programming (ECOOP 2020)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik. 2020.

[82] Madumal, P. et al. “Explainable Reinforcement Learning through a Causal Lens”. In: *Proc. 34th Conf. on Artificial Intelligence (AAAI 2020)*. AAAI Press, 2020, pp. 2493–2500.

[83] Marsalek, J. et al. “Urban Drainage Systems: Design and Operation”. In: *Water Science and Technology* vol. 27 (1993), pp. 31–70.

[84] Marsalek, J. “Evolution of Urban Drainage: From Cloaca Maxima to Environmental Sustainability”. In: *Acqua e Citta, I Convegno Nazionale di Idraulica Urbana*. 2005, pp. 1–22.

[85] Mavridis, C. N. and Baras, J. S. “Vector quantization for adaptive state aggregation in reinforcement learning”. In: *2021 American Control Conf. (ACC)*. IEEE. 2021, pp. 2187–2192.

[86] McDonnell, B. E. et al. “PySWMM: The Python Interface to Stormwater Management Model (SWMM)”. In: *Journal of Open Source Software* vol. 5 (2020), p. 2292.

[87] Messikommer, N., Song, Y., and Scaramuzza, D. “Contrastive initial state buffer for reinforcement learning”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 2866–2872.

[88] Michie, D. and Chambers, R. A. “BOXES: An experiment in adaptive control”. In: *Machine intelligence* vol. 2, no. 2 (1968), pp. 137–152.

[89] Mnih, V. and Kavukcuoglu, K. *Methods and Apparatus for Reinforcement Learning*. US Patent #20150100530A1. Apr. 2015.

[90] Mnih, V. et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *Proc. 33nd Intl. Conf. on Machine Learning (ICML 2016)*. Vol. 48. JMLR Workshop and Conf. Proceedings. JMLR.org, 2016, pp. 1928–1937.

[91] Mnih, V. et al. “Human-Level Control through Deep Reinforcement Learning”. In: *Nature* vol. 518 (2015), pp. 529–533.

[92] Mnih, V. et al. “Playing Atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[93] Mohibullah, W. and Julie, S. “Developing An Agent Model of a Missing Person in the Wilderness”. In: Oct. 2013, pp. 4462–4469.

[94] Montgomery, W. et al. “Reset-free guided policy search: Efficient deep reinforcement learning with stochastic initial states”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 3373–3380.

[95] Moore, A. W. “Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces”. In: *Machine Learning Proceedings 1991*. Elsevier, 1991, pp. 333–337.

[96] Moore, A. W. “Efficient memory-based learning for robot control”. PhD thesis. University of Cambridge, UK, 1990.

[97] Mullapudi, A. et al. “Deep Reinforcement Learning for the Real Time Control of Stormwater Systems”. In: *Advances in Water Resources* vol. 140 (June 2020), p. 103600.

[98] Negm, A., Ma, X., and Aggidis, G. “Deep Reinforcement Learning Challenges and Opportunities for Urban Water Systems”. In: *Water Research* vol. 253 (Apr. 2024), p. 121145.

[99] Nicol, S. and Chadès, I. “Which states matter? An application of an intelligent discretization method to solve a continuous POMDP in conservation biology”. In: *PloS one* vol. 7, no. 2 (2012), e28993.

[100] Oikarinen, T. et al. “Robust Deep Reinforcement Learning through Adversarial Loss”. In: *Advances in Neural Information Processing Systems*. Ed. by Ranzato, M. et al. Vol. 34. Curran Associates, 2021, pp. 26156–26167.

[101] Oroojlooy, A. and Hajinezhad, D. “A review of cooperative multi-agent deep reinforcement learning”. In: *Applied Intelligence* vol. 53, no. 11 (2023), pp. 13677–13722.

[102] Pang, Q., Yuan, Y., and Wang, S. “MDPFuzz: Testing Models Solving Markov Decision Processes”. In: *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2022. ACM Press, 2022, pp. 378–390.

[103] Pasareanu, C. S. et al. “Symbolic PathFinder: integrating symbolic execution with model checking for Java bytecode analysis”. In: *Autom. Softw. Eng.* vol. 20, no. 3 (2013), pp. 391–425.

[104] Passino, K. “Biomimicry of bacterial foraging for distributed optimization and control”. In: *IEEE Control Systems Magazine* vol. 22, no. 3 (2002), pp. 52–67.

[105] Pathak, D. et al. “Curiosity-driven exploration by self-supervised prediction”. In: *International conference on machine learning*. PMLR. 2017, pp. 2778–2787.

[106] Puiutta, E. and Veith, E. M. S. P. “Explainable Reinforcement Learning: A Survey”. In: *Proc. 4th Intl. Cross-Domain Conf. (CD-MAKE 2020)*. Vol. 12279. Lecture Notes in Computer Science. Springer, 2020, pp. 77–95.

[107] Qin, X. et al. “Multi-Agent Cooperative Target Search Based on Reinforcement Learning”. In: *Journal of Physics: Conference Series* vol. 1549, no. 2 (2020), p. 022104.

[108] Raffin, A. et al. *Stable baselines3*. https://stable-baselines3.readthedocs.io/. 2019.

[109] Raileanu, R. and Rocktäschel, T. “Ride: Rewarding impact-driven exploration for procedurally-generated environments”. In: *arXiv preprint arXiv:2002.12292* (2020).

[110] Riccio, V. et al. “DeepMetis: Augmenting a Deep Learning Test Set to Increase its Mutation Score”. In: *36th IEEE/ACM International Conference on Automated Software Engineering (ASE 2021)*. 2021, pp. 355–367.

[111] Romdhana, A. et al. “IFRIT: Focused Testing through Deep Reinforcement Learning”. In: *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 2022, pp. 24–34.

[112] Ruderman, A. et al. “Uncovering Surprising Behaviors in Reinforcement Learning via Worst-case Analysis”. In: *Safe Machine Learning workshop at ICLR 2019*. 2019.

[113] Saddiqi, M. M. et al. “Smart Management of Combined Sewer Overflows: From an Ancient Technology to Artificial Intelligence”. In: *WIREs Water* vol. 10 (2023), e1635.

[114] Saliba, S. M. et al. “Deep Reinforcement Learning with Uncertain Data for Real-Time Stormwater System Control and Flood Mitigation”. In: *Water* vol. 12 (Nov. 2020), p. 3222.

[115] Schulman, J. et al. “Proximal Policy Optimization Algorithms”. In: *ArXiv* vol. abs/1707.06347 (2017).

[116] Schulman, J. et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[117] Schulman, J. et al. “Trust region policy optimization”. In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[118] Seipp, J. and Helmert, M. “Counterexample-guided Cartesian abstraction refinement for classical planning”. In: *Journal of Artificial Intelligence Research* vol. 62 (2018), pp. 535–577.

[119] Semadeni-Davies, A. et al. “The impacts of climate change and urbanisation on drainage in Helsingborg, Sweden: Suburban stormwater”. In: *Journal of Hydrology* vol. 350, no. 1 (2008), pp. 114–125.

[120] Senanayake, M. et al. “Search and tracking algorithms for swarms of robots: A survey”. In: *Robotics and Autonomous Systems* vol. 75 (2016), pp. 422–434.

[121] Sharma, A. et al. “Communication and networking technologies for UAVs: A survey”. In: *Journal of Network and Computer Applications* vol. 168 (2020), p. 102739.

[122] Sharma, A. et al. “Autonomous reinforcement learning via subgoal curricula”. In: *Advances in Neural Information Processing Systems* vol. 34 (2021), pp. 18474–18486.

[123] Strehl, A. L. and Littman, M. L. “An analysis of model-based interval estimation for Markov decision processes”. In: *Journal of Computer and System Sciences* vol. 74, no. 8 (2008), pp. 1309–1331.

[124] Su, J. et al. “Effectively Generating Vulnerable Transaction Sequences in Smart Contracts with Reinforcement Learning-guided Fuzzing”. In: *37th IEEE/ACM International Conference on Automated Software Engineering (ASE 2022)*. to appear. IEEE Press, 2022.

[125] Sun, Y. et al. “Concolic testing for deep neural networks”. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 2018, pp. 109–119.

[126] Sutton, R. S. “Learning to Predict by the Methods of Temporal Differences”. In: *Mach. Learn.* vol. 3, no. 1 (1988), pp. 9–44.

[127] Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. 2nd. The MIT Press, 2018.

[128] Szita, I. “Reinforcement Learning in Games”. In: *Reinforcement Learning*. Vol. 12. Adaptation, Learning, and Optimization. Springer, 2012, pp. 539–577.

[129] Tappler, M. et al. “Search-Based Testing of Reinforcement Learning”. In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*. Ed. by Raedt, L. D. International Joint Conferences on Artificial Intelligence Organization, July 2022, pp. 503–510.

[130] Tarbouriech, J. et al. “No-regret exploration in goal-oriented reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9428–9437.

[131] Tian, W. et al. “Combined Sewer Overflow and Flooding Mitigation Through a Reliable Real-Time Control Based on Multi-Reinforcement Learning and Model Predictive Control”. In: *Water Resources Research* vol. 58 (2022), e2021WR030703.

[132] Tran, H.-D. et al. “Safety verification of cyber-physical systems with reinforcement learning control”. In: *ACM Transactions on Embedded Computing Systems (TECS)* vol. 18, no. 5s (2019), pp. 1–22.

[133] Tufano, R. et al. “Using Reinforcement Learning for Load Testing of Video Games”. In: *Proc. IEEE/ACM 44th International Conference on Software Engineering (ICSE 2022)*. ACM Press, 2022.

[134] Türker, U. C. et al. “Efficient state synchronisation in model-based testing through reinforcement learning”. In: *Proc. 36th IEEE/ACM International Conference on Automated Software Engineering (ASE 2021)*. 2021, pp. 368–380.

[135] Uchendu, I. et al. “Jump-start reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 34556–34583.

[136] Uther, W. T. B. and Veloso, M. M. “Tree Based Discretization for Continuous State Space Reinforcement Learning”. In: *Proc. 15th National Conf. on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conf. (AAAI 98, IAAI 98)*. AAAI Press / The MIT Press, 1998, pp. 769–774.

[137] Vardhan, H. and Sztipanovits, J. “Rare Event Failure Test Case Generation in Learning-Enabled-Controllers”. In: *2021 6th International Conference on Machine Learning Technologies*. ICMLT 2021. Jeju Island, Republic of Korea: ACM Press, 2021, pp. 34–40.

[138] Verdier, C. F. et al. “Near Optimal Control With Reachability and Safety Guarantees”. In: *IFAC-PapersOnLine* vol. 52, no. 11 (2019), pp. 230–235.

[139] Voogd, E. et al. “Symbolic Semantics for Probabilistic Programs”. In: *Proc. 20th Intl. Conf. on Quantitative Evaluation of Systems (QEST 2023)*. Vol. 14287. Lecture Notes in Computer Science. Springer, 2023, pp. 329–345.

[140] Vyetrenko, S. and Xu, S. “Risk-Sensitive Compact Decision Trees for Autonomous Execution in Presence of Simulated Market Response”. In: *arXiv preprint arXiv:1906.02312* (2019).

[141] Wang, C. et al. “Reinforcement Learning for Flooding Mitigation in Complex Stormwater Systems during Large Storms”. In: *IEEE EUROCON 2021 - 19th International Conference on Smart Technologies*. July 2021, pp. 274–279.

[142] Wang, P. et al. “Decentralized navigation with heterogeneous federated reinforcement learning for UAV-enabled mobile edge computing”. In: *IEEE Transactions on Mobile Computing* (2024).

[143] Wang, X. and Fang, X. “A multi-agent reinforcement learning algorithm with the action preference selection strategy for massive target cooperative search mission planning”. In: *Expert Systems with Applications* vol. 231 (2023), p. 120643.

[144] Watkins, C. J. C. H. “Learning from delayed rewards”. In: (1989).

[145] Wei, H., Corder, K., and Decker, K. “Q-learning acceleration via state-space partitioning”. In: *Proc. 17th Intl. Conf. on Machine Learning and Applications (ICMLA 2018)*. IEEE. 2018, pp. 293–298.

[146] Wenger, S. J. et al. “Twenty-six key research questions in urban stream ecology: an assessment of the state of the science”. In: *Journal of the North American Benthological Society* vol. 28 (2009), pp. 1080–1098.

[147] Whiteson, S. *Adaptive Representations for Reinforcement Learning*. Vol. 291. Studies in Computational Intelligence. Springer, 2010.

[148] Wu, M. et al. “A game-based approximate verification of deep neural networks with provable guarantees”. In: *Theor. Comput. Sci.* vol. 807 (2020), pp. 298–329.

[149] Wu, Y., Low, K. H., and Lv, C. “Cooperative Path Planning for Heterogeneous Unmanned Vehicles in a Search-and-Track Mission Aiming at an Underwater Target”. In: *IEEE Transactions on Vehicular Technology* vol. 69, no. 6 (2020), pp. 6782–6787.

[150] Xia, J. et al. “Cooperative multi-target hunting by unmanned surface vehicles based on multi-agent reinforcement learning”. In: *Defence Technology* vol. 29 (2023), pp. 80–94.

[151] Yin, Z. et al. “Forecasting and Optimization for Minimizing Combined Sewer Overflows Using Machine Learning Frameworks and Its Inversion Techniques”. In: *Journal of Hydrology* vol. 628 (Jan. 2024), p. 130515.

[152] Yu, C. et al. “Reinforcement learning in healthcare: A survey”. In: *ACM Computing Surveys (CSUR)* vol. 55, no. 1 (2021), pp. 1–36.

[153] Yue, L. et al. “Deep Reinforcement Learning for UAV Intelligent Mission Planning”. In: *Complexity* vol. 2022 (Mar. 2022), p. 3551508.

[154] Zelvelder, A. E., Westberg, M., and Främling, K. “Assessing Explainability in Reinforcement Learning”. In: *Proc. Third Intl. Workshop on Explainable and Transparent AI and Multi-Agent Systems (EXTRAAMAS 2021)*. Vol. 12688. Lecture Notes in Computer Science. Springer, 2021, pp. 223–240.

[155] Zha, D. et al. “Rank the episodes: A simple approach for exploration in procedurally-generated environments”. In: *arXiv preprint arXiv:2101.08152* (2021).

[156] Zhang, S. et al. “FIGCPS: Effective Failure-inducing Input Generation for Cyber-Physical Systems with Deep Reinforcement Learning”. In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2021, pp. 555–567.

[157] Zhang, Z., Tian, W., and Liao, Z. “Towards Coordinated and Robust Real-Time Control: A Decentralized Approach for Combined Sewer Overflow and Urban Flooding Reduction Based on Multi-Agent Reinforcement Learning”. In: *Water Research* vol. 229 (Feb. 2023), p. 119498.

[158] Zheng, Y. et al. “Automatic Web Testing Using Curiosity-Driven Reinforcement Learning”. In: *Proc. IEEE/ACM 43rd International Conference on Software Engineering (ICSE 2021)*. ACM Press, 2021, pp. 423–435.

[159] Zolfagharian, A. et al. “Search-Based Testing Approach for Deep Reinforcement Learning Agents”. In: arXiv, 2022.

Appendices

APPENDIX A
Collection of papers

Multi-Agent Reinforcement Learning for Search-and-Rescue with Cooperative Rotation Maneuver

Danyal Yorulmaz, Tobias Gad Spoorendonk, Mohsen Ghaffari*, Andrzej Wąsowski

Computer Science Departement, IT University of Copenhagen, Denmark

Abstract

Cooperative target search is an area of research with applications in both military and civilian fields. One such application is search-and-rescue by unmanned aerial vehicles. We extend the prior results of Wang & Fang (2023) on using reinforcement learning for this problem with several new contributions. First, we propose a new collision avoidance strategy based on a cooperative rotation maneuver for an aerial vehicle with a limited range of motion, more realistic than the originally proposed. Second, we evaluate the use of reinforcement learning, and in particular the proximal policy optimization for this problem, under uncertainty of location of the targets and obstacles. The results of our experiments demonstrate that the cooperative rotation maneuver significantly reduces the frequency of collisions while maintaining the effectiveness of the search process.

Keywords: Reinforcement Learning, Multi-agent Systems, Search-and-Rescue

1. Introduction

Reinforcement learning is an experience-based learning method that is well suited for optimization problems in a complex and unknown environment (Sutton & Barto, 2014). It gained popularity for learning controllers in many domains, including robotics manipulation, navigation, electronic systems, and in search-and-rescue (Tang et al., 2024; Wang et al., 2024; Ghaffari & Afsharchi, 2021; Kulkarni et al., 2020). This study addresses the problem of target search by Unmanned Aerial Vehicles (UAVs). The objective is to identify the optimal search strategy in terms of the route length, risk, or efficiency (Yue et al., 2022). In particular, we concentrate on cooperative search-and-rescue, where emergency responders need to locate and assist civilians in distress. Given the unpredictable nature of the

**Corresponding Author:* Mohsen Ghaffari, (+45) 71559083, mohg@itu.dk
Email addresses: dayo@itu.dk (Danyal Yorulmaz), tosp@itu.dk (Tobias Gad Spoorendonk), mohg@itu.dk (Mohsen Ghaffari), wasowski@itu.dk (Andrzej Wąsowski)

Differences	**Our model**	**Wang & Fang (2023)**
Obstacles in the arena	Yes	No
Realistic collision avoidance	Yes	No
Learning Algorithm	PPO	Action preference selection
Neural network	Multi-Layer Perceptron	Recurrent neural network

Table 1: Differences between the models described here, and by Wang & Fang (2023). Realistic collision avoidance means that instantaneous arbitrary rotation is not required.

well-being in such situations, it is imperative that the responders find them as fast as possible. For this reason one would like to use multiple UAVs (Senanayake et al., 2016). This, however, means that the UAVs must coordinate with each other to avoid collisions and avoid duplication of work (Wu et al., 2020; Cai et al., 2013).

Deep reinforcement learning is used for the cooperative target search to tame the inherently large state spaces (Wang & Fang, 2023; Jin et al., 2019; Qin et al., 2020; Yue et al., 2022). Gandhi et al. (2017) use the deep deterministic policy gradients to yield results in continuous time and action spaces. The use of neural networks and deep learning was explored by Lygouras et al. (2019). They guarantee safe take-off, navigation, and landing of the UAV on a fixed target. However, the common collision avoidance methods are either infeasible for realistic UAVs (Wang & Fang, 2023), or require a lot of failure data to learn the collision-free path (Gandhi et al., 2017). Additionally, these authors assume that the targets are at fixed positions at the training and rescue time. In reality, the locations of the targets are unknown during policy execution. This is a search problem after all!

In this paper, we contribute (i) a new collision avoidance strategy for multiple UAVs with a realistically limited cost and range of motion. Furthermore, we evaluate the effectiveness of a *proximal policy optimization* (Schulman et al., 2017) for a search-and-rescue problems with this collision avoidance strategy, and under (ii) varying degrees of complexity and uncertainty about the targets' location.

Related Work. Cooperative target search is a classic problem that has received a lot of attention in recent years (Senanayake et al., 2016; Wang & Fang, 2023). Search operations are inherently complex, due to their unpredictable nature (Cooper, 2005). Many works use biologically inspired meta-heuristics (Dorigo et al., 1999; Passino, 2002; Jevtić et al., 2010). As multiple agents can collaborate effectively, *multi-agent reinforcement learning* (MARL) has the potential to improve the execution of search-and-rescue operations (Yue et al., 2022; Wang & Fang, 2023; Cai et al., 2013; Hou et al., 2024). In MARL, agents can adjust their search strategy in response to new information from the environment obtained by any other agent (Oroojlooy & Hajinezhad, 2023; Ding et al., 2020). The target search requires use of technologies such as communication, trajectory optimization, obstacle avoidance, and cooperative control (Wang & Fang, 2023). It is important to respect the physical limitations of the agents, which we address in this paper, by making the collision avoidance metric realistic (cf. Tbl. 1).

Proximal policy optimization (PPO) shows good potential for the search-and-rescue problem (Yue et al., 2022; Cai et al., 2013; Xia et al., 2023; Wang et al., 2024). Yue et al. (2022) use PPO to learn a strategy to suppress the enemy's aerial defense. Cai et al. (2013) address the problem of cooperative navigation, modeling the navigation policy as a combination of dynamic target selection and collision avoidance. Xia et al. (2023) focus on multi-target search by unmanned surface vehicles using a distributed partially observable multi-target hunting PPO algorithm. However, the collision avoidance issue has not been addressed adequately in the above works. To address the issue, Gandhi et al. (2017) penalize the agent for not following a collision-free path to the target. This unfortunately requires that the agents collide many times before they learn to avoid collisions. Wang & Fang (2023) propose to directly force the agents to move in the opposite direction when in vicinity of another agent. This requires that UAVs are able to rotate 180° instantly, which is not realistic for many types of drones, especially underactuated drones that are used in long-distance operations. For this reason, Wang & Fang (2023) limit the angle of rotation that the agent can perform in a single time-step. Moreover, the above papers assume that the target locations are fixed during training and execution (Lygouras et al., 2019; Wang & Fang, 2023). To address these shortcomings, we present a new collision avoidance method and train policies under uncertainty of target location and a varying complexity of the environment.

2. Background

In reinforcement learning an agent interacts with an environment by taking actions and observing responses in order to learn a behavioral policy π (Sutton & Barto, 2014). The agent operates in state space $\mathcal{S}$ with dynamics governed by a Markov decision process. Once an action $a_t \in \mathcal{A}$ is taken in state $s_t \in \mathcal{S}$, the agent moves to some state s_{t+1} and receives a reward $\mathcal{R}(s_t, a_t, s_{t+1}) \in \mathbb{R}$. This process is repeated until the objective is reached. An optimal policy π maps states to actions in a way maximizing the expected long-term expected cumulative reward.

Proximal Policy Optimization (PPO) is a type of policy gradient algorithm, meaning it optimizes its policy function (Schulman et al., 2017). This optimization is done by updating the parameters of its policy function. The agent interacts with the environment and gathers trajectories consisting of states, actions, rewards, and next states. These collected trajectories are used to compute the advantage function and other statistics (Schulman et al., 2017). It then does mini-batch updates over multiple epochs, which includes updating the policy and value functions using stochastic gradient ascent on the surrogate objective (Schulman et al., 2017). We use an actor-critic style PPO, where the actor represents the policy function and the critic represents the value function (Schulman et al., 2017).

3. Method

The environment is defined as a rectangular arena $W_B \times H_B$ and two sets, targets K and obstacles O. An agent $i \in I$ has state (x_i, y_i, θ_i), its position in a 2D-plane

Symbol	Definition
ζ	*target detection range*: any target within this range of an agent is seen as found
η	*safety distance*: the distance below which agents are considered to have collided
c	*collision avoidance distance*: collision avoidance activates if agents are this close
v	*velocity*: the speed at which the agents move
$\Delta\theta_{\max}$	*maximum yaw angle*: the limit of heading angle change in a time unit

Table 2: Agent definition parameters

and a rotation angle. Agents move with constant velocity v and cannot leave the arena. Table 2 summarizes the key constants controlling the behavior. A target $k \in K$ is defined as a location of $(x_k, y_k) \in ([0, W_B], [0, H_B])$. A target is detected if the distance between the target and the ith agent d_{ik} is below ζ: $d_{ik} \leq \zeta$. A detected target is no longer relevant in the search process. Each obstacle $o \in O$ is a rectangle with a center (x_o, y_o) and dimensions $W_o \times H_o$. Targets and obstacles are fixed in their position and can not move. In centralized cooperative MARL the global state is the knowledge of all agents: $\mathcal{S} = ((x_1, y_1, \theta_1), \ldots, (x_{|I|}, y_{|I|}, \theta_{|I|}))$.

Similarly, the action space is the product of actions of all agents $\mathcal{A} = \prod_{i \in I} A^i$ where $A^i = \{-\frac{3}{4}, -\frac{2}{4}, -\frac{1}{4}, 0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}\}$ are the possible moves of the ith agent. Each action $a \in \mathcal{A}$ can be performed in any state and represents the pose rotation of the ith agent in radians. For velocity v and action $a_t \in \mathcal{A}$ the transition function is:

$$\mathcal{T}(s_t, a_t) = \prod_{i \in I} T((x_t^i, y_t^i, \theta_t^i), a_t^i) \quad \text{where} \tag{1}$$

$$T(x_t^i, y_t^i, \theta_t^i, a_t^i) := \begin{cases} x_{t+1}^i = x_t^i + v\cos(\theta_t^i + a_t^i) \\ y_{t+1}^i = y_t^i + v\sin(\theta_t^i + a_t^i) \\ \theta_{t+1}^i = \theta_t^i + a_t^i \end{cases} \tag{2}$$

The reward function $\mathcal{R} = R_p + R_s$ combines the search reward and the time penalty. The time penalty $R_p = -t/T$ is given to agents for each step of the operation that does not terminate, where t is the current time step and T is the maximum duration of an episode. The search reward $R_s = r_o + r_1 + r_2 + r_b + r_m + r_c$ is split into six different values: odor effect, finding a target, finding all targets, boundary penalty, movement cost, and collision penalty. The odor effect rewards proximity to the target, see Fig. 1. It guides the agents towards the targets. The ranges of the odor effect have to be carefully defined, as too large ranges easily compound leading to sub-optimal strategies. A too small range has negative effect on the training time (Wang & Fang, 2023). For the distance d_{ik} and detection range ζ define the odor effect as (Wang & Fang, 2023):

$$r_o = \begin{cases} 0 & d_{ik} > \zeta + \zeta/2.3 \\ 1 & \zeta + \zeta/3.5 < d_{ik} < \zeta + \zeta/2.3 \\ 2 & \zeta + \zeta/7 < d_{ik} < \zeta + \zeta/3.5 \\ 3 & \zeta < d_{ik} < \zeta + \zeta/7 \end{cases} \tag{3}$$

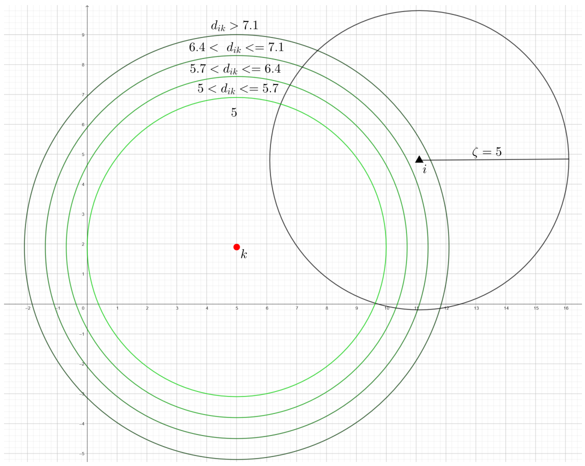


Figure 1: Visual representation of the odor effect around target k. The target detection range ζ for agent i is set to 5 in this example

Finding a target gives a reward $r_1 = 5n_I$ where n_I is the number of yet-undetected targets. Finding all targets gives a reward $r_2 = 100$, hitting the bounds of the arena or an obstacle gives a reward $r_b = -3$, moving gives a negative reward of $r_m = -n_I$, a collision between agents gives a reward based on the angle and distance between the colliding agents. If d_{ij} denotes the distance between agent i and agent j, $\bar{\mathbf{I}}$ is the standard direction vector, usually $(0, 1)$, α is the distance adjustment weight and β the direction adjustment weight, then the collision penalty is as follows (Wang & Fang, 2023). The parameters α and β, set to 1 and 0.1 respectively, are used to dictate the impact of the distance and direction, respectively.

$$r_c = -\alpha|d_{ij}| - \frac{\beta}{\pi}\arccos\left(\frac{\langle \mathbf{I_i}, \bar{\mathbf{I}} \rangle}{\|\mathbf{I_i}\| \cdot \|\bar{\mathbf{I}}\|}\right) \tag{4}$$

We consider three different types of collisions:

1. *Collision between agents.* The colliding agents are allowed to continue their previous movement uninterrupted after a penalty.
2. *Collision with boundaries.* The position of a colliding agent is updated to the nearest point within the bounds of the arena.
3. *Collision with obstacles.* The agent's position is updated to the nearest point on the boundary of the obstacle.

Collision avoidance can be handled in many ways. One option is to generate a repelling force between agents (Wang & Fang, 2023). This can work well, and mostly ensures that collisions are avoided. However, this assumes that the agent can move in any direction, at any time. This is not feasible for underactuated vehicles, which are limited in changes to the heading angle. For this reason, we propose a new collision avoidance mechanism that does not require full actuation.

The proposed collision avoidance mechanism, the *Cooperative Rotating Maneuver* (CRM), operates as follows. Let $i \in I$ be an agent, and let E be the set of agents that are closer than c to i, so i shall attempt to avoid collisions with them. Then define a set $V = \{[x_i - x_e, y_i - y_e]^\intercal \mid e \in E\}$ and $\vec{v}_f = \sum_{\vec{v} \in V} \hat{v}$, where $\hat{v}$ is $\vec{v}$ normalized. Vector $\vec{v}_f$ is the desired new heading vector for agent i. The desired heading angle φ^θ is then $\operatorname{atan2}(\vec{v}_f)$. Finally, the new heading angle is calculated by, ensuring that any change in the heading angle θ^i is limited to $\Delta\theta_{\max}$.

$$\theta^i_{t+1} = \begin{cases} (\theta^i_t - \Delta\theta_{\max}) & \varphi^\theta_t \leq (\theta^i_t - \Delta\theta_{\max}) \\ (\theta^i_t + \Delta\theta_{\max}) & \varphi^\theta_t \geq (\theta^i_t + \Delta\theta_{\max}) \\ \varphi^\theta_t & (\theta^i_t - \Delta\theta_{\max}) < \varphi^\theta_t < (\theta^i_t + \Delta\theta_{\max}) \end{cases} \tag{5}$$

4. Evaluation

We design experiments to answer the following research questions:

RQ1 To what extent do the different target and obstacle uncertainty modes affect the average performance of PPO in cooperative target search with CRM?

RQ2 To what extent do numbers of targets and obstacles affect the average performance of PPO in cooperative target search with CRM?

RQ3 How well does CRM perform for a varying number of agents?

To answer **RQ1** we assess the impact of uncertainty on the rate of completion of search. We collect data about average reward and time-to-completion in four scenarios: **a)** obstacles and targets in fixed positions during training and evaluation, **b)** obstacles placed randomly in each episode but fixed targets, **c)** fixed obstacles but randomly placed targets in each episode, and **d)** obstacles and targets randomly placed. To answer **RQ2**, we collect the mean number of steps-to-complete the mission for increasing number of obstacles and targets. To answer **RQ3**, we quantify the number of collisions between agents while varying the number of obstacles, agents, and collision avoidance distance between experiments. We use the mechanism proposed by Wang & Fang (2023) and no mechanism as baselines.

Table 3 collects the problem configuration for the experiments. We use PPO, for training the agents, as implemented in the CleanRL library (Huang et al., 2022). We use a fully connected Multi-Layer Perceptron with two hidden layers of size 64 with hyperbolic tangent activation functions in between. The neural network using Pytorch (Vu et al., 2022). The MARL environment is defined using PettingZoo (Terry et al., 2021) and SuperSuit (Terry et al., 2020). We run the experiments on Intel(R) Core(TM) i7-10510U CPU @ GHz 2.30 GHz, 16.0 GB RAM.

	Definition	**Value**		**Definition**	**Value**
W_B	width of search area	20	n_L	number of obstacles	2
H_B	height of search area	20	M_K	target spawn mode	1
W_O	width of obstacles	4	M_L	obstacle spawn mode	0
H_O	height of obstacles	4	T	max. steps in an episode	100
ζ	target detection Range	5	*timesteps*	#steps/experiment	10^7
c	collision avoidance distance	4	*horizon*	horizon limit	200
η	safety distance	1	$S_{minibatch}$	size of each mini-batch	800
v	velocity	1	*epoch*	number of epochs	4
$\Delta\theta_{\max}$	max. Δ yaw angle	$\frac{\pi}{4}$	γ	discount factor	0.99
n_I	number of agents	2	ϵ	clipping surrogate coeff.	0.1
n_K	number of targets	3	λ	GAE lambda param.	0.95

Table 3: Parameter values used in experiments

At the beginning of each episode, the agents, targets, and obstacles are placed. The agents spawn in the bottom region of the search area, spread evenly across the x-axis. To answer **RQ2**, we allow targets and obstacles to spawn in different ways, depending on the designated spawn mode: when $M_K = 0$, all targets will spawn deterministically in the same location every episode, when $M_K = 1$, all targets will spawn randomly every episode. Respectively, $M_L = 0$ and $M_L = 1$ spawn obstacles deterministically and randomly. The parameters *horizon*, $S_{minibatch}$, *epoch*, ϵ, and λ are all PPO-related parameters (Schulman et al., 2017), which will not be altered during the experiments, except $S_{minibatch}$ which, for implementation reasons: $S_{minibatch} = 800$ for even n_I, while $S_{minibatch} = 750$ for odd n_I. It has been shown that $S_{minibatch}$ does not have a large impact on training accuracy for MLP networks (Keskar et al., 2017).

RQ1: Uncertainty of targets and obstacles. Table 4 (right) shows the impact of uncertainty in target and obstacle spawn mode on the average number of steps required to complete a mission, for four possible combinations of $M_k = 0, 1$ and $M_L = 0, 1$. The time required to complete a mission when the targets remain constant is considerably shorter than when they are randomly generated. This is to be expected, given that the change from constant to random target spawn alters the nature of the problem from path finding to area coverage. A path finding problem can be optimized to a high degree, if the target location is known. In contrast, an area coverage problem cannot be optimized to the same extent, since the worst case scenario is to search the entire area.

Random obstacle placement with fixed targets is, only 0.4 steps slower than fixed placement on average. This is because the probability of an obstacle randomly appearing on the optimal fixed path between an agent and a target is low. In the event that an obstacle does spawn on the agent's path, it is not problematic for the agent, as it can simply increase the cost and fly higher past the obstacle at the cost of some negative reward. This is seen in Tbl. 4 (left), where the average

Reward		Obstacles const.	Obstacles rand.
Targets	const.	0.5	0.46
Targets	rand.	0.4	0.26

Time		Obstacles const.	Obstacles rand.
Targets	const.	15	16
Targets	rand.	27	32

Table 4: Mean reward per episode with different values of M_K and M_L (Left), Mean steps to completion with different values of M_K and M_L (Right).

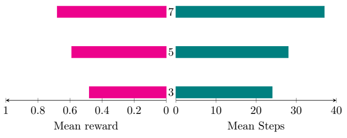
return is affected significantly more by the random obstacles with constant targets than the average steps to completion would suggest.

If the targets are randomly placed, the impact of obstacle placement is more significant: the difference between fixed and random obstacle spawn is five steps on average. The projected path of the agent is much longer for the case of an area-coverage problem. Consequently, it is more probable that an obstacle will spawn on the optimal path. Moreover, the absence of accurate directional information results in the frequent occurrence of collisions with the same obstacle. This is also demonstrated by the episodic return, as seen in Tbl. 4 (left).

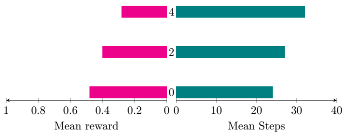
RQ2: Effectiveness analysis of complexity. Figure 2a shows a significant correlation between an increase in the number of targets and the time required to locate all targets. This is an inherent consequence of the revised objective, which now includes the additional goal of locating a greater number of targets. As illustrated in Fig. 2a, an increase in the number of targets is accompanied by an increase in the mean reward. This is consistent with expectations, as locating a target results in a positive reward, and the average number of additional steps required per new target is less than the reward for locating a target. The relatively minor increase in the number of steps required per new target can be attributed to the fact that the search area is of a comparatively limited.

Figure 2b shows the negative correlation between the number of obstacles and the average steps to completion. This is a logical consequence of the fact that obstacles impede the optimal routes, compelling the agents to traverse alternative paths to achieve coverage. Furthermore, Fig. 3a illustrates the correlation between collisions and the number of obstacles. Given that the episodic return incorporates both search time and collision frequency into account, it is unsurprising that Fig. 2b demonstrates a negative correlation between the number of obstacles and average episodic reward.

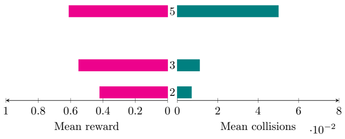
RQ3: Effectiveness of improved collision avoidance mechanism. In Tbl. 5 and Fig. 4 we measure the average collision frequencies, and rewards for both collision avoidance mechanisms, while varying the number of agents and collision avoidance distance. As you can see in Tbl. 5, the collision frequency of our collision avoidance mechanism is slightly higher than the collision avoidance



(a) An analysis of the impact of n_K on search reward and time

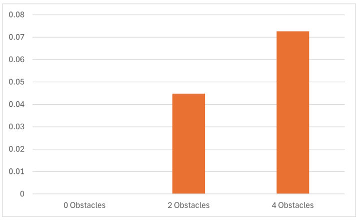


(b) An analysis of the impact of n_L on search reward and steps to completion

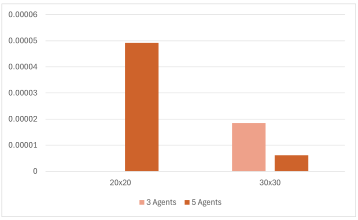


(c) An analysis of the amount of collisions and mean reward with no collision avoidance and different n_I

Figure 2: Analyses of the n_K, n_L, and collision distance.



(a) Mean collisions per step with different values of n_L



(b) Mean collisions per step with different values $W_B \times H_B$ and n_I

Figure 3: Analysis of collisions with varying number of obstacles and size of the search area.

mechanism presented by (Wang & Fang, 2023). This is to be expected, since their collision avoidance creates a repelling force between agents, forcing them away, while our collision avoidance merely instructs the agent how to move to best avoid collisions. This is a cost of operating under a constraint of the agent being not fully actuated (the baseline is not feasible for all drones). Furthermore, Fig. 4 shows that there is a negative correlation between the collision avoidance distance and the average reward for our collision avoidance. It is for this reason that this study defines $c = 4$ as the default, which seems to be around the point where the optimal balance between frequency of collisions and average rewards is found given c, and $W_B \times H_B$. With this being said, our collision avoidance mechanism does, on average, slightly outperform the collision avoidance mechanism proposed by Wang & Fang (2023)

Looking at Tbl. 5, a question arises. Why is the frequency of collisions noticeably lower when $n_I = 3$ than when $n_I = 2$ or $n_I = 5$? The reason for this is directly

	2 Ag., 3 Av.	2 Ag., 4 Av.	2 Ag., 5 Av.	3 Ag., 3 Av.	3 Ag., 4 Av.	3 Ag., 5 Av.	5 Ag., 3 Av.	5 Ag., 4 Av.	5 Ag., 5 Av.
CRM	6.2	0.0	0.0	0.9	0.0	0.0	3.3	0.5	0.1
Wang & Fang (2023)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 5: Mean collisions per step with different collision modes and values of n_I and c. All reported values are scaled by 10^{-4}.

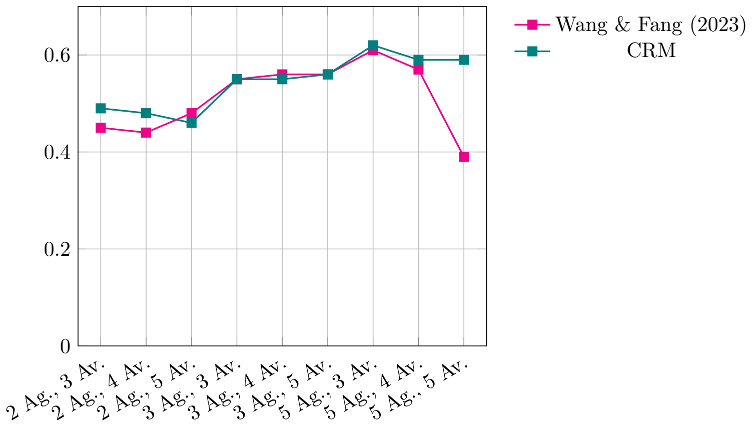


Figure 4: Mean reward per episode with different collision modes and values of n_I and c

tied to the cause of collisions. When there are few agents, the main cause of collisions is when two agents enter the collision avoidance distance while facing directly toward each other. When this happens, the projected corrected courses for both agents intersect as a result of the limited turning range, resulting in a collision. When $n_I = 2$, the agents spawn far from each other. As such, by the time they meet, there is more randomness in the potential directions they could be facing. Meanwhile, when $n_I = 5$, the agents meet each other a lot sooner, meaning it is a lot less likely for any agents to be facing directly toward each other, but since there are more agents, it is more likely for complex and risky situations. For $n_I = 3$ reaches a good balance given the scale of the environment. This changes of course as W_B, H_B increases. This becomes clear as you evaluate the collision frequencies of $n_I = 3, 5$ for $W_B, H_B = 30$, as shown in Fig. 3b. Now $n_I = 5$ has very few collisions, for the same reason $n_I = 3$ had when $W_B, H_B = 20$, while $n_I = 3$ has a higher collision frequency.

Figure 2c shows a strong correlation between the amount of agents and the amount of collisions when the agents have no collision avoidance. In addition, we see that the mean reward increases with the number of agents. This can be explained by the fact that the penalty for a collision is, when compared to the reward for finding all targets, not that big. Table 5 and Fig. 2c shows that our collision avoidance mechanism decreases collisions substantially, while Fig. 4 show no meaningful change in mean reward.

The number of collisions decreases quite a lot when the collision avoidance distance increases (Fig. 2c and Tbl. 5), the average episodic reward when using our collision avoidance mechanism is not really affected (Figure 4).

Assumptions and limitations. We assumed a simplified search area, namely a rectangle. Any search area can fit inside a rectangle, however if the shape was not a rectangle, our solution would not be efficient, due to searching outside the defined search area.

Making targets stationary is an assumption we know to be incorrect in many cases (Mohibullah & Julie, 2013). However, there are also some cases where the search targets cannot move, such as natural disaster cases where civilians are trapped or injured (Bortolin, 2024). In these cases, our model still holds.

Finally, we assume that all agents have communication with each other at all times, which is reasonable as the technology improves (Sharma et al., 2020).

We have discretized the action space. This may not affect the general idea; rather, it was done for the purpose of better understanding the trained behavior. It is possible that a continuous action space may require more training time, but the overall argument should remain valid.

5. Conclusion

This study has successfully established a cooperative target-seeking model based on the model presented by Wang & Fang (2023), and implemented an improved collision-avoidance mechanism that is realistic in a scenario where the range of motion is limited. The mechanism has been shown to be capable of avoiding collisions while finding targets. Still a complete collision avoidance under the set mobility constraint has not been achieved. In the future we would address the assumptions and limitations listed in Sect. 4. We would also like to pursue further enhancements to CRM. It would be interesting to give greater weight to closer agents in the collision avoidance process and to prioritise agents that are more difficult to avoid (imminent frontal collisions are particularly challenging to avoid). Finally, the collision avoidance mechanism should take into account the obstacles.

Acknowledgement. This work was partially funded by DIREC (Digital Research Centre Denmark), a collaboration between the eight Danish universities and the Alexandra Institute supported by the Innovation Fund Denmark.

References

Bortolin, M. (2024). 55 – Urban search and rescue. In G. Ciottone (Ed.), *Ciottone's Disaster Medicine (Third Edition)* (pp. 359–363). New Delhi: Elsevier. (Third edition ed.). URL: `https://www.sciencedirect.com/science/article/pii/B9780323809320000550`. doi:`https://doi.org/10.1016/B978-0-323-80932-0.00055-0`.

Cai, Y., Yang, S. X., & Xu, X. (2013). A combined hierarchical reinforcement learning based approach for multi-robot cooperative target searching in complex unknown environments. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)* (pp. 52–59). doi:`10.1109/ADPRL.2013.6614989`.

Cooper, D. C. (2005). *Fundamentals of search and rescue*. Jones & Bartlett Learning.

Ding, Z., Huang, Y., Yuan, H., & Dong, H. (2020). Introduction to reinforcement learning. In H. Dong, Z. Ding, & S. Zhang (Eds.), *Deep Reinforcement Learning: Fundamentals, Research and Applications* (pp. 49–50). Singapore: Springer Singapore. URL: `https://doi.org/10.1007/978-981-15-4095-0_2`. doi:`10.1007/978-981-15-4095-0_2`.

Dorigo, M., Caro, G. D., & Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial Life*, *5*, 137–172. doi:`10.1162/106454699568728`.

Gandhi, D., Pinto, L., & Gupta, A. (2017). Learning to fly by crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 3948–3955). IEEE.

Ghaffari, M., & Afsharchi, M. (2021). Learning to shift load under uncertain production in the smart grid. *International Transactions on Electrical Energy Systems*, *31*, e12748.

Hou, Y., Zhao, J., Zhang, R., Cheng, X., & Yang, L. (2024). Uav swarm cooperative target search: A multi-agent reinforcement learning approach. *IEEE Transactions on Intelligent Vehicles*, *9*, 568–578. doi:`10.1109/TIV.2023.3316196`.

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., & Araújo, J. G. (2022). Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, *23*, 1–18. URL: `http://jmlr.org/papers/v23/21-1342.html`.

Jevtić, A., Gazi, P., Andina, D., & Jamshidi, M. (2010). Building a swarm of robotic bees. (pp. 1 – 6).

Jin, Y., Zhang, Y., Yuan, J., & Zhang, X. (2019). Efficient multi-agent cooperative navigation in unknown environments with interlaced deep reinforcement learning. In *ICASSP 2019 - 2019 IEEE International Conference* *on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 2897–2901). doi:10.1109/ICASSP.2019.8682555.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. arXiv:1609.04836.

Kulkarni, S., Chaphekar, V., Chowdhury, M. M. U., Erden, F., & Guvenc, I. (2020). Uav aided search and rescue operation using reinforcement learning. In *2020 SoutheastCon* (pp. 1–8). IEEE volume 2.

Lygouras, E., Santavas, N., Taitzoglou, A., Tarchanidis, K., Mitropoulos, A., & Gasteratos, A. (2019). Unsupervised human detection with an embedded vision system on a fully autonomous uav for search and rescue operations. *Sensors*, *19*, 3542.

Mohibullah, W., & Julie, S. (2013). Developing an agent model of a missing person in the wilderness. (pp. 4462–4469). doi:10.1109/SMC.2013.759.

Oroojlooy, A., & Hajinezhad, D. (2023). A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, *53*, 13677–13722. URL: https://doi.org/10.1007/s10489-022-04105-y. doi:10.1007/s10489-022-04105-y.

Passino, K. (2002). Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Systems Magazine*, *22*, 52–67. doi:10.1109/MCS.2002.1004010.

Qin, X., Li, X., Liu, Y., Zhou, R., & Xie, J. (2020). Multi-agent cooperative target search based on reinforcement learning. *Journal of Physics: Conference Series*, *1549*, 022104. URL: https://dx.doi.org/10.1088/1742-6596/1549/2/022104. doi:10.1088/1742-6596/1549/2/022104.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv:1707.06347.

Senanayake, M., Senthooran, I., Barca, J. C., Chung, H., Kamruzzaman, J., & Murshed, M. (2016). Search and tracking algorithms for swarms of robots: A survey. *Robotics and Autonomous Systems*, *75*, 422–434. URL: https://www.sciencedirect.com/science/article/pii/S0921889015001876. doi:https://doi.org/10.1016/j.robot.2015.08.010.

Sharma, A., Vanjani, P., Paliwal, N., Basnayaka, C. M., Jayakody, D. N. K., Wang, H.-C., & Muthuchidambaranathan, P. (2020). Communication and networking technologies for uavs: A survey. *Journal of Network and Computer Applications*, *168*, 102739. URL: https://www.sciencedirect.com/science/article/pii/S1084804520302137. doi:https://doi.org/10.1016/j.jnca.2020.102739.

Sutton, R. S., & Barto, A. G. (2014). Reinforcement learning: An introduction, . URL: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

Tang, C., Abbatematteo, B., Hu, J., Chandra, R., Martín-Martín, R., & Stone, P. (2024). Deep reinforcement learning for robotics: A survey of real-world successes. *arXiv preprint arXiv:2408.03539*, .

Terry, J., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L. S., Dieffendahl, C., Horsch, C., Perez-Vicente, R. et al. (2021). Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, *34*, 15032–15043.

Terry, J. K., Black, B., & Hari, A. (2020). Supersuit: Simple microwrappers for reinforcement learning environments. *arXiv preprint arXiv:2008.08932*, .

Vu, D., Ngo, B., & Phan, H. (2022). Hybridnets: End-to-end perception network. arXiv:2203.09035.

Wang, P., Yang, H., Han, G., Yu, R., Yang, L., Sun, G., Qi, H., Wei, X., & Zhang, Q. (2024). Decentralized navigation with heterogeneous federated reinforcement learning for uav-enabled mobile edge computing. *IEEE Transactions on Mobile Computing*, .

Wang, X., & Fang, X. (2023). A multi-agent reinforcement learning algorithm with the action preference selection strategy for massive target cooperative search mission planning. *Expert Systems with Applications*, *231*, 120643. URL: https://www.sciencedirect.com/science/article/pii/S0957417423011454. doi:https://doi.org/10.1016/j.eswa.2023.120643.

Wu, Y., Low, K. H., & Lv, C. (2020). Cooperative path planning for heterogeneous unmanned vehicles in a search-and-track mission aiming at an underwater target. *IEEE Transactions on Vehicular Technology*, *69*, 6782–6787. doi:10.1109/TVT.2020.2991983.

Xia, J., Luo, Y., Liu, Z., Zhang, Y., Shi, H., & Liu, Z. (2023). Cooperative multi-target hunting by unmanned surface vehicles based on multi-agent reinforcement learning. *Defence Technology*, *29*, 80–94.

Yue, L., Yang, R., Zhang, Y., Yu, L., & Wang, Z. (2022). Deep reinforcement learning for UAV intelligent mission planning. *Complexity*, *2022*, 3551508.

RESEARCH ARTICLES

Minimizing Combined Sewer Overflows with Online Model-Predictive Reinforcement Learning

Esther Hahyeon Kim,[a] Mohsen Ghaffari,[b] Martijn Angelo Goorden,[a]
Andreas Holck Høeg-Petersen,[a] Kim Guldstrand Larsen,[a]
Thomas Dyhre Nielsen,[a] Andrzej Wsowski,[b]

[a] Department of Computer Science, Aalborg University, Aalborg, Denmark
[b] Department of Computer Science, IT University of Copenhagen, Denmark

ARTICLE HISTORY
Compiled September 25, 2024

ABSTRACT
This paper addresses the challenges posed by urban stormwater runoff in light of increased urbanization and climate change, which strain traditional stormwater infrastructure. It focuses on mitigating Combined Sewer Overflows (CSOs) by maximizing urban runoff storage in stormwater tunnels during Wastewater Treatment Plant (WWTP) capacity overloads. Unlike passive rule-based control, this research explores adaptive control systems that utilize weather forecasts and dynamic strategies. It introduces a novel control synthesis approach, combining Model Predictive Control (MPC) and Q-learning, to optimize CSO management based on real-time weather predictions. Simulated evaluations (using EPA-SWMM, focused on the Hvidovre stormwater tunnel in Copenhagen, Denmark) show significant improvements in CSO management: 17% over classic Q-learning and 34% over Q-learning with the Uppaal model, and 72% over Rule-Based Control (RBC). Our method, MPC Q-learning, dynamically incorporates weather conditions, outperforming other Q-learning approaches.

KEYWORDS
Stormwater management, Combined sewer overflows, Model predictive control, Control synthesis, Reinforcement learning, Clean Water and Sanitation, Climate Action

1. Introduction

Increasing urbanization and impervious surfaces in cities contribute to rising wastewater runoff and the release of pollutants into the surrounding ecosystems (Ellis and Marsalek 1996; Wenger et al. 2009). Stormwater systems manage runoff to prevent flooding and reduce environmental impact, but climate change strains their effectiveness (Semadeni-Davies et al. 2008; Alamdari et al. 2020). Urban stormwater infrastruc-

E. H. Kim. Email: hki@cs.aau.dk
M. Ghaffari. Email: mohg@itu.dk
M. A. Goorden. Email: mgoorden@cs.aau.dk
A. H. H.-Petersen. Email: ahhp@cs.aau.dk
K. G. Larsen. Email: kgl@cs.aau.dk
T. D. Nielsen. Email: tdn@cs.aau.dk
A. Wsowski. Email: wasowski@itu.dk

ture can be separated or combined with sewers. Combined sewer systems (CSSs) often generate combined sewer overflow (CSO) during storms, mixing sewage and stormwater, necessitating treatment before discharge (Marsalek et al. 1993). Various infrastructural solutions, collectively termed Best Management Practices, aim to mitigate the impacts of urban runoff (Marsalek 2005). These solutions generally aim to reduce stormwater inflow, increase combined stormwater and sewage storage, and treat overflows. This paper focuses on minimizing CSO by storing it before treatment, as retrofitting existing infrastructure for changing rainfall is costly. Consequently, water utility companies seek better methods to utilize their current infrastructure.

In this paper, we study the feasibility of using reinforcement learning (RL) based controllers (Sutton and Barto 2018) for realizing a storage-release controller, taking weather forecasts into account. In particular, we compare a number of learning strategies for such a controller using a CSO storage system for the area west of Copenhagen, the so-called Hvidovre pipeline. We train a controller using Model Predictive Control (MPC) in conjunction with Q-Learning (Watkins 1989), one of RL algorithms, against an *abstract model* of the water environment and weather within the tool Uppaal Stratego (David et al. 2015). We also train an alternative controller using standard Q-Learning directly using a SWMM model (Huber, Rossman, and Dickinson 2005; McDonnell et al. 2020) as an oracle, without an abstract model. We compare the models against a baseline existing rule-based controller (RBC) using a SWMM model. We find that the MPC Q-learning is better than the three other strategies. It gives comparable performance to training directly against SWMM but achieves convergence much more quickly.

Real-Time Control of Urban Water Management. Real-time control (RTC) is a crucial strategic tool in urban water management (García et al. 2015; Kerkez et al. 2016). It leverages real-time sensor data such as rainfall and water levels to efficiently operate systems and preemptively address potential issues. RTC primarily aims to minimize CSO, manage stormwater quality, and optimize water supply and demand. RTC can be categorized into RBC and optimization-based control. RBC relies on the expertise of operational staff or offline optimization processes to determine actuator set-points. However, its fixed configuration limits its ability to adapt to varying rainfall-runoff scenarios (Lin et al. 2020). MPC extensively researched as an optimization-based method, integrates system models with rainfall forecasts and optimization algorithms to recursively calculate optimal control actions (Lund et al. 2018; El Ghazouli et al. 2021; Tian et al. 2022). Although MPC has been extensively studied, practical implementations are limited. Challenges include hardware instability, significant computational resource consumption in real-time optimization, and uncertainty in rainfall forecasting (Lund et al. 2020, 2018).

Reinforcement-Learning-Based Control. Control synthesis based on RL offers a new approach to water management. This methodology focuses on modeling complex environments and training agents to make optimal real-time control decisions (Sutton and Barto 2018). In RL, two main approaches are commonly used: direct learning and indirect learning (Guan et al. 2021). Direct learning involves the agent interacting with the environment to directly learn a policy that maximizes rewards (Bowes et al. 2020, 2022; Zhang, Tian, and Liao 2023; Bowes et al. 2020, 2022). The agent observes states, selects actions to maximize rewards based on these observations, and learns iteratively through trial and error. This approach is particularly useful when the environment is complex or not fully known beforehand. Finding the path that maximizes rewards requires many trials and experiments, and it does not guarantee safety. In contrast, indirect learning utilizes a model of the environment to predict states and rewards, improving policies based on these predictions. This method can enhance learning efficiency when the model accurately represents the environment (Tian et al. 2022; Wang et al. 2021; Saddiqi et al. 2023; Luo et al. 2023). For instance, in water resource management, modeling the stormwater system allows for simulation-based optimization of control strategies before implementation, showcasing the applicability of indirect learning in complex real-world scenarios. The accuracy of the environmental model significantly affects learning performance. If the model is inaccurate, it can limit policy improvement. The choice between these approaches depends on the specific characteristics and requirements of the problem at hand, playing a crucial role in various applications of RL. Online combined RL involves the agent interacting with the environment in real-time, collecting data continuously, and updating the model incrementally. This approach balances exploration and exploitation to improve performance through iterative processes, allowing the agent to adapt to real-time changes in the environment, though it can be costly and risky. In contrast, Offline Learning uses a pre-collected, fixed dataset for one-time training, without the opportunity for new state or action exploration. While it cannot reflect real-time changes, it offers the advantages of safer training and cost and resource efficiency.

Control Based on Deep Reinforcement Learning. Deep reinforcement learning (DRL) has emerged as a leading technique, demonstrating enhanced flood mitigation. One of the earliest efforts to apply DRL for managing stormwater systems was investigated by Mullapudi et al. (2020), who used Deep Q-learning (Mnih et al. 2015) to control a system during heavy rainfall. Later Mullapudi et al. (2020) evaluated how the RL agent controls the multiple stormwater basins. Due to the risks associated with using a trial-and-error approach like RL in real-world applications, the usage of this work is limited to simulations of an actual system. The study conducted by Saliba et al. (2020) demonstrates how another DRL algorithm, specifically Deep Deterministic Policy Gradients, enhances flood mitigation compared to a passive control system considering data uncertainty of both state and forecast data. Yin et al. (2024) present a method that enables a single neural network for both CSO prediction and optimization tasks. Tian et al. (2022) train five distinct DRL models, including PPO, with varied architectures, with control actions selected by a voting mechanism. Negm, Ma, and Aggidis (2024) provide a comprehensive review of DRL applications in water systems. Despite being a recent survey, it only references a few papers related to stormwater system control. None of the mentioned studies address the comparison between offline learning and model predictive control reinforcement learning in this context. Consistently, Fu et al. (2022) note that the application of deep learning techniques is still in its early stages, with most studies relying on benchmark networks, synthetic data, and laboratory or pilot systems to test performance.

The rest of this paper is organized as follows: In Sect. 2, we describe our method and introduce our case study on urban water management, detailing the control problem and system model. The experimental results are presented in Sect. 3 with a comparison between RBC, direct Q-learning, offline MPC Q-learning, and online MPC Q-learning along with a discussion and analysis. Finally, Sect. 4 concludes the paper with discussions and future research challenges.

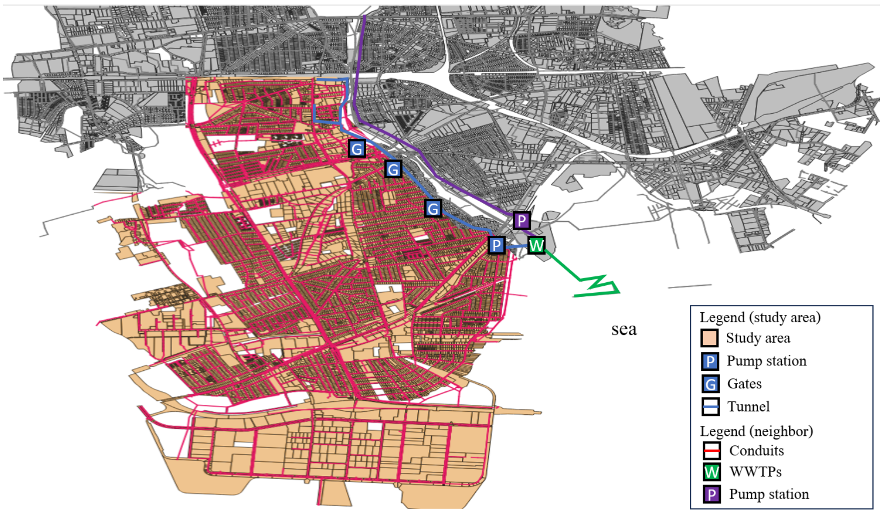


Figure 1.: The essential elements in our case study, including the Hvidovre tunnel

2. Methodology

Case Study. The case study area concerns the stormwater tunnel in Hvidovre municipality of Copenhagen, Denmark (Fig. 1). This case involves the management of runoff from a densely populated residential area spanning 2,241 hectares. The wastewater from this urban region, along with rainwater, is collected in the Hvidovre tunnel, depicted by blue solid lines in Fig. 1, which constitutes the CSS. The tunnel, ranging in diameter from 1.6 to 2.5 meters and stretching a total length of 3.2 kilometers, is segmented into four sections. Each section is equipped with three gates (indicated by blue 'G') and AMN (Åmarken Nord) pump station (indicated by blue 'P'). There are two pumps: the default pump and the emergency pump, each with different performance characteristics. The system measures the water levels at the gates, the AMN pump station, and the flow rate at the wastewater treatment plant (WWTP, indicated by green 'W'). It controls the gates and the AMN pump operation to regulate the inflow rate into the WWTP, ultimately minimizing CSOs. The WWTP consists of two stages: a physical treatment stage with a maximum treatment capacity of 5.0 m^3/sec and a biological treatment stage with a maximum treatment capacity of 2.77 m^3/sec. The purified wastewater, having passed through both stages, is discharged into the sea.

Rule-based Control (RBC). The current RBC in the Hvidovre tunnel is reactive. It responds to the present conditions (water level of the pump storage and gates) rather than proactively considering future events, such as a forecast of heavy rain. It keeps all tunnel gates completely open during dry weather and light rain. As runoff passes through the tunnel and reaches the AMN pump station, all the water is channeled towards the WWTP. While the WWTP has treatment capacity limits, these do not pose a problem under light weather conditions. However, heavy rainfall can lead to challenges for CSSs in general and the Hvidovre tunnel in particular. The primary concern is flooding, which happens when underground pipes in residential areas become filled with water, potentially leading to wastewater overflow. To address this, residential area's runoff is directed to the Hvidovre tunnel situated in low-lying areas, aiding in alleviating flooding problems that most severely affect residential areas.

The second major concern is overflow. Currently, the RBC of the AMN pump station is subject to regulations set by the Copenhagen municipality, which allow overflow up to five times annually. When the water level in the AMN pump station storage rises to 6.3 meters, untreated wastewater overflows into the nearby Damhusåen river. Further, if the inflow of wastewater exceeds the biological treatment capacity of the WWTP, the biologically untreated excess is released into the sea (bypass). When the AMN pump stops, wastewater accumulates in the AMN storage, and overflow occurs when the storage level continues to rise. To prevent this, discharge into the WWTP begins when the AMN storage level reaches 5.3 m, at a rate of 1.0 m^3/sec, as the bypass is less harmful to the environment than overflow. If the AMN storage tank level does not reach 5.3 meters, the water is retained until the WWTP restores its biological treatment capacity, at which point the AMN pump initiates draining of the Hvidovre tunnel.

Maintaining a low level in the AMN storage can reduce overflow and bypass. The three gates (G1, G2, G3) in the Hvidovre tunnel divide the sloped tunnel into four sections, maximizing the amount of wastewater, so the tunnel functions as a storage device. According to the RBC, when the water level in the AMN storage exceeds 4.2 m, all gates are simultaneously closed. Then, as the water level of AMN storage decreases to normal water levels (2.7 m), G3 is the first to be opened. As the water level continues to drop ($W_{G3} = 0.36$ m) due to the opening of G3, G2 is opened, and when the water level further decreases ($W_{G2} = 0.48$ m) due to the opening of G2, G1 is opened. Furthermore, in scenarios of exceptionally intense rainfall resulting in water levels at each gate reaching the overflow thresholds ($W_{G1} = 3.39$ m, $W_{G2} = 4.23$ m, $W_{G3} = 4.26$ m), an emergency gate control mechanism is activated to open the gates.

2.1. *Methodology*

Reinforcement Learning. RL is a powerful framework for solving optimal control problems, where an agent should learn how to act in an unknown environment (Sutton and Barto 2018). This type of learning is defined for Markov Decision Processes (MDPs) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T})$. MDPs work in discrete time: at each time step t, the controller receives a representation of the environment's state $s_t \in \mathcal{S}$, and takes an action $a_t \in \mathcal{A}$ based on a policy $\pi : \mathcal{S} \to \mathcal{A}$. The action changes the environment's state according to $\mathcal{T}(s_t, a_t) = s_{t+1}$, possibly in a stochastic manner, and this transition results in a reward $r_t \in \mathcal{R}$. The optimal control objective is then to maximize from each initial state the expected cumulative reward. The problem is thus one of sequential decision-making, optimizing the long-term performance. RL achieves this via the action-value function $Q^\pi : (\mathcal{S}, \mathcal{A}) \to \mathbb{R}$, which is the discounted expected return of rewards given a state, action, and policy. There are several methods available to apply for reinforcement learning problems such as Q-Learning and SARSA (Sutton and Barto 2018). Q-learning is one of the most known algorithms based on the Temporal Difference method (Sutton and Barto 2018), which updates the Q-value immediately and allows using the updated version later in the same episode for choosing a new policy. The Q-value will be updated using the following equation (Sutton and Barto 2018).

$$Q_{t+1}(s_t, a_t) = (1 - \alpha)Q_t(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)] \quad (1)$$

where, α is the learning rate, and γ is the discount factor. Once the learning process has converged to the optimal policy, the optimal policy for each state in the environment can be extracted from the Q-table.

UPPAAL Stratego. Uppaal Stratego is a model checking and verification tool designed to model real-time systems (David et al. 2015). For hybrid systems with control variables, the tool is capable of learning a near-optimal control strategy, using statistical model checking to give probabilistic guarantees for the properties of the system under the learned strategy. Uppaal Stratego employs a non-deep reinforcement learning algorithm that differs from classical Q-learning, in that the continuous state space is discretized during learning via an *online refinement scheme* based on the variance in the expected reward in different regions of the state space (Jaeger et al. 2019).

MPC Q-Learning vs Q-Learning. Q-learning entails training a model utilizing either a simulator or available data, followed by its application to real-world problem-solving (the evaluation step). During this process, the agent learns the optimal Q-function, which determines the best actions to maximize rewards given specific states and actions. In evaluation, the agent uses the learned Q-function to choose optimal actions and solve problems accordingly. Q-learning is effective when the given simulator or data comprehensively covers all potential scenarios. It requires an accurate model, focusing initially on handling all possible situations. Conversely, MPC Q-learning introduces interaction between the agent and the environment during both training and evaluation phases. Even after the initial training phase, the model continues to improve and optimize based on real-time feedback from the environment. In the evaluation phase, the agent interacts with the environment in real-time to make decisions. This real-time interaction allows the agent to adapt more quickly and choose optimal actions in response to dynamic changes in the environment. MPC Q-learning is particularly effective in addressing uncertainties and variability that may occur in real-world environments. It allows continuous refinement of the model based on actual feedback.

2.2. *Modeling*

We conduct a co-simulation by integrating the RL-framework Uppaal Stratego with the domain-specific simulator SWMM (McDonnell et al. 2020), an open-source, physically-based dynamic rainfall-runoff model extensively utilized in urban stormwater management for decades. We model the CSS in SWMM, along with our case study and neighboring infrastructures, which reflects the real-world system in our research.

We outline the system's RL framework modeling based on Q-learning and MPC Q-learning. The experimental setup and related materials, including the SWMM model and RL framework modeling using Uppaal Stratego, are available online.[1]

The System Model for Q-Learning. We model the CSS case study as an MDP as described in Sect. 2.1. Specifically, in this paper the agent is a centralized controller for the gates and pump of the Hvidovre tunnel. The gates split the entire tunnel into four sections. The system's state can be modeled as $\mathcal{S} = \{(x_1, x_2, x_3, x_4, r_0, r_1, r_2, r_3, r_4) \mid \forall i : x_i \in \mathbb{R}, \forall j : r_j \in \mathbb{R}\}$, where x_i is the water level in the i-th section and r_h is the average rain precipitation prediction in the next h-th hour. The action set is $\mathcal{A} =$

[1]DOI: 10.5281/zenodo.11191543

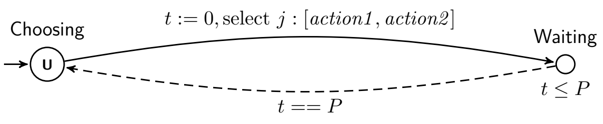

Choosing
$t := 0, \text{select } j : [\textit{action1}, \textit{action2}]$
Waiting
U
$t \leq P$
$t == P$

Figure 2.: Model of a controller, which can change modes periodically.

$\{(a_1, a_2, a_3, a_4) \mid \forall i : a_i \in \{0, 1\}\}$, where a_1, a_2, a_3 corresponds to the control setting (open or closed) of the first, second and third gate, respectively. Action component a_4 refers to the pumping station, which can either be on or off. The reward function is defined in Sect. 3.

Let's explain one step of the transition of the model. At time t, we observe the water level in each section from our SWMM model, and predict the rain scenario for $t \leq h < t + 5$. Then, according to these observations, we select a new controller for the SWMM model and at time $t + 1$, we can collect the overflow and bypass that occurred in the SWMM. As time t increases during the given simulation time horizon, the update of the Q-table using the designated reward function continues in accordance with the outlined procedure.

The System Model in UPPAAL STRATEGO for MPC Q-learning. We synthesized a control strategy for gates for this model using UPPAAL STRATEGO (David et al. 2015) with online learning. The system was developed using the STOMPC framework (Goorden et al. 2022), which integrates UPPAAL STRATEGO for synthesizing a control strategy of the gates and the pump, and SWMM for reflecting real-system.

Fig. 2 illustrates the controller model assigned to each of these gates. The Choosing location serves as the starting point for the gate and pump controller's operations. The transition from the Choosing to the Waiting locations occurs immediately when the system starts (the Choosing location is an urgent location, indicated by ∪, in which time cannot progress), and the clock t is set to 0. To model that a control decision is taken periodically with period P, the controller waits in the Waiting location. Once the controller has waited P time units, the transition from Waiting to Choosing is taken and the process repeats. In the model, we specify the possible control modes of gates as $[\textit{open}, \textit{close}]$ and set the interval P at 10 min. The controller model for the pump operates similarly to that of the gates, except that the feasible control modes are $[\textit{on}, \textit{off}]$.

Each section of the tunnel is simplified in the UPPAAL STRATEGO model used for online learning: they are approximated using the model of a tank. The water volume in each section V_i can be described using

$$\frac{\mathrm{d}V_i}{\mathrm{d}t} = \begin{cases} 0 & \text{if } (w_i = 0) \wedge (Q_{in,i} \leq Q_{out,i}), \\ 0 & \text{if } (w_i \geq W_i) \wedge (Q_{in,i} \geq Q_{out,i}), \\ Q_{in,i} - Q_{out,i} & \text{else}, \end{cases} \tag{2}$$

where $Q_{in,i}$ is the inflow to section i, $Q_{out,i}$ the outflow, w_i the current water level, and W_i the maximum water level of the i-section. The first boundary case in this equation occurs when the outflow $Q_{out,i}$ is greater than the inflow $Q_{in,i}$, and the water level w_i is (already) zero. The second case describes a full-capacity situation during which the inflow $Q_{in,i}$ is greater than the outflow $Q_{out,i}$, hence no additional water can accumulate in the section. The inflow $Q_{in,i}$ into section i consists of the outflow from the previous section $Q_{out,i-1}$ (except the first tunnel section) and the runoff coming from the urban catchment connected to this section. The outflow $Q_{out,i}$ fluctuates based on the gate opening decided by the controller.

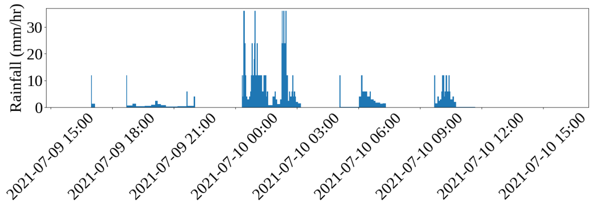
Eq. (2) can also be used to describe the water volume dynamics at the pump station w_p depending on its inflow $Q_{in,p}$ and outflow $Q_{out,p}$. The inflow into the pump station is the outflow from the last section of the tunnel ($Q_{out,4}$), while the outflow depends on the mode chosen by the pump controller. To respond to rapid changes in environmental conditions (in our case, weather), we have implemented MPC (Camacho and Alba 2013), which predicts the future states of a system using a predictive model and generates optimal control signals based on these predictions. This method is primarily used to find the best control inputs while considering various variables and constraints. Since environmental conditions like weather are highly volatile, it is crucial to reflect these changes in real-time so that the control system can respond appropriately. MPC generates control signals through an iterative process. In each control horizon, the latest data is used to predict the system's state, and the optimal control signals for the next horizon are calculated based on this prediction. To achieve this, the latest sensor data on actual water levels is utilized to accurately determine the current state, and weather forecast data is used to predict future states. Prediction horizons and control horizons are key elements of MPC; during the prediction horizon, the future state of the system is predicted, and during the control horizon, the control signals to be applied are determined.

3. Experimental Results

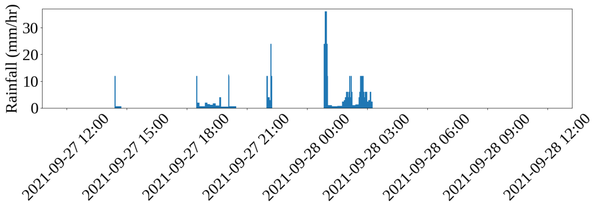
We now evaluate the effectiveness of four control approaches: (i) MPC Q-learning, (ii) classic Q-learning, (iii) Q-learning using UPPAAL model, and (iv) Rule-based control (RBC). Our assessment focuses on the control objectives of removing overflow at the pump station and reducing bypasses into WWTP. We compare the performance of different controllers synthesized using various methods by applying them to the PySWMM model (McDonnell et al. 2020). To ensure the robustness and validity of our evaluation, we employ k-fold cross-validation for RL methods (Burman 1989). This validation technique verifies the model's reliability by assessing its performance on data segments not seen during training. Specifically, we employ five distinct rainfall scenarios, training the model on four scenarios in each iteration while assessing its effectiveness on the scenario which is not used for training to avoid over-fitting in learning.

Weather Forecast. To facilitate a meaningful comparison among the various controllers, we selected five significant rainfall events from 2021. These events were documented for their impact on overflow and bypass occurrences caused by heavy rainfall by water utility companies. These selected events are obtained through historical data from rain gauges near our study site. Fig. 3 provides details of each 24-hour rain event, including total precipitation (mm) and cumulative inflow from all urban catchments into the Hvidovre tunnel (m^3). The precipitation data were collected at one-minute intervals and sourced from the Danish Meteorological Institute (DMI).[2] To train Q-learning and MPC Q-learning based control, we applied a 10% uncertainty factor to the data in Fig. 3 to generate weather forecasts based on historical rain data.

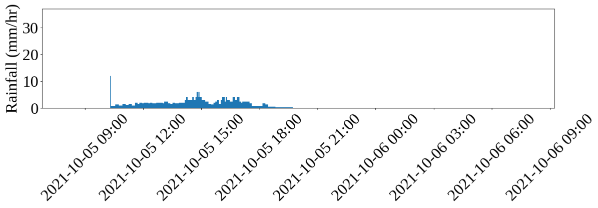
[2] `dmi.dk`



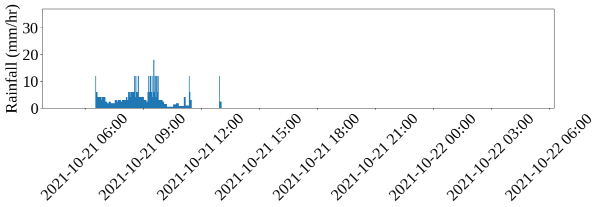
(a) Urban catchments of 62,311 m^3 and total precipitation of 35.80 mm.



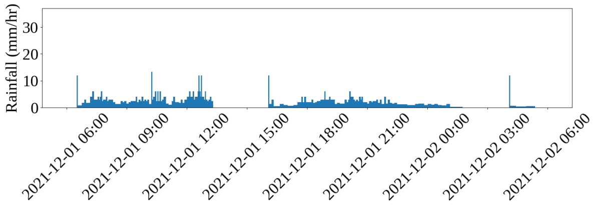
(b) Urban catchments of 20,418 m^3 and total precipitation of 16.60 mm.



(c) Urban catchments of 26,979 m^3 and total precipitation of 17.20 mm.



(d) Urban catchments of 30,856 m^3 and total precipitation of 17.60 mm.



(e) Urban catchments of 65,329 m^3 and total precipitation of 36.60 mm.

Figure 3.: Rain scenarios used for the experiments.

	MPC Q-learning (UPPAAL)		Q-learning (Classic)		Q-learning (UPPAAL)		Rule-based	
	O_T(m^3)	B_T(m^3)	O_T(m^3)	B_T(m^3)	O_T(m^3)	B_T(m^3)	O_T(m^3)	B_T(m^3)
Rain 1	0	30,460 ± 180	0	37,860 ± 160	0	38,236 ± 80	16,200	24,840
Rain 2	0	2,195 ± 55	0	2,520 ± 51	0	3,155 ± 37	0	5,330
Rain 3	0	4,701 ± 78	0	5,580 ± 90	0	5,774 ± 53	0	6,280
Rain 4	0	5,160 ± 70	0	6,059 ± 90	0	6,923 ± 30	0	7,200
Rain 5	0	31,510 ± 240	0	37,668 ± 300	0	44,041 ± 190	16,920	27,300

Table 1.: The evaluation results over rain events in prioritizing overflow O_T than the bypass of pump station B_T with cost function using ($\mathcal{C}_1 > \mathcal{C}_2$).

Prioritizing Control Objectives. Currently, the AMN pumps operate under RBC. The goal of our case study is to adjust the operation of gates and pumps optimally based on the cost function Eq. (3) to prevent total accumulated overflow at the pump station (O_T) and reduce the cumulative amount of untreated bypass (B_T) discharged from the WWTP to the sea, during each rainfall event. Below the $cost_t$ represents the cost calculated at each time step t; integrating this over the full period T results in the total cost.

$$cost_t = \underbrace{\mathcal{C}_1 \times \mathrm{O}_t}_{\text{Part 1: overflow}} + \underbrace{\mathcal{C}_2 \times \mathrm{B}_t}_{\text{Part 2: bypass}} + \underbrace{\mathcal{C}_3 \times |\mathrm{M}_t - \mathrm{M}_{t-1}|}_{\text{Part 3: changing the control mode}} \tag{3}$$

The constant weights $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ indicate the relative importance of the different objectives, i.e. $\mathcal{C}_1 > \mathcal{C}_2 > \mathcal{C}_3$. The first part of the cost function penalizes the overflow amount O_t at timestep t. The second part of the cost function indicates the bypass amount B_t at t. Finally, the last part of the cost function has been designed to incentivize the pump operation not to change the mode if it is unnecessary. To achieve this, we compare the control mode of the current pump M_t, with the control mode from the previous time step M_{t-1}, and apply a penalty for frequent changes in control modes.

Tbl. 1 presents the experimental results of MPC Q-learning, Q-learning using UPPAAL model, classic Q-learning, and RBC. The results obtained from MPC Q-learning demonstrate our proposed methodology's efficacy. Conversely, the outcomes of classic Q-learning and UPPAAL model Q-learning serve as a standard against which to measure the performance of the learning-based approach, while the results of the RBC serve as a benchmark for assessing real-system performance. We assess the performance of each control method using a rain scenario that the system had not previously encountered, employing k-fold cross-validation. Each experiment was repeated five times, and the mean values and standard errors, with a 95% confidence interval, are presented in Tbl. 1. The RBC-based result shows overflow O_T in two extreme rain scenarios (Rain 1 and Rain 5), while both Q-learning and MPC Q-learning have no overflow O_T in any case. Across all rain scenarios, when considering bypass (B_T), MPC Q-learning consistently outperforms classic Q-learning by an average of 17%, Q-learning using UPPAAL by 34%, and rule-based control (RBC) by 72%. This performance improvement results in no overflow (O_T) and minimal bypass (B_T), aligning with the desired control objectives. These findings confirm the effectiveness of MPC Q-learning and Q-learning-based controllers in achieving primary operational goals.

Fig. 4 presents qualitative outcomes of the MPC Q-learning, Q-learning controllers, and RBC under the first rain scenario as detailed in Fig. 3. Rain1 is a scenario characterized by heavy and intense rainfall. Although the actual system has three gates,

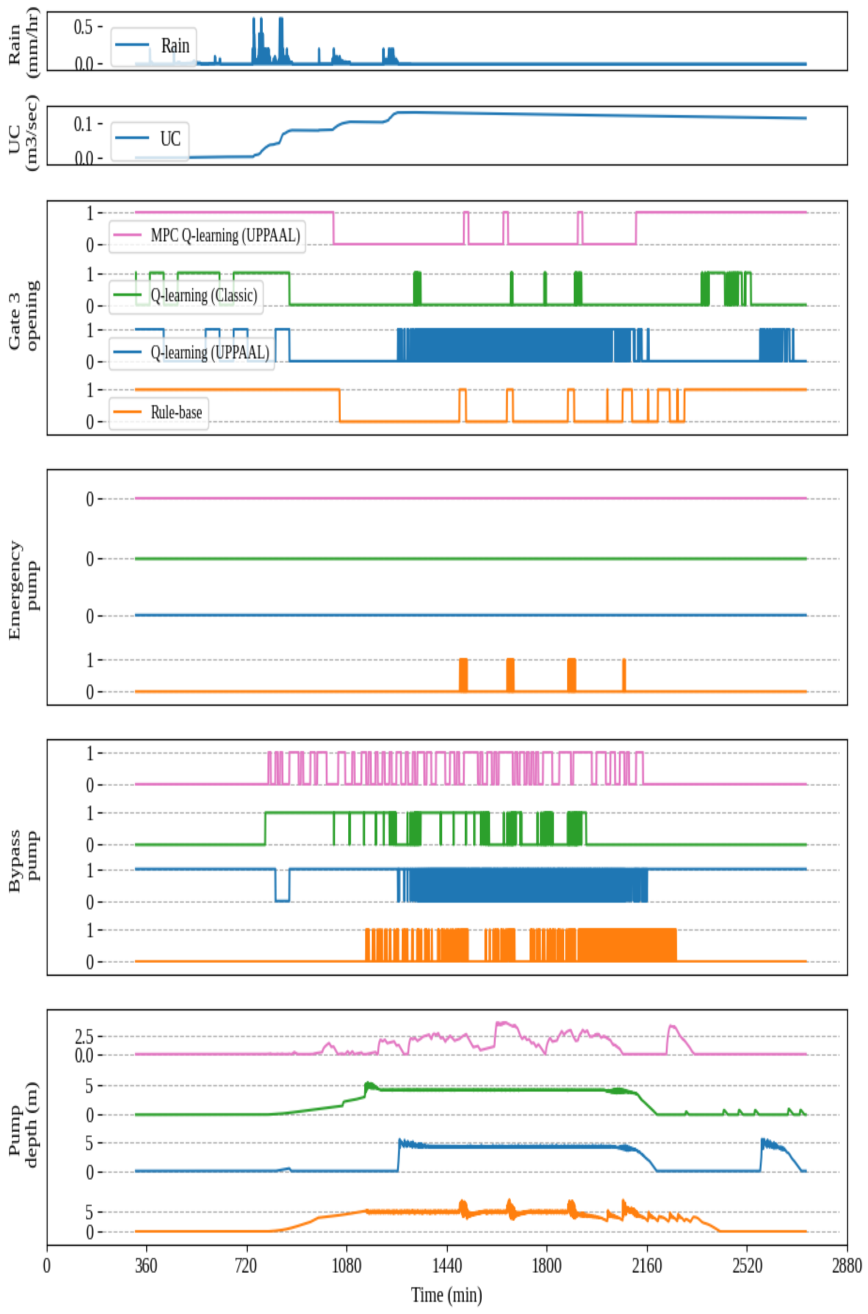


Figure 4.: Results of MPC Q-learning using UPPAAL model, Q-learning, Q-learning using UPPAAL model, and RBC applied control strategies in response to Rain Scenario 1.

the control behavior of all three gates is similar, so the current figure illustrates the control behavior of gate 3. In the case of MPC Q-learning, it is observed that the gate remains closed until the water level is high to minimize gate operations and maximize water retention, compared to classic Q-learning and Uppaal model based Q-learning. Compared to RBC, it is noted that the gate opens more quickly to release water after detecting the end of the heavy rain, as shown by the control trend in the latter part of the simulation. The last three plots in the figure show information about the pump station. Notably, the emergency pump only operates in the RBC scenario. MPC Q-learning and Q-learning trigger a bypass earlier than RBC, preventing the emergency pump from operating by maintaining a lower water level at the pump station. This decision is guided by incorporating weather forecasts into the control synthesis to better prepare for heavy rain. In the case of MPC Q-learning control, it can be observed that the control behavior frequency is lower compared to Q-learning based control, and the amount of bypass is also less, as shown in Tbl. 1.

Applicability of Learning Across Diverse Objectives Objectives and prioritization can vary significantly across different case studies in water management. Additional experiments have been conducted with varying prioritized objectives, as shown in Tbl. 2, where $\mathcal{C}_1 < \mathcal{C}_2$ or $\mathcal{C}_1 = \mathcal{C}_2$ are employed using the cost function in Eq. (3). Note that these results are not comparable to those in Tbl. 1.

We apply the constant $\mathcal{C}_1 < \mathcal{C}_2$ to the cost function Eq. (3), giving O_T a lesser penalty than B_T across five rain scenarios. As shown in Tbl. 2, in all scenarios, MPC Q-learning consistently achieves superior performance compared to both model-based and classic Q-learning. On average, MPC Q-learning shows a 25% improvement over classic Q-learning and a 51% improvement over Uppaal model Q-learning.

Furthermore, experiments using $\mathcal{C}_1 = \mathcal{C}_2$ in the cost function Eq. (3) apply equal penalties to both overflow and bypass amounts. Consequently, the controller has the flexibility to select either overflow or bypass for water discharge, aggregating O_T and B_T in Tbl. 2. The findings indicate that MPC Q-learning consistently delivers superior outcomes, demonstrating enhanced optimization compared to Q-learning, even under conditions where $\mathcal{C}_1$ equals $\mathcal{C}_2$. On average, MPC Q-learning outperforms classic Q-learning by 27% and Uppaal model Q-learning by 55%.

Discussion. We found the performance of RL-based controllers (MPC Q-learning using Uppaal model, Q-learning using Uppaal model, and classic Q-learning) outperforms RBC. The RL-based controller, which incorporates weather forecasts, demonstrates its capability to prevent overflow, particularly during periods of intense rainfall (refer to the Rain1, Rain5 scenario in Fig. 3). By showing that the performance of MPC Q-learning exceeds that of other controllers, it can be seen that in the case of the CSO management system, which is closely related to rain scenarios, RL applying the online approach shows better performance. In RL, the choice of cost function is crucial for learning, as discussed in Sect. 3. The outcomes can vary depending on the priorities set within the cost function. Our study introduces an initial objective towards developing a robust RL-based control method applicable to urban water systems.

	MPC Q-learning (UPPAAL)		**Q-learning (Classic)**		**Q-learning (UPPAAL)**	
$\mathcal{C}_1 < \mathcal{C}_2$	O_T(m^3)	B_T(m^3)	O_T(m^3)	B_T(m^3)	O_T(m^3)	B_T(m^3)
Rain 1	26,232 ± 210	3,024 ± 120	26,592 ± 230	4,531 ± 190	26,537 ± 170	6,135 ± 90
Rain 2	3,132 ± 140	0	4,076 ± 190	0	4,637 ± 130	0
Rain 3	5,136 ± 130	0	6,720 ± 160	0	7,988 ± 80	0
Rain 4	5,352 ± 110	0	6,170 ± 130	0	8,028 ± 90	0
Rain 5	31,176 ± 180	8,472 ± 120	34,869 ± 240	11,548 ± 160	38,356 ± 210	15,249 ± 120
$\mathcal{C}_1 = \mathcal{C}_2$	$O_T + B_T$(m^3)		$O_T + B_T$(m^3)		$O_T + B_T$(m^3)	
Rain 1	29,460 ± 180		34,386 ± 200		39,771 ± 160	
Rain 2	2,916 ± 110		4,082 ± 230		5,249 ± 90	
Rain 3	4,848 ± 80		5,988 ± 160		7,562 ± 70	
Rain 4	4,524 ± 100		5,633 ± 80		6,831 ± 80	
Rain 5	30,804 ± 180		38,730 ± 150		45,211 ± 160	

Table 2.: The results are divided into two categories: (up) pump station overflow is prioritized over bypass ($\mathcal{C}_1 < \mathcal{C}_2$), and (down) pump station overflow and bypass are treated equally ($\mathcal{C}_1 = \mathcal{C}_2$).

4. Conclusion

This study conducted experiments using RL to improve the mitigation of sewage overflows in pumping stations and gates of CSSs during storm events. We introduce a controller synthesis for CSSs using a novel approach called MPC Q-learning, which leverages the Uppaal model. This method outperforms other synthesized controllers, including Q-learning using the Uppaal model, classic Q-learning without model, and RBC. To evaluate the policies provided by each real-time control system more realistically, we developed a configuration of the SWMM model to represent the actual infrastructure system. We used the k-fold method for evaluation, ensuring that rainfall scenarios not used in training were used for evaluation. RBC was used as a baseline to compare against the learning-based controllers. Five different rainfall scenarios were used to compare the robustness of each controller. The key contributions of this study are summarized as follows.

- RL-based control strategies can more effectively mitigate CSOs in both the pumps and gates of CSSs compared to the RBC used in real systems.
- Among different RL-based controllers, MPC Q-learning using the Uppaal model proved to be the most suitable controller for adapting to unknown environmental conditions.

In our current study, we have chosen to use the Q-learning algorithm from the various available RL algorithms. However, in future research, we aim to assess the performance of RL-based control using a range of RL algorithms. Additionally, we plan to implement a safety-guaranteed shield to ensure that the learning-based control adheres to safety constraints and does not violate them.

Acknowledgement(s)

We extend our gratitude to HOFOR, the Danish water utility company, for furnishing us with a case study. Special appreciation goes to Lone Bo Jørgensen, Hydraulic Specialist at HOFOR, for her invaluable assistance in modeling based on the real system.

Disclosure statement

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This research is funded by the Digital Research Centre Denmark (DIREC), Innovation Fund Denmark.

References

Alamdari, Nasrin, David J. Sample, Andrew C. Ross, and Zachary M. Easton. 2020. "Evaluating the Impact of Climate Change on Water Quality and Quantity in an Urban Watershed Using an Ensemble Approach." *Estuaries and Coasts* 43: 56–72. https://doi.org/10.1007/s12237-019-00649-4.

Bowes, Benjamin D., Arash Tavakoli, Cheng Wang, Arsalan Heydarian, Madhur Behl, Peter A. Beling, and Jonathan L. Goodall. 2020. "Flood Mitigation in Coastal Urban Catchments Using Real-Time Stormwater Infrastructure Control and Reinforcement Learning." *Journal of Hydroinformatics* 23: 529–547. https://doi.org/10.2166/hydro.2020.080.

Bowes, Benjamin D., Cheng Wang, Mehmet B. Ercan, Teresa B. Culver, Peter A. Beling, and Jonathan L. Goodall. 2022. "Reinforcement Learning-Based Real-Time Control of Coastal Urban Stormwater Systems to Mitigate Flooding and Improve Water Quality." *Environmental Science: Water Research & Technology* 8: 2065–2086. https://doi.org/10.1039/D1EW00582K.

Burman, Prabir. 1989. "A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods." *Biometrika* 76: 503–514.

Camacho, Eduardo F, and Carlos Bordons Alba. 2013. *Model predictive control.* Springer.

David, Alexandre, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikučionis, and Jakob Haahr Taankvist. 2015. "Uppaal Stratego." In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, 206–211.

El Ghazouli, Khalid, Jamal El Khatabi, Aziz Soulhi, and Isam Shahrour. 2021. "Model Predictive Control Based on Artificial Intelligence and EPA-SWMM Model to Reduce CSOs Impacts in Sewer Systems." *Water Science and Technology* 85: 398–408. https://doi.org/10.2166/wst.2021.511.

Ellis, J.B., and J. Marsalek. 1996. "Overview of urban drainage: environmental impacts and concerns, means of mitigation and implementation policies." *Journal of Hydraulic Research* 34: 723–732. https://doi.org/10.1080/00221689609498446.

Fu, Guangtao, Yiwen Jin, Siao Sun, Zhiguo Yuan, and David Butler. 2022. "The Role of Deep Learning in Urban Water Management: A Critical Review." *Water Research* 223: 118973. https://doi.org/10.1016/j.watres.2022.118973.

García, L., J. Barreiro-Gomez, E. Escobar, D. Téllez, N. Quijano, and C. Ocampo-Martinez. 2015. "Modeling and real-time control of urban drainage systems: A review." *Advances in Water Resources* 85: 120–132. https://doi.org/https://doi.org/10.1016/j.advwatres.2015.08.007.

Goorden, Martijn A., Peter G. Jensen, Kim G. Larsen, Mihhail Samusev, Jií Srba, and Guohan Zhao. 2022. "STOMPC: Stochastic Model-Predictive Control withăUppaal Stratego." In *Automated Technology for Verification and Analysis*, edited by Ahmed Bouajjani, Luká Holík, and Zhilin Wu, Lecture Notes in Computer Science, 327–333. Springer International Publishing.

Guan, Yang, Shengbo Eben Li, Jingliang Duan, Jie Li, Yangang Ren, Qi Sun, and Bo Cheng. 2021. "Direct and indirect reinforcement learning." *International Journal of Intelligent Systems* 36 (8): 4439–4467.

Huber, Wayne C, Lewis A Rossman, and Robert E Dickinson. 2005. "EPA storm water management model, SWMM5." *Watershed models* 338: 359.

Jaeger, Manfred, Peter Gjøl Jensen, Kim Guldstrand Larsen, Axel Legay, Sean Sedwards, and Jakob Haahr Taankvist. 2019. "Teaching Stratego to Play Ball: Optimal Synthesis for Continuous Space MDPs." In *Automated Technology for Verification and Analysis*, edited by Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, Cham, 81–97. Springer International Publishing.

Kerkez, Branko, Cyndee Gruden, Matthew Lewis, Luis Montestruque, Marcus Quigley, Brandon Wong, Alex Bedig, et al. 2016. "Smarter stormwater systems." .

Lin, Pengfei, Jinjun You, Hong Gan, and Ling Jia. 2020. "Rule-based object-oriented water resource system simulation model for water allocation." *Water Resources Management* 34:

3183–3197.

Lund, Nadia Schou Vorndran, Morten Borup, Henrik Madsen, Ole Mark, and Peter Steen Mikkelsen. 2020. “CSO reduction by integrated model predictive control of stormwater inflows: a simulated proof of concept using linear surrogate models.” *Water Resources Research* 56 (8): e2019WR026272.

Lund, Nadia Schou Vorndran, Anne Katrine Vinther Falk, Morten Borup, Henrik Madsen, and Peter Steen Mikkelsen. 2018. “Model predictive control of urban drainage systems: A review and perspective towards smart real-time water management.” *Critical Reviews in Environmental Science and Technology* 48: 279–339.

Luo, Xinran, Pan Liu, Qian Xia, Qian Cheng, Weibo Liu, Yiyi Mai, Chutian Zhou, Yalian Zheng, and Dianchang Wang. 2023. “Machine Learning-Based Surrogate Model Assisting Stochastic Model Predictive Control of Urban Drainage Systems.” *Journal of Environmental Management* 346: 118974. https://doi.org/10.1016/j.jenvman.2023.118974.

Marsalek, J., T. O. Barnwell, W. Geiger, M. Grottker, W. C. Huber, A. J. Saul, W. Schilling, and H. C. Torno. 1993. “Urban Drainage Systems: Design and Operation.” *Water Science and Technology* 27: 31–70. https://doi.org/10.2166/wst.1993.0291.

Marsalek, Jiri. 2005. “Evolution of Urban Drainage: From Cloaca Maxima to Environmental Sustainability.” In *Acqua e Citta, I Convegno Nazionale di Idraulica Urbana*, 1–22.

McDonnell, Bryant E., Katherine Ratliff, Michael E. Tryby, Jennifer Jia Xin Wu, and Abhiram Mullapudi. 2020. “PySWMM: The Python Interface to Stormwater Management Model (SWMM).” *Journal of Open Source Software* 5: 2292. https://doi.org/10.21105/joss.02292.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, et al. 2015. “Human-Level Control through Deep Reinforcement Learning.” *Nature* 518: 529–533. https://doi.org/10.1038/nature14236.

Mullapudi, Abhiram, Matthew J Lewis, Cyndee L Gruden, and Branko Kerkez. 2020. “Deep reinforcement learning for the real time control of stormwater systems.” *Advances in water resources* 140: 103600.

Negm, Ahmed, Xiandong Ma, and George Aggidis. 2024. “Deep Reinforcement Learning Challenges and Opportunities for Urban Water Systems.” *Water Research* 253: 121145. https://doi.org/10.1016/j.watres.2024.121145.

Saddiqi, M. Matin, Wanqing Zhao, Sarah Cotterill, and Recep Kaan Dereli. 2023. “Smart Management of Combined Sewer Overflows: From an Ancient Technology to Artificial Intelligence.” *WIREs Water* 10: e1635. https://doi.org/10.1002/wat2.1635.

Saliba, Sami M., Benjamin D. Bowes, Stephen Adams, Peter A. Beling, and Jonathan L. Goodall. 2020. “Deep Reinforcement Learning with Uncertain Data for Real-Time Stormwater System Control and Flood Mitigation.” *Water* 12: 3222. https://doi.org/10.3390/w12113222.

Semadeni-Davies, Annette, Claes Hernebring, Gilbert Svensson, and Lars-Göran Gustafsson. 2008. “The impacts of climate change and urbanisation on drainage in Helsingborg, Sweden: Suburban stormwater.” *Journal of Hydrology* 350 (1): 114–125. https://doi.org/10.1016/j.jhydrol.2007.11.006.

Sutton, Richard S., and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* 2nd ed. The MIT Press.

Tian, Wenchong, Zhenliang Liao, Guozheng Zhi, Zhiyu Zhang, and Xuan Wang. 2022. “Combined Sewer Overflow and Flooding Mitigation Through a Reliable Real-Time Control Based on Multi-Reinforcement Learning and Model Predictive Control.” *Water Resources Research* 58: e2021WR030703. https://doi.org/10.1029/2021WR030703.

Wang, Cheng, Benjamin D. Bowes, Peter A. Beling, and Jonathan L. Goodall. 2021. “Reinforcement Learning for Flooding Mitigation in Complex Stormwater Systems during Large Storms.” In *IEEE EUROCON 2021 - 19th International Conference on Smart Technologies*, July, 274–279.

Watkins, Christopher John Cornish Hellaby. 1989. “Learning from Delayed Rewards.” PhD diss., King’s College, Oxford.

Wenger, Seth J., Allison H. Roy, C. Rhett Jackson, Emily S. Bernhardt, Timothy L. Carter, Solange Filoso, Catherine A. Gibson, et al. 2009. “Twenty-six key research questions in urban stream ecology: an assessment of the state of the science.” *Journal of the North American Benthological Society* 28: 1080–1098.

Yin, Zeda, Yasaman Saadati, Arturo S. Leon, M. Hadi Amini, Linlong Bian, and Beichao Hu. 2024. “Forecasting and Optimization for Minimizing Combined Sewer Overflows Using Machine Learning Frameworks and Its Inversion Techniques.” *Journal of Hydrology* 628: 130515. https://doi.org/10.1016/j.jhydrol.2023.130515.

Zhang, Zhiyu, Wenchong Tian, and Zhenliang Liao. 2023. “Towards Coordinated and Robust Real-Time Control: A Decentralized Approach for Combined Sewer Overflow and Urban Flooding Reduction Based on Multi-Agent Reinforcement Learning.” *Water Research* 229: 119498. https://doi.org/10.1016/j.watres.2022.119498.

Formal Specification and Testing for Reinforcement Learning

MAHSA VARSHOSAZ, IT University of Copenhagen, Denmark
MOHSEN GHAFFARI, IT University of Copenhagen, Denmark
EINAR BROCH JOHNSEN, University of Oslo, Norway
ANDRZEJ WĄSOWSKI, IT University of Copenhagen, Denmark

The development process for reinforcement learning applications is still exploratory rather than systematic. This exploratory nature reduces reuse of specifications between applications and increases the chances of introducing programming errors. This paper takes a step towards systematizing the development of reinforcement learning applications. We introduce a formal specification of reinforcement learning problems and algorithms, with a particular focus on temporal difference methods and their definitions in backup diagrams. We further develop a test harness for a large class of reinforcement learning applications based on temporal difference learning, including SARSA and Q-learning. The entire development is rooted in functional programming methods; starting with pure specifications and denotational semantics, ending with property-based testing and using compositional interpreters for a domain-specific term language as a test oracle for concrete implementations. We demonstrate the usefulness of this testing method on a number of examples, and evaluate with mutation testing. We show that our test suite is effective in killing mutants (90% mutants killed for 75% of subject agents). More importantly, almost half of all mutants are killed by generic write-once-use-everywhere tests that apply to *any* reinforcement learning problem modeled using our library, without any additional effort from the programmer.

CCS Concepts: • **Theory of computation → Program specifications**; • **Software and its engineering → Software testing and debugging**.

Additional Key Words and Phrases: specification-based testing, reinforcement learning, Scala

ACM Reference Format:
Mahsa Varshosaz, Mohsen Ghaffari, Einar Broch Johnsen, and Andrzej Wąsowski. 2023. Formal Specification and Testing for Reinforcement Learning. *Proc. ACM Program. Lang.* 7, ICFP, Article 193 (August 2023), 34 pages. https://doi.org/10.1145/3607835

1 INTRODUCTION

"Applications of reinforcement learning are still far from routine and typically require as much art as science" (Sutton and Barto [2018]). The development process for reinforcement learning (RL) applications is exploratory rather than systematic, which reduces reuse between applications and increases the chances of introducing errors into particular implementations, lowering trustworthiness and effectiveness. This is especially important in areas where reinforcement learning is used to control physical devices (e.g., embedded or cyber-physical systems, and robots). Techniques and tools for systematic quality assurance of reinforcement learning applications are rare in the field.

Authors' addresses: Mahsa Varshosaz, IT University of Copenhagen, Copenhagen, Denmark, mahv@itu.dk; Mohsen Ghaffari, IT University of Copenhagen, Copenhagen, Denmark, mohg@itu.dk; Einar Broch Johnsen, University of Oslo, Oslo, Norway, einarj@ifi.uio.no; Andrzej Wąsowski, IT University of Copenhagen, Copenhagen, Denmark, wasowski@itu.dk.

2475-1421/2023/8-ART193
https://doi.org/10.1145/3607835

Reinforcement learning is a method of machine learning, in which an agent experiments interactively with an environment, receiving rewards for these interactions. The most popular learning algorithms are *model-free*, as they do not need a complete model of the environment but learn from sampling executions. The goal of a learning algorithm is to find an optimal behavior policy for the agent which maximizes long-term expected reward. Reinforcement learning research often focuses on evaluating the quality of obtained policies. However, the results of even the most carefully designed evaluation experiments have little value, unless the evaluated methods are implemented correctly. Our long-term goal is to make the development and test of reinforcement learning applications more systematic and the outcomes more trustworthy by enabling easily accessible automatic testing. In this paper, we take a step towards a direct formal specification of correctness for reinforcement learning problems (agents) and the *learning algorithms* themselves, as opposed to the policies that they output.

We focus on *temporal difference* (TD) reinforcement learning methods [Sutton 1988], a large and well-established class of model-free methods. The majority of reinforcement-learning algorithms in use are TD algorithms. The TD algorithms update an estimate of a state action value function using a number of sampling steps and an estimate (bootstrapping). The *temporal difference* in the name refers to the fact that an update for a state-and-action expected reward is performed not immediately but after one or more execution steps. As we focus on the correctness of the algorithm and the agent, we settle for simple representations of value estimation—discrete tables. However, the core structure of our specification appears relevant for approximating methods.

We perform a domain analysis of the TD algorithms domain, leading to a formal specification. We characterize commonalities and differences between different reinforcement learning problems and between TD algorithms. We pay special attention to the update step in the algorithms, commonly described using so-called *backup diagrams* [Sutton 1988]. Our domain analysis formally defines *bdl*, or a *back-up diagram language*—a compositional, domain-specific language for describing updates in reinforcement learning. An interpreter for *bdl*, formalized in a denotational style, serves as a specification of correctness for updates in individual TD algorithms. A compositional denotational definition of this interpreter is what allows us to characterize many TD algorithms at once.

The obtained specification may be used for testing and verification (after embedding in a suitable formal system). We use testing to demonstrate and evaluate it here. We implement Q-Learning, SARSA, and Expected SARSA along with a number of case studies extracted from text books and papers. We add a number of tests derived from the formal modeling of agents and implement the *bdl* interpreter to serve as an oracle in property-based tests in the style of Quickcheck [Claessen and Hughes 2000]. Using an interpreter as an oracle for testing a concrete implementation is a technique well-known to compiler engineers, but rarely used outside this community. Our applicative strongly-typed implementation of these algorithms in Scala 3 is concise and traceable to the formal definitions in the paper. All code and tests are available online.

The test harness can be imposed on different algorithms and can be extended with properties specific for an agent. We try it on three algorithms and nine agents. We experience that the framework can carry the implementation of various kinds of problems, and that the cost of customizing the tests (especially the cost of providing custom generators for property-based tests of agents) is not high. Furthermore, an experiment based on mutation testing [DeMillo et al. 1978] demonstrates that the test harness kills a vast majority of programs with randomly injected faults. Crucially, about half of all mutants for each problem are killed by *generic tests* that are written once and reused for all new agents, just by invocation. No additional effort from the programmers implementing new agents is required.

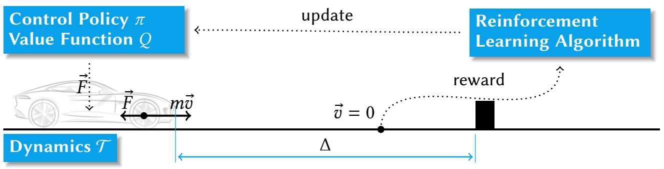


Fig. 1. Example: A car moves with velocity $\vec{v}$ towards a fixed obstacle at distance Δ, learning how to brake. The control policy chooses a deceleration with which to brake. The agent receives a reward based on the location where stopped and updates the policy

We continue with a motivating example in Sect. 2. The technical contributions are:

- *A formal specification of key elements of reinforcement learning problems and TD algorithms*, including a detailed, self-contained, fully-formal definition of the update step (Sect. 4). The specification is probabilistic, applicative, compositional, and executable. We are not aware of any comparably detailed formal specification of reinforcement learning to date.
- *A translation of the above specification into a test harness* (Sect. 5), implemented (Sect. 6) following the paradigm of property-based testing. The test harness includes a general interpreter of *bdl* terms—formal models of updates for TD learning algorithms. It is used as an oracle. To our best knowledge this is the first property-based test suite for reinforcement learning.
- *An evaluation of the test harness* on SARSA and Expected SARSA for several agents using mutation testing (Sect. 7). The evaluation shows that the test harness is able to kill half of the generated mutants "for free," i.e. without any customization of tests for new agents.

We discuss limitations (Sect. 8) and related work (Sect. 9), and conclude in Sect. 10.

2 MOTIVATING EXAMPLE: AN AGENT AND A LEARNING ALGORITHM

Consider a moving car that needs to stop before reaching a static obstacle (Fig. 1). We want to apply RL techniques to this problem and teach the car's controller to avoid a collision with the obstacle. Hence, we formulate a reinforcement learning problem to which we can apply a RL algorithm.

In our problem formulation, the car controller (the agent) interacts with the road and the obstacle (the dynamic environment) to learn the control policy from experience. The states describe the possible positions and velocities of the car, and the actions the possible changes in velocity. The transition function then computes the effect of an action on a state. The goal is to avoid collision with the obstacle. We need to formulate a reward function reflecting this goal! Clearly, hitting the obstacle should give a negative reward; for other states and actions we might want to penalize unnecessarily sharp braking actions. How can we check that our transition and reward functions fit our intended reinforcement learning problem?

In this paper, we give a formal definition for a class of reinforcement learning problems, for which we implement a test harness using property-based testing. A good test suite that ensures the basic properties of RL problems, can significantly accelerate the task definition process. In fact, there are many potential pitfalls in defining such reinforcement learning problems. Some tests reflect implicit assumptions about the class of RL problems whereas others are specific to a particular RL problem. An example of the *generic* case is a test expressing that a transition from a so-called observable state (i.e., a state known to the reinforcement learning algorithm) will lead to another observable state. Such domain properties reflect real development problems in debugging RL implementations, based on our experience and on discussions with RL developers, and testing for simple properties helps eliminate the main bugs fast. An example of a *specific* problem property is, for our braking car example, that the car cannot move backwards by braking. Problem-specific tests are needed to check that the transition and reward functions properly capture the characteristics of the problem domain.

Once the problem has been defined, we can apply a reinforcement learning algorithm to it and obtain a policy. The car should learn to halt before the obstacle and to avoid unnecessary sharp braking. Throughout the learning process, the car repeatedly starts from different states (i.e., positions and velocities), selects different actions (i.e., values of deceleration) and observes the resulting reward and change of state. Sometimes the car stops before the obstacle, sometimes it crashes. A reinforcement learning algorithm estimates the long-term effect of taking an action in a specific state. The estimate is updated by considering new rewards observed from the agent's interactions with the environment. This update step constitutes the core of such RL algorithms; the details of how and when to perform the update vary depending on the specific algorithm.

One example of a RL algorithm is SARSA (Fig. 2). The algorithm considers episodes that are sequences of states S_i and actions A_i that can be selected in those states, leading to a final state. A value function Q assigns a long-term reward estimation to state–action pairs. SARSA is an on-policy learning algorithm because the policy depends on Q when it selects an action in a given state. The rate by which learning affects Q is given by α, while γ defines the long-term reward discount.

Even though the standard presentation of SARSA (Fig. 2) makes an impression of an imperative program, a programmer would quickly notice that one cannot implement it without much additional knowledge. For instance, the meaning of "observe" and "reward" in Line 3 requires understanding what is an agent, what operations it supports, and with what semantics. The link between ε and π in l. 4 is not explicit. The policy π, mapping states to optimal actions, needs knowing in which state to select A_{t+1}. Also value function Q needs to be involved, as it defines which action is the best (so π and Q are dependent). All these problems could be fixed in Fig. 2 by refining the pseudo code, but they get difficult to fix for more advanced temporal difference algorithms, for which the update equation in Line 5 considers multiple states and actions at different times. The monolithic presentation no longer scales. Without a formal compositional specification, it is not only hard to implement the algorithms, but even harder to say whether the implementation is correct. Finally, it is non-obvious that the program is probabilistic and that several identifiers and expressions represent random variables.

As TD learning algorithms follow a common structure, we can capture this structure in a language and build a probabilistic interpreter for it to serve as a specification and an oracle. A run of this interpreter given a term defines the update of a particular TD algorithm. The term describing the SARSA update (l. 5) is $\mathsf{sample}^{\gamma}\ \mathsf{Update}^{\alpha}\ \mathsf{sample}$, and a denotational semantics precisely defines its meaning.

In this paper, we formalise TD learning algorithms and build a test harness for them. We first define properties that should apply generally to the considered class of TD algorithms. An example of such a property, is that the action selected in any state is distributed according to the specified policy and Q-table, which we check with a statistical test. Similarly, we test the update of the Q-table, using our interpreter as an oracle. An erroneous update will typically not crash, but produce a wrong value; a broken update step might go unnoticed for a long time, only manifesting itself in subpar results from the learning process. Our test suite facilitates the detection of such subpar behavior.

3 PRELIMINARIES

We follow the notation of Sutton and Barto [2018] and conventions of lambda-calculus; e.g., we omit parentheses around function arguments and write $f x$ instead of $f(x)$. Sets and types are boldfaced.

A probability distribution over a finite set $\mathbf{A}$ is a function $\mathrm{Pr}_{\mathbf{A}} \in \mathbf{A} \to [0;1]$ that assigns some probability mass to each element $a \in \mathbf{A}$, and that satisfies the usual axioms of probability. We write pmf $\mathbf{A}$ for the set of all *probability mass functions* (or distributions) over the set $\mathbf{A}$. If $\mathbf{A}$ is continuous, we write pdf $\mathbf{A}$ for the set of all *probability density functions* $\mathrm{Pr}_{\mathbf{A}} \in \mathbf{A} \to \mathbb{R}_{+} \cup \{0\}$ over $\mathbf{A}$. Since the co-domains of functions in this paper will often be such probability mass or density functions, we

```
1: Initialize $S_t$, select action $A_t$ using the current policy $\pi$ ($\varepsilon$-greedy)
2: **while** $S_t$ is not final **do**
3:     Execute $A_t$, observe the next state $S_{t+1}$ and reward $R_{t+1}$
4:     Select next action $A_{t+1}$ using the $\varepsilon$-greedy policy derived from $Q$
5:     $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
6:     $S_t \leftarrow S_{t+1}, A_t \leftarrow A_{t+1}$
```

Fig. 2. A standard impure presentation of the popular SARSA algorithm following the style adopted in the literature [Sutton and Barto 2018]. The pseudocode shows one learning episode, until the agent reaches a final state. Normally a large number of episodes is run, updating the same global Q-table. The most characteristic part of SARSA is the update rule in Line 5. It uses the current value $Q(S_{t+1}, A_{t+1})$ as an approximation of the future long-run reward. The action A_{t+1} is selected probabilistically according to the current policy.

parenthesize expressions that return distributions, to emphasize this; e.g., if $f\ x\ y$ returns a distribution over $Z \ni z$, we write $(f\ x\ y)\ z$ for the probability (or density) assigned to z by the function $f\ x\ y$.

A probability mass function $\mathrm{Pr}_{\mathbf{A}}$ and a function $f : \mathbf{A} \longrightarrow \text{pmf } \mathbf{B}$ induce a probability distribution over the set $\mathbf{B}$ following the chain rule (by picking a value $a \in \mathbf{A}$ according to $\mathrm{Pr}_{\mathbf{A}}$, applying f to a, then picking a value in $\mathbf{B}$ according to the obtained distribution).

$$(\mathrm{Pr}_{\mathbf{A}} \mathbin{>\!\!\!>\!\!=} f) \in \text{pmf } \mathbf{B} \qquad (\mathrm{Pr}_{\mathbf{A}} \mathbin{>\!\!\!>\!\!=} f)\ b = \sum_{a \in \mathbf{A}} \mathrm{Pr}_{\mathbf{A}}\ a \cdot (f\ a)\ b \tag{1}$$

Some readers might find it useful to know that the left-associative composition operator $\mathbin{>\!\!\!>\!\!=}$ between a distribution and an into-distribution function is an instance of a monadic bind [Ramsey and Pfeffer 2002]. If $\mathrm{Pr}_{\mathbf{A}}$ is a pdf and $f : \mathbf{A} \longrightarrow \text{pdf } \mathbf{B}$ is measurable, we similarly define:

$$(\mathrm{Pr}_{\mathbf{A}} \mathbin{>\!\!\!>\!\!=} f) \in \text{pdf } \mathbf{B} \qquad (\mathrm{Pr}_{\mathbf{A}} \mathbin{>\!\!\!>\!\!=} f)\ b = \int_{a \in \mathbf{A}} \mathrm{Pr}_{\mathbf{A}}\ a \cdot (f\ a)\ b \tag{2}$$

The integral (technically, the measure) is used to formally describe the behavior of a RL system. Our implementation replaces the computation of probability measures with estimation by sampling (so the source domains are also discrete in any given run).

Given two functions $f \in \mathbf{A} \longrightarrow \text{pmf } \mathbf{B}$ and $g \in \mathbf{B} \longrightarrow \text{pmf } \mathbf{C}$ their Kleisli composition $f \mathbin{>\!\!\!>\!\!\!>} g$ is the unique function from $\mathbf{A} \longrightarrow \text{pmf } \mathbf{C}$ given below, Eq. (3). The Kleisli composition "flattens" (multiplies) nested probability distributions during function composition.

$$f \mathbin{>\!\!\!>\!\!\!>} g = \lambda a. f\, a \mathbin{>\!\!\!>\!\!=} g \tag{3}$$

Given a constant $x \in \mathbf{X}$, we write $\mathrm{Det}\ x$ for the unique deterministic distribution such that $(\mathrm{Det}\ x)\ x' = 1$ iff $x = x'$ and zero otherwise (the Dirac distribution).

From the reinforcement learning and testing perspective, it does not matter whether one performs statistical tests following the Bayesian or the frequentist tradition [Gelman and Shalizi 2013]. Somewhat arbitrarily we make the Bayesian choice [Gelman et al. 2013; Kruschke 2014]. Given a prior distribution with parameter θ over a random variable X, and given a number of observations δ of values drawn from X, a Bayesian analysis rests on computing or estimating a posterior belief distribution on θ given the observations. One typically uses Bayes' theorem: $\Pr(\theta \mid \delta) \approx \Pr(\delta \mid \theta) \cdot \Pr(\theta)$, where $\Pr(\delta \mid \theta)$ is known as the likelihood function. The likelihood is typically easy to formulate as it is generative: for each value of θ it can give a probability of generating δ. In the context of testing probabilistic programs, the likelihood will be given by the program semantics. Once the posterior distribution is established, one queries it for probabilities of relevant facts: for instance what is the belief that $\theta \geq 0.9$. In general, the posterior can be an arbitrarily complex function, but if the prior is specifically selected for the likelihood (a *conjugate* prior), a kind of closure is obtained: the prior and the posterior have the same format. For conjugate priors, often an easy syntactic rule exists for updating the prior given the observations. We exploit such in this paper for Bernoulli and Gaussian priors to write simple, but computationally efficient tests.

4 A FORMAL SPECIFICATION OF REINFORCEMENT LEARNING

In reinforcement learning, an agent interacts with an environment in order to learn its behavior by trial-and-error. The agent's aim is to find a policy that maximizes a measure of reward [Kaelbling et al. 1996]. A *reinforcement learning problem* defines the agent's state space, dynamics (transition relation), and the rewards. The problem is handed over to a *learning algorithm* to search for the optimal policy, given the problem. The first key technical development of this paper is to formalize both the RL problems and the RL algorithms in order to allow reason and testing about the correctness of learning processes.

4.1 Reinforcement Learning Problems (Agents)

Definition 4.1. A *Reinforcement Learning Problem* is represented by a tuple ($\overline{\mathbf{State}}$, $\overline{\mathbf{State}}_0$, $\mathbf{Action}$, $\mathbf{State}, O, \mathcal{T}, \mathcal{R}, \mathcal{F}$) where:

- $\mathbf{r_1}$: $\overline{\mathbf{State}}$ is a possibly infinite set of states of the environment and the agent combined,
- $\mathbf{r_2}$: $\overline{\mathbf{State}}_0 \in \text{pdf}\,\overline{\mathbf{State}}$ is a density function defining probability for initial states,
- $\mathbf{r_3}$: $\mathbf{Action}$ is a finite set of actions that an agent can take,
- $\mathbf{r_4}$: $\mathbf{State}$ is a finite set of observable states,
- $\mathbf{r_5}$: $O \in \overline{\mathbf{State}} \longrightarrow \mathbf{State}$ is a total observation function,
- $\mathbf{r_6}$: $\mathcal{T} \in \overline{\mathbf{State}} \longrightarrow \mathbf{Action} \longrightarrow \text{pdf}\,\overline{\mathbf{State}}$ is the transition probability function,
- $\mathbf{r_7}$: $\mathcal{R} \in \overline{\mathbf{State}} \longrightarrow \mathbf{Action} \longrightarrow \mathbb{R}$ is the reward function, and
- $\mathbf{r_8}$: $\mathcal{F} \in \mathbf{State} \longrightarrow \{0, 1\}$ is a predicate defining which observable states are final for a training episode. Initial states are not final, i.e., if $\overline{\mathbf{State}}_0\ (\overline{S}) > 0$ then not $\mathcal{F}\ (O\ \overline{S})$.

The agent perceives the world through a discrete observation function O ($\mathbf{r_5}$), mapping the possibly continuous state space $\overline{\mathbf{State}}$ of the environment ($\mathbf{r_1}$) to a finite state space $\mathbf{State}$ of the learning algorithm ($\mathbf{r_4}$). For the rest of the paper, it is a useful to remember that the identifiers with a line over refer to the actual state space of the environment, while those without refer to the state space abstraction that the agent can observe (the observable state space). The reward function $\mathcal{R}\ \overline{S}\ A$ defines the reward received by the agent ($\mathbf{r_7}$) when arriving at state $\overline{S}$ after taking the action A. The transition function $\mathcal{T}$ defines a distribution of successor states $\mathcal{T}\ \overline{S}\ A$ given a source state $\overline{S}$ and an action A, see $\mathbf{r_6}$. The function $\mathcal{T}$ captures the stochastic environment behavior (e.g., noise) by returning a distribution. Its values are density functions, to account for the continuous state of the environment. Generally, $\mathcal{T}$ defines a partially observable Markov Decision Process (MDP) with rewards and the composition $\mathcal{T} \ggg O$ projects it onto a finite state MDP.

Given a reinforcement learning problem, a *run* is a sequence $\overline{S}_0\, A_0\, R_1 \cdots \overline{S}_{t-1} A_{t-1} R_t \cdots$, where all transitions have non-zero probability: $(\mathcal{T}\ \overline{S}_i\ A_i)\ \overline{S}_{i+1} > 0$ and $R_{i+1} = \mathcal{R}\ \overline{S}_{i+1}\ A_i$. Each state $\overline{S}_i$ in a run marks a discrete time *epoch* in which the system performs the action A_i and receives a reward R_{i+1}.

Definition 4.2. A RL problem is *episodic* iff every run from an initial state eventually reaches some final state $\overline{S}$, so $\mathcal{F}\ (O\ \overline{S}) = 1$, see $\mathbf{r_8}$ in Def. 4.1. Otherwise the problem is *non-episodic.*

Example 4.3. To exemplify the definitions, we instantiate Def. 4.1 for the car example of Fig. 1.

- $\mathbf{r_1}$: The set of states of the environment is $\overline{\mathbf{State}} = [0.0, 15.0] \times [0.0, 10.0]$, where $[0.0, 15.0]$ is the interval of possible positions and $[0.0, 10.0]$ is the interval of possible velocities. For a state $\overline{S} \in \overline{\mathbf{State}}$, we write $\overline{S}.p$ for its position and $\overline{S}.v$ for its velocity. See Fig. 3.

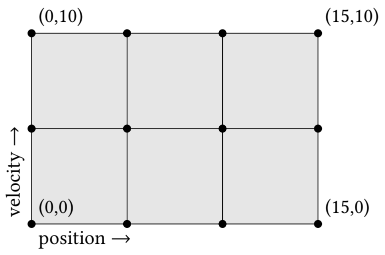


Fig. 3. The state space for the breaking car example of Fig. 1. Each state is a pair of numbers representing the position of the car and its forward velocity. The environment state space is the entire gray rectangle. The black bullet points represent the observable states. The observation function O maps the interior of each of the small squares (including its bottom and left edges) to the observable state in its bottom-left corner.

- $\mathbf{r_2}$: The car can start in any state with equal probability, so $\overline{\mathbf{State}}_0$ is a uniform distribution over the gray rectangle in Fig. 3: $\overline{\mathbf{State}}_0 = \text{Uniform}\ (\overline{\mathbf{State}})$.
- $\mathbf{r_3}$: The set of actions is $\mathbf{Action} = \{-10.0, -5.0, -2.5, -0.5, -0.05, -0.01, -0.001\}$ represents possible deceleration (negative acceleration) rates corresponding to increasingly gentle braking.
- $\mathbf{r_4}$: The set of observable states is $\mathbf{State} = \{0.0, 5.0, 10.0, 15.0\} \times \{0.0, 5.0, 10.0\}$, so even though the environment admits infinitely many position–velocity combinations, the agent can only observe 12 pairs, represented as black points in Fig. 3.
- $\mathbf{r_5}$: The observation function discretizes each car state by bringing each of its components to the largest multiple of five smaller than the value (in Fig. 3 each square to is discretized to its bottom-left corner):

$$O\ \overline{S} = S \quad \text{where} \quad S.p = 5 \cdot \left\lfloor \overline{S}.p/5 \right\rfloor \text{ and } S.v = 5 \cdot \left\lfloor \overline{S}.v/5 \right\rfloor .$$

- $\mathbf{r_6}$: The transition function $\mathcal{T}$ selects the next state given a predecessor state $\overline{S}_t$ and an action A_t. In this example, the successor state $\overline{S}_{t+1}$ is calculated deterministically based on the car dynamics (assuming that the car moves for a time step δ):

$$\overline{S}_{t+1}.p = \overline{S}_t.p + \overline{S}_t.v\delta + A_t\delta^2/\,2 \quad \text{and} \quad \overline{S}_{t+1}.v = \overline{S}_t.v + A_t\delta\ .$$

  Since the successor state is computed deterministically we get $\mathcal{T}\ \overline{S}_t\ A_t = \text{Det}\ (\overline{S}_{t+1})$, a Dirac distribution over successor states.
- $\mathbf{r_7}$: The reward for reaching a state $\overline{S}_{t+1}$ after taking the action A_t is

$$\mathcal{R}\ \overline{S}_{t+1}\ A_t = -100 \quad \text{if} \quad \overline{S}_{t+1}.p \geq 10 \quad \text{and} \quad \mathcal{R}\ \overline{S}_{t+1}\ A_t = A_t \quad \text{otherwise.}$$

  The model assumes that the obstacle is found at position 10 (the rightmost edge in Fig. 3). The first case penalizes a crash. The second case (no crash) says that a sharper deceleration results in a lower reward; the reward is proportional to the action value in this case, to penalize violently abrupt braking.
- $\mathbf{r_8}$: The braking car problem is episodic. The car is in a final state if either it has come to a full stop or it has crashed, so $\mathcal{F}\ S$ is true if and only if $S.v = 0.0 \,||\, S.p \geq 10$. □

A *policy* for an agent defines its behavior at any state. It maps observed states to actions that an agent takes in those states. Policies are derived from richer *value functions* estimating the ultimate reward of an execution started by each action in a state, $Q \in \mathbf{State} \times \mathbf{Action} \to \mathbb{R}$. We let $\mathbf{Q}$ denote the set of all value functions. A policy then becomes a probability function that, given a value function, represents the distribution of plausible actions in each state.

$$\pi \in \mathbf{Q} \to \mathbf{State} \to \text{pmf}\ \mathbf{Action} \tag{4}$$

A greedy policy returns the action maximizing expected reward for a given state deterministically (with probability one). A greedy policy is not effective for learning, as it may follow globally sub-optimal actions, especially at an early stage of learning. A popular policy in RL algorithms is the ε-greedy policy that forces the agent to try random actions with small probability $\varepsilon \in [0,1]$ and use the locally best action otherwise:

$$(\pi\ Q_t\ S_t)\ A_t = \begin{cases} 1 - \varepsilon + \varepsilon \cdot |\mathbf{Action}|^{-1} & A_t = \arg\max_A Q_t\ (S_t,\ A) \\ \varepsilon \cdot |\mathbf{Action}|^{-1} & \text{otherwise} \end{cases} \tag{5}$$

Solving a RL problem means finding the policy that maximizes the expected reward over many episodes. For non-episodic tasks a discounted expected reward over an infinite run is maximized.

4.2 Temporal Difference Learning Algorithms

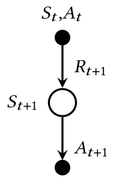
We will now summarize the update methods of RL algorithms following the style adopted in the literature of the field. We will rectify the shortcomings of this style in later sections.

In implementations, the value function is often represented as a table with a value for each state-action pair, a so-called Q-table. (In deep reinforcement learning, it is approximated using a neural network, but we are not concerned with this here, as we are formalising the core structure of the algorithms.) Learning happens by experimentation: an agent tries an action, receives a reward, and updates the Q-table entry for that action. How precisely an update is made is the very essence of each reinforcement learning algorithm. A large class subsuming most common designs are the *temporal-difference* (TD) algorithms, which update the value function after accumulating the reward over a (possibly singleton) sequence of actions. Given a state S_t and action A_t, the general update rule for a TD prediction method is (written as an imperative assignment, as commonly practiced in the literature of the field):

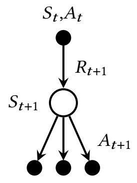
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \cdot (G_t - Q(S_t, A_t))\ . \tag{6}$$

The new state-action value estimation is the old estimate $Q(S_t, A_t)$ corrected by its error against the new estimate G_t, discounted by a coefficient α. The *return* G_t is the newly obtained expected reward value. Its calculation varies, depending on the specific TD method used. The *TD error* value $\alpha(G_t - Q(S_t, A_t))$ represents the difference between the new estimate and the old estimate of the expected reward, where $\alpha \in [0,1]$ is the learning rate defining the trade-off between learning and remembering. If $\alpha = 1$ then the old estimate is entirely forgotten and the new estimate is adopted. If $\alpha = 0$ the new estimate is ignored: no learning happens. Other values of α control the speed with which the new experiences influence the current policy. Concrete update rules for different TD algorithms are instances of the general rule given in Eq. (6) with different returns G_t.

Example 4.4. The popular SARSA algorithm [Rummery and Niranjan 1994] is a TD algorithm. The pseudocode is shown in Fig. 2 (Sect. 2). Given a Q-table and the learning rate α, it performs the following steps for a prescribed number of episodes. The second but last line, performs the update. The term R_{t+1}+$\gamma Q(S_{t+1}, A_{t+1})$ defines the return G_t in this case, where $\gamma \in [0,1]$ is the so-called discount factor, weighing immediate rewards vs future rewards. □



Reinforcement learning researchers often visualize the calculation of the return in a so called backup diagram. The diagram for SARSA is shown to the right. The top arrow means that the reward R_{t+1} is obtained first by sampling the environment using action A_t from state S_t. The second arrow means that the action A_{t+1} is sampled from the policy and its current value estimate in $Q(S_{t+1}, A_{t+1})$ is used to compute the return and update $Q(S_t, A_t)$. In general, a backup diagram is a stylized list of steps, where all but last interact with the environment, while the last one uses a prior estimate. The proliferation of backup diagrams in conference presentations, lectures, and teaching materials on reinforcement learning is testimony to the pivotal role of the update and its structure for understanding the RL algorithms.



Example 4.5. The Expected SARSA algorithm [Van Seijen et al. 2009] follows the same steps as SARSA (Example 4.4) just with a different update rule. It uses an estimate for all possible actions in the second step instead of the single best action chosen by the current policy π. The summation below computes an expectation over all actions in the next step discounted by factor γ. A backup diagram for the Expected SARSA update rule is shown to the right. Note that the second step differs from SARSA, reflecting the use of expectation instead of following the current policy greedily.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)) \ . \qquad \square$$

SARSA and Expected SARSA are *on-policy* algorithms as they improve the same policy that it is following to select subsequent actions (a decision making policy). In other algorithms, the updated policy and the followed policy might differ (off-policy learning).

Example 4.6. Q-learning is a popular and effective learning algorithm similar to SARSA [Watkins 1989]. Its update equation is the same as that of SARSA and Expected SARSA if the updated policy π is greedy ($\varepsilon = 0$) [De Asis et al. 2018]. Q-learning can be seen as an off-policy version of SARSA, that allows choosing actions using an ε-greedy policy ($\varepsilon > 0$), but performing an update using an estimation based on a greedy policy ($\varepsilon = 0$). However, in the scope of this paper, where we focus on semantics of the update procedure, both algorithms have the same specification. The difference between Q-learning and SARSA manifests when multiple updates are composed iteratively (see Sect. 8). $\square$

In all the above algorithms the update rule is similar and the main difference is in the calculation of the return. The update is based on a single step ($n = 1$), the received return, and an estimation of the value for the state-action pair for the next step in the Q-table. The entries in the table serve as a proxy for the remaining achievable long-term reward. All these algorithms can be generalized to perform an update after $n > 1$ steps, however the imperative non-compositional presentation gets very unwieldy. For this reason, we postpone the generalization to the formal compositional definition below.

4.3 Formalizing Temporal Difference Reinforcement Learning Algorithms

Despite the mathematical notation, the standard presentation of these algorithms in the literature remains relatively informal and hides many intricacies. Crucially, in this style, the update equations get more and more complex, eventually spanning several lines for the most complex methods in the classic textbook of Sutton and Barto [2018]. Even though the temporal difference algorithms share semantic similarity, the monolithic presentation makes it difficult to appreciate and exploit this similarity in formal reasoning and testing. This not only makes the reinforcement learning algorithms difficult to implement, but also hinders formal reasoning about and testing of desirable behaviors.

Let us point out a few challenges concretely. For example, R_{t+1} above is the immediate reward when an agent transitions to a state as a result of taking an action. This reward should be calculated using the model of the agent and the environment, so the functions $\mathcal{T}$, $\mathcal{O}$, and $\mathcal{R}$ in our formalization of the RL problems, but this is not visible in the standard presentation. Similarly, A_{t+1} denotes an action selected using the policy π, but the standard presentation does not relate A_{t+1} and π in any way. This relation is non-trivial; at runtime, the policy is not fixed, as it depends on the Q-table, which is regularly updated. In some reinforcement learning algorithms the policy also undergoes temporal changes, for instance, the exploration ratio ε is decreased over time. Which version of the policy π should select A_{t+1}? Similarly, the action selection depends on the current state but you cannot see this in the equation. Finally, while it appears that A_{t+1} is a concrete value, semantically it stands for a *random variable*, as the functions π and $\mathcal{T}$ are stochastic. This is the case for most of the elements in the algorithm, including states S_t and rewards R_t. An additional complication is that for TD algorithms looking beyond one step ahead, the update uses values from past time epochs (past iterations of the loop).

Obviously, a test or a correctness proof has to specify these time dependencies, values, and probability distributions explicitly and precisely. Instead of hiding meanings in variable names and indices, one needs a specification robust to alpha-renaming that makes dependencies explicit. For this reason we will now formalize the calculation of the *return* for a large class of TD algorithms. It is possible to handle many languages simultaneously, by defining a small term language to formally specify them, and defining the semantics of the update rule as an interpreter for this term language. Our term language for update rules is inspired by the backup diagrams. If RL researchers find them important and informative, they must convey important information from the domain expertise perspective. Unfortunately, the diagrams themselves are quite informal, not clearly compositional, and lack details. To address these issues we propose a textual syntax generated by the following grammar.

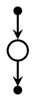
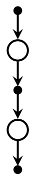
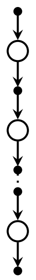
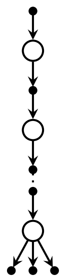
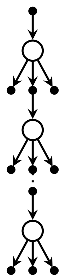
$$\begin{aligned} est &::= \mathsf{sample}^{\gamma} \mid \mathsf{expectation}^{\gamma} \\ bdl &::= est^{+}\ \mathsf{Update}^{\alpha}(\mathsf{sample} \mid \mathsf{expectation})\ . \end{aligned} \tag{7}$$

An *est* term, corresponding to a single segment in a backup diagram, represents an estimation step parametrized by a discount factor $\gamma \in [0, 1]$. It estimates state-action values by sampling (sample^{γ}) or by averaging ($\mathsf{expectation}^{\gamma}$) state-action-value estimates over all possible actions in a state. A *bdl* term, corresponding to an entire diagram, combines a non-empty sequence of estimation steps (est^{+}) with a final update—the last segment in each diagram. Update^{α} should be seen as infix operator parametrized by a learning rate $\alpha \in [0, 1]$, one for a sampling update and one for an expectation update.

Example 4.7. Table 1 lists backup diagrams (first column), their *bdl* abstract syntax (second column), and Sutton-and-Barto-style return calculations for five examples of temporal difference algorithms (third column); only the first one was discussed above. The *bdl* expression for 1-step SARSA is: $\mathsf{sample}^{\gamma}\ \mathsf{Update}^{\alpha}\ \mathsf{sample}$. As shown in Example 4.4, the update step includes updating the Q-value of a state-action pair (S_t, A_t) using the reward and the Q-value of the state-action pair (S_{t+1}, A_{t+1}) resulting from one time policy sampling. Similarly, for the 2-step SARSA, the diagrams represent two sampling steps composed with a sampling update: $\mathsf{sample}^{\gamma}\mathsf{sample}^{\gamma}\ \mathsf{Update}^{\alpha}\ \mathsf{sample}$. In contrast, the update in the last step of n-step Expected SARSA is calculated by taking an expectation of values for all possible actions instead of using the value for the sampled action. In the third column, $G_{t:t+n}$ and $\pi(a|s)$ are notations adopted by Sutton and Barto [2018] to denote, respectively, the n-step return, from time step t to t+n, and the probability of taking action a in state s following policy π. □

We can now specify the update step of the TD algorithms compositionally by giving formal semantics to elements of a *bdl* term. We map each basic element type to a function. The semantics of an entire backup diagram (and thus of an update) is given by a function composition. This style can be directly implemented in a functional programming language and used, for instance, for testing.

Table 1. Examples: A representation of updates in TD learning as a backup diagram, a *bdl* term, and a return calculation (G_t in Eq. (6)) after Sutton and Barto [2018]. The index T denotes the final time step in an episode.

Diagram	**BDL (abstract syntax)**	**Return ($G_{t:t+n}$) in Update formulae** [Sutton and Barto 2018]
	**1-step SARSA** sample$^\gamma$ Update$^\alpha$ sample	$R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1})$
	**2-step SARSA** sample$^\gamma$sample$^\gamma$ Update$^\alpha$ sample	$R_{t+1} + \gamma R_{t+2} + \gamma^2 Q_{t+1}(S_{t+2}, A_{t+2})$
	**n-step SARSA** (sample$^\gamma$)^n Update$^\alpha$ sample	$\begin{cases} R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} \\ \quad + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) & \text{if } n \geq 1 \wedge t < T - n \\ R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots + \\ \quad \gamma^{T-t-1} R_T & \text{otherwise} \end{cases}$
	**n-step expected SARSA** (sample$^\gamma$)^n Update$^\alpha$ expectation	$\begin{cases} R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} \\ \quad + \gamma^n \sum_a \pi(a \mid S_{t+1}) Q_{t+n-1}(S_{t+1}, a) & \text{if } t < T - n \\ R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots \\ \quad + \gamma^{T-t-1} R_T & \text{otherwise} \end{cases}$
	**n-step tree backup** (expectation$^\gamma$)^n Update$^\alpha$ expectation	$\begin{cases} R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a \mid S_{t+1}) Q_{t+n-1}(S_{t+1}, a) \\ \quad + \gamma \pi(A_{t+1} \mid S_{t+1}) G_{t+1:t+n} & \text{if } t < T \ \wedge n \geq 2 \\ R_{t+1} + \gamma \sum_a \pi(a \mid S_{t+1}) Q_t(S_{t+1}, a) & \text{if } n = 1 \\ R_T & \text{if } t = T - 1 \end{cases}$

Even more interestingly, an *interpreter* for *bdl* terms can be implemented in functional style and used for testing updates of direct implementations of TD algorithms.

An estimation step, say sample$^\gamma$, is based on a current table $Q_t \in \mathbf{Q}$, a source state $\overline{S}_t \in \overline{\mathbf{State}}$, an action $A_t \in \mathbf{Action}$ that the system decided to perform, a return value $G_{:t} \in \mathbb{R}$ from the preceding estimation steps, and a discount factor $\gamma_t \in [0, 1]$ for this and subsequent steps. It runs a step of the environment $\mathcal{T}$ and returns the reached successor state, the obtained reward value, and the action to perform in the next step. To make estimation steps compositional, the current discount factor, which can depend on the step, and the cumulative discounted reward up to this step (current return) are included as a part of the semantic domain. Since the environment may contain stochastic behavior and the action selection may be probabilistic (as we use ε-greedy policies), the semantics of an estimation step is actually a multivariate probability distribution over target states, actions, returns, and discount factors, resulting in the following semantic domain.

$$[\![\cdot]\!]_{est} \in \mathbf{Q} \to \overline{\mathbf{State}} \times \mathbf{Action} \times \mathbb{R} \times \mathbb{R} \to \text{pmf}\ (\overline{\mathbf{State}} \times \mathbf{Action} \times \mathbb{R} \times \mathbb{R}) \tag{8}$$

The semantic function for a sampling estimation step is:

$$\begin{aligned}
[\![\text{sample}^\gamma]\!]_{est}\ Q_t\ =\ & \lambda(\overline{S}_t, A_t, G_{:t}, \gamma_t).\ \mathcal{T}\ \overline{S}_t\ A_t\ \gg\!= && \text{(run the system in state } \overline{S}_t \text{ executing action } A_t\text{)}\\
& \lambda\overline{S}_{t+1}.\ \text{Det}\left(\mathcal{R}\ \overline{S}_{t+1}\ A_t\right)\ \gg\!= && \text{(reward } R_{t+1} \text{ for the action and the resulting state)}\\
& \lambda R_{t+1}.\ \text{Det}\left(O\ \overline{S}_{t+1}\right) \gg\!= && \text{(observe discrete state } S_{t+1} \text{ reached)}\\
& \lambda S_{t+1}.\ \pi\ Q_t\ S_{t+1} \gg\!= && \text{(select the next action } A_{t+1} \text{ following policy } \pi\text{)}\\
& \lambda A_{t+1}.\ \text{Det}\ (G_{:t} + \gamma_t R_{t+1})\ \gg\!= && \text{(discount accumulated return by } \gamma_t \text{ and add } R_{t+1}\text{)}\\
& \lambda G_{:t+1}.\ \text{Det}\left(\gamma_t \cdot \gamma\right)\ \gg\!= && \text{(accumulate the discount factor)}\\
& \lambda\gamma_{t+1}.\ \text{Det}\left(\overline{S}_{t+1}, A_{t+1}, G_{:t+1}, \gamma_{t+1}\right)
\end{aligned} \tag{9}$$

First, the step function is applied to the environment state $\overline{S}_t$ and action A_t. This results in a distribution over successor states, and we bind a successor to $\overline{S}_{t+1}$. The reward is computed deterministically for a pair of successor state $\overline{S}_{t+1}$ and the performed action A_t, the result bound to R_{t+1}. The observable target state S_{t+1} is obtained using a deterministic function O, and the next action is selected on policy π obtaining a non-trivial distribution again. At this point, we compute the return by adding the prior return $G_{:t}$ with the discounted reward. Finally, the discount factor is updated by this step's discount rate γ. In the last line all the four elements, i.e., next state $\overline{S}_{t+1}$, next action A_{t+1}, return $G_{:t+1}$, and discount factor γ_{t+1} to be used in the next step, are returned.

An advantage of this presentation is that it makes it explicit where the values of S_{t+1}, A_{t+1}, and R_{t+1} come from, which steps are deterministic, and which result in proper random variables (those not generated by Det). The first four function applications in Eq. (9), show how these values are obtained.

The semantics of an expectation step is defined similarly below. The differences between sampling and expectations are highlighted in blue—the expectation step uses an expected value of the estimated return instead of a sample reward when calculating the return.

$$\begin{aligned}
&[\![\text{expectation}^{\gamma}]\!]_{est}\, Q_t = \\
&\quad \lambda(\overline{S}_t, A_t, G_{:t}, \gamma_t).\ \mathcal{T}\ \overline{S}_t\ A_t \gg= && \text{(run the system step)}\\
&\quad \lambda\overline{S}_{t+1}.\ \mathrm{Det}\left(\mathcal{R}\ \overline{S}_{t+1}\ A_t\right) \gg= && \text{(calculate reward)}\\
&\quad \lambda R_{t+1}.\ \mathrm{Det}(O\ \overline{S}_{t+1}) \gg= && \text{(perform observation)}\\
&\quad \lambda S_{t+1}.\ \pi\ Q_t\ S_{t+1} \gg= && \text{(select next action)}\\
&\quad \lambda A_{t+1}.\ \mathrm{Det}\left(G_{:t} + \gamma_t\left[R_{t+1} + \sum_{A \neq A_{t+1}} (\pi\ Q_t\ S_{t+1})\ A \cdot Q_t(S_{t+1}, A)\right]\right) \gg= && \text{(update return with expected reward)}\\
&\quad \lambda G_{:t+1}.\ \mathrm{Det}\left(\gamma_t \cdot \gamma \cdot (\pi\ Q_t\ S_{t+1})\ A_{t+1}\right) \gg= && \text{(discount weighed by prob. of } A_{t+1}\text{)}\\
&\quad \lambda\gamma_{t+1}.\ \mathrm{Det}\left(\overline{S}_{t+1}, A_{t+1}, G_{:t+1}, \gamma_{t+1}\right)
\end{aligned} \tag{10}$$

The next return $G_{:t+1}$ in Eq. (10) is computed as a sum of current return $G_{:t}$, the discounted immediate reward R_{t+1} obtained by taking action A_t, and the expected return from alternative actions which are not selected as next action. This expectation is also discounted. The next discount factor is computed by multiplying the current discount factor with the constant discount factor for the step and the probability of taking the next action A_{t+1}. This calculation of discount factor is common for algorithms such as n-step tree backup [Sutton and Barto 2018].

The estimation steps can be composed into larger sequences of the same type.

$$[\![est_1 \cdots est_k]\!]_{est^+} \in \mathbf{Q} \to \overline{\textbf{State}} \times \textbf{Action} \times \mathbb{R} \times \mathbb{R} \to \text{pmf}\ (\overline{\textbf{State}} \times \textbf{Action} \times \mathbb{R} \times \mathbb{R}) \tag{11}$$

The composition is computed using the standard Kleisli composition:

$$\begin{aligned}
&[\![est_1 \cdots est_k]\!]_{est^+}\ Q_t\ = \\
&\quad \lambda(\overline{S}_t, A_t, G_{:t}, \gamma_t).\ [\![est_1]\!]_{est}\ (Q_t)\ (\overline{S}_t, A_t, G_{:t}, \gamma_t)\ \gg= && \text{(the 1. estimation step)}\\
&\quad \lambda(\overline{S}_{t+1}, A_{t+1}, G_{:t+1}, \gamma_{t+1}).\ [\![est_2]\!]_{est}\ (Q_t)\ (\overline{S}_{t+1}, A_{t+1}, G_{:t+1}, \gamma_{t+1})\ \gg= && \text{(composed with the 2. estimation)}\\
&\quad \vdots\\
&\quad \lambda(\overline{S}_{t+k-1}, A_{t+k-1}, G_{:t+k-1}, \gamma_{t+k-1}). && \text{(composed with the } k\text{th estimation)}\\
&\qquad [\![est_k]\!]_{est}\ (Q_t)\ (\overline{S}_{t+k-1}, A_{t+k-1}, G_{:t+k-1}, \gamma_{t+k-1})
\end{aligned} \tag{12}$$

The same can be stated compactly using the Kleisli composition of functions:

$$[\![est_1 \cdots est_k]\!]_{est^+}\ Q_t\ = [\![est_1]\!]_{est}\ Q_t\ \ggg [\![est_2]\!]_{est}\ Q_t\ \ggg \cdots \ggg [\![est_k]\!]_{est}\ Q_t\ . \tag{13}$$

From the functional programming perspective, the above may appear an obvious step, but we are not aware of a similar observation in the reinforcement learning literature, where the update procedures for each algorithm tend to be presented monolithically and implemented from scratch, obscuring the common structure of TD algorithms. This also means that in a correctness proof, it is difficult to generalize from properties of a single estimate step to the entire composed update.

Finally, we define the meaning of an update step, which performs a sequence of estimation steps, given an initial state and action, estimates the value of a final action in the sequence (in the right-hand-side operand) using the Q-table, and finally performs an update of the respective entry of the table. After an update the agent lands in a new target state and has chosen the next action (for on-policy algorithms). The type of the semantics reflects this: we start with a Q-table, a state and an action, and land (with some stochastic disturbance) in a new Q-table, state, and a subsequent action to execute.

$$[\![est^k \ \mathsf{Update}^\alpha \ (\mathsf{sample} \mid \mathsf{expectation})]\!]_{bdl} \in \mathbf{Q} \times \overline{\mathbf{State}} \times \mathbf{Action} \longrightarrow \mathrm{pmf}(\mathbf{Q} \times \overline{\mathbf{State}} \times \mathbf{Action}) \tag{14}$$

Let us define this function, for the case of the sampling update first.

$$\begin{aligned}
&[\![est^k \ \mathsf{Update}^\alpha \ \mathsf{sample}]\!]_{bdl} = \\
&\quad \lambda(Q_t, \overline{S}_t, A_t).\ [\![est^k]\!]_{est^+}\ (Q_t)\ (\overline{S}_t, A_t, 0, 1) \ \mathrel{\gg\!=} && \text{(execute the estimation steps)} \\
&\quad \lambda(\overline{S}_{t+k}, A_{t+k}, G_{t:t+k}, \gamma_{t+k}).\ \mathrm{Det}\left(O\ \overline{S}_t, O\ \overline{S}_{t+k}\right) \ \mathrel{\gg\!=} && \text{(observe initial \& final state)} \\
&\quad \lambda(S_t, S_{t+k}).\ \mathrm{Det}\,(G_{t:t+k} + \gamma_{t+k} Q_t(S_{t+k}, A_{t+k})) \ \mathrel{\gg\!=} && \text{(return based on } A_{t+k}) \\
&\quad \lambda G_{t:t+k+1}.\ \mathrm{Det}\,(Q_t\,[(S_t, A_t) \mapsto Q_t(S_t, A_t) + \alpha\,[G_{t:t+k+1} - Q_t(S_t, A_t)]]) \ \mathrel{\gg\!=} && \text{(update)} \\
&\quad \lambda Q_{t+1}.\ \mathrm{Det}\left(Q_{t+1}, \overline{S}_{t+k}, A_{t+k}\right) && (15)
\end{aligned}$$

The notation $Q_t[x \mapsto y]$ in the penultimate line means a table entry substitution. It denotes a new Q-table, obtained from Q_t by replacing the entry at position x to contain the value y. All other entries remain unchanged. The first line of the above semantics executes all the estimation steps of the algorithm (if any), then establishes what is the observable initial and target states, and uses the Q-table entry of the target state to update the entry for the initial state–action pair.

An update with an expectation estimate is defined similarly below in Eq. (16). The highlighted difference is in the computation of the return $G_{t:t+k+1}$. In Eq. (15), the discounted estimated Q_t value for the last pair of state and action (S_{t+k}, A_{t+k}) is added in the update. In Eq. (16), a discounted expectation over the Q_t values over all actions in the next state S_{t+k} is used instead. The discount factor uses the accumulated update and the local contribution (cf. Eqs. (9) and (10)).

$$\begin{aligned}
&[\![est^k \ \mathsf{Update}^\alpha \ \mathsf{expectation}]\!]_{bdl} = \\
&\quad \lambda(Q_t, \overline{S}_t, A_t).\ [\![est^k]\!]_{est^+}\ (Q_t)\ (\overline{S}_t, A_t, 0, 1) \ \mathrel{\gg\!=} && \text{(execute the estimation steps)} \\
&\quad \lambda(\overline{S}_{t+k}, A_{t+k}, G_{t:t+k}, \gamma_{t+k}).\ \mathrm{Det}\left(O\ \overline{S}_t, O\ \overline{S}_{t+k}\right) \ \mathrel{\gg\!=} && \text{(observe initial \& final state)} \\
&\quad \lambda(S_t, S_{t+k}).\ \mathrm{Det}\,(G_{t:t+k} + \gamma_{t+k} \textstyle\sum_A (\pi\ S_{t+k}\ A) Q_t(S_{t+k}, A)) \mathrel{\gg\!=} && \text{(expected return)} \\
&\quad \lambda G_{t:t+k+1}.\ \mathrm{Det}\,(Q_t\,[(S_t, A_t) \mapsto Q_t(S_t, A_t) + \alpha\,[G_{t:t+k+1} - Q_t(S_t, A_t)]]) \ \mathrel{\gg\!=} && \text{(update)} \\
&\quad \lambda Q_{t+1}.\ \mathrm{Det}\left(Q_{t+1}, \overline{S}_{t+k}, A_{t+k}\right) && (16)
\end{aligned}$$

The above functions provide compositional specifications for backup diagrams and for TD updates. As the Kleisli composition is associative, the above semantics is insensitive to the order of composing the functions. Since the above definitions are self-contained (all dependencies are explicit in the function constructions), they eliminate ambiguities seen in typical descriptions and can support automatic test of and formal reasoning about implementations. For example:

Theorem 4.8. *For a greedy policy π (with $\varepsilon = 0$) SARSA and Expected SARSA perform updates in the same manner, i.e.,* $[\![\mathsf{sample}^\gamma \ \mathsf{Update}^\alpha \ \mathsf{sample}]\!]_{bdl} = [\![\mathsf{sample}^\gamma \ \mathsf{Update}^\alpha \ \mathsf{expectation}]\!]_{bdl}$.

Proof. The action A_{t+k} in Eq. (15) is always bound to the action with the highest entry in the Q-table, as the policy π is greedy and it assigns probability 1 to the highest value action (cf. Eq. (9)). The return calculation in Eq. (15) returns the value in the Q-table discounted by γ_{t+k}. Similarly, as π is greedy, it is a Dirac distribution, so in the return calculation in Eq. (16), it will have value 1 for exactly one action A which is the action with the highest entry of $Q(S_{t+k}, A)$. Consequently, $A = A_{t+k}$ and the sum in Eq. (16) collapses to the same term as in Eq. (9). Ultimately, the proof rests on the fact that the mean and the mode of a Dirac-distributed random variable are the same. □

Recall that the update of Q-Learning, corresponds to an update of these algorithms if the policy π is greedy. Consequently, we can use the same specification to test the update in the Q-learning implementations. Without a formal specification it would be hard to make a similar statement.

5 SPECIFICATION-BASED TESTING OF REINFORCEMENT LEARNING

The presented properties can be used as a start of a formal verification project for an RL application. However, verification can quickly become intricate with specifics of the chosen learning algorithm. As we want to maintain a broad focus on many algorithms, while keeping the paper accessible and (relatively) short, we turn to automated testing for demonstration of our specification. We use the formal definitions in Sect. 4 to derive tests. Our long-term goal is to build a parameterized test harness and a method to write testable properties, to broadly lay the grounds for automating testing for RL.

The tests are organized in two main categories: the *tests of problem* definitions and the *tests of algorithms*. Problem tests check consistency properties of the agent and environment model, and of their interaction. We further distinguish between the *specific problem* tests, relevant only for a specific RL problem, and *generic tests* for problem definitions—the properties that should hold in general for all RL problem definitions. The algorithm tests check whether the learning algorithm behaves according to its design specification. In the following, we first focus on generic problem tests and then move to problem-specific and algorithm tests. Even though they are practically important, we devote less attention to the specific problem properties, as these are not reusable for larger groups of users. Their discussion quickly gets lost in the intricacies of a specific problem. In contrast, the generic problem properties and the tests of algorithms benefit more users directly, and can be pre-implemented in a library. We demonstrate deriving a selection of tests.

All tests below are described as abstract universal properties, basically logical statements. This way we minimize pollution by details of the programming language and style. We discuss how to concretize them as executable code in Sect. 6.

5.1 Testing Reinforcement Learning Problems (Agents)

Generic Problem Properties. These properties should hold for all implementations of RL problems. They are derived from the formal definition of a reinforcement learning problems in Sect. 4.1. We begin with the totality of the observation function O (Def. 4.1, requirement $\mathbf{r_5}$), a classic case of a *domain constraint.* The observation function O links the environment and the agent with the state space of the learning algorithm—as the algorithm only 'sees' and 'learns' about the observable states. The function O should be total in the sense that every system state should have a translation to an observable state; otherwise some system trajectories will lead to crashes or unexpected runs of the learning algorithm. Note that for most applications the domain constraints are not automatically enforced by the type system, as often only subsets of a type's values are legal (say the set of positive floats as opposed to all floats). Typically, these subsets are not tracked by the type system, thus it is natural to resort to testing. [1]

$$\forall \bar{S} \in \overline{\mathbf{State}}.\ O\ \bar{S} \in \mathbf{State}\ , \tag{17}$$

$$\forall \bar{S}_0 \in \overline{\mathbf{State}}.\ \overline{\mathbf{State}}_0\ \bar{S}_0 > 0 \rightarrow O\ \bar{S}_0 \in \mathbf{State}\ . \tag{18}$$

Equation (17) is testable if we can generate elements of $\overline{\mathbf{State}}$ and check membership in $\mathbf{State}$. In discrete RL, observable state spaces are finite and relatively small, so the latter is easily achieved by

[1]In a strongly typed programming language totality of functions can be reflected in types. Unfortunately, pragmatics often prevents it. The environment states and observations are generated by a physical system, or a simulator, which may be implemented in another programming language than the learning algorithms or the test harness. For instance, in some of our Scala-based projects we use Java implementations of the environment to leverage the existing infrastructure.

enumerating the observable states. However, in the case of approximate learning, specifically in continuous state spaces, employing a membership predicate becomes more practical and feasible. That is, finding a proper observable state space is not straightforward and requires approximating based on the environment model or problem. This requirement also applies to initial system states, which follows by combining $\mathbf{r_2}$ and $\mathbf{r_5}$. When testing it is useful to specify this requirement separately, to ensure that the property is tested on the initial states. See Eq. (18); recall that $\overline{\mathbf{State}}_0$ is a probability density function. The precondition that a density is positive ensures that the state is attainable.

We further require that the initial state is not final. Equation (19) tests the interaction of the pmf $\overline{\mathbf{State}}_0$ (requirement $\mathbf{r_2}$) and the predicate $\mathcal{F}$ (requirement $\mathbf{r_8}$):

$$\forall \overline{S}_0 \in \overline{\mathbf{State}}.\ \overline{\mathbf{State}}_0\ \overline{S}_0 > 0 \ \rightarrow\ \neg\mathcal{F}\ (\mathcal{O}\ \overline{S}_0)\ . \tag{19}$$

For the transition function $\mathcal{T}$, a domain constraint is obtained by combining requirements $\mathbf{r_5}$, $\mathbf{r_6}$ and $\mathbf{r_2}$ in Eq. (20). Together with the properties above, the totality of observation leads to a *closure* property—starting in an observable state, we end in an observable state:

$$\forall \overline{S}_t,\ \overline{S}_{t+1} \in \overline{\mathbf{State}}.\ \forall A \in \mathbf{Action}.\ (\mathcal{T}\ \overline{S}_t\ A_t)\ \overline{S}_{t+1} > 0 \ \rightarrow\ \mathcal{O}\ \overline{S}_{t+1} \in \mathbf{State}\ . \tag{20}$$

For episodic agents (Def. 4.2), we also test termination. Termination is difficult to establish by testing, but a random exploration strategy with a timeout is effective for simple RL problems. We implement it as a generic problem property with a timeout parameter (measured in discrete time epochs) in the model of episodic problems. This way different agents can be tested against different time horizons.

Some properties emerge from the composition of an algorithm and the problem definition, so one needs to involve both components (sections 4.1 and 4.3) in testing. One such example is a key property for reinforcement learning—*convergence.* Its violation can be caused by errors in the learning implementation, but also by overly liberal reward functions, a part of the problem definition. While convergence is not testable generally,[2] some special cases can be tested. In the context of a finite implementation we can test whether the accumulated reward values are representable in the range of double numbers (no overflow errors). An overflow of reward estimation during learning is effectively a sign of divergence. A test for overflow of rewards can be constructed by performing iterations of updates from various initial Q-tables. Given a learning algorithm u and a natural number of iterations n

$$\begin{aligned}
&\forall Q_t \in \mathbf{Q}.\ \forall S_t \in \mathbf{State}.\ \forall A_t \in \mathbf{Action}.\\
&\quad \forall Q \in \mathbf{Q}.\ [(\llbracket u \rrbracket_{bdl})^n (Q_t, S_t, A_t)]\ Q > 0\\
&\qquad \rightarrow \forall S \in \mathbf{State}.\ \forall A \in \mathbf{Action}.\ Q(S, A) \text{ is in the floating point range,}
\end{aligned} \tag{21}$$

where the n-times iteration composition of $\llbracket u \rrbracket_{bdl}$ is defined using Kleisli-composition (an iteration with monadic bind). The property states that if a Q-table Q is reachable via n iterations of the learning loop from an initial configuration (q_t, S_t, A_t), all values stored in its cells should be in the floating point range. The floating point range test can be implemented by tracking overflow exceptions or by checking the Q-table entries for `NaN` and `infinity`, depending on the programming language used. Also the range of the generated Q-tables (the first quantifier) needs to be reasonable (as in regularized, realistic), to avoid values very close to the overflow/underflow limit. What is realistic range of reward values is problem-dependent.

The domain properties listed above, although very simple, constitute real development problems in debugging RL implementations. When implementing applications we often encountered these errors and other RL developers confirmed this to us anecdotally. Such simple tests are able to capture and help diagnose many confusions in practice. It appears that testing for simple properties helps

[2] Convergence is also hard to prove in a formal system, and even otherwise—the convergence of deep reinforcement learning remains an open problem [Sutton and Barto 2018].

to eliminate the main bugs fast. This is in contrast to complex properties of the algorithm, discussed in Sect. 5.2, which tend to cause fewer problems in practice. Algorithms are typically implemented well, and are run (and thus debugged) many times by many users, but problem formulations are new for every RL task one undertakes and thus prone to new bugs. A good test suite ensuring their basic properties significantly accelerates the task definition process.

Specific Problem Properties. These tests capture idiosyncratic properties of the problem domain; they cannot be formulated without a concrete problem. We include a few cases for our running example to complete the picture, starting with the physics of braking. Equation (22) states that the car's position never becomes negative, Eq. (23) expresses that a stopped car cannot be moved by braking , and Eq. (24) states that a forward-moving car cannot move backward by braking. All three properties reflect actual bugs experienced by us in our first model of physics for this example:

$$\forall \overline{S}_1, \overline{S}_2 \in \overline{\mathbf{State}}.\ \forall A \in \mathbf{Action}.\ (\mathcal{T}\ \overline{S}_1\ A)\ \overline{S}_2 \geq 0\ \rightarrow\ \overline{S}_2.p \geq 0 \tag{22}$$

$$\forall \overline{S}_1, \overline{S}_2 \in \overline{\mathbf{State}}.\ \forall A \in \mathbf{Action}.\ [\overline{S}_1.v = 0 \land (\mathcal{T}\ \overline{S}_1\ A)\ \overline{S}_2 > 0] \rightarrow \overline{S}_1.p = \overline{S}_2.p \tag{23}$$

$$\forall \overline{S}_1, \overline{S}_2 \in \overline{\mathbf{State}}.\ \forall A \in \mathbf{Action}.\ [\overline{S}_1.v > 0\ \land (\mathcal{T}\ \overline{S}_1\ A)\ \overline{S}_2 > 0] \rightarrow \overline{S}_2.p \geq \overline{S}_1.p \tag{24}$$

The next two tests capture basic intuitions about rewards. Equation (25) says that the further the car is from the obstacle the larger the reward; it is then easier to brake in time, if the velocities are the same. Equation (26) states that lower velocity should yield higher rewards, as it is easier to stop by braking from lower velocities, if in the same position.

$$\forall \overline{S}_1, \overline{S}_2 \in \overline{\mathbf{State}}.\ \forall A \in \mathbf{Action}.\ \overline{S}_1.p \leq \overline{S}_2.p \land \overline{S}_1.v = \overline{S}_2.v \rightarrow \mathcal{R}\,\overline{S}_1\,A \geq \mathcal{R}\,\overline{S}_2\,A \tag{25}$$

$$\forall \overline{S}_1, \overline{S}_2 \in \overline{\mathbf{State}}.\ \forall A \in \mathbf{Action}.\ \overline{S}_1.p = \overline{S}_2.p \land \overline{S}_1.v \leq \overline{S}_2.v \rightarrow \mathcal{R}\,\overline{S}_1\,A \geq \mathcal{R}\,\overline{S}_2\,A \tag{26}$$

In summary, problem-specific tests are needed to check that the transition function and the reward function capture the domain characteristics and problem objectives.

5.2 Testing Reinforcement Learning Algorithms

The specification in Sect. 4.3 enables us to derive correctness tests for the learning algorithms. We start with algorithm-independent properties that apply widely to TD learning methods. The simplest tests enforce domain constraints on initialization and update steps. Let Q_0 be the initial value of a Q-table. We test whether Q_0 is defined for all state action pairs in Eq. (27) and that all entries are initialized to zero—a common choice—in Eq. (28).

$$\mathbf{dom}\ Q_0 = \mathbf{State} \times \mathbf{Action}\ , \tag{27}$$

$$\forall S_0 \in \mathbf{State}.\ \forall A_0 \in \mathbf{Action}.\ Q_0\ (S_0,\ A_0) = 0.0\ . \tag{28}$$

We establish a domain property for the policy used to select actions. Below, π represents some implementation of a policy; we enforce adherence to the specification of Eq. (4).

$$\forall Q_t \in \mathbf{Q}.\ \forall S_t \in \mathbf{State}.\ \forall A_t \in \mathbf{Action}.\ \ [(\pi\ Q_t\ S_t)\ A_t > 0] \rightarrow A_t \in \mathbf{Action}\ . \tag{29}$$

While ensuring domain constraints is relevant, the essence of each policy lies in its probabilistic behavior. In an on-policy ε-greedy learning algorithm (the class considered in this paper), the policy selects a random action with probability ε and otherwise it greedily follows the highest-value action. This requirement can be cast as a probability distribution test. For every state S_t, the selected action A_t should be distributed according to the distribution $\pi\,Q_t\,S_t$. In Eq. (30), we derive a Boolean random variable that tracks selecting the highest value action. We check whether this random variable is distributed according to a Bernoulli distribution with a parameter derived from Eq. (5).

$$\forall Q_t \in \mathbf{Q}.\ \forall S_t \in \mathbf{State}.\ \left[(\pi\, Q_t\, S_t) \gg\!= (\lambda A_t.\, A_t \neq \arg\max_A Q_t\,(S_t, A))\right] \sim \mathrm{Bern}\left(\varepsilon \cdot \frac{|\mathbf{Action}| - 1}{|\mathbf{Action}|}\right) \tag{30}$$

In practice during testing, the policy is not available as a symbolic representation of a distribution, but as a sampling algorithm. Therefore, testing the above law requires a statistical test. In our test harness for reinforcement learning, we perform a Bayesian test here. We use a weak prior (a Beta distribution) which encodes that the actual parameter of the Bernoulli distribution is essentially unknown. We collect a sample of executions of the policy and estimate the posterior belief in this parameter given the outcomes of these executions, whether the maximum value action or another action has been selected. This can be calculated analytically for a Beta prior using the conjugate update rule for a Bernoulli likelihood [Kruschke 2014]. We check whether in the obtained posterior distribution over values concentrates 0.94 of the probability mass in a small *credible interval* containing $\varepsilon \cdot (1 - |\mathbf{Action}|^{-1}|)$ (also known as a *high density interval* [Kruschke 2014]).

As argued in the previous section, the Q-table update is the key element of reinforcement learning—an update is the defining piece of logic for every reinforcement learning algorithm. Second, updates are executed with high frequency. A learning procedure often involves hundreds of thousands of episodes, with many epochs per episode, each epoch containing an update. Third, the success criterion for an update is not easily observable: An update would typically not crash, but "just" produce a wrong floating point number. For this reason, a broken update step can remain unnoticed for a long time, only manifesting in hard to explain subpar results from learning, for instance a slower convergence or higher variance (instability)—both very hard to assess. Consequently, a thorough testing of the update step appears prudent.

The specification of an update defines a function that, given a Q-table Q_t, a system state $\overline{S}_t$, and an action A_t, returns a multivariate density function over successor Q-tables, target states, and subsequent actions (cf. Sect. 4.3). Let `update` denote the implementation of the update function and $[\![u]\!]$ the prescribed semantics of this function for the algorithm under test (cf. Sect. 4.3, Eqs. (15) and (16)). The following requirement states that the implementation and the specification produce the same multivariate distribution:

$$\forall Q_t \in \mathbf{Q}.\ \forall \overline{S}_t \in \overline{\mathbf{State}}.\ \forall A_t \in \mathbf{Action}.\ \mathtt{update}\ (Q_t,\ \overline{S}_t,\ A_t) = [\![u]\!]_{bdl}(Q_t, \overline{S}_t, A_t)\ , \tag{31}$$

where `update` stands for the implementation of the update function for the concrete learning algorithm. Two aspects of Eq. (31) warrant further discussion: first, how is the equality established (recall that this is an equality on multivariate distributions), second, how the specification $[\![u]\!]_{bdl}$ is concretely provided to the test. We handle these points in order.

First, an update produces a multivariate distribution over target states, next actions, and Q-tables. Out of the three components, the target state selection belongs to specific problem tests (Sect. 5.1), the next-action selection follows the policy (see Eq. (30)). Let us focus on the marginal representing the change in the Q-table entry $Q_t\ (O\,\overline{S}_t,\ A_t)$, as the most interesting here. We derive the two random variables representing this update using the implementation and the specification.

$$\begin{gathered}\forall Q_t \in \mathbf{Q}.\ \forall \overline{S}_t \in \overline{\mathbf{State}}.\ \forall A_t \in \mathbf{Action}.\\ \left[\mathtt{update}\ (Q_t,\ \overline{S}_t,\ A_t) \gg\!= g\right] = \left[[\![u]\!]_{bdl}\,(Q_t,\ \overline{S}_t,\ A_t) \gg\!= g\right]\\ \text{where } g = \lambda(Q_{t+1},\ \overline{S}_{t+1},\ A_{t+1}).\ Q_{t+1}\ (O\,\overline{S}_t, A_t)\ .\end{gathered} \tag{32}$$

Then we test statistically whether the two distributions agree. Assuming that the updates are normally distributed, allows us to compare the two samples with a simple Gaussian model. This test can be implemented in many ways. We perform a Bayesian belief propagation again. We generate a statistical sample of update results from both the implementation and the specification, compute their point-wise difference, and derive a posterior for the difference using an uninformative normal prior (a conjugate). Then we check whether the posterior for the difference is concentrated around zero with high probability (credibility interval 0.94) [Kruschke 2014].

To perform this test we need the specification u. We use three strategies in our library to provide this specification. We list them in the order of increasing trustworthiness. First, we can *directly implement a specification* as a function in a functional language, directly following the equations of Sect. 4.3 instantiated for a concrete algorithm u, and using a sampling-based implementation of the probability monad (we use our own implementation here). This approach might be useful if testing an implementation in a different style, say a Python implementation. When testing a purely functional implementation of a reinforcement learning algorithm, we end up testing essentially the same code against itself, as the specification and implementation are identical. Second, we can use an implementation of the *update function of another algorithm*. For instance, as discussed in Sect. 4, we can test Q-learning using an update function of SARSA, and a greedy policy π. Testing implementations against each other is a well established tradition—this here is a special case of differential testing for an expected update value.

The third strategy is the most interesting one from the programming language perspective. It is crucial that the equations in Sect. 4.3 are executable, given a sampling implementation of the probability monad. We can thus implement an *interpreter* for *bdl* terms, which given a particular specification term, performs the update accordingly following the model of the update. This allows using the interpreter to test many TD algorithms, the same way as, for example, interpreters are used to test compilers and partial evaluators (where interpreted code is also serving as an oracle for a concrete specialized code). We show fragments of our interpreter in Sect. 6.

6 IMPLEMENTATION

We implemented the above test harness in Scala, and used it to test implementations of SARSA, Q-Learning, and Expected SARSA, along with several example reinforcement learning problems extracted from text books and research papers. The entire project is about 2.3K lines of purely functional Scala 3 code using the Cats and ScalaCheck libraries;[3] including the algorithms, all example problems, and tests. Our infrastructure is extensible, it facilitates modular development of reinforcement applications, and continuous testing during the development. Many tests are reusable and can be instantiated for new algorithms and problems. The project is open source and available[4].

We follow the *property-based testing* (PBT) methodology of Claessen and Hughes [2000] to bridge from the formal specification world to the concrete applications. This methodology is useful both for testing and for development of formal specifications, as it facilitates static type checking and immediate randomized execution of formal properties. This way obvious, and even some non-obvious, specification errors can be detected automatically. At the same time the specifications can be used as tests.

In PBT, program properties are written as executable predicates over input data. Testing a property involves generating inputs automatically, evaluating a predicate on the inputs, and checking whether it holds on all the inputs. For each input data variable we need a test case generator. These are typically associated with the types in PBT, at least in strongly typed languages. The PBT testing libraries provide generators for standard types, and since generators are compositional, it is relatively cheap to add custom ones, as we also show below. PBT testing libraries are available for most mainstream programming languages.

[3] https://typelevel.org/cats/
[4] https://github.com/itu-square/symsim

```
forAll { (s_t: State) =>                       // ∀ s_t ∈ State
  forAll { (a_t: Action) =>                    // ∀ a_t ∈ Action
    for s_t1 <- agent.step (s_t) (a_t)         // s_t1 ← sample(𝒯 s_t a_t)
        d1    = agent.observe (s_t1)           // d1 ← 𝒪 s_t1
    yield observableStates.contains (d1)       // d1 ∈ State
} }
```

Fig. 4. Testing that a state which can be reached by a step can also be correctly observed (a domain constraint). The comments in the right column relate the test to Eq. (20)

```
val positions = Gen.choose[Double] (0.0, 10.0)
val velocities = Gen.choose[Double] (0.0,10.0)
val actions = Gen.oneOf (Car.instances.enumAction.membersAscending)

forAll (velocities, positions, actions) { (v, p, a) =>     // ∀ S_1 = (v,p) ∈ State. ∀A ∈ Action
  for s_2 <- Car.step (CarState (v, p)) (a)                // S_2 ← sample(𝒯 S_1 A)
  yield (v > 0 ==> s_2.p >= p)                             // v > 0 implies S_2.p ≥ S_1.p
}
```

Fig. 5. A car shall not move backwards by braking. The comments in the rightmost column trace to Eq. (24)

```
forAll { (q: Q, a_t: Action) =>
  val trials = for s_t  <- agent.initialize
                   a_t1 <- chooseAction (q) (agent.discretize (s_t))
  yield a_t1 != bestAction (q) (agent.discretize (s_t))
  val successes = trials.take (episodes).count { _ == true }
  val failures = episodes - successes
  val cdfEpsilon =  Beta (2 + successes, 2 + failures).cdf (epsilon)
  cdfEpsilon >= 0.94
}
```

Fig. 6. An ε-greedy on-policy action choice follows the best action greedily with probability below ε, cf. Eq. (30)

We now demonstrate how the properties of Sect. 5 can be tested. Let us begin with the basic domain constraint that if a state can be reached, its observation is a valid observable state, as defined in Eq. (20) (a generic problem property). The test is shown in Fig. 4. Notice that the implementation is very close to the logical description of the property—the comments make the link explicit. The biggest difference is that instead of quantifying over S_{t+1} and checking whether it is reachable with positive density, we sample the value of S_{t+1} from the successor state distribution—if the sample is obtained, we are guaranteed that its density was positive. Notably, in an implementation of the test, the logical quantifiers (`forAll`) are implemented as samplers of test data. Instead of proving the property, the framework evaluates it on several hundreds of input cases.

To support generation of test cases in properties like the one of Fig. 4, the test harness requires that the problem definition includes generators for actions and states. Below we show the default generator for the states of the braking car example using ScalaCheck:

```
val genCarState: Gen[CarState] = for
   v <- Gen.choose (0.0, Double.MaxValue)
   p <- Gen.choose (0.0, Double.MaxValue)
yield CarState (v, p)
```

Figure 5 shows an example of a specific problem test for our running example—checking whether the braking car does not go backwards while moving, as specified in Eq. (24). In this example, we use custom generators (defined in lines 1–3) to control the domains of states, velocities, and actions to come from some specific ranges. The actual property derives a Boolean random variable representing test success like before, except that we also use the ScalaCheck pre-condition operator ==>.

Finally, Fig. 6 shows a simple implementation of the learning algorithm property specified in Eq. (30), checking that when following an ε-greedy policy, a random action is selected with probability ε with high belief. In this test, random Q-tables are generated using our custom generator (not shown). As we cannot compare distributions directly, we derive a Boolean random variable `trials`, true whenever an action has not been selected greedily. We count the number of successes in a sample and calculate a posterior for the bias parameter (l. 7). Line 8 checks whether this posterior has almost all values below ε (probability mass 0.94).[5]

Finally, the test of the update distribution of Eq. (32) follows a similar statistical design. Here the more interesting aspect is how the update oracle is provided—as mentioned before this can be done by interpreting our term language. Figure 7 shows all the sampling cases of our *bdl* interpreter, implementing the semantic rules of Sect. 4.3. The listing has four parts: the *bdl* abstract syntax implementation (lines 1–7), an estimation step definition (lines 8–18), a sequential Kleisli composition of multiple estimation steps (lines 19–23), and finally a definition of an update step (lines 24–34). The reader can convince themselves that the interpreter indeed follows closely and formally the definitions of the algorithms, and it is fairly easy to establish a one-to-one correspondence (in our implementation the transition $\mathcal{T}$ and reward $\mathcal{R}$ are combined in a single call, e.g. in Line 13). A small term in the *bdl* abstract syntax can be provided to the interpreter to simulate an update of any TD learning algorithm. A similar interpreter can be naturally implemented in any functional programming language.

7 EXPERIMENTAL EVALUATION

We now assess the applicability of the specification and evaluate the effectiveness of the resulting test harness. We discuss how these tests can be reused and as a result reduce the cost of testing in such setups. In particular we address the following research questions:

RQ1 Is the specification general enough to accommodate diverse reinforcement learning problems?
RQ2 How effective is the test harness in finding bugs in reinforcement learning problems?
RQ3 To what extent can generic problem properties be used to reduce the cost of testing for reinforcement learning problems?

To answer **RQ1**, we implement a range of small and medium-sized case studies (first 5 columns in Tbl. 2). We give a brief description of each case study in the following. (Our code and case studies for the experiments in this section are open source and publicly available [Varshosaz et al. 2023])

Case studies. The Unit Agent is the smallest problem in the collection, featuring a single state and a constant reward. We have defined and implemented it to facilitate testing and debugging of learning on a minimal case. Braking Car is the running example of this paper [Vardhan and Sztipanovits 2021]. The somewhat similar Golf learns what club and force to use to hit the target in a minimum number of rounds. The Mountain Car aims to learn how to obtain enough momentum to move up a steep climb [Moore 1990]. Maze requires an agent to find a safe path to a goal location in a 2D maze [Russell and Norvig 2016]. Windy Grid is similar, but includes randomized dislocation of the agent (a wind) [Sutton and Barto 2018]. Cliff walking is another example in which the goal of the agent is to move from a point in a map to another without walking into a cliff area [Sutton and

[5]For a Bayesian statistics afficionado, we remark that this test would be better done using a proper region-of-practical-equivalance (ROPE) for ε [Kruschke 2014], but we use a simpler test here to avoid longer discussions of statistics.

```
// BDL abstract syntax cf. Eq. (7) p. 10
enum Est:
  case Sample (gamma: Double)
  case Expectation (gamma: Double)
enum Upd:
  case SampleU, ExpectationU
case class Update (est: List[Est], alpha: Double, update: Upd)

// Semantics of a sampling estimation step cf. Eq. (9) p. 12
def sem (est: Est) (q_t: Q) (s_t: State, a_t: Action, g_t: Double, gamma_t: Double)
  : Randomized[(State, Action, Double, Double)] = est match
  case Sample (gamma) =>
    for (s_tt, r_tt) <- agent.step (s_t) (a_t)
        os_tt         = agent.observe (s_tt)
        a_tt         <- vf.chooseAction (epsilon) (q_t) (os_tt)
        g_tt          = g_t + gamma_t * r_tt
        gamma_tt      = gamma_t * gamma
    yield (s_tt, a_tt, g_tt, gamma_tt)
  case Expectation (gamma) => ...

// Kleisli composition of mutliple estimation steps cf. Eq. (13) p. 13
def sem (ests: List[Est]) (q_t: Q) (s_t: State, a_t: Action, g_t: Double, gamma_t: Double)
  : Randomized[(State, Action, Double, Double)] =
  ests.foldM (s_t, a_t, g_t, gamma_t)
    { case ((s_t, a_t, g_t, gamma_t), e) => sem (e) (q_t) (s_t, a_t, g_t, gamma_t) }

// Semantics of a sampling update cf. Eq. (15) p. 14
def learningEpoch (bdl: Update, q_t: Q, s_t: State, a_t: Action): Randomized[(Q,State,Action)] =
  bdl.update match
  case SampleU =>
    for (s_tk, a_tk, g_tk, gamma_tk) <- sem (bdl.est) (q_t) (s_t, a_t, 0.0, 1.0)
        (os_t, os_tk)                 = (agent.observe (s_t), agent.observe (s_tk))
        g_tkk                         = g_tk + gamma_tk * q_t (os_tk, a_tk)
        q_tt_value                    = q_t (os_t, a_t) + bdl.alpha * (g_tkk - q_t (os_t, a_t))
        q_tt                          = q_t.updated (os_t, a_t, q_tt_value)
    yield (q_tt, s_tk, a_tk)
  case ExpectationU => ...
```

Fig. 7. The abstract syntax and interpreter for BDL (only the sampling parts; the expectation parts differ minimally as shown in the equations in Sect. 4.3). Given a term representing an update of a particular reinforcement learning algorithm, this interpreter serves as a probabilistic correctness oracle for testing updates.

Barto 2018]. The k-arm bandit represents a class of state-less agents which admit k actions [Sutton and Barto 2018]. Pumping is a larger industrial case study developed together with a public utility company operating pumping stations for drinking water. The controller must satisfy the public water consumption while the water table does not run dry or get polluted. The pumping case has a larger state space with 92160 observable states.

In response to **RQ1**, we note that the types implementing the concepts of our formal specification facilitate reinforcement learning problems of various scale. The framework allows for learning policies by running different learning algorithms and test the applications. Each application has a test suite and generic tests can be reused between applications. In total, all these case studies required as little as 68 lines of code for the generator implementations. Thus the generators are

Table 2. Experiment results. *K-arm bandit is the class of stateless randomized problems. Unit agent was used in testing learning algorithms but it has no tests itself and was unmutated as it represents an artificial problem.

	state space size		episodic	gen. size	test cases	mutants			time	mutation
agent	**continuous**	**observable**		**[LOC]**	**[#]**	**killed**	**survived**	**invalid**	**[s]**	**score [%]**
Pumping	$\mathbb{R}\times\mathbb{R}\times\mathbb{R}\times\mathbb{R}$	92160	✓	17	31	43	16	0	1787	73
Mountain Car	$\mathbb{R}\times\mathbb{R}$	121	×	11	20	24	7	1	34	77
Braking Car	$\mathbb{R}\times\mathbb{R}$	12	✓	9	17	25	0	2	43	100
Windy Grid	-	70	✓	6	17	7	0	1	31	100
Cliff Walking	-	38	✓	6	13	31	1	0	5	97
Simple Maze	-	12	✓	7	15	26	2	0	8	93
Golf	-	10	✓	6	12	9	1	0	7	90
K-arm Bandit*	-	2	✓	3	5	2	0	4	7	100
Unit Agent	-	1	✓	3	-	-	-	-	-	-

not hard or expensive to develop. This highlights the advantage of using the types provided by the specification that enables reuse in implementing reinforcement learning problems.

To answer **RQ2** and **RQ3**, we evaluate the adequacy and the effectiveness of the test suite, using mutation testing [DeMillo et al. 1978; Hamlet 1977]. During mutation testing, variants of a program, called *mutants*, are generated by applying syntactic changes, a class of fault injections. The objective of mutation testing is to measure the ability of a test suite in distinguishing between the output of the original program and its mutants. The program output is often defined as the observable return values, thrown exceptions and program crashes (in the context of unit testing). An outcome of mutation testing is a *mutation score* that represents the ability of the test suite to discriminate between mutants and the correct code [Papadakis et al. 2019]. In the following we explain how the experiments are designed and discuss the results.

Experiment design. We generate mutants of reinforcement learning algorithms, agents and environments using Stryker,[6] a mutation testing tool that supports several programming languages. Stryker4s is the version of the tool for Scala. We use it to inject faults in the implementation of the reinforcement learning problem and run tests. Stryker supports syntactic transformation rules, *mutators*, for *boolean literals*, *conditional expressions*, *equality operator*, *logical operator*, *method expressions*, *regex* and *string literals*. For each mutant Stryker reports a result: *survived*, means that all tests passed, *killed* means that at least one test failed, *timeout* means that the tests have not terminated before timeout, and *no coverage* means the mutation was not detected by tests—no test failed. Besides these, Stryker also reports invalid mutants e.g., mutants causing runtime errors. Stryker computes a mutation score as:

$$\text{mutation score} = \frac{\#\text{killed} + \#\text{timout}}{\#\text{valid} \ \times \ 100} \tag{33}$$

Mutators for *arithmetic operations* are not supported by Stryker for Scala, so it does not generate mutants of reinforcement learning algorithms (other mutators do not change the code as their target operations do not exist in the algorithm). This is a limitation applied conservatively for Scala as arithmetic operators are function calls and, depending on the code, mutating these operators can lead to mutants that are *stillborn* (i.e., syntactically illegal). To mitigate for this weakness, we manually (using a script) mutate our implementations of reinforcement learning algorithms following the

[6] https://stryker-mutator.io/

strategy used by Stryker's arithmetic mutators for other languages. In general, these changes are among the common changes applied by arithmetic mutators [Papadakis et al. 2019]: swapping plus for minus or vice-versa (m_1) and swapping division for multiplication or vice versa (m_2).

Execution and results. We generate mutants for all case studies described in Tbl. 2, except for the unit agent which is not a real agent but rather an algebraic construct used to test the algorithms (it has a singleton state and a singleton policy space). We use Stryker4s version 0.14.3, which is the latest available version that supports Scala 3. The mutants of the reinforcement learning algorithms are generated semi-automatically using a Python script which applies arithmetic mutators to inject faults in the implementation (other mutators are not applicable for these implementations as explained above). To answer **RQ2**, we run Stryker on each of the case studies individually 10 times. The experiments are performed on a Macbook Pro with 2,3 GHz Quad-Core Intel Core i5 processor and 16 GB RAM.

The results can be found in the six right-most columns of Tbl. 2. The test cases column shows the total number of tests that are run against each mutant. This number includes six generic problem tests (laws) that should hold for all agents. The time reported in the table is the average of the time for Stryker running tests on mutated files in 10 experiment runs. As the mutation score can change due to the randomisation in tests, we report the lowest mutation score in case of changes between runs (only in two cases the mutation score changed between runs).

In response to **RQ2**, we note that the results of evaluation show that in 75% of the cases the mutation score is above 90%. One of the common reasons for surviving mutants is lack of tests for extreme values which is due to the limitation of test data generators. Additionally, there are mutants, for example in the pump case, that are result of applying changes in a function in which the outcome is conditionally selected from overlapping intervals. Hence, writing tests that distinguish changes in the conditions is not feasible.

To address **RQ3**, we re-run the experiments excluding the tests specific to the agents and using only generic tests. These tests express properties (laws) that should hold for any reinforcement learning problem, specifically agent and algorithm in this framework. The results of performing mutation testing using generic problem tests are presented in Fig. 8. In this figure the number of killed and survived mutants are depicted (the number of invalid mutants remain the same as in Tbl. 2).

To answer **RQ3**, we note that the results in Fig. 8 (right) show that in all cases except for pump example the mutation score is above 48%. We observe that mutants are detected by tests that perform sanity checks on the output of the functions responsible for discretizing (observing) the state space and identifying when an agent is in a final state. The results show that these tests can be effective in finding a subset of bugs and providing them can give an advantage to the developer to avoid rewriting the tests which reduces the cost of testing as a result.

Figure 8 (left) illustrates the results of performing mutation testing for SARSA and Expected SARSA algorithms. For each algorithm seven tests (laws) are executed. In SARSA algorithm, five faults are injected including three by m_1 and two by m_2 mutations. All mutants with m_1 changes are killed by tests that are generated based on the specification of SARSA algorithm as explained in Sect. 5. The mutants with m_2 changes are stillborn and are caught by the Scala type checker due to a type mismatch introduced by the change. In Expected SARSA algorithm, six faults are injected including three by m_1 and three by m_2. Similarly the three mutants with m_1 changes are killed and the three mutants with m_2 changes are caught by the Scala type checker. As an additional observation in relation to **RQ3**, we note that in this case all valid mutants are killed by the tests designed for the learning algorithms while the stillborn mutants are caught by Scala type checker. Hence, considering the valid mutants, a full detection of bugs is achieved by the tests designed for

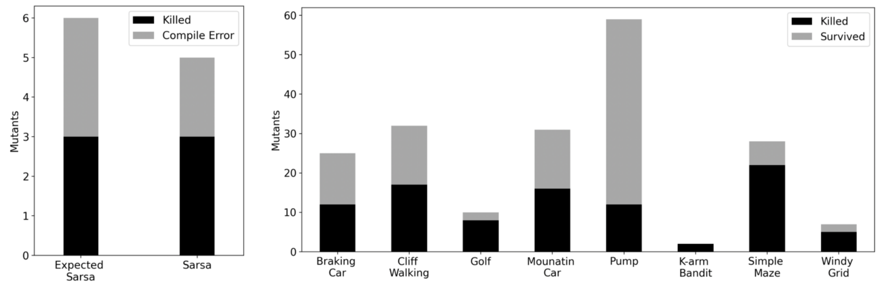


Fig. 8. Mutation results with generic tests for SARSA/Expected SARSA (left) and case studies (right).

testing TD algorithms. These test cases can be parameterized and reused between algorithms and as a result can contribute to reducing the cost of testing the reinforcement learning setups.

8 DISCUSSION

Supporting Other Reinforcement Learning Methods. We focused on testing on-policy TD learning, and exploited an equivalence of updates to test also Q-learning, which is an off-policy algorithm. In general, specifying off-policy learning requires a refinement of *bdl* semantics to support two policies simultaneously. There are no technical obstacles to it, besides maintaining the simplicity of exposition. In Eqs. (9), (10), (15) and (16), one needs to distinguish the policy π used for estimation in the final update from the one that is used to select the next action for execution. Presently, the same policy is used for both. We do intend to implement this in our Scala harness soon.

Support for *Monte Carlo* methods and dynamic programming can be added rather directly. Monte Carlo methods are very similar to n-step TD methods (Tbl. 1). The main difference is that the update only happens at the end of an episode, i.e., when a terminal state is reached. At this point the actual return is known and the state action values can be updated. The sequencing of estimations for episodic systems is a special case of the sequencing of estimations introduced in Eq. (15). To describe the termination of episodes in a final state, without a pre-specified number of steps, a Kleene-star-like variant of the update equation could be introduced in *bdl*. We sketch its definition below. Note that the update-until-termination does not require the final prediction step, as all rewards are known until a termination of a run.

$$
\begin{aligned}
&[\![est\ \text{Update} *]\!]_{bdl}\ S_0\ A_0\ G_t\ \gamma_t = \\
&\quad \lambda \left(Q_t, \overline{S}_t, A_t\right).\ \mathbf{if}\ \mathcal{F}\ (O\ \overline{S}_t) && \text{(at the end of the episode?)} \\
&\qquad \mathbf{then}\ \text{Det}\left(Q_t\ [(S_0, A_0) \mapsto G_t]\ , \overline{S}_t, A_t\right) && \text{(update and stop if in final)} \\
&\qquad \mathbf{else}\ [\![est]\!]_{est}\ (Q_t)\ (\overline{S}_t, A_t, G_t, \gamma_t)\ \ggcurly && \text{(execute the estimation step)} \\
&\qquad\quad \lambda(\overline{S}_{t+1}, A_{t+1}, G_{t:t+1}, \gamma_{t+1}).\ \text{Det}\left(O\ \overline{S}_{t+1}\right)\ \ggcurly && \text{(observe the resulting state)} \\
&\qquad\quad \lambda S_{t+1}.\ [\![est\ \text{Update} *]\!]_{bdl}\ S_0\ A_0\ G_{t:t+1}\ \gamma_{t+1}\ (Q_t, \overline{S}_{t+1}, A_{t+1}) && \text{(iterate again)}
\end{aligned}
\tag{34}
$$

Here, the state S_0 is the origin state of the Monte Carlo update (initially the same as $O\ \overline{S}_t$), and A_0 is the initial action (same as the first A_t). The values of G_t and γ_t, the initial accumulated reward and the initial compounded discount factors, should be initialized to zero and one, respectively. The essential difference from Eq. (15) is found in the first three lines. We check whether the agent has arrived at a final state and terminate with an update, if so. Otherwise we iterate again, maintaining the current accumulated reward and discount factor. The above can be defined as the least fixed point of a Scott-continuous operator (the core of the equation above) in a suitable domain. These are standard steps though, so we omit the details, in favour of a program-like recursive presentation.

For dynamic-programming-based methods (DP), the learner requires an explicit probabilistic model of the environment. Hence, in the update rules one should use an MDP model instead of sampling to obtain the successor states, rewards and their probabilities. The DP algorithms commonly use a state value function $V \in \mathbf{State} \to \mathbb{R}$, instead of a Q-table. A state value function represents the total amount of reward that an agent can accumulate over the episodes starting in that state. The value of each state is updated iteratively, by accumulating the immediate reward and value of all possible successor states, i.e., the states that are reachable in one step. To formalize this, the same structure of the equations can be kept, but with a unary (not binary) value function. Moreover, one can eliminate the probability monad, as the dynamic programming update is deterministic (or uses Dirac distributions if maintaining the same structure with probabilistic algorithms is desirable).

Testing the TD(λ) algorithm requires modeling *eligibility traces* [Sutton and Barto 2018] in the *bdl* semantics and the interpreter. The eligibility traces are a mechanism which allows to execute multiple TD-updates simultaneously, similarly to pipe-lining in CPUs. Once eligibility traces are handled in the *bdl* interpreter it will graduate from being a test oracle to being a general TD learning algorithm, parameterized by an updated specification, which is interesting for future work.

It is also interesting to generalize to different value function representations, to allow *approximate learning* with neural networks. As *bdl* abstracts from the representation of the Q-table, a neural network can in principle be used; structurally the specification should not change much. For example Actor-critic methods still follow the pattern of TD(0), SARSA, and Q-Learning [Sutton and Barto 2018]. As action selection is already probabilistic in our model (ε-exploration), for continuous actions we just need to switch from sampling a discrete distribution to a continuous one. The challenge lies in formalizing gradient-based updates on neural networks, which are much more sophisticated than simple assignments to a table cell, and require non-trivial further work; similarly for popular newer policy iteration methods like PPO [Schulman et al. 2017]. To address this limitation, we plan to investigate using methods taken from the differentiable programming languages field in the future.

Software Testing with Statistics. As machine learning gains popularity, we face more and more programs with probabilistic correctness requirements that have to be tested statistically. Reinforcement learning is one such example. This requires development of new experience in software testing. The tests we used are certainly simple, even simplistic, as we prioritized efficiency and simplicity over strength. Assuming conjugate priors is too strong in general [McElreath 2020]. Monte-Carlo posterior estimation is a possible alternative—unfortunately this would slow the tests down by several orders of magnitude. The performance problem is exacerbated, if more precise tests are used. We tested for equivalence of expectations, but the variance is also relevant for reinforcement learning, especially that learning algorithms may agree on the expectation but differ on variance. Comparing variance would help to kill mutants that maintain the same expectation at the cost of slower convergence, which manifests in a higher variance of reward estimations. Unfortunately, it is computationally much harder to estimate variance as precisely as the mean. In general, the credibility (strength) of statistical tests correlates with their computational cost. More samples are required for stronger conclusions.

Statistical tests are by their very nature flaky [Luo et al. 2014]. They can fail occasionally, disturbing continuous integration, or mutation scoring like in our experiments (mutation runs the tests many times, increasing the chance of failure occurring). There is an inherent tension between flakiness of tests and their ability to kill mutants and to detect bugs. One can set weaker thresholds for credibility of a test, which will make it fail less or practically never, but it will also make it accept larger deviations from the spec and miss more bugs. One can strengthen the test to kill more mutants, but this will increase the frequency at which the test will fail randomly on the correct code. This trade-off makes it very difficult to set the hyper parameters (thresholds) for statistical tests. We have done this by trial-and-error, but more systematic methods are needed.

9 RELATED WORK

A large body of recent work studies the use of RL and deep RL to improve testing processes. Such techniques [Liu et al. 2022; Romdhana et al. 2022; Su et al. 2022; Tufano et al. 2022; Türker et al. 2021; Zhang et al. 2021; Zheng et al. 2021] are applied for testing a variety of systems (e.g., video games, web applications, and cyber physical systems). In contrast, we are concerned with the opposite problem—applying testing to reinforcement learning.

Many authors focus on testing Machine Learning (ML) algorithms broadly. For example, optimizing stochastic regression tests in ML projects [Dutta et al. 2021], augmenting a deep learning test set to increase its mutation score [Riccio et al. 2021], testing bias in ML software [Chakraborty et al. 2021], pointwise robustness in deep neural networks [Wu et al. 2020], concolic testing for deep neural networks [Sun et al. 2018], formally verifying safety properties of deep reinforcement learning system [Ivanov et al. 2020]. The present paper does not contribute to testing neural networks (even if they are a representation of value functions used in RL) but addresses testing the correctness of RL problems and algorithm implementations by providing specifications for their basic blocks.

In the field of RL, a key topic of focus in prior work is the reliability assessment of a trained agent. Adversarial ML is used to understand the behavior of models and algorithms in the presence of failure inducing contexts and behaviors. Huang et al. investigate impact of the effectiveness of adversarial examples on a deep RL algorithm [Huang et al. 2017]. Lin et al. introduce strategically timed attacks on RL agents [Lin et al. 2017]. Amirloo et al. propose to guide adversarial sampling by a predictor trained along with the agent to predict the probability of failure [Abolfathi et al. 2021]. Vardhan and Sztipanovits use a generative model to find failure scenarios [Vardhan and Sztipanovits 2021]. Ruderman et al. study the worst-case analysis to detect the directions in which agents may have failed to generalize while learning the policy [Ruderman et al. 2019]. To overcome the small adversarial perturbations on the agent's inputs, Oikarinen et al. propose to train RL agents with improved robustness against l_p-norm bounded adversarial attacks [Oikarinen et al. 2021]. In this line of work, the focus is on optimality and generality of the obtained policies. However, they side step the problem of correctness of the RL implementations used to learn the policies. In contrast, we follow a modular testing strategy, not unlike unit testing, for low-level properties of individual elements in RL applications, hoping that this exposes problems early and close to their origins. Furthermore, this way we also hope to inspire work on formal verification of RL, as properties follow the style more often used in verification.

Another line of work, surveyed by García and Fernández [2015], addresses the synthesis and update of models that preserve safety properties by either modifying the optimality criterion or the exploration process. In particular, safety properties can be ensured for RL algorithms by incorporating a *shielding* mechanism that prevents the algorithm from taking actions that could lead to unsafe outcomes, by means of techniques such as control barrier functions [Alshiekh et al. 2018], logically constrained learning [Hasanbeig et al. 2018] and safe permissive schedulers [Junges et al. 2016]. Justified speculative control proves the shields safe by means of deductive verification [Fulton and Platzer 2018, 2019]. Jansen et al. consider probabilistic shielding to ensure the safety of RL agents [Jansen et al. 2020]. Tappler et al. combine automata learning and shielding into a method that enables RL agents to acquire a model of the environment and enforces safety constraints [Tappler et al. 2022b]. Although both shielding and testing are concerned with the correctness of RL, shielding techniques are complementary to testing: they aim at enforcing correctness by constraining the learning process rather than at exposing bugs. Whereas shielding techniques mainly focus on constraining the actions of reinforcement learning problems, our work addresses property-based testing both for reinforcement learning problems and for the learning algorithms themselves.

Other software engineering methods have been applied to test and verify RL agents, including black-box fuzzing [Pang et al. 2022], search-based testing [Tappler et al. 2022a], mutation testing [Lu et al. 2021], deductive reasoning [Déletang et al. 2021], using machine learning models and genetic algorithms to test policies [Zolfagharian et al. 2023]. Alur et al. have studied the formal specifications of RL tasks [Jothimurugan et al. 2019] and of multi-agent RL problems [Jothimurugan et al. 2022], transforming task specifications in RL [Alur et al. 2022], RL algorithms in abstract decision processes [Jothimurugan et al. 2021b], and compositional RL from logical specifications [Jothimurugan et al. 2021a] are other cases that software engineering to RL. In contrast, our work is concerned with providing a direct formal specification of correctness for RL problems and algorithms themselves, as opposed to the policies that they output. We develop a property-based test harness for all elements of a RL problem and algorithm. We are not aware of similar formal definitions of RL problems that are precise and self-contained and nor of prior uses of property-based testing for RL.

10 CONCLUSION

We have presented a formal specification of the different components of reinforcement learning, targeting temporal difference methods. The formalization enables us to derive an associated test harness, reusable across a large class of reinforcement learning applications based on Q-learning, SARSA, etc. Somewhat unusually for the reinforcement learning context, the testing harness embeds an interpreter of formal models for update equations as an oracle (a practice well recognized in programming language engineering). The test harness has been evaluated on several algorithms and agents using mutation testing, showing good baseline effectiveness. Our implementations (including all algorithms, laws, case study examples, and test scripts) is available as an open source project.

Formal verification and testing of learning algorithms is a fast growing research field. As specification is the first step towards verification, this paper may help researchers working on verification of probabilistic programs to tackle reinforcement learning. We also hope that the presentation of tests in Sections 5 and 6 can serve as a tutorial on how and what to test when developing a RL system.

ACKNOWLEDGMENTS

Partially funded by DIREC (Digital Research Centre Denmark), a collaboration between the eight Danish universities and the Alexandra Institute supported by the Innovation Fund Denmark.

DATA AVAILABILITY STATEMENT

The experiment data and the scripts to reproduce them are available at Varshosaz et al. [2023]. The implementation of symsim and its test suite are an open source project available at https://github.com/itu-square/symsim.

A APPENDIX

In order to facilitate translating our specification to formal verification systems and to other testing frameworks, we include an integrated definition below.

A.1 A Formal Specification of Reinforcement Learning

A.1.1 Reinforcement Learning Problems. In the following we add the definitions for formalising a reinforcement learning problem.

Definition A.1. A *Reinforcement Learning Problem* is represented by a tuple $(\overline{\mathbf{State}}, \overline{\mathbf{State}}_0, \mathbf{Action}, \mathbf{State}, \mathcal{O}, \mathcal{T}, \mathcal{R}, \mathcal{F})$ where:

- $\mathbf{r_1}$: $\overline{\mathbf{State}}$ is a possibly infinite set of states of the environment and the agent combined,
- $\mathbf{r_2}$: $\overline{\mathbf{State}}_0 \in \text{pdf}\ \overline{\mathbf{State}}$ is a density function defining probability for initial states,
- $\mathbf{r_3}$: $\mathbf{Action}$ is a finite set of actions that an agent can take,
- $\mathbf{r_4}$: $\mathbf{State}$ is a finite set of observable states,
- $\mathbf{r_5}$: $\mathcal{O} \in \overline{\mathbf{State}} \to \mathbf{State}$ is a total observation function,
- $\mathbf{r_6}$: $\mathcal{T} \in \overline{\mathbf{State}} \to \mathbf{Action} \to \text{pdf}\ \overline{\mathbf{State}}$ is the transition probability function,
- $\mathbf{r_7}$: $\mathcal{R} \in \overline{\mathbf{State}} \to \mathbf{Action} \to \mathbb{R}$ is the reward function, and
- $\mathbf{r_8}$: $\mathcal{F} \in \mathbf{State} \to \{0, 1\}$ is a predicate defining which observable states are final for a training episode. Initial states are not final, i.e., if $\overline{\mathbf{State}}_0\ (\overline{S}) > 0$ then not $\mathcal{F}\ (\mathcal{O}\ \overline{S})$.

Definition A.2. A RL problem is *episodic* iff every run from an initial state eventually reaches some final state $\overline{S}$, so $\mathcal{F}\ (\mathcal{O}\ \overline{S}) = 1$. Otherwise the problem is *non-episodic.*

Policy π is a probability function that, given a value function, represents the distribution of plausible actions in each state.

$$\pi \in \mathbf{Q} \to \mathbf{State} \to \text{pmf}\ \mathbf{Action} \tag{RL 3}$$

Action selection based on an ε-greedy algorithm is defined as:

$$(\pi\ Q_t\ S_t)\ A_t = \begin{cases} 1 - \varepsilon + \varepsilon \cdot |\mathbf{Action}|^{-1} & A_t = \arg\max_A Q\ (S_t,\ A) \\ \varepsilon \cdot |\mathbf{Action}|^{-1} & \text{otherwise} \end{cases} \tag{RL 4}$$

A.1.2 Temporal Difference Learning Algorithms. The general update rule for a TD prediction method is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))\ . \tag{RL 5}$$

A.1.3 Formalising Temporal Difference Algorithms. The proposed abstract syntax for backup diagrams is generated by the following grammar:

$$\begin{array}{lll} est & ::= & \mathsf{sample}^{\gamma} \mid \mathsf{expectation}^{\gamma} \\ bdl & ::= & est^{+}\ \mathsf{Update}^{\alpha}(\mathsf{sample} \mid \mathsf{expectation})\ . \end{array} \tag{RL 6}$$

The semantic domain of an estimation step is:

$$[\![\ \cdot\ ]\!]_{est} \in \mathbf{Q} \to \overline{\mathbf{State}} \times \mathbf{Action} \times \mathbb{R} \times \mathbb{R} \to \text{pmf}\ (\overline{\mathbf{State}} \times \mathbf{Action} \times \mathbb{R} \times \mathbb{R}) \tag{RL 7}$$

The semantic function for a sampling estimation step is:

$$\begin{aligned} &[\![\mathsf{sample}^{\gamma}]\!]_{est}\ Q_t\ = \\ &\quad \lambda(\overline{S}_t, A_t, G_{:t}, \gamma_t).\ \mathcal{T}\ \overline{S}_t\ A_t\ \gg\!= \\ &\quad \lambda\overline{S}_{t+1}.\ \text{Det}\left(\mathcal{R}\ \overline{S}_{t+1}\ A_t\right)\ \gg\!= \\ &\quad \lambda R_{t+1}.\ \text{Det}\left(\mathcal{O}\ \overline{S}_{t+1}\right) \gg\!= \\ &\quad \lambda S_{t+1}.\ \pi\ Q_t\ S_{t+1} \gg\!= \\ &\quad \lambda A_{t+1}.\ \text{Det}\ (G_{:t} + \gamma_t R_{t+1})\ \gg\!= \\ &\quad \lambda G_{:t+1}.\ \text{Det}\ (\gamma_t \cdot \gamma)\ \gg\!= \\ &\quad \lambda\gamma_{t+1}.\ \text{Det}\left(\overline{S}_{t+1}, A_{t+1}, G_{:t+1}, \gamma_{t+1}\right) \end{aligned} \tag{RL 8}$$

The semantic function for an expectation estimation step is:

$$
\begin{aligned}
&[\![\text{expectation}^{\gamma}]\!]_{est}\; Q_t \;=\\
&\quad \lambda(\overline{S}_t, A_t, G_{:t}, \gamma_t).\; \mathcal{T}\;\overline{S}_t\;A_t \;\gg=\\
&\quad \lambda\overline{S}_{t+1}.\;\mathrm{Det}\left(\mathcal{R}\;\overline{S}_{t+1}\;A_t\right) \;\gg=\\
&\quad \lambda R_{t+1}.\;\mathrm{Det}(O\;\overline{S}_{t+1}) \;\gg=\\
&\quad \lambda S_{t+1}.\;\pi\;Q_t\;S_{t+1} \;\gg=\\
&\quad \lambda A_{t+1}.\;\mathrm{Det}\left(G_{:t} + \gamma_t\left[R_{t+1} + \textstyle\sum_{A\neq A_{t+1}}(\pi\,Q\,S_{t+1})\,A\cdot Q(S_{t+1},A)\right]\right) \;\gg=\\
&\quad \lambda G_{:t+1}.\;\mathrm{Det}\left(\gamma_t\cdot\gamma\cdot(\pi\,Q\,S_{t+1})\,A_{t+1}\right) \;\gg=\\
&\quad \lambda\gamma_{t+1}.\;\mathrm{Det}\left(\overline{S}_{t+1}, A_{t+1}, G_{:t+1}, \gamma_{t+1}\right)
\end{aligned}
\tag{RL 9}
$$

The semantic domain for a sequence of composed estimation steps:

$$
[\![est_1\cdots est_k]\!]_{est^+} \in \mathbf{Q}\to\overline{\mathbf{State}}\times\mathbf{Action}\times\mathbb{R}\times\mathbb{R}\to\mathrm{pmf}\,(\overline{\mathbf{State}}\times\mathbf{Action}\times\mathbb{R}\times\mathbb{R})
\tag{RL 10}
$$

The semantics of composed estimation steps is defined as:

$$
\begin{aligned}
&[\![est_1\cdots est_k]\!]_{est^+}\; Q_t \;=\\
&\quad \lambda(\overline{S}_t, A_t, G_{:t}, \gamma_t).\;[\![est_1]\!]_{est}\;(Q_t)\;(\overline{S}_t, A_t, G_{:t}, \gamma_t) \;\gg=\\
&\quad \lambda(\overline{S}_{t+1}, A_{t+1}, G_{:t+1}, \gamma_{t+1}).\;[\![est_2]\!]_{est}\;(Q_t)\;(\overline{S}_{t+1}, A_{t+1}, G_{:t+1}, \gamma_{t+1}) \;\gg=\\
&\quad \vdots\\
&\quad \lambda(\overline{S}_{t+k-1}, A_{t+k-1}, G_{:t+k-1}, \gamma_{t+k-1}).\\
&\qquad [\![est_k]\!]_{est}\;(Q_t)\;(\overline{S}_{t+k-1}, A_{t+k-1}, G_{:t+k-1}, \gamma_{t+k-1})
\end{aligned}
\tag{RL 11}
$$

The compact version of the above composition is:

$$
[\![est_1\cdots est_k]\!]_{est^+}\; Q_t \;=\; [\![est_1]\!]_{est}\;Q_t \;\ggg\; [\![est_2]\!]_{est}\;Q_t \;\ggg\;\cdots\;\ggg\; [\![est_k]\!]_{est}\;Q_t
\tag{RL 12}
$$

The semantic domain of an update step is:

$$
[\![est^k\;\text{Update}^{\alpha}\;(\text{sample}\mid\text{expectation})]\!]_{bdl} \in \mathbf{Q}\times\overline{\mathbf{State}}\times\mathbf{Action}\to\mathrm{pmf}\,(\mathbf{Q}\times\overline{\mathbf{State}}\times\mathbf{Action})
\tag{RL 13}
$$

The semantic function for an update step finalized with a sample is:

$$
\begin{aligned}
&[\![est^k\;\text{Update}^{\alpha}\;\text{sample}]\!]_{bdl} =\\
&\quad \lambda(Q_t, \overline{S}_t, A_t).\;[\![est^k]\!]_{est^+}\;(Q_t)\;(\overline{S}_t, A_t, 0, 1) \;\gg=\\
&\quad \lambda(\overline{S}_{t+k}, A_{t+k}, G_{t:t+k}, \gamma_{t+k}).\;\mathrm{Det}\left(O\;\overline{S}_t, O\;\overline{S}_{t+k}\right) \;\gg=\\
&\quad \lambda(S_t, S_{t+k}).\;\mathrm{Det}\left(G_{t:t+k} + \gamma_{t+k}\cdot Q(S_{t+k}, A_{t+k})\right) \;\gg=\\
&\quad \lambda G_{t:t+k+1}.\;\mathrm{Det}\left(Q_t(S_t, A_t)\mapsto Q(S_t, A_t) + \alpha\left[G_{t:t+k+1} - Q(S_t, A_t)\right]\right) \;\gg=\\
&\quad \lambda Q_{t+1}.\;\mathrm{Det}(Q_{t+1}, \overline{S}_{t+k}, A_{t+k})
\end{aligned}
\tag{RL 14}
$$

The semantic function for an update step finalized with an expectation is:

$$
\begin{aligned}
&[\![est^k \text{ Update}^{\alpha} \text{ expectation}]\!]_{bdl} = \\
&\quad \lambda(Q_t, \overline{S}_t, A_t).\ [\![est^k]\!]_{est^{+}}\ (Q_t)\ (\overline{S}_t, A_t, 0, 1)\ \gg= \\
&\quad \lambda(\overline{S}_{t+k}, A_{t+k}, G_{t:t+k}, \gamma_{t+k}).\ \text{Det}\left(\mathcal{O}\ \overline{S}_t, \mathcal{O}\ \overline{S}_{t+k}\right)\ \gg= \\
&\quad \lambda(S_t, S_{t+k}).\ \text{Det}\ (G_{t:t+k} + \gamma_{t+k} \textstyle\sum_A (\pi\ S_{t+k+1}\ A) \cdot Q_t(S_{t+k}, A)) \gg= \\
&\quad \lambda G_{t:t+k+1}.\ \text{Det}\ (Q_t(S_t, A_t) \mapsto Q_t(S_t, A_t) + \alpha\ [G_{t:t+k+1} - Q_t(S_t, A_t)])\ \gg= \\
&\quad \lambda Q_{t+1}.\ \text{Det}(Q_{t+1}, \overline{S}_{t+k}, A_{t+k})
\end{aligned}
\qquad \text{(RL 15)}
$$

A.2 Selected Tests for Reinforcement Learning

A.2.1 Testing RL Problems. Every system state should have a translation into an observable state.

$$\forall \overline{S} \in \overline{\textbf{State}}.\ \mathcal{O}\ \overline{S} \in \textbf{State}\ , \qquad \text{(RL 16)}$$

$$\forall \overline{S}_0 \in \overline{\textbf{State}}.\ \overline{\textbf{State}}_0\ \overline{S}_0 > 0 \longrightarrow \mathcal{O}\ \overline{S}_0 \in \textbf{State}\ . \qquad \text{(RL 17)}$$

An initial state is never a final state:

$$\forall \overline{S}_0 \in \overline{\textbf{State}}.\ \overline{\textbf{State}}_0\ \overline{S}_0 > 0 \longrightarrow \neg\mathcal{F}\ (\mathcal{O}\ \overline{S}_0)\ . \qquad \text{(RL 18)}$$

Starting in an observable state, taking an action, an agent ends in an observable state.

$$\forall \overline{S}_t \in \overline{\textbf{State}}.\ \forall A_t \in \textbf{Action}.\ \mathcal{O}\ (\mathcal{T}\ \overline{S}_t\ A_t) \in \textbf{State} \qquad \text{(RL 19)}$$

A.2.2 Testing RL Algorithms. An initial Q-table Q_0 is defined for all state action pairs and all entries are initialized to zero.

$$\textbf{dom}\ Q_0 = \textbf{State} \times \textbf{Action} \qquad \text{(RL 20)}$$

$$\forall S_0 \in \textbf{State}.\ \forall A_0 \in \textbf{Action}.\ Q_0\ (S_0,\ A_0) = 0.0 \qquad \text{(RL 21)}$$

A policy π includes valid actions.

$$\forall Q_t \in \textbf{Q}.\ \forall S_t \in \textbf{State}.\ \forall A_t \in \textbf{Action}.\ \ [(\pi\ Q_t\ S_t)\ A_t > 0] \longrightarrow A_t \in \textbf{Action} \qquad \text{(RL 22)}$$

The probability distribution test for the ε-greedy algorithm.

$$
\begin{aligned}
&\forall Q_t \in \textbf{Q}.\ \forall S_t \in \textbf{State}. \\
&\quad \left[(\pi\ Q_t\ S_t)\ \gg= (\lambda A_t.\ A_t \neq \underset{A}{\arg\max}\ Q_t\ (S_t, A)) \right] = \text{Bern}\left(\varepsilon \cdot \frac{|\textbf{Action}| - 1}{|\textbf{Action}|} \right)
\end{aligned}
\qquad \text{(RL 23)}
$$

The implementation and the specification produce the same multivariate distribution.

$$
\begin{aligned}
&\forall Q_t \in \textbf{Q}.\ \forall \overline{S}_t \in \overline{\textbf{State}}.\ \forall A_t \in \textbf{Action}. \\
&\quad \texttt{update}\ (Q_t,\ \overline{S}_t,\ A_t) = [\![u]\!]_{bdl}\ (Q_t,\ \overline{S}_t,\ A_t)
\end{aligned}
\qquad \text{(RL 24)}
$$

The updates are normally distributed as they are affected by the noise in many executions of the agent.

$$
\begin{aligned}
&\forall Q_t \in \textbf{Q}.\ \forall \overline{S}_t \in \overline{\textbf{State}}.\ \forall A_t \in \textbf{Action}. \\
&\quad \left[\texttt{update}\ (Q_t,\ \overline{S}_t,\ A_t) \gg= g \right] = \left[[\![u]\!]_{bdl}\ (Q_t,\ \overline{S}_t,\ A_t) \gg= g \right] \\
&\quad \text{where } g = \lambda(Q_{t+1},\ \overline{S}_{t+1},\ A_{t+1}).\ Q_{t+1}\ (\mathcal{O}\ \overline{S}_t, A_t)
\end{aligned}
\qquad \text{(RL 25)}
$$

REFERENCES

Elmira Amirloo Abolfathi, Jun Luo, Peyman Yadmellat, and Kasra Rezaee. 2021. CoachNet: An Adversarial Sampling Approach for Reinforcement Learning. In *NeurIPS2019 Workshop on Safety and Robustness in Decision Making*. arXiv. https://doi.org/10.48550/ARXIV.2101.02649

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe reinforcement learning via shielding. In *Proc. AAAI Conference on Artificial Intelligence*, Vol. 32. AAAI Press. https://doi.org/10.1609/aaai.v32i1.11797

Rajeev Alur, Suguman Bansal, Osbert Bastani, and Kishor Jothimurugan. 2022. A Framework for Transforming Specifications in Reinforcement Learning. In *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday (Lecture Notes in Computer Science, Vol. 13660)*, Jean-François Raskin, Krishnendu Chatterjee, Laurent Doyen, and Rupak Majumdar (Eds.). Springer. https://doi.org/10.1007/978-3-031-22337-2_29

Joymallya Chakraborty, Suvodeep Majumder, and Tim Menzies. 2021. Bias in Machine Learning Software: Why? How? What to Do?. In *Proc. 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. ACM Press, 429––440. https://doi.org/10.1145/3468264.3468537

Koen Claessen and John Hughes. 2000. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *5th ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*. ACM Press, 268–279. https://doi.org/10.1145/357766.351266

Kristopher De Asis, J. Fernando Hernandez-Garcia, G. Zacharias Holland, and Richard S. Sutton. 2018. Multi-Step Reinforcement Learning: A Unifying Algorithm. In *Proc. 32nd AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence (AAAI'18/IAAI'18/EAAI'18)*. AAAI Press, Article 354, 8 pages. https://doi.org/10.1609/aaai.v32i1.11631

Richard A DeMillo, Richard J Lipton, and Frederick G Sayward. 1978. Hints on test data selection: Help for the practicing programmer. *Computer* 11, 4 (1978), 34–41. https://doi.org/doi:10.1109/C-M.1978.218136

Saikat Dutta, Jeeva Selvam, Aryaman Jain, and Sasa Misailovic. 2021. TERA: Optimizing Stochastic Regression Tests in Machine Learning Projects. In *Proc. 30th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual, Denmark) *(ISSTA 2021)*. ACM Press, 413––426. https://doi.org/10.1145/3460319.3464844

Grégoire Déletang, Jordi Grau-Moya, Miljan Martic, Tim Genewein, Tom McGrath, Vladimir Mikulik, Markus Kunesch, Shane Legg, and Pedro A. Ortega. 2021. Causal Analysis of Agent Behavior for AI Safety. arXiv. https://doi.org/10.48550/ARXIV.2103.03938

Nathan Fulton and André Platzer. 2018. Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. In *Proc. Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18)*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 6485–6492. https://doi.org/10.1609/aaai.v32i1.12107

Nathan Fulton and André Platzer. 2019. Verifiably safe off-model reinforcement learning. In *Proc. 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2019) (Lecture Notes in Computer Science, Vol. 11427)*. Springer, 413–430. https://doi.org/10.1007/978-3-030-17462-0_28

Javier García and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.* 16 (2015), 1437–1480. https://dl.acm.org/doi/10.5555/2789272.2886795

Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. 2013. *Bayesian data analysis*. CRC press.

Andrew Gelman and Cosma Rohilla Shalizi. 2013. Philosophy and the practice of Bayesian statistics. *Brit. J. Math. Statist. Psych.* 66, 1 (2013), 8–38. https://doi.org/10.1111/j.2044-8317.2011.02037.x

Richard G. Hamlet. 1977. Testing Programs with the Aid of a Compiler. *IEEE Transactions on Software Engineering* 3, 4 (1977), 279–290. https://doi.org/10.1109/TSE.1977.231145

Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. 2018. Logically-Correct Reinforcement Learning. *CoRR* abs/1801.08099 (2018). arXiv:1801.08099 http://arxiv.org/abs/1801.08099

Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial Attacks on Neural Network Policies. arXiv. https://doi.org/10.48550/ARXIV.1702.02284

Radoslav Ivanov, Taylor J Carpenter, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. 2020. Case study: verifying the safety of an autonomous racing car with a neural network controller. In *Proc. 23rd International Conference on Hybrid Systems: Computation and Control*. 1–7. https://doi.org/10.1145/3365365.3382216

Nils Jansen, Bettina Könighofer, Sebastian Junges, Alex Serban, and Roderick Bloem. 2020. Safe Reinforcement Learning Using Probabilistic Shields. In *Proc. 31st International Conference on Concurrency Theory (CONCUR 2020) (LIPIcs, Vol. 171)*, Igor Konnov and Laura Kovács (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 3:1–3:16. https://doi.org/10.4230/LIPIcs.CONCUR.2020.3

Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. 2019. A Composable Specification Language for Reinforcement Learning Tasks. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. 2021a. Compositional Reinforcement Learning from Logical Specifications. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 10026–10039.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. 2022. Specification-Guided Learning of Nash Equilibria with High Social Welfare. In *Proc. 34th International Conference on Computer Aided Verification (CAV 2022) (Lecture Notes in Computer Science, Vol. 13372)*, Sharon Shoham and Yakir Vizel (Eds.). Springer, 343–363. https://doi.org/10.1007/978-3-031-13188-2_17

Kishor Jothimurugan, Osbert Bastani, and Rajeev Alur. 2021b. Abstract Value Iteration for Hierarchical Reinforcement Learning. In *Proc. 24th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 130)*, Arindam Banerjee and Kenji Fukumizu (Eds.). PMLR, 1162–1170.

Sebastian Junges, Nils Jansen, Christian Dehnert, Ufuk Topcu, and Joost-Pieter Katoen. 2016. Safety-Constrained Reinforcement Learning for MDPs. In *Proc. 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2016) (Lecture Notes in Computer Science, Vol. 9636)*, Marsha Chechik and Jean-François Raskin (Eds.). Springer, 130–146. https://doi.org/10.1007/978-3-662-49674-9_8

Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. 1996. Reinforcement Learning: A Survey. *J. Artif. Intell. Res.* 4 (1996), 237–285. https://doi.org/10.1613/jair.301

John Kruschke. 2014. *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan.* Academic Press.

Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. 2017. Tactics of Adversarial Attack on Deep Reinforcement Learning Agents. In *Proc. 26th International Joint Conference on Artificial Intelligence (IJCAI'17).* AAAI Press, 3756–3762. https://doi.org/10.24963/ijcai.2017/525

Junrui Liu, Yanju Chen, Bryan Tan, Isil Dillig, and Yu Feng. 2022. Learning Contract Invariants Using Reinforcement Learning. In *Proc. 37th IEEE/ACM International Conference on Automated Software Engineering, (ASE 2022).* ACM Press, 63:1–63:11. https://doi.org/10.1145/3551349.3556962

Yuteng Lu, Weidi Sun, and Meng Sun. 2021. Mutation Testing of Reinforcement Learning Systems. In *Proc. 7th International Symposium on Dependable Software Engineering: Theories, Tools, and Applications (SETTA 2021) (Lecture Notes in Computer Science, Vol. 13071)*, Shengchao Qin, Jim Woodcock, and Wenhui Zhang (Eds.). Springer, 143–160. https://doi.org/10.1007/978-3-030-91265-9_8

Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. 2014. An empirical analysis of flaky tests. In *Proc. 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE-22)*, Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne D. Storey (Eds.). ACM Press, 643–653. https://doi.org/10.1145/2635868.2635920

Richard McElreath. 2020. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan* (2nd ed.). CRC Press.

Andrew W. Moore. 1990. Efficient memory-based learning for robot control. Ph.D. Thesis, University of Cambridge.

Tuomas Oikarinen, Wang Zhang, Alexandre Megretski, Luca Daniel, and Tsui-Wei Weng. 2021. Robust Deep Reinforcement Learning through Adversarial Loss. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, 26156–26167.

Qi Pang, Yuanyuan Yuan, and Shu Wang. 2022. MDPFuzz: Testing Models Solving Markov Decision Processes. In *Proc. 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2022).* ACM Press, 378–390. https://doi.org/10.1145/3533767.3534388

Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. 2019. Chapter Six - Mutation Testing Advances: An Analysis and Survey. Advances in Computers, Vol. 112. Elsevier, 275–378. https://doi.org/10.1016/bs.adcom.2018.03.015

Norman Ramsey and Avi Pfeffer. 2002. Stochastic lambda calculus and monads of probability distributions. In *Proc. 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2002)*, John Launchbury and John C. Mitchell (Eds.). ACM Press, 154–165. https://doi.org/10.1145/503272.503288

Vincenzo Riccio, Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. DeepMetis: Augmenting a Deep Learning Test Set to Increase its Mutation Score. In *36th IEEE/ACM International Conference on Automated Software Engineering (ASE 2021).* 355–367. https://doi.org/10.1109/ASE51524.2021.9678764

Andrea Romdhana, Mariano Ceccato, Alessio Merlo, and Paolo Tonella. 2022. IFRIT: Focused Testing through Deep Reinforcement Learning. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST).* 24–34. https://doi.org/10.1109/ICST53961.2022.00013

Avraham Ruderman, Richard Everett, Bristy Sikder, Hubert Soyer, Jonathan Uesato, Ananya Kumar, Charlie Beattie, and Pushmeet Kohli. 2019. Uncovering Surprising Behaviors in Reinforcement Learning via Worst-case Analysis. In *Safe Machine Learning workshop at ICLR 2019.*

G. A. Rummery and M. Niranjan. 1994. On-line Q-learning Using Connectionist Systems. *Technical Report CUED/F-INFENF/TR* (1994). https://cir.nii.ac.jp/crid/1573668924277769344

Stuart J Russell and Peter Norvig. 2016. *Artificial intelligence: A modern approach.* Pearson Education Limited.

John Schulman, Filip Wolski, Pra fulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *ArXiv* abs/1707.06347 (2017). https://doi.org/10.48550/arXiv.1707.06347

Jianzhong Su, Hong-Ning Dai, Lingjun Zhao, Zibin Zheng, and Xiapu Luo. 2022. Effectively Generating Vulnerable Transaction Sequences in Smart Contracts with Reinforcement Learning-guided Fuzzing. In *Proc. 37th IEEE/ACM International Conference on Automated Software Engineering (ASE 2022)*. ACM Press, 36:1–36:12. https://doi.org/10.1145/3551349.3560429

Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic testing for deep neural networks. In *Proc. 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. 109–119. https://doi.org/10.1145/3238147.3238172

Richard S. Sutton. 1988. Learning to Predict by the Methods of Temporal Differences. *Mach. Learn.* 3, 1 (1988), 9–44. https://doi.org/10.1023/A:1022633531479

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (2nd ed.). The MIT Press.

Martin Tappler, Filip Cano Córdoba, Bernhard K. Aichernig, and Bettina Könighofer. 2022a. Search-Based Testing of Reinforcement Learning. In *Proc. Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)*, Lud De Raedt (Ed.). International Joint Conferences on Artificial Intelligence Organization, 503–510. https://doi.org/10.24963/ijcai.2022/72

Martin Tappler, Stefan Pranger, Bettina Könighofer, Edi Muskardin, Roderick Bloem, and Kim G. Larsen. 2022b. Automata Learning Meets Shielding. In *Proc. 11th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles (ISoLA 2022) (Lecture Notes in Computer Science, Vol. 13701)*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer, 335–359. https://doi.org/10.1007/978-3-031-19849-6_20

Rosalia Tufano, Simone Scalabrino, Luca Pascarella, Emad Aghajani, Rocco Oliveto, and Gabriele Bavota. 2022. Using Reinforcement Learning for Load Testing of Video Games. In *Proc. IEEE/ACM 44th International Conference on Software Engineering (ICSE 2022)*. ACM Press. https://doi.org/10.1145/3510003.3510625

Uraz Cengiz Türker, Robert M. Hierons, Mohammad Reza Mousavi, and Ivan Y. Tyukin. 2021. Efficient state synchronisation in model-based testing through reinforcement learning. In *Proc. 36th IEEE/ACM International Conference on Automated Software Engineering (ASE 2021)*. 368–380. https://doi.org/10.1109/ASE51524.2021.9678566

Harm Van Seijen, Hado Van Hasselt, Shimon Whiteson, and Marco Wiering. 2009. A theoretical and empirical analysis of Expected SARSA. In *Proc. Symposium on Adaptive Dynamic Programming and Reinforcement Learning*. IEEE, 177–184. https://doi.org/10.1109/ADPRL.2009.4927542

Harsh Vardhan and Janos Sztipanovits. 2021. Rare Event Failure Test Case Generation in Learning-Enabled-Controllers. In *2021 6th International Conference on Machine Learning Technologies* (Jeju Island, Republic of Korea) *(ICMLT 2021)*. ACM Press, 34–40. https://doi.org/10.1145/3468891.3468897

Mahsa Varshosaz, Mohsen Ghaffari, Einar Broch Johnsen, and Andrzej Wasowski. 2023. *Formal Specification and Testing for Reinforcement Learning (Supplementary Material)*. https://doi.org/10.5281/zenodo.8083298

Christopher John Cornish Hellaby Watkins. 1989. Learning from delayed rewards. (1989).

Min Wu, Matthew Wicker, Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. 2020. A game-based approximate verification of deep neural networks with provable guarantees. *Theor. Comput. Sci.* 807 (2020), 298–329. https://doi.org/10.1016/j.tcs.2019.05.046

Shaohua Zhang, Shuang Liu, Jun Sun, Yuqi Chen, Wenzhi Huang, Jinyi Liu, Jian Liu, and Jianye Hao. 2021. FIGCPS: Effective Failure-inducing Input Generation for Cyber-Physical Systems with Deep Reinforcement Learning. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 555–567. https://doi.org/10.1109/ASE51524.2021.9678832

Yan Zheng, Yi Liu, Xiaofei Xie, Yepang Liu, Lei Ma, Jianye Hao, and Yang Liu. 2021. Automatic Web Testing Using Curiosity-Driven Reinforcement Learning. In *Proc. IEEE/ACM 43rd International Conference on Software Engineering (ICSE 2021)*. ACM Press, 423–435. https://doi.org/10.1109/ICSE43902.2021.00048

Amirhossein Zolfagharian, Manel Abdellatif, Lionel C. Briand, Mojtaba Bagherzadeh, and Ramesh S. 2023. A Search-Based Testing Approach for Deep Reinforcement Learning Agents. *IEEE Transactions on Software Engineering* (2023), 1–22. https://doi.org/10.1109/TSE.2023.3269804 To appear.

Received 2023-03-01; accepted 2023-06-27

Symbolic State Partitioning for Reinforcement Learning

Anonymous

Abstract. Tabular reinforcement learning methods cannot operate directly on continuous state spaces. One solution for this problem is to partition the state space. A good partitioning enables generalization during learning and more efficient exploitation of prior experiences. Consequently, the learning process becomes faster and produces more reliable policies. However, partitioning introduces approximation, which is particularly harmful in the presence of nonlinear relations between state components. An ideal partition should be as coarse as possible, while capturing the key structure of the state space for the given problem. This work extracts partitions from the environment dynamics by symbolic execution. We show that symbolic partitioning improves state space coverage with respect to environmental behavior and allows reinforcement learning to perform better for sparse rewards. We evaluate symbolic state space partitioning with respect to precision, scalability, learning agent performance and state space coverage for the learnt policies.

Keywords: Reinforcement Learning · Symbolic Execution · State Space Partitioning.

1 Introduction

Reinforcement learning is a form of active learning, where an agent learns to make decisions to maximize a reward signal. The agent interacts with an environment and takes actions based on its current state. The environment rewards the agent, which uses the reward value to update its decision-making policy. Reinforcement learning has applications in many domains: robotics [19], gaming [40], electronics [14], and healthcare [50]. This method can automatically synthesize controllers for many challenging control problems [39], however dedicated approximation techniques, hereunder deep learning, are needed for continuous state spaces. Unfortunately, despite many spectacular success with continuous problems, Deep Reinforcement Learning suffers from low explainability and lack of convergence guarantees. At the same time discrete (tabular) learning methods have been shown to be more explainable [23, 33, 47, 51] and to yield policies for which it is easier to assure safety [12, 16, 44]; for instance using formal verification [1, 17, 41]. Thus, finding a good state-space representation for discrete learning remains an active research area [3, 8, 15, 22, 24, 31, 48].

To adapt a continuous state space for discrete learning, one exploits partial observability, and merges regions of the state space into discrete partitions, each representing a subset of the states of the agent. Ideally, all states in a partition should capture meaningful aspects of the environment—best if they share the same optimal action in the optimal policy. Consequently, a good partitioning depends on the problem at hand. For instance, in safety critical environments, it is essential to identify small "singularities"—regions that require special handling—even if they are very small. Otherwise, if such regions are included in larger partitions, the control policy will not be able to distinguish them from the surrounding, leading to high variance at operation time and slow convergence of learning.

The trade-off between the size of the partitioning and the optimality and convergence of reinforcement learning remains a challenge [3, 8, 22, 24, 31, 48]. Policies obtained for coarse-grained partitionings are unreliable. Large fine-grained partitionings make reinforcement learning slow. The dominant methods are *tiling* and *vector quantization* [22, 24, 31, 48]; both are not adaptive to the structure of the state space. They ignore nonlinear dependencies between state components even though quadratic behaviors are common in control systems. So far, the shape of the state partitions has hardly been studied.

In this work, we investigate the use of *symbolic execution* to extract approximate adaptive partitionings that reflect the problem dynamics. *Symbolic execution* [7, 18] is a classic foundational technique for dynamic program analysis originating in the software engineering and deductive verification research, commonly used for test input generation [45] and in interactive theorem provers (e.g. [2]). A symbolic executor generates a set of *path conditions* (PC), constraints that must hold for each execution path of the program to be taken. These conditions partition the state space of the executed program into groups that share the same execution path. Our hypothesis is that *the path conditions obtained by symbolic execution of an environment model (the step and reward functions) provide a useful state space partition for reinforcement learning.* The branches in the environment program likely reflect important aspects of the problem dynamics that should be respected by an optimal policy. We test this hypothesis by:

- Defining a symbolic partitioning method and establishing its basic theoretical properties. This method, SymPar, is adaptive to the problem semantics, general (not developed for a specific problem), and automatic (given a symbolically executable environment program).
- Implementing the method on top of the Symbolic PathFinder, an established symbolic executor for Java programs (JVM programs) [32]
- Evaluating SymPar empirically against other offline and online partitioning approaches, and against deep reinforcement learning methods. The experiments show that symbolic partitioning can allow the agent to learn better policies than with the baselines.

To the best of our knowledge, this is the first ever attempt to use symbolic execution to breath semantic knowledge into an otherwise statistical reinforcement learning process. We see it as an interesting case of a transfer of concepts from software engineering and formal methods to machine learning. It does break with the tradition of reinforcement learning to treat environments as black-boxes. It is however consistent with common practice of using reinforcement learning for software defined problems and with pre-training robotic agents in simulators, as software problems and simulators are amenable to symbolic execution.

The paper proceeds as follows. Section 2 reviews the relevant state of the art. Section 3 recalls the required preliminaries and definitions. Our state space abstraction method is detailed in Sect. 4. In Sect. 5 we present the evaluation design, and then discuss the experiment results (Sect. 6). Finally, Sect. 7 concludes the paper and presents future work.

2 Related Work

We study partitioning, or a discrete abstraction, of the state space in reinforcement learning by mapping from a continuous state space to a discrete one or by aggregating discrete states. To the best of our knowledge, the earliest use of partitioning, was the BOXES system [25]. The Parti-game algorithm [29] automatically partitions state spaces but applies only to tasks with known goal regions and requires a greedy local controller. While tile coding is a classic method for partitioning [4], it often demands extensive engineering effort to avoid misleading the agent with suboptimal partitions. Lanzi and coauthors [21] extended learning classifier systems to use tile coding. Techniques such as vector quantization [22,24,31,48] and decision trees [37,42,49] lack adaptability to the properties of the state space and may overlook non-linear dependencies among state components. Techniques that gradually refine a coarse partitioning during the learning process [3,8,15,24,48] are time-intensive, and require generating numerous partitions to achieve better approximations near the boundaries of nonlinear functions. Unlike other methods, SymPar incurs no direct learning cost (it is offline), requires no engineering effort (it is automated), and is not problem specific in contrast to some of the existing techniques (it is general). It produces a partitioning that effectively captures non-linear dependencies as well as narrow partitions, without incurring additional costs or increasing the number of partitions at the boundaries.

The concept of bisimulation metrics [10,11] defines two states as being behaviorally similar if they (1) yield comparable immediate rewards and (2) transition to states that are behaviorally aligned. Bisimulation metrics have been employed to reduce the dimensionality of state spaces through the aggregation of states. However, they have not been extensively explored due to their high computational costs. Moreover, note that bisimulation-minimization-based state-space-abstraction is too fine-grained for the problem at hand. It requires that any states lumped together exhibit the same behavior. This is an unnecessary constraint from the reinforcement learning perspective, which takes no preference over behaviors provided that they lead to the same long-term reward. As long as the same long-term reward estimate is expected for the same (best) local action in two states, it is theoretically sufficient for the two states to be lumped together. For this reason it is worth exploring weaker principles than bisimulation metrics for reducing dimensionality.

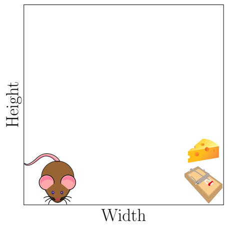


Fig. 1. Navigation environment. A mouse agent in a continuous rectangular board needs to find the cheese, while not stepping on the trap.

3 Background

Reinforcement Learning. A Partially Observable Markov Decision Process is a tuple $\mathcal{M} = (\overline{\mathcal{S}}, \overline{\mathcal{S}}_0, \mathcal{A}, \mathcal{S}, \mathcal{O}, \mathcal{T}, \mathcal{R}, \mathcal{F})$, where $\overline{\mathcal{S}}$ is a set of states, $\overline{\mathcal{S}}_0 \in \text{pdf}\, \overline{\mathcal{S}}$ is a probability density function for initial states, $\mathcal{A}$ is a finite set of actions, $\mathcal{S}$ is a finite set of observable states, $\mathcal{O} \in \overline{\mathcal{S}} \to \mathcal{S}$ is a total observation function, $\mathcal{T} \in \overline{\mathcal{S}} \times \mathcal{A} \to \text{pdf}\, \overline{\mathcal{S}}$ is the transition probability function, $\mathcal{R} \in \overline{\mathcal{S}} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\mathcal{F} \in \mathcal{S} \to \{0, 1\}$ is a predicate defining final states. The task is to find a policy $\pi : \mathcal{S} \to \text{Dist}(\mathcal{A})$ that maximizes the expected accumulated reward [39].

Example 1 *Consider a room with dimensions $W \times H$, a mouse in its bottom-left corner, a mousetrap in the bottom-right corner, and a piece of cheese next to the mousetrap (Fig. 1). The mouse moves with a fixed velocity in four directions: up, down, left, right. Its goal is to find the cheese but avoid the trap. The states $\overline{\mathcal{S}}$ are ordered pairs representing the mouse's position in the room. The set of initial states $\overline{\mathcal{S}}_0$ is fixed to $(1, 1)$, a Dirac distribution. We define the actions as the set of all possible movements for the mouse: $\mathcal{A} = \{(d, v) : d \in \mathcal{D}, v \in \mathcal{V}\}$, where $\mathcal{D} = \{U, D, R, L\}$ and $\mathcal{V} = \{r_1, r_2, \ldots, r_n \mid r_i \in \mathbb{R}^+\}$. $\mathcal{S}$ can be any partitioning of the room space and $\mathcal{O}$ is the map from the real position of the mouse to the partition containing it. Our goal is to find the partitioning, i.e., $\mathcal{S}$ and $\mathcal{O}$. The reward function $\mathcal{R}$ is zero when mouse finds the cheese, and -1 otherwise. For simplicity, we let the environment be deterministic, so $\mathcal{T}$ is a deterministic movement of the mouse from a position by a given action to a new position. The final state predicate $\mathcal{F}$ holds for the cheese and trap positions and not otherwise.*

Symbolic Execution is a program analysis technique that systematically explores program behaviors by solving symbolic constraints obtained from conjoining the program's branch conditions [18]. Symbolic execution extends normal execution by running the basic operators of a language using symbolic inputs (variables) and producing symbolic formulas as output. A symbolic execution of a program produces a set of *path conditions*—logical expressions that encode conditions on the input symbols to follow a particular path in the program.

$$\begin{array}{rrcl}
\text{S-ASSIGN} & (x := e, \sigma, k, \phi) & \rightarrow & (\text{skip}, \sigma[x \mapsto \sigma e], k, \phi) \\
\text{S-IF-T} & (\text{if } b\ s_1 \text{ else } \ s_2, \sigma, k, \phi) & \rightarrow & (s_1, \sigma, k, \phi \wedge \ \sigma b) \\
\text{S-IF-F} & (\text{if } b\ s_1 \text{ else } \ s_2, \sigma, k, \phi) & \rightarrow & (s_2, \sigma, k, \phi \wedge \ \sigma \neg b) \\
\text{S-WHILE-T} & (\text{while } b\ s, \sigma, k, \phi) & \rightarrow & (s\ ;\ \text{while } b\ s, \sigma, k, \phi \wedge \ \sigma b) \\
\text{S-WHILE-F} & (\text{while } b\ s, \sigma, k, \phi) & \rightarrow & (\text{skip}\ ;\ \sigma, k, \phi \wedge \ \sigma \neg b) \\
\text{S-SMLP} & (x \sim \text{rnd}, \sigma, k, \phi) & \rightarrow & (\text{skip}, \sigma[x \mapsto y_k], k+1, \phi)
\end{array}$$

Fig. 2. Symbolic execution rules for an idealized probabilistic language. Each judgement is a quadruple: the program, the symbolic store (σ), the sample index (k), the current path condition (ϕ).

For a program over input arguments $I = \{v_1, v_2, \ldots, v_k\}$, a path condition $\phi \in PC(I')$ is a quantifier free logical formula defined on $I' = \{\vartheta_1, \vartheta_2, \ldots, \vartheta_k\}$, where each symbolic variable ϑ_i corresponds to v_i.

We sketch a definition of symbolic execution for a minimal language. [5] provides more details. Let V be the set of program variables and Ops be a set of arithmetic operations, $x \in V$, $n \in \mathbb{R}$, and $op \in$ Ops. We consider programs generated by the following grammar:

$$\begin{aligned}
&e ::= x \mid n \mid op(e_1, \ldots, e_n) \\
&b ::= \text{True} \mid \text{False} \mid b_1 \text{ AND } b_2 \mid b_1 \text{ OR } b_2 \mid \neg b \mid b_1 \leq b_2 \mid e_1 < e_2 \\
&s ::= x := e \mid x \sim \text{rnd} \mid s_1; s_2 \mid \text{if } b\ s_1 \text{ else } s_2 \mid \text{while } b\ s \mid \text{skip}
\end{aligned} \tag{1}$$

A symbolic store, denoted by σ maps input program variables $I \subseteq V$ to expressions, generated by productions e above. An update to a symbolic store is denoted $\sigma[x := e]$. It replaces the entry for variable x with the expression e. An expression can be interpreted in a symbolic store by applying (substituting) its mapping to the expression syntax (written $e\sigma$).

Figure 2 gives the symbolic execution rules for the above language, in terms of traces (it computes a path condition ϕ for a terminating trace). In the reduction rules, ϕ represents the path condition and k denotes the sampling index. The first rule defines the symbolic assignment; An assignment does not change the path conditions, but updates the symbolic store σ. When encountering conditional statements, the symbolic executor splits into two branches. For the true case (rule S-IF-T) the path condition is extended with the head condition of the branch, for the false case (S-IF-F), the path condition is extended with the negation of the branch condition. Similarly, for a *while* loop two branches are generated, with an analogous effect on path conditions. The last rule executes the randomized sampling statement. It simply allocates a new symbolic variable y_k for the unknown result of sampling, and advances the sampling index [46]. Figure 4 shows the path conditions obtained by applying similar rules to above for the code to the left (Fig. 3). The first path condition $PC^{(U,1)}$ corresponds to the branch where condition `d==1` is true in the program.

```
W = 10 # Width
H = 10 # Height
def step(x, y, d, v):
  if d == 1: # UP
    if y < H:
      return x, y+v
  if d == 2: # DOWN
    if y > 1:
      return x, y-v
  if d == 3: # LEFT
    if x > 1:
      return x-v, y
  if d == 4: # RIGHT
    if x < W:
       return x+v, y
  return x, y

def reward(x, y, d, v):
  if x == W:
    if y == 2:
      return 0.0 # Cheese
    elif y == 1:
      return -1000.0 # Mousetrap
  return -1.0
```

Fig. 3. The environment program for the navigation problem (Fig. 1)

$PC^{(U,1)}$
$y < 10\ \wedge x = 10\ \wedge y + 1 = 2$
$y < 10\ \wedge x = 10\ \wedge y + 1 \neq 2$
$y < 10\ \wedge x \neq 10$
$y \geq 10\ \wedge x = 10$
$y \geq 10\ \wedge x \neq 10$

$PC^{(D,1)}$
$y > 1\ \wedge x = 10\ \wedge y - 1 = 2$
$y > 1\ \wedge x = 10\ \wedge y - 1 \neq 2\ \wedge y - 1 = 1$
$y > 1\ \wedge x = 10\ \wedge y - 1 \neq 2\ \wedge y - 1 \neq 1$
$y > 1\ \wedge x \neq 10$
$y \leq 1\ \wedge x = 10\ \wedge y = 1$
$y \leq 1\ \wedge x \neq 10$

$PC^{(L,1)}$
$x > 1\ \wedge x - 1 \neq 10$
$x \leq 1\ \wedge x \neq 10$

$PC^{(R,1)}$
$x < 10\ \wedge x + 1 = 10\ \wedge y = 2$
$x < 10\ \wedge x + 1 = 10\ \wedge y \neq 2\ \wedge y = 1$
$x < 10\ \wedge x + 1 = 10\ \wedge y \neq 2\ \wedge y \neq 1$
$x < 10\ \wedge x + 1 \neq 10$
$x \geq 10\ \wedge x = 10\ \wedge y = 2$
$x \geq 10\ \wedge x = 10\ \wedge y \neq 2\ \wedge y = 1$
$x \geq 10\ \wedge x = 10\ \wedge y \neq 1\ \wedge y \neq 1$

Fig. 4. Path conditions collected by symbolic execution for the program of Fig. 3

The above rules can be used to prove basic properties of the symbolic execution. For example, as each path condition contains conjunction of *different* branch conditions in a program, the path conditions of the same program are mutually exclusive [5].

There exist practical symbolic executors for full scale programming languages. Even though we defined the concept at the level of syntax, the two most popular symbolic executors operate on compiled bytecode [6,32]. In presence of loops and recursion, symbolic execution does not terminate. To halt symbolic execution, we can set a predefined timeout, an iteration limit, or a program statement limit. This produces an approximation of the set of path conditions.

4 Partitioning Using Symbolic Execution

We present the idea using a single agent with the environment modeled as a computer program. The program (*Env*) is implementing a single step-transition ($\mathcal{T}$) in the environment with the corresponding reward ($\mathcal{R}$). We use symbolic execution to analyze the environment program *Env*, then partition the state space using the obtained path conditions. The partitioning serves as the observation function $\mathcal{O}$. The entire process is automatic and generic—we follow the same procedure for all problems.

Example 2 *Figure 3 shows the environment program for the* 10×10 *navigation problem (Example 1). For simplicity, we assume the agent can move one unit in each direction, so* $\mathcal{V} = \{1\}$ *and* $\mathcal{A} = \{U, D, R, L\} \times \mathcal{V}$*. The path conditions in Fig. 4 are obtained by symbolically executing the step and reward functions using symbolic inputs* x *and* y *and a concrete input from* $\mathcal{A}$*. Using path conditions in partitioning requires a translation from the symbolic executor syntax into the programming language used for implementing partitioning, as the executor will generate abstract value names.*

A good partitioning maintains the Markov property, so that the same action is optimal for all unobservable states abstracted by the same partition. Unfortunately, this means that a good partitioning can be selected only once we know a good policy—after learning. To overcome this, SymPar heuristically bundles states into the same partition if they induce the same execution path in the environment program. We use an off-the-shelf symbolic executor to extract all possible path conditions from *Env*, by using $\overline{\mathcal{S}}$ as symbolic input and actions from $\mathcal{A}$ as concrete input. The result is a set PC of path conditions for each concrete action: $PC = \{PC^{a_0}, PC^{a_1}, \ldots, PC^{a_m}\}$, where $PC^a = \{PC^a_0, PC^a_1, \ldots, PC^a_{k_a}\}$. The set PC^a contains the path conditions computed for action a, and k_a is the number of all path conditions obtained by running *Env* symbolically, for a concrete action a.

It is instructive to reflect on the partitionings obtained this way, contrasting them with bisimulation-metrics-based lumping. First, bisimulation minimization is essentially impossible for environment models expressed as computer programs—in our case a Markov Chain would be required. However, obtaining an explicit Markov Chain representation for any non-trivial environment is infeasible as of today. We would thus need symbolic bisimulation-minimization methods. In contrast, symbolic execution performs a mixed syntactic-semantic decomposition of the input state space by path conditions. This process is chiefly driven by the syntax of the program, yet it is semantically informed via the branch conditions. The obtained partitioning might be unsound from the Markov Chain bisimulation perspective, but it tends to produce coarser partitionings than bisimulation minimization. Symbolic execution also has the advantage of scaling better on environments expressed as computer programs.

Running the environment program for any concrete state satisfying a condition PC^a_i with action a will execute the same program path. However, the partitioning for reinforcement learning needs to be action independent (the same for all actions). So the path conditions cannot be used as partitions, as they are. Consider $PC^{a_1}_i \in PC^{a_1}$ and $PC^{a_2}_j \in PC^{a_2}$, arbitrary path conditions for some actions a_1, a_2. To make sure that the same program path will be taken from a concrete state for both actions, we need to create a partition that corresponds to the intersection of both path conditions: $PC^{a_1}_i \wedge PC^{a_2}_j$. In general, each set in PC defines a

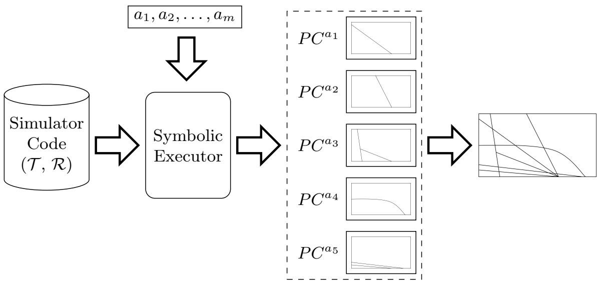


Fig. 5. Overview of SymPar.

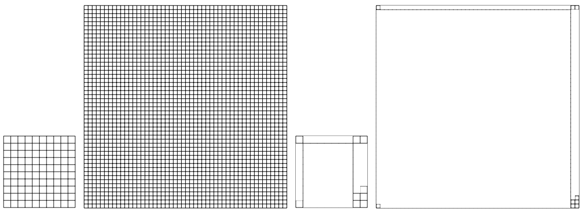


Fig. 6. Using tile coding (left) and SymPar (right) for 10×10 and 50×50 navigation

partitioning of the state space for a different actions. To make them compatible, we need to compute the unique coarsest partitioning finer than those induced by the path conditions for any action, which is a standard operation in order theory [9]. In this case, this amounts to computing all intersections of partitions for all actions, and removing the empty intersections using an SMT check.

The process of symbolic state space partitioning is summarized in Fig. 5 and Alg. 1. SymPar executes the environment program symbolically. For each action, a set of path conditions is collected. In the figure, $|\mathcal{A}| = 5$ and, accordingly, five sets of path conditions are collected (shown as rectangles). Each rectangle is divided into a group of regions, each of which maps to a path condition. Thus, the rectangles illustrate the state space that is discretized by the path conditions. Note that the border of each region can be a unique path condition (an expression with equality relation) or a part of neighbour regions (an expression with inequality relation). The final partitioning is shown as another rectangle that contains the overlap between the regions from the previous step.

Example 3 *Figure 6 (left) shows the partitioning of the Navigation problem using tile coding [4] for two room sizes. Numerous cells share the same policy, prompting the question of why they should be divided. SymPar achieves a much*

Algorithm 1 Partitioning with Symbolic Execution (SymPar)

Input: Env, $\mathcal{A}$

Output: $\mathcal{P}$ (a partitioning of $\overline{\mathcal{S}}$)

```
1: $PC \leftarrow \emptyset$
2: for $a \in \mathcal{A}$ do
3:     $PC^a, \Psi \leftarrow$ SymExec ($Env$, symbolic $\overline{\mathcal{S}}$, concrete $a$)    // $\Psi$ is the set of sampling variables
4:     Add distribution support constraints for all variables $\overline{\mathcal{S}} \in \Psi$ to $PC^a$
5:     Existentially quantify all sampling variables in $PC^a$    // may introduce overlaps of conditions
6:     $PC'^a \leftarrow \emptyset$
7:     for $p, q \in PC^a$ do if SAT ($p \wedge q$) $PC'^a \leftarrow PC'^a \cup \{p \wedge q\}$
8:     $PC^a \leftarrow PC'^a$
9: $\mathcal{P} \leftarrow PC^{\mathcal{A}[0]}$
10: for $a \in \mathcal{A} - \{\mathcal{A}[0]\}$ do
11:     $\mathcal{P}' \leftarrow \emptyset$
12:     for $p \in \mathcal{P}, q \in PC^a$ do if SAT ($p \wedge q$) then $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{p \wedge q\}$
13:     $\mathcal{P} \leftarrow \mathcal{P}'$
14: return $\mathcal{P}$
```

coarser partitioning than the initial tiling, by discovering that for many tiles the dynamics is the same (right).

We handle stochasticity of the environment by allowing environment programs to be probabilistic and then following rule S-SMLP in symbolic execution (Fig. 2). We introduce a new symbolic variable whenever a random variable is sampled in the program [20, 46]. Consequently, our path conditions also contain these sampling variables. To make the process more reliable, one can generate constraints, limiting them to the support of the distribution. For example, for sampling from a uniform distribution $U[\alpha, \beta]$, the sampling variable n_v is subject to two constraints: $n_v \geq \alpha$ and $n_v \leq \beta$. In order to be able to compute the partitioning over state variables only, as above, we existentially quantify the sampling variables out. This may introduce overlaps between the conditions, so we compute their intersection at this stage before proceeding.

Since the entire setup uses logical representations and an SMT solver, we exploit it further to generate witnesses for all partitions, even the smallest ones. We use them to seed reinforcement learning episodes, ensuring that each partition has been visited at least once. Consequently the agent is certain to learn from all the paths in the environment program. This can be further improved by constraining with a reachability predicate (not used in our examples).

Properties of SymPar. SymPar on the specifics of the environment implementation. Distinct implementations of the simulated environment may result in different partitioning outcomes for a given problem. On the other hand, the outcome is independent of the size of state space. Recall that in Fig. 6 (right) the number of partitionings is the same for the small and the large room.

To argue for the totality of our partitions, we first need to discuss completeness of the symbolic execution. De Boer and Bonsangue prove that symbolic execution is complete [5]. This result allows us to state the following theorem:

Theorem 1. *The obtained partitioning $\mathcal{P}$ is total for loop-free programs: $\forall \overline{s} \in \overline{\mathcal{S}}\ \exists!\, \mathcal{P}_0 \in \mathcal{P} \cdot\ \overline{s} \in \mathcal{P}_0$.*

In practice, symbolic execution is not complete, as most interesting programs with loops have infinitely many symbolic paths. This is easily overcome, by adding an extra path conditions, the complement of the computed ones, to cover for the unexplored paths.

The cost of SymPar amounts to exploring all paths in the program symbolically and then computing the coarsest partitioning. The symbolic execution involves generating a number of paths exponential in the number of branch points in the program (and at each branch one needs to solve an SMT problem—which is in principle undecidable, but works well for many practical problems). A practical approach is to bound the depth of exploration of paths by symbolic executor for more complex programs. Computing the coarsest partitioning requires solving $|\mathcal{P}|^{|\mathcal{A}|}$ number of SMT problems where $|\mathcal{P}|$ is the upper bound on the number of partitions (symbolic paths) and $|\mathcal{A}|$ is the number of actions. The other operations involved in this process such as computing and storing the path conditions in the required syntax are polynomial and efficient in practice.

Theorem 2. *Let PC^a be the set of path conditions produced by SymPar for each of the actions $a \in \mathcal{A}$. The size of the final partitioning $\mathcal{P}$ returned by SymPar is bounded from below by each $|PC^a|$ and from above by $\prod_{a\in\mathcal{A}} |PC^a|$.*

The theorem follows from the fact that $\mathcal{P}$ is finer than any of the PC^as and the algorithm for computing the coarsest partitioning finer then a set of partitionings can in the worst case intersect each partition in each set PC^a with all the partitions in the partitionings of the other actions.

Note that SymPar is a heuristic and approximate method. To appreciate this, define the optimal partitioning to be the unique partitioning in which each partition contains all states with the same action in the optimal policy (the optimal partition is an inverse image of the optimal policy for all actions). The partitionings produced by SymPar are neither always coarser or always finer than the optimal one. This can be shown with simple counterexamples. For an environment with only one action, the optimal partitioning has only one partition as the optimal policy maps the same action for all states. But Sympar will generate more than one partition (a finer partitioning) if the simulation program contains branching. For problems without branching in the simulator such as cart pole problem, Sympar produces only one partition. However, the optimal partitioning contains more than one partition as optimal actions for all states in the state space are not the same. To understand the significance of this approximation in practice, we evaluate SymPar empirically against the existing methods.

5 Evaluation Setup

The partitioning of the state space faces a trade-off: on one hand, the granularity of the partitioning should be fine enough to distinguish crucial differences between states in the state space. On the other hand, this granularity should be chosen to avoid a combinatorial explosion, where the number of possible partitions becomes unmanageably large. Achieving this balance is essential for efficient and effective learning. In this section, we explore this trade-off and evaluate the performance of our implementation in SymPar empirically. To this aim, we address the following research questions:

RQ1 *How much smaller are the SymPar partitionings compared to other methods, and how do these smaller partitionings impact learning performance?*
RQ2 *How does the granularity of the partitioning affect the learning performance?*
RQ3 *How does SymPar scale with increasing state space sizes?*
RQ4 *How well does SymPar group together behaviorally similar states?*

We compare SymPar to CAT-RL [8] (online partitioning) and with tile coding techniques (offline partitioning) for different examples [39]. Tile coding is a classic technique for partitioning. It splits each feature of the state into rectangular tiles of the same size. Although there are new problem specific versions, we opt for the classic version due to its generality.

To answer **RQ1**, we measure (a) the *size of partitioning*, (b) the *failure and success rates* and (c) the *accumulated reward* during learning. Being offline, our approach is hard to compare with online methods, since the different parameters may affect the results. Therefore, we separate the comparison with offline and online algorithms. For offline algorithms, we first find the number of abstract states using SymPar and partition the state space using tile coding accordingly (i.e., the number of tiles is set to the smallest square number greater than the number of partitions from SymPar). Then, we use standard Q-learning for these partitionings, and compare their performance. For online algorithms, we compute the running time for SymPar and its learning process, run CAT-RL for the same amount of time, and compare their performance. Obviously, if the agent observes a failing state, the episode stops. This decreases the running time. Finally, we compare the accumulated reward for SymPar with well-known algorithms DQN [27], A2C [26], PPO [36], using the Stable-Baselines3 implementations[1] [34]. These comparisons are done for two complementary cases: (1) randomly selected states and (2) states that are less likely to be chosen by random selection. The latter are identified by SymPar's partitioning. We sample states from different partitions obtained by SymPar and evaluate the learning process by measuring the accumulated reward.

To answer **RQ2**, we create different learning problems with various partitioning granularity by changing the search depth for the symbolic execution. We then compare the maximum accumulated reward of the learned policy to gain an understanding of the learning performance for the given abstraction.

[1] https://github.com/DLR-RM/stable-baselines3

	SymPar					**Tile Coding**				
	$\lvert\mathcal{S}\rvert$	**Succ**	**Fail**	$\mathbf{T_{out}}$	**Opt**	$\lvert\mathcal{S}\rvert$	**Succ**	**Fail**	$\mathbf{T_{out}}$	**Opt**
	(#)	(%)	(%)	(%)	(%)	(#)	(%)	(%)	(%)	(%)
Simple Maze	33	74.9	<0.1	25.0	5.0	10^4	6.0	7.1	86.9	0.0
MA Navigation	130	5.8	82.6	11.6	0.0	10^4	0.0	99.6	0.4	0.0
Wumpus World 1	73	18.4	0.0	81.6	2.1	8^4	9.6	0.0	90.4	0.0
Wumpus World 2	52	37.3	22.9	39.8	4.2	64	19.1	33.2	47.7	0.0
Navigation	51	13.2	4.8	82.0	<0.1	64	0.0	0.0	100.0	0.0
Braking Car	81	89.1	10.9	0.0	29.8	81	82.0	18.0	0.0	14.9
Mountain Car	70	82.2	0.0	17.8	61.3	81	59.4	0.0	40.6	14.7
Random Walk	184	61.2	11.1	27.7	44.0	196	6.5	5.1	88.4	<0.1

Table 1. Partitioning size and learning performance. Discrete cases above bar, continuous below.

To answer **RQ3**, we compare the number of partitions when increasing the state space of problems.

To answer **RQ4**, we select five random partitions from the partitioning obtained by SymPar, and five random concrete states from each partition. Then, we feed the concrete states as initial states to RL, and compute the accumulated reward using the policy that obtained from a trained model, assuming the training converged to the optimal policy. This way we can check how different the concrete states are with regard to performance.

Test Problems. The **Navigation** problem with a room (continuous) size of 10×10. The **Simple Maze** is a discrete environment (100 × 100) including blocks, goal and trap, in which a robot tries to find the shortest and safest route to the goal state [39]. **Braking Car** describes a car moving towards an obstacle with a given velocity and distance. The goal is to stop the car to avoid a crash with minimum braking pressure [43]. The **Multi-Agent Navigation** environment (10×10 grid) contains two agents attempting to find safe routes to a goal location. They must arrive to the goal position at the same time [38]. The **Mountain Car** aims to learn how to obtain enough momentum to move up a steep slope [28]. The **Random Walk** in continuous space is an agent with noisy actions on an infinite line [39]. The agent aims to avoid a hole and reach the goal region. **Wumpus World** [35] is a grid world (1: 64×64, 2: 16×16) in which the agent should avoid holes and find the gold.

6 Results

6.1 RQ1: Partitioning Size

Tables 1 and 2 show that SymPar consistently outperforms both tile coding (offline) and CAT-RL (online) on discrete state space in terms of success and

	SymPar				CAT-RL			
	$\lvert\mathcal{S}\rvert$	**Succ**	**Fail**	$\mathbf{T_{out}}$	$\lvert\mathcal{S}\rvert$	**Succ**	**Fail**	$\mathbf{T_{out}}$
	(#)	(%)	(%)	(%)	(#)	(%)	(%)	(%)
Mountain Car	70	82.2	0.0	17.8	16	78.7	0.0	21.3
Wumpus World 1	73	18.4	0.0	81.6	157	2.7	0.0	97.3
Wumpus World 2	52	37.3	22.9	39.8	22	14.5	30.2	55.3
Braking Car	81	89.1	10.9	0.0	127	34.0	66.0	0.0

Table 2. Partitioning size and learning performance with SymPar and CAT-RL online partitioning.

failure rates, and reduced number of timeouts ($\mathbf{T_{out}}$) during learning. Also, the agents using SymPar partitionings obtain maximum reward more often than with tiling (**Opt**), cf. Tbl. 1. Note that in Tbl. 1, the size of partitionings is substantially biased in favour of tiling. Nevertheless, SymPar enables better learning. In Tbl. 2, CAT-RL obtains smaller partitionings in the first and third cases in the same amount of time as SymPar but the quality of learning is worse. For the other test problems, SymPar is better than CAT-RL in both the size and learning performance.

For randomly selected states, the three left plots in Fig. 7, show that the agents trained by SymPar obtain a better normalized cumulative reward and subsequently converge faster to a better policy than the best competing approaches (other results, cf. Appendix, Fig. 10). The three right plots in the figure show the accumulated reward when starting from unlikely states (small partitions) for the best competing approaches (more in Appendix, Fig. 11) Here, we expect to observe a good policy from algorithms that capture the dynamics of environment. Interestingly, the online technique CAT-RL struggles when dealing with large sets of initial states. This can be seen in, e.g., the training for Braking Car, where each episode introduces new positions and velocities.

6.2 RQ2: Granularity vs Learning

The plots in Fig. 8 show that a higher granularity of partitionings yields a higher accumulated reward achieved with the optimal policy.

The plots in Fig. 9 show the shapes of partitions obtained by SymPar for Braking Car and Simple Maze. The first plot represents different partitions with different colors. Notably, the green and purple partitions depict partitioning expressions that contain a non-linear relation between the components of the state space (position and velocity). Besides, close to the x-axis, narrow partitions are discernible, depicted in yellow and pink. To illustrate the partitions obtained for Simple Maze, the expressions are translated into a 10×10 grid. The maze used for Fig. 9(b) differs from the one before, by including additional obstacles in the environment. These two visualizations shed light on the intricacies of state space partitioning and hint at the logical explainability of the partitionings obtained by SymPar.

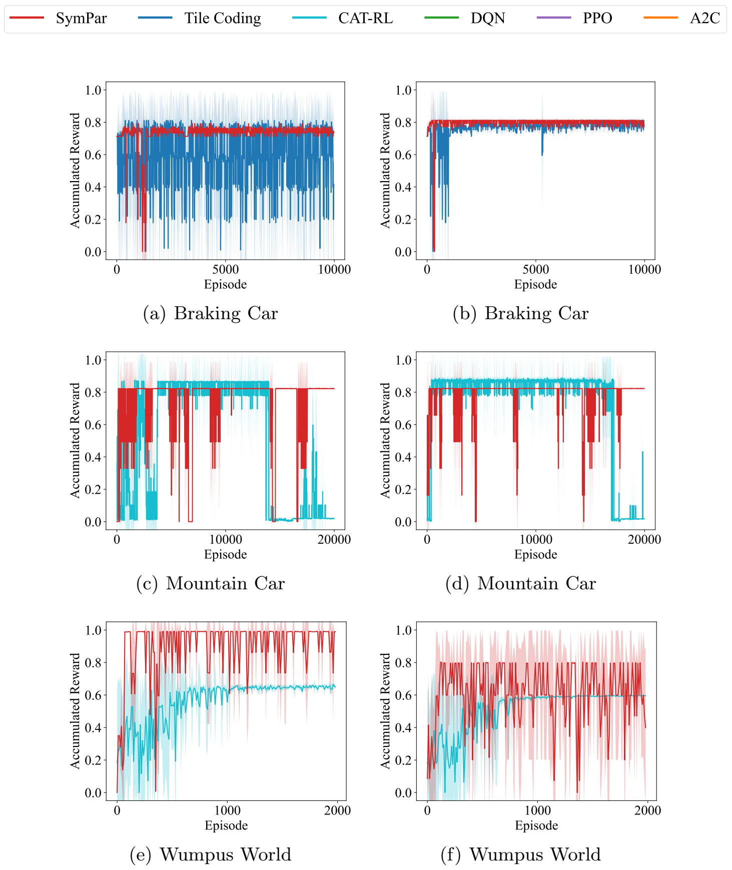


Fig. 7. Normalized cumulative reward per episode while evaluating ten random states (Left), and less likely states (Right). The best approach for each case is shown; see figures 10 and 11 for the rest.

6.3 RQ3: Scalability

Table 4 shows that the number of SymPar partitions is independent of the size of the state space. However, this does not imply the universal applicability of the same partitioning across different sizes. The conditions specified within the parti-

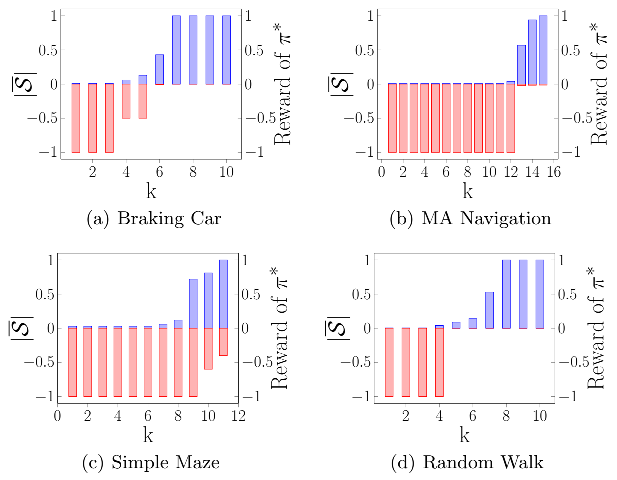


Fig. 8. Normalized granularity of states and its performance for symbolic execution with search depth k.

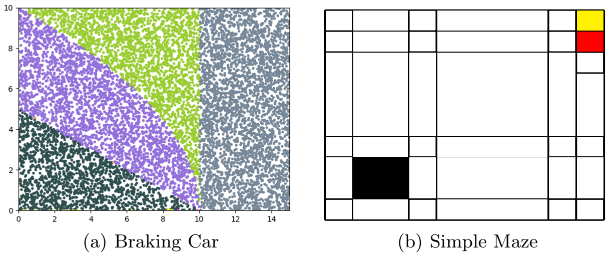


Fig. 9. Partitionings with SymPar for the Braking Car and Simple Maze.

tions are size-dependent. Consequently, when analyzing environments with different sizes for a given problem, running SymPar is necessary to ensure the appropriate partitioning, even though the total number of partitions remains the same.

6.4 RQ4: Partitioning as an Abstraction

Table 3 presents the variance in accumulated rewards for concrete states across various partitions. The findings demonstrate a notable consistency in accumulated rewards among states within the same partition, indicating minimal divergence.

	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$
Braking Car	$-0.05 \pm 0.0\%$	$-0.01 \pm 0.0\%$	$-0.5 \pm 0.0\%$	$-10.0 \pm 0.0\%$	$-10.01 \pm 0.0\%$
Mountain Car	$996.1 \pm 0.1\%$	$975.5 \pm 0.2\%$	$979.04 \pm 0.2\%$	$986.7 \pm 0.2\%$	$981.6 \pm 0.2\%$
Wumpus World	$486.8 \pm 0.3\%$	$490.0 \pm 0.2\%$	$477.6 \pm 0.2\%$	$475.0 \pm 0.0\%$	$495.8 \pm 0.1\%$

Table 3. Assessment of similarity of concrete states within partitions.

	$\lvert\overline{\mathcal{S}}\rvert$	$\lvert\mathcal{S}\rvert$		$\lvert\overline{\mathcal{S}}\rvert$	$\lvert\mathcal{S}\rvert$		$\lvert\overline{\mathcal{S}}\rvert$	$\lvert\mathcal{S}\rvert$
Simple Maze	10×10	33	**Wumpus World**	64×64	73	**Navigation**	10×10	51
	$10^2 \times 10^2$	33		$10^2 \times 10^2$	73		$10^2 \times 10^2$	51
	$10^3 \times 10^3$	33		$10^3 \times 10^3$	73		$10^3 \times 10^3$	51

Table 4. Size of state space and partitioning for test problems.

	Off	**Auto**	**Dyn**	**NonL**	**NarrP**	**SInd**
SymPar	✓	✓	✓	✓	✓	✓
CAT-RL	×	✓	✓	×	×	×
Tiling	✓	×	×	×	×	×

Table 5. Capabilities and properties.

This is particularly evident when the mean and normalized standard deviation are compared, which demonstrates that the standard deviation is considerably smaller in relation to the mean accumulated reward.

Summary. Our experiments show distinct advantages of SymPar over the other approaches, cf. Tbl. 5. It is an offline (**Off**) automated (**Auto**) approach, which captures the dynamics of the environment (**Dyn**), and maps the nonlinear relation between components of the state into their representation (**NonL**). SymPar can detect narrow partitions (**NarrP**) without excessive sampling and generates a logical partitioning that is independent of the specific size of the state space (**SInd**). This comprehensive comparison underscores the robust capabilities of SymPar across various dimensions, positioning it as a versatile and powerful approach compared to CAT-RL and Tile Coding.

Limitations. SymPar can handle only environments that are implemented as programs. It will also perform weakly in environments with very limited branching; e.g., like Cart Pole [39]. The simulation of Cart Pole only branches on final states; its dynamics is a physical formula over the position and velocity of the cart, and the angle and angular velocity of the pole. The path conditions found by symbolic execution are of little help here.

Technical Details. The implementation of SymPar (will be publicly available upon the acceptance) uses Symbolic PathFinder[2] [32] as its symbolic executor, Z3[3] [30] as its main SMT-Solver and the SMT-solver DReal[4] [13] to handle non-linear functions such as trigonometric functions.

7 Conclusion

We presented SymPar—a new generic offline method for partitioning state spaces in reinforcement learning through a comprehensive analysis of the environment's dynamics. SymPar operates automatically, in practice streamlining the process. In contrast to related work, SymPar's partitions effectively capture the semantics of the environment dynamics. SymPar accommodates non-linear environmental behaviors by using adaptive partition shapes, instead of rectangular tiles. Our experiments demonstrate that SymPar improves state space coverage with respect to environmental behavior and allows reinforcement learning to better handle with sparse rewards. However, since SymPar analyzes the simulator of the environment, it is sensitive to the implementation of the environment model. The performance of the underlying tools, including the symbolic executor and SMT solvers, also affect the effectiveness of SymPar for complex simulators with long execution paths. In the future, we would like to address these limitations and consider using symbolic execution also for online partitioning.

References

1. Adelt, J., Herber, P., Niehage, M., Remke, A.: Towards safe and resilient hybrid systems in the presence of learning and uncertainty. In: Proc. 11th Intl. Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles (ISoLA 2022). Lecture Notes in Computer Science, vol. 13701, pp. 299–319. Springer (2022). https://doi.org/10.1007/978-3-031-19849-6_18
2. Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Schmitt, P.H., Ulbrich, M. (eds.): Deductive Software Verification - The KeY Book - From Theory to Practice, Lecture Notes in Computer Science, vol. 10001. Springer (2016). https://doi.org/10.1007/978-3-319-49812-6, https://doi.org/10.1007/978-3-319-49812-6
3. Akrour, R., Veiga, F., Peters, J., Neumann, G.: Regularizing reinforcement learning with state abstraction. In: Proc. Intl. Conf. on Intelligent Robots and Systems (IROS). pp. 534–539. IEEE (2018)
4. Albus, J.S.: Brains, behavior, and robotics. BYTE Books (1981)
5. de Boer, F.S., Bonsangue, M.M.: Symbolic execution formally explained. Formal Aspects Comput. **33**(4-5), 617–636 (2021). https://doi.org/10.1007/S00165-020-00527-Y, https://doi.org/10.1007/s00165-020-00527-y
6. Cadar, C., Dunbar, D., Engler, D.R.: KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. 8th Symposium on Operating Systems Design and Implementation (OSDI 2008). pp. 209–224. USENIX

[2] https://github.com/SymbolicPathFinder

[3] https://github.com/Z3Prover/z3

[4] https://github.com/dreal/dreal4

Association (2008), http://www.usenix.org/events/osdi08/tech/full_papers/cadar/cadar.pdf
7. Clarke, L.A.: A program testing system. In: Proc. 1976 Annual Conf. pp. 488–491. ACM (1976). https://doi.org/10.1145/800191.805647
8. Dadvar, M., Nayyar, R.K., Srivastava, S.: Conditional abstraction trees for sample-efficient reinforcement learning. In: Proc. 39th Conf. on Uncertainty in Artificial Intelligence. Proc. Machine Learning Research, vol. 216, pp. 485–495. PMLR (2023)
9. Davey, B.A., Priestley, H.A.: Introduction to lattices and order. Cambridge University Press, Cambridge (1990), http://www.worldcat.org/search?qt=worldcat_org_all&q=0521367662
10. Ferns, N., Panangaden, P., Precup, D.: Metrics for finite markov decision processes. In: UAI. vol. 4, pp. 162–169 (2004)
11. Ferns, N., Panangaden, P., Precup, D.: Bisimulation metrics for continuous markov decision processes. SIAM Journal on Computing **40**(6), 1662–1714 (2011)
12. Fulton, N., Platzer, A.: Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In: Proc. 32nd Conf. on Artificial Intelligence (AAAI-18). pp. 6485–6492. AAAI Press (2018). https://doi.org/10.1609/AAAI.V32I1.12107
13. Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: Proc. 24th Intl. Conf. on Automated Deduction (CADE-24). Lecture Notes in Computer Science, vol. 7898, pp. 208–214. Springer (2013). https://doi.org/10.1007/978-3-642-38574-2_14
14. Ghaffari, M., Afsharchi, M.: Learning to shift load under uncertain production in the smart grid. Intl. Transactions on Electrical Energy Systems **31**(2), e12748 (2021)
15. Jaeger, M., Jensen, P.G., Larsen, K.G., Legay, A., Sedwards, S., Taankvist, J.H.: Teaching Stratego to play ball: Optimal synthesis for continuous space MDPs. In: Proc. 17th Intl. Symposium on Automated Technology for Verification and Analysis (ATVA 2019). Lecture Notes in Computer Science, vol. 11781, pp. 81–97. Springer (2019). https://doi.org/10.1007/978-3-030-31784-3_5
16. Jansson, A.D.: Discretization and representation of a complex environment for on-policy reinforcement learning for obstacle avoidance for simulated autonomous mobile agents. In: Proc. 7th Intl. Congress on Information and Communication Technology. Lecture Notes in Networks and Systems, vol. 464, pp. 461–476. Springer (2023)
17. Jin, P., Tian, J., Zhi, D., Wen, X., Zhang, M.: Trainify: A CEGAR-driven training and verification framework for safe deep reinforcement learning. In: Proc. 34th Intl. Conf. on Computer Aided Verification (CAV 2022). Lecture Notes in Computer Science, vol. 13371, pp. 193–218. Springer (2022). https://doi.org/10.1007/978-3-031-13185-1_10
18. King, J.C.: Symbolic execution and program testing. Communications of the ACM **19**(7), 385–394 (1976)
19. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The Intl. Journal of Robotics Research **32**(11), 1238–1274 (2013)
20. Kozen, D.: Semantics of probabilistic programs. In: Proc. 20th Annual Symposium on Foundations of Computer Science (SFCS 1979). pp. 101–114. IEEE Computer Society (1979). https://doi.org/10.1109/SFCS.1979.38, https://doi.org/10.1109/SFCS.1979.38
21. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: Classifier prediction based on tile coding. In: Proc. Genetic and Evolutionary Computation Conf. (GECCO 2006). pp. 1497–1504. ACM (2006). https://doi.org/10.1145/1143997.1144242

22. Lee, I.S., Lau, H.Y.: Adaptive state space partitioning for reinforcement learning. Engineering applications of artificial intelligence **17**(6), 577–588 (2004)
23. Madumal, P., Miller, T., Sonenberg, L., Vetere, F.: Explainable reinforcement learning through a causal lens. In: Proc. 34th Conf. on Artificial Intelligence (AAAI 2020). pp. 2493–2500. AAAI Press (2020). https://doi.org/10.1609/AAAI.V34I03.5631
24. Mavridis, C.N., Baras, J.S.: Vector quantization for adaptive state aggregation in reinforcement learning. In: 2021 American Control Conf. (ACC). pp. 2187–2192. IEEE (2021)
25. Michie, D., Chambers, R.A.: Boxes: An experiment in adaptive control. Machine intelligence **2**(2), 137–152 (1968)
26. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: Proc. 33nd Intl. Conf. on Machine Learning (ICML 2016). JMLR Workshop and Conf. Proceedings, vol. 48, pp. 1928–1937. JMLR.org (2016), http://proceedings.mlr.press/v48/mniha16.html
27. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
28. Moore, A.W.: Efficient memory-based learning for robot control. Ph.D. thesis, University of Cambridge, UK (1990). https://doi.org/10.1.1.17.2654
29. Moore, A.W.: Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In: Machine Learning Proceedings 1991, pp. 333–337. Elsevier (1991)
30. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: Proc. 14th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008). Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24
31. Nicol, S., Chadès, I.: Which states matter? an application of an intelligent discretization method to solve a continuous POMDP in conservation biology. PloS one **7**(2), e28993 (2012)
32. Pasareanu, C.S., Visser, W., Bushnell, D.H., Geldenhuys, J., Mehlitz, P.C., Rungta, N.: Symbolic PathFinder: integrating symbolic execution with model checking for Java bytecode analysis. Autom. Softw. Eng. **20**(3), 391–425 (2013). https://doi.org/10.1007/S10515-013-0122-2
33. Puiutta, E., Veith, E.M.S.P.: Explainable reinforcement learning: A survey. In: Proc. 4th Intl. Cross-Domain Conf. (CD-MAKE 2020). Lecture Notes in Computer Science, vol. 12279, pp. 77–95. Springer (2020). https://doi.org/10.1007/978-3-030-57321-8_5
34. Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., Dormann, N.: Stable baselines3 (2019), https://stable-baselines3.readthedocs.io/
35. Russell, S.J., Norvig, P.: Artificial intelligence a modern approach. London (2010)
36. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
37. Seipp, J., Helmert, M.: Counterexample-guided cartesian abstraction refinement for classical planning. Journal of Artificial Intelligence Research **62**, 535–577 (2018)
38. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. Artificial Intelligence **219**, 40–66 (2015)
39. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, 2nd edn. (2018)

40. Szita, I.: Reinforcement learning in games. In: Reinforcement Learning, Adaptation, Learning, and Optimization, vol. 12, pp. 539–577. Springer (2012). https://doi.org/10.1007/978-3-642-27645-3_17
41. Tran, H.D., Cai, F., Diego, M.L., Musau, P., Johnson, T.T., Koutsoukos, X.: Safety verification of cyber-physical systems with reinforcement learning control. ACM Transactions on Embedded Computing Systems (TECS) **18**(5s), 1–22 (2019)
42. Uther, W.T.B., Veloso, M.M.: Tree based discretization for continuous state space reinforcement learning. In: Proc. 15th National Conf. on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conf. (AAAI 98, IAAI 98). pp. 769–774. AAAI Press / The MIT Press (1998), http://www.aaai.org/Library/AAAI/1998/aaai98-109.php
43. Varshosaz, M., Ghaffari, M., Johnsen, E.B., Wąsowski, A.: Formal specification and testing for reinforcement learning. Proc. ACM Program. Lang. **7**(ICFP) (aug 2023). https://doi.org/10.1145/3607835, https://doi.org/10.1145/3607835
44. Verdier, C.F., Babuška, R., Shyrokau, B., Mazo, M.: Near optimal control with reachability and safety guarantees. IFAC-PapersOnLine **52**(11), 230–235 (2019). https://doi.org/https://doi.org/10.1016/j.ifacol.2019.09.146
45. Visser, W., Pasareanu, C.S., Pelánek, R.: Test input generation for java containers using state matching. In: Pollock, L.L., Pezzè, M. (eds.) Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2006, Portland, Maine, USA, July 17-20, 2006. pp. 37–48. ACM (2006). https://doi.org/10.1145/1146238.1146243, https://doi.org/10.1145/1146238.1146243
46. Voogd, E., Johnsen, E.B., Silva, A., Susag, Z.J., Wąsowski, A.: Symbolic semantics for probabilistic programs. In: Proc. 20th Intl. Conf. on Quantitative Evaluation of Systems (QEST 2023). Lecture Notes in Computer Science, vol. 14287, pp. 329–345. Springer (2023). https://doi.org/10.1007/978-3-031-43835-6_23
47. Vyetrenko, S., Xu, S.: Risk-sensitive compact decision trees for autonomous execution in presence of simulated market response. arXiv preprint arXiv:1906.02312 (2019). https://doi.org/10.48550/ARXIV.1906.02312
48. Wei, H., Corder, K., Decker, K.: Q-learning acceleration via state-space partitioning. In: Proc. 17th Intl. Conf. on Machine Learning and Applications (ICMLA 2018). pp. 293–298. IEEE (2018)
49. Whiteson, S.: Adaptive Representations for Reinforcement Learning, Studies in Computational Intelligence, vol. 291. Springer (2010). https://doi.org/10.1007/978-3-642-13932-1, https://doi.org/10.1007/978-3-642-13932-1
50. Yu, C., Liu, J., Nemati, S., Yin, G.: Reinforcement learning in healthcare: A survey. ACM Computing Surveys (CSUR) **55**(1), 1–36 (2021)
51. Zelvelder, A.E., Westberg, M., Främling, K.: Assessing explainability in reinforcement learning. In: Proc. Third Intl. Workshop on Explainable and Transparent AI and Multi-Agent Systems (EXTRAAMAS 2021). Lecture Notes in Computer Science, vol. 12688, pp. 223–240. Springer (2021). https://doi.org/10.1007/978-3-030-82017-6_14

8 Appendix / Supplementary Material

8.1 Proofs of properties of SymPar

Theorem 1. *The obtained partitioning $\mathcal{P}$ is total for loop-free programs:*

$$\forall \overline{s} \in \overline{\mathcal{S}} \cdot \exists!\, \mathcal{P}_0 \in \mathcal{P} \cdot \overline{s} \in \mathcal{P}_0.$$

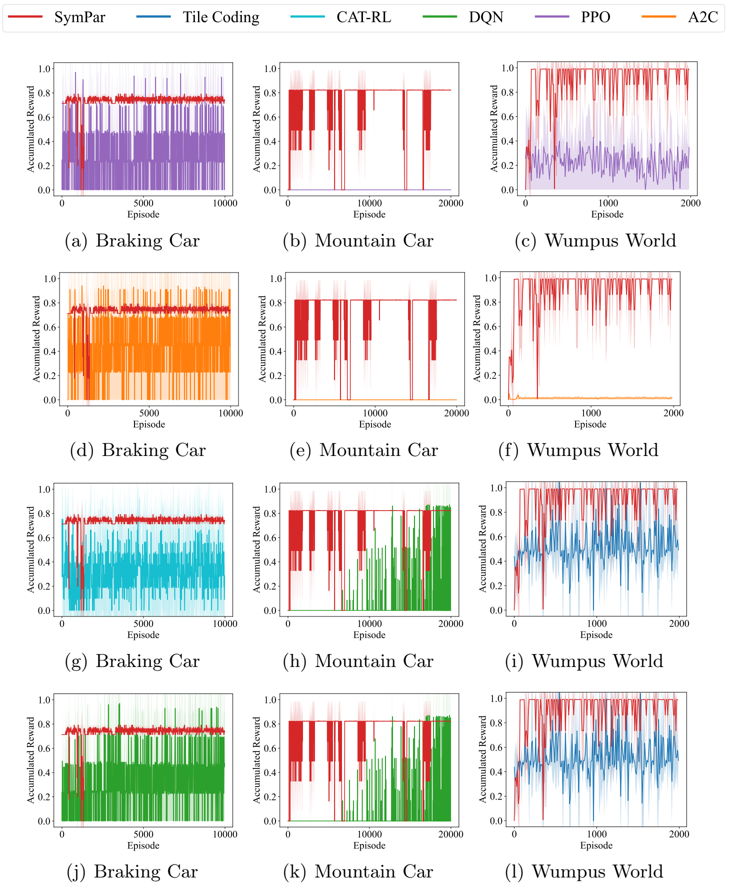
Proof. The theorem follows from the fact that the partitioning generated by SymPar is obtained from first running the simulated environment symbolically and collecting the path conditions. By design, all path conditions produced by a complete terminating symbolic execution run is a partitioning. The final partitioning is obtained by intersecting these partitionings to obtain the unique coarsest partitioning finer than each of them. This partitioning is known from order theory [9] to be unique and it is total and pairwise disjoint. Hence, it can be inferred that each concrete state in the partitioned state space, $\overline{s} \in \overline{\mathcal{S}}$, is represented by at least one partition $\mathcal{P}_0 \in \mathcal{P}$.

Theorem 2. *Let PC^a be the set of path conditions produced by SymPar for each of the actions $a \in \mathcal{A}$. The size of the final partitioning $\mathcal{P}$ returned by SymPar is bounded from below by each $|PC^a|$ and from above by $\prod_{a\in\mathcal{A}} |PC^a|$.*

Proof. The theorem follows from the fact that $\mathcal{P}$ is finer than any of the PC^as and the algorithm for computing the coarsest partitioning finer then a set of partitionings can in the worst case intersect each partition in each set PC^a with all the partitions in the partitionings of the other actions.

8.2 Additional Results

To make the comparison with DQN, A2C and PPO fair, we used the same running time as for SymPar, which resulted in lower performance for these approaches. The fluctuation observed in the plots suggest that they may need more iterations and possibly more customized architectures. A2C and PPO, which are proper for problems with continuous action space, behave as expected.



(a) Braking Car (b) Mountain Car (c) Wumpus World

(d) Braking Car (e) Mountain Car (f) Wumpus World

(g) Braking Car (h) Mountain Car (i) Wumpus World

(j) Braking Car (k) Mountain Car (l) Wumpus World

Fig. 10. Normalized cumulative reward per each episode while evaluating ten random states.

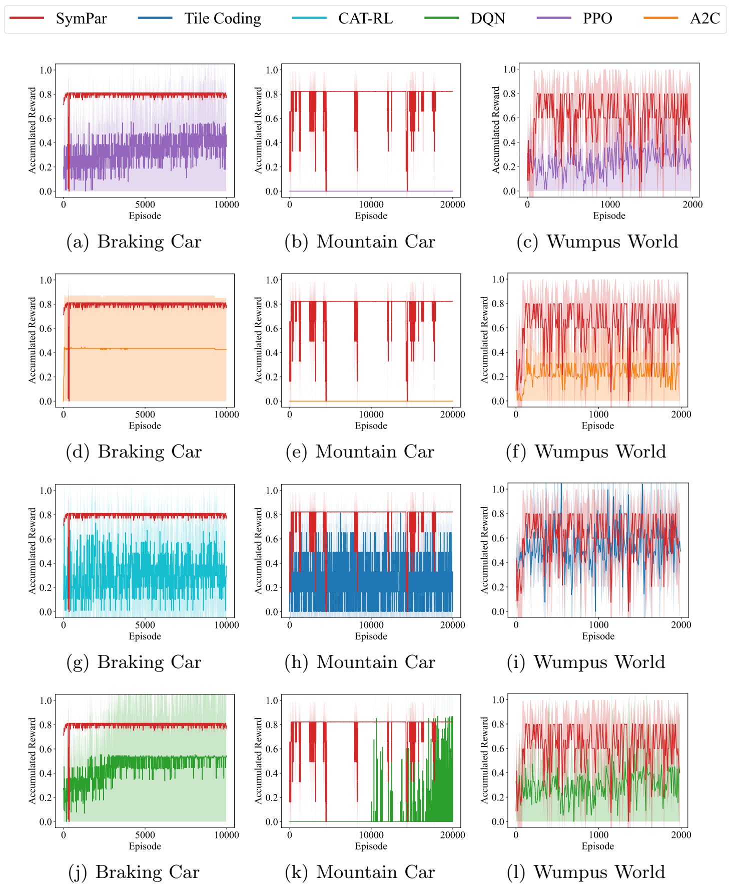


Fig. 11. Normalized cumulative reward per each episode while evaluating ten less likely states.

Symbolic State Seeding Improves Coverage Of Reinforcement Learning

Mohsen Ghaffari[a,*], Cong Chen[a], Mahsa Varshosaz[a], Einar Broch Johnsen[b], Andrzej Wsowski[a]

[a]*Computer Science Departement, IT-University of Copenhagen, Copenhagen, Denmark*

[b]*Department of Informatics, University of Oslo, Oslo, Norway*

Abstract

In practice, reinforcement learning should complete the training process at a given point in time. For this reason, a reinforcement agent is only capable of exploring the most probable scenarios and not the entirety of the environment, which results in a limited understanding of the situation. One potential approach to address this problem is to explore the agent in rare but important situations. For example, one might initialize the agent to observe states that are likely to result in different outcomes. In this paper, we propose a method for seeding initial states, called SymSeed, for the class of reinforcement learning applications that the simulation environment is available, either in a black-box or white-box manner. To this end, we analyze the simulator of the environment using symbolic execution, then generate initial states to ensure that the agent would learn how to act in those states. We evaluate our approach by feeding the generated states into well-known reinforcement learning algorithms including DQN, PPO, A2C, and CAT-RL.

Keywords: Reinforcement Learning, Symbolic Execution, Initialization

1. Introduction

Reinforcement learning (RL) is a framework to develop controllers through iterative training (Sutton & Barto, 2018). In practice, the training of a reinforcement learning is often constrained by limited time, restricting exposure to only consider the most likely scenarios out of a large space. Limited exploration narrows the agent's understanding of the full environment and can lead to several challenges: (1) disastrous results for safety-critical systems where a safe policy is needed for every possible state, (2) the agent may get stuck in local optima, (3) policies may become overly sensitive to initial and observed states, and (4) dealing with sparse rewards becomes more difficult.

Exploration is a fundamental aspect of reinforcement learning (Tarbouriech et al., 2020; Fruit & Lazaric, 2017; Kolter & Ng, 2009). Common approaches include ϵ-greedy exploration (Sutton & Barto, 2018), count-based exploration (Bellemare et al., 2016; Strehl & Littman, 2008; Machado et al., 2020), curiosity-based exploration (Pathak et al., 2017), or methods specifically designed for exploring sparse reward contextual MDPs (Raileanu & Rocktäschel, 2020; Zha et al., 2021). All these methods begin by seeding initial states using a uniform distribution over a subset of the state space. Such a uniform selection of initial states can be problematic for several reasons. First, the

*Mohsen Ghaffari
Email addresses: mohg@itu.dk (Mohsen Ghaffari), coch@itu.dk (Cong Chen), mahv@itu.dk (Mahsa Varshosaz), einarj@ifi.uio.no (Einar Broch Johnsen), wasowski@itu.dk (Andrzej Wsowski)

agent may spend a considerable amount of time exploring areas of the state space that are free of immediate reward. Second, the agent may overfit to regions that it frequently encounters initially, while neglecting others that are critical in later stages of training. Third, the agent might focus on suboptimal areas, thereby reducing the utility of early policies. In contrast, our objective is to enhance the exploration process by leveraging prior knowledge about the dynamics of the agent's environment. In short, this paper seeks to address the following questions:

How can insights into a set of states that are likely to vary significantly be leveraged for reinforcement learning? Can we identify states that are less likely to be sampled during reinforcement learning training?

We hypothesize that insights into the structure of the state space will enable more efficient exploration. Specifically, initializing the reinforcement learning agent in expert-provided states ensures that these states are explored, allowing the agent to learn appropriate policies for critical situations. This approach can also help the agent reach a goal state more quickly, effectively distributing reward values to intermediary states, which is particularly useful in scenarios with sparse rewards.

How can we provide reinforcement learning with such expert knowledge? To answer this question, we can use software analysis tools that extract such knowledge. In particular, symbolic execution meets our needs. Symbolic execution is a popular technique in program analysis, which can be used to automatically generate test inputs for programs or detect hidden problems in an implementation, guaranteeing high coverage of the analyzed code (Baldoni et al., 2018; Cadar et al., 2008; Ball et al., 2015; Chen et al., 2021).

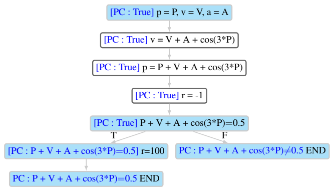
This paper proposes an initialization strategy for reinforcement learning applications, called SymSeed, where an environment simulator—either black–box or white–box—is available. We leverage symbolic execution to analyze the environment simulator and generate initial states, ensuring that the agent learns effective policies in those states. A pre-training phase in simulation is often a prerequisite to apply reinforcement learning in real-world scenarios, such as robotics. It can be impractical as well as financially prohibitive to train the agent in a real-world environment. In this work, we focus on problems where such an environment simulator is available. This allows unsafe states to be observed as often as necessary, which helps for learning optimal policies for those states.

2. Related Work

Although the initial states of a trained model play an instrumental role in its performance (Andrychowicz et al., 2020; Jang & Kim, 2022), state seeding for reinforcement learning has received much less attention than policy initialization (Abel et al., 2018; Uchendu et al., 2023; Barreto et al., 2020; Kraemer & Banerjee, 2014). Policy initialization, which is useful for policy optimization, makes the agent learn a policy that is roughly analogous to an initial policy, but does not trigger a more comprehensive exploration of the environment; thus, the exploration of the state space remains limited to a specific range. The reinforcement learning algorithm may get stuck in local optima, impeding its ability to explore the state space effectively. Furthermore, states that are less likely to be observed may be entirely bypassed with policy initialization; this represents a significant risk for safety-critical systems. Another line of work that also emphasizes the importance of the initial states, optimizes an ensemble of policies over different "slices" of the state space (Ghosh et al., 2017). In contrast, we develop a method for seeding initial states to reinforcement learning that lead to better state exploration.

```python
def step(p, v, a):
  v += a+math.cos(3*
    p)
  p += v
  r = -1.0
  if p == 0.5:
    r = 100
  return p, v, r
```

(a) A simple step function.


[PC : True] p = P, v = V, a = A
[PC : True] v = V + A + cos(3*P)
[PC : True] p = P + V + A + cos(3*P)
[PC : True] r = -1
[PC : True] P + V + A + cos(3*P)=0.5
T
F
[PC : P + V + A + cos(3*P)=0.5] r=100
PC : P + V + A + cos(3*P)≠0.5 END
PC : P + V + A + cos(3*P)=0.5 END

(b) Execution tree generated with symbolic input p = P, v = V, a = A.

Figure 1: Example of symbolic execution for a step function.

In robotics, where agents are commonly trained on simulators, there is work on mapping the trained initial states from the simulation level to states in the real world (Montgomery et al., 2017; Sharma et al., 2021). This work does not address the initial states that an agent would take at the training level in the simulator but focuses on the initial states that an agent, such as a robot, should adopt when retrained or tested in the real world. In robotics, another branch of study attempts to learn the reset function or initial states (Kim et al., 2022). Similarly, Messikommer et al. (2024) select states from past experiences and use these to initialize the agent in the environment, thereby guiding it toward a more informative state. This work demonstrates the significance of maintaining a sufficiently limited initial state to facilitate the repetition of previously explored states. Moreover, the set of initial states should be sufficiently expansive to guarantee that the agent has adequately explored the state space, which is not considered in the above-mentioned works.

3. Background

Symbolic Execution is a software analysis technique used to automate software testing to find program errors (Păsăreanu, 2020). It extends normal execution by running the basic operators of a language using symbolic inputs (King, 1976). When doing so, the values of program variables become mathematical expressions over the symbolic inputs. For each path executed through the program, the analysis maintains a symbolic *path condition* which encodes the conditions on the inputs for the execution to follow that path. Each path condition is built by accumulating the branch conditions encountered during the execution of the program.

Figure 1a shows an example of a simple *step function*, and Fig. 1b the corresponding symbolic execution tree for this function, generated by symbolic execution in a step-by-step manner. As illustrated, the execution tree carries a path condition PC in each node. The tree's leaf nodes show the logical expressions that must be satisfied to execute the corresponding execution path in the program.

Markov Decision Processes (MDPs) are a discrete-time stochastic control structures, which assume that the future state is only related to the present state and is independent of the past states (Bellman, 1957). Formally, an MDP is a tuple $(\mathcal{S}, \mathcal{S}_0, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{F})$, where $\mathcal{S}$ is a set of states, $\mathcal{S}_0 \in \text{pdf}\,\mathcal{S}$ a probability density function for initial states, $\mathcal{A}$ a finite set of actions, $\mathcal{T} \in \mathcal{S} \times \mathcal{A} \to \text{pdf}\,\mathcal{S}$ the transition probability function for successor states for transitions from a given state with a given action, $\mathcal{R} \in \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function, and $\mathcal{F} \in \mathcal{S} \to \{0, 1\}$ a predicate defining final states.

Reinforcement Learning is concerned with tasks in which an agent interacts with an environment through actions, observations and rewards (Sutton & Barto, 2018). A reinforcement learning problem can be modeled using an MDP, in which the task is to find a *policy* π that selects actions in different states to maximize the expected accumulated reward (so reinforcement learning is a statistical method for solving MDPs). To learn an optimal policy π^*, the action-value function can be represented in, e.g., a Q-table (Watkins & Dayan, 1992), which we update using the equation

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] , \tag{1}$$

where r_{t+1} is the reward received after taking action $a_t \in \mathcal{A}$ in state $s_t \in \overline{\mathcal{S}}$, $0 < \alpha \leq 1$ is the *learning rate*, $0 < \gamma \leq 1$ is the *discount factor*, $\max_{a'} Q(s_{t+1}, a')$ is the maximum Q-value for the next state $s_{t+1} \in \overline{\mathcal{S}}$ across all possible actions $a' \in \mathcal{A}$. The optimal action-value function $Q^*(s, a)$ can then be obtained by $Q^*(s, a) = \max_\pi Q^\pi(s, a)$.

In the sequel, we give a brief overview of four reinforcement learning algorithms that will be used for the evaluation of the methodology proposed in this paper.

Deep Q-Network (DQN) learning has been is to overcome challenges arising when the dimensions of state space increase (Mnih, 2013). Unlike traditional Q-learning, which relies on a Q-table, DQN leverages Deep Neural Networks (DNNs) to approximate value functions. In each episode, the DQN updates its parameters by learning from the agent's experiences, allowing it to estimate the expected cumulative reward more efficiently. The DNN model is optimized to approximate the optimal action-value function by adjusting its parameters based on observed state transitions and rewards.

Asynchronous Advantage Actor-Critic (A3C) is a reinforcement learning algorithm where multiple agents (or copies of the environment) work in parallel to learn a task. These agents operate independently and asynchronously, meaning they collect experiences at different times (Mnih et al., 2016). Each agent uses an *actor* network to decide on actions and a *critic* network to evaluate how good those actions are, based on the current state. By working together, the agents share their learned experiences with a global model, allowing it to learn more efficiently and effectively.

Proximal Policy Optimization (PPO) is a policy gradient algorithm that combines ideas from Asynchronous Actor-Critic (A2C-having multiple workers) and Trust Region Policy Optimization (TRPO-using a trust region to improve the actor) (Schulman et al., 2015). It iteratively learns a parameterized policy π_θ. In standard implementations, PPO regularizes policy updates with clipped probability ratios, and parameterizes policies with either continuous Gaussian distributions or discrete Softmax distributions (Schulman et al., 2017).

Conditional Abstraction Trees for Sample-Efficient Reinforcement Learning (CAT-RL) is a top-down approach for constructing state abstractions while carrying out reinforcement learning. Starting with state variables and a simulator, it dynamically computes an abstraction based on the dispersion of temporal difference errors in abstract states as the agent continues acting and learning.

4. Symbolic State Seeding (SymSeed)

We propose a methodology for generating a set of initial states for reinforcement learning algorithms by analyzing the environment dynamics, which we assume to be simulated by a computer program. To this end, the environment simulator is executed symbolically to extract path conditions (PCs), using an off-the-shelf tool. Each PC is a logical expression over the input variables of the program, in this case the state and the action of the reinforcement learning agent. We can then solve each PC using an SMT solver, resulting in a concrete state and action that together satisfy

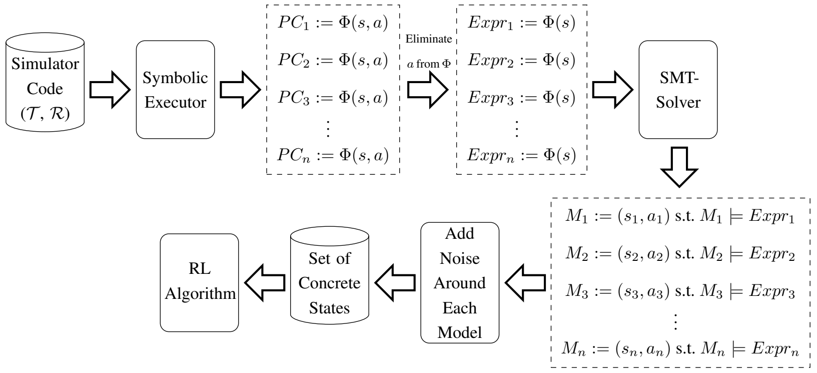

Simulator Code $(\mathcal{T}, \mathcal{R})$
Symbolic Executor
$PC_1 := \Phi(s,a)$
$PC_2 := \Phi(s,a)$
$PC_3 := \Phi(s,a)$
$PC_n := \Phi(s,a)$
Eliminate a from Φ
$Expr_1 := \Phi(s)$
$Expr_2 := \Phi(s)$
$Expr_3 := \Phi(s)$
$Expr_n := \Phi(s)$
SMT-Solver
$M_1 := (s_1, a_1)$ s.t. $M_1 \models Expr_1$
$M_2 := (s_2, a_2)$ s.t. $M_2 \models Expr_2$
$M_3 := (s_3, a_3)$ s.t. $M_3 \models Expr_3$
$M_n := (s_n, a_n)$ s.t. $M_n \models Expr_n$
Add Noise Around Each Model
Set of Concrete States
RL Algorithm

Figure 2: Overview of the approach.

the given PC. Finally, we introduce noise around these states in order to increase the number of samples, and use all the obtained states as initial states for a reinforcement learning algorithm, see Fig. 2.

The environment simulator is a program that takes the current state of the agent and its action, and computes the next state that the agent reaches and the immediate reward obtained for the current transition, a so-called *step function*. In other words, the environment simulator implements a single-step transition of the MDP for the given problem, reflecting the environment dynamics. Remark that the simulator does not necessarily need to be a faithful rendition of the real environment, in full detail. On the contrary, abstract environment models are often used for pre-training simulations.

The symbolic execution of a program explores the program's feasible execution paths and generates a set of PCs; each PC characterizes a specific execution path within the program. To execute a program symbolically, two key elements must be in place: the program itself (or the byte code of the program) and the variables within the program that are to be treated as symbolic variables. The symbolic execution of a given step function captures the environment dynamics in a set of PCs, where each PC becomes a logical expression over the state and action variables. The specific techniques used to solve the PCs may vary depending on the chosen SMT solver. When asked for the solution to a PC expression, the solver returns concrete values for the variables of the formula that can satisfy this logical expression. These concrete values can be used as initial states for learning.

SMT solvers typically yield one solution for a given formula, even if multiple solutions exist. Although feasible, obtaining more solutions from an SMT solver is time-consuming, as the formula to be solved typically grows with each iteration. Instead, we obtain a single solution from the SMT solver and then add a small amount of noise to this solution to obtain multiple states. It is highly probable (heuristically) that this increases the number of samples for each PC.

The result of solving each PC using the SMT solver, is a set of concrete values for state and action variables that can satisfy the given PC. Remark that although these solutions include values for both state and action variables, actions are not initialised in reinforcement learning. Consequently, the action values are excluded from the answers, yielding a set of concrete values for state variables. Next, we introduce noise around each state value to avoid sample bias. Ultimately, this set can now be fed into learning algorithms, either tabular or deep methods.

5. Evaluation

5.1. Experimental setup

The following research questions are addressed in order to evaluate the performance and efficacy of the approach presented in this paper.

RQ1. Can SymSeed decrease the number of visited states and yet improve the reward?
RQ2. Does SymSeed help to avoid local optima?
RQ3. Does SymSeed improve the performance in the presence of sparse rewards?

To answer these questions, we apply SymSeed to well-known RL algorithms DQN (Mnih, 2013), A3C (Mnih et al., 2016), PPO (Schulman et al., 2017), using the Stable-Baselines3[1] implementations (Raffin et al., 2019), and CAT-RL (Dadvar et al., 2023). To assess the performance of SymSeed, we conduct a series of experiments several classic case studies. The training of each agent is conducted using three distinct initialization strategies: *(a)* a uniform distribution over the entire state space, *(b)* a uniform distribution over the set of states generated by SymSeed, and *(c)* a combination of (a) and (b), whereby the percentage of each is controlled.

To answer **RQ1**, we collect the visited states during training for each initialization strategy. Additionally, we measure the mean of the accumulated reward of all episodes.

To answer **RQ2**, we designed a series of examples, e.g., the Safari Car test case, which demonstrate how local optima may impede an agent's progress if the environment is not adequately explored. We say that the agent has succeeded if it identifies the global optimum. In these experiments, the success rate was measured during training for each initialization strategy.

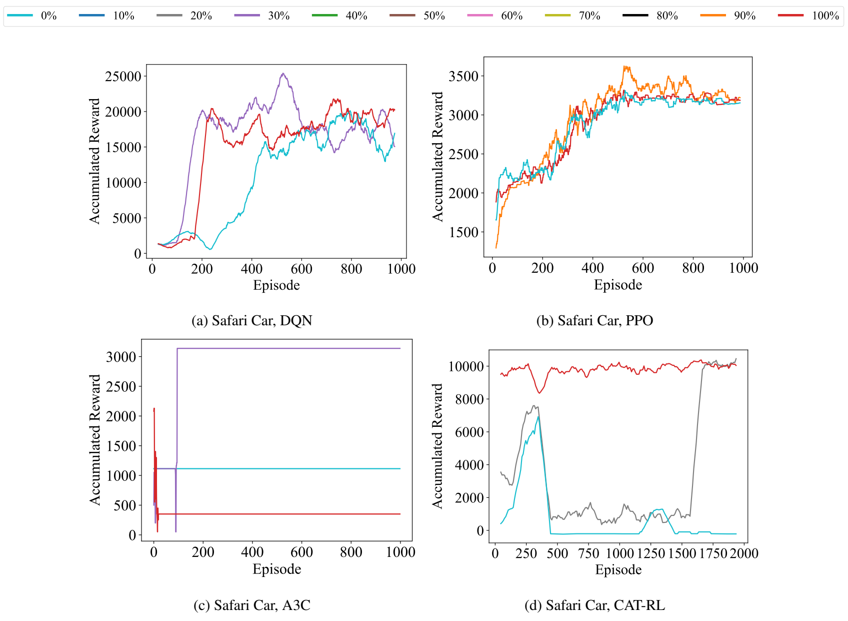
To answer **RQ3**, we designed a series of examples based on Office World, in which the agent only obtains a reward in final states (i.e., no intermediate rewards). The goal is to show how SymSeed helps the agent to find goal states in early episodes and to expand awareness of these goal states to the rest of states. We measure the accumulated reward to evaluate the trained policy every ten episodes for ten randomly selected states.

Test Problems. The **Office World 1** problem is a grid map comprising four distinct rooms at its corners. The objective is to collect and deliver mail and coffee to the designated office location, resulting in a positive reward; otherwise, no reward is given. The **Office World 2** is analogous to Office World 1, with the exception that the location of the goal is situated at the farthest distance from the start position of the agent. The **Office World 3** represents a combination of the first and second office worlds, incorporating two distinct goal states. Nevertheless, the agent will only succeed in one of the two possible outcomes, with the other office acting as a local optima. The **Safari Car** aims to learn how to obtain enough momentum to move up two steep slopes, similar dynamics to the mountain car (Moore, 1990), but only with two steep slopes. The agent obtains a small positive reward when it moves up from the first steep slope, and it obtains the highest reward when it moves up from the second steep slope; otherwise, it obtains -1 for each move.

5.2. Results

RQ1. Figure 3 shows the best mixing strategy for initializing the RL algorithms with SymSeed seeded states, complete initialization with SymSeed (100%), and not using SymSeed at all (0%)

[1]https://github.com/DLR-RM/stable-baselines3



(a) Safari Car, DQN

(b) Safari Car, PPO

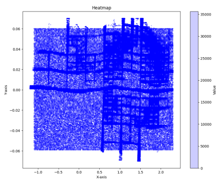
(c) Safari Car, A3C

(d) Safari Car, CAT-RL

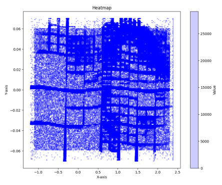
Figure 3: RL algorithms evaluated for ten uniformly random initial states, every 100 episodes. The baseline (0%), all initial states generated by SymSeed (100%), and the best mixed method are selected.

which serves as the baseline. The comparison of achieved accumulated rewards for each of these strategies (Fig. 3) shows that SymSeed has enhanced the performance of all the studied RL algorithms, resulting in higher rewards. Furthermore, SymSeed obtains higher rewards more rapidly than state initializations that generated at random across the entire state space. For the same experiments, an analysis of the visited states is presented in Table 1. The results demonstrate that the initial states provided by SymSeed allow RL algorithms (in this experiment, we present the results of CAT-RL) to explore a smaller number of new states compared to the baseline (0%), while achieving a higher reward. Figure 4 displays heatmap plots of visited states for each strategy while training in the safari car environment using CAT-RL. The weight of points in the plots refers to the frequency with which a given state has been visited. As the training for all strategies is 100,000 episodes, a higher number in this plot indicates a lower number of new states visited. Accordingly, initializing RL algorithms with SymSeed provided states not only accelerates the reward accumulation but also enhances exploration efficiency by focusing on more relevant regions of the state space. This targeted exploration allows the RL algorithms to achieve better performance with reduced computational overhead compared to random state initializations.

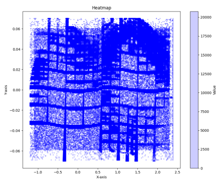
RQ2. The success rate for each of the aforementioned initialization strategies is shown in Fig. 5, which illustrates that each of those algorithms exhibits a superior success rate in the presence of SymSeed-provided initial states than in their absence. However, there is no consistency in the observed outcomes when the percentage of SymSeed seeded states is increased. In some cases, including more SymSeed states led to improved performance, while in other cases, the opposite was true. Furthermore, while SymSeed-provided initial states generally enhance success rates across RL algorithms, the optimal balance of SymSeed states versus random states varies depending on the



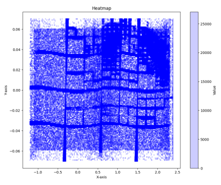
(a) visited states in 0%



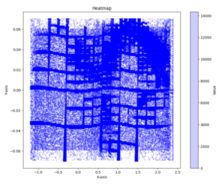
(b) visited states in 10%



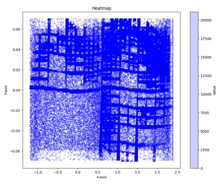
(c) visited states in 20%



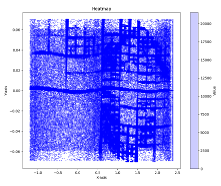
(d) visited states in 30%



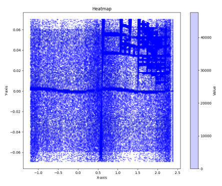
(e) visited states in 40%



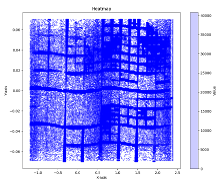
(f) visited states in 50%



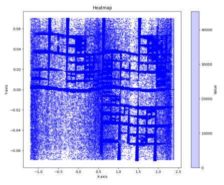
(g) visited states in 60%



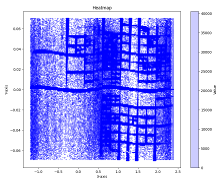
(h) visited states in 70%



(i) visited states in 80%



(j) visited states in 90%

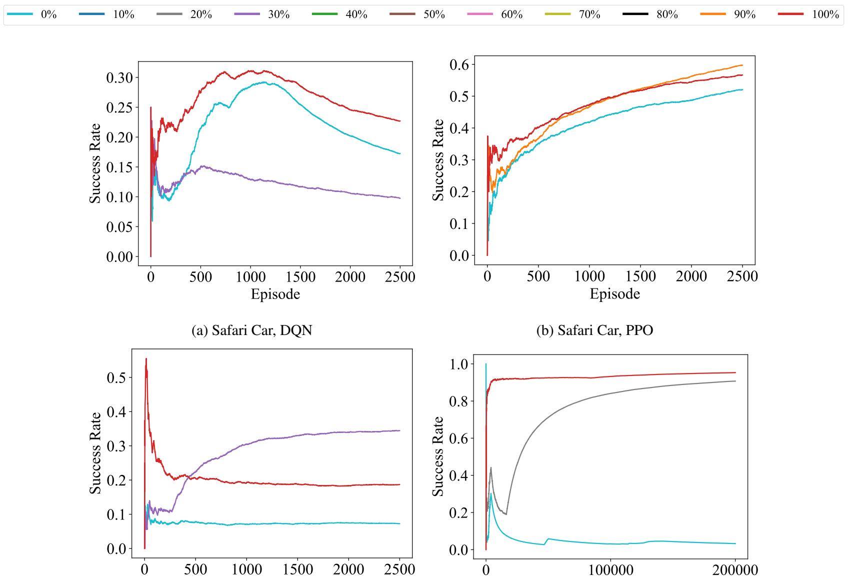


(k) visited states in 100%

Figure 4: Visited states of Safari car example during 100K episodes training of CAT-RL.

	Num new states	**Max Frequency**	**Mean Frequency**	**Std**
0%	159028	35662	5.64	127.03
10%	176573	29131	10.82	112.74
20%	194564	20766	11.15	80.69
30%	153059	27072	12.84	249.43
40%	178155	14351	11.93	87.79
50%	177352	21108	10.82	117.84
60%	149587	21543	4.99	118.35
70%	109722	47648	11.7	307.09
80%	160338	40829	11.72	264.9
90%	138977	45349	11.74	329.14
100%	145279	40533	9.31	267.8

Table 1: Visited states for Safari Car using CAT-RL.



(a) Safari Car, DQN

(b) Safari Car, PPO

(c) Safari Car, A3C

(d) Safari Car, CAT-RL

Figure 5: The success rate is calculated during the learning by counting success for all training episodes. The baseline (0%), all initial states generated by SymSeed (100%), and the best mixed method are selected.

specific algorithm and environment. This indicates that the integration of SymSeed states should be carefully calibrated, as an increase does not always correlate with improved performance, suggesting a nuanced interaction between state initialization and algorithm dynamics.

RQ3. Figure 7 shows that CAT-RL initialized with SymSeed seeded states, converges to an optimal policy faster than with other initialization strategies. This phenomenon can be attributed to the fact that the agent may start from states that are in close proximity to the final states (given that

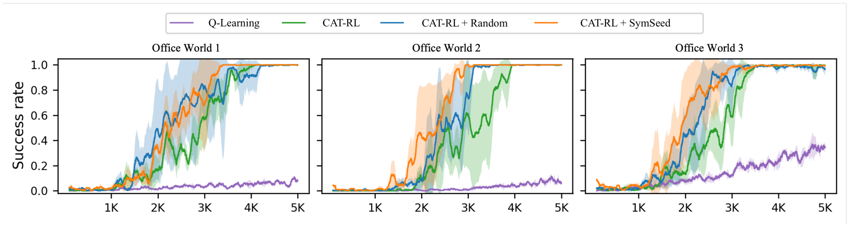


Figure 6: Success rate for 3 runs averaged over the last 100 training episodes of 4 methods on 3 Office World domains with 5K episodes.

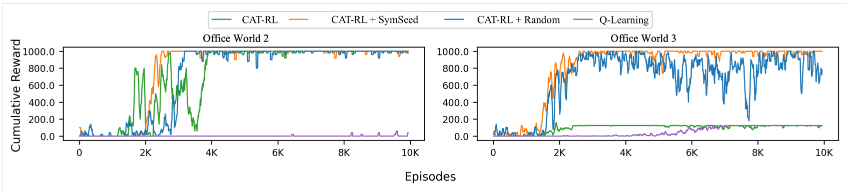


Figure 7: Cumulative reward for a single run of running 4 methods on Office World-modified map and Office World-variant map with 10K episodes.

in many cases, the final states are conditionally defined, and SymSeed is capable of acquiring this knowledge through symbolic execution). This allows the agent to observe the final state at an early stage, thereby enabling the distribution of the reward across the neighboring states of the final state. Consequently, the impact of the reward is distributed rapidly, which is equivalent to an environment with a non-sparse reward. Subsequently, initializing with SymSeed-seeded states provides a strategic advantage by facilitating early exposure to rewarding states, thereby accelerating policy convergence. This advantage highlights the potential of SymSeed to optimize RL training efficiency by transforming sparse reward environments into effectively denser ones, improving both learning speed and stability.

6. Conclusion

The findings of this study indicate that reinforcement learning algorithms are sensitive to initial states. The acquisition of additional knowledge about the environment can facilitate more optimal seeding of initial states. The paper shows that the enhanced seeding enabled by SymSeed facilitates more effective exploration and performance of RL algorithms. In this context, a pre-analysis with an environment simulator allows for the generation of a set of initial states for RL algorithms that can enhance exploration and facilitate more effective response to sparse rewards.

References

Abel, D., Jinnai, Y., Guo, S. Y., Konidaris, G., & Littman, M. (2018). Policy and value transfer in lifelong reinforcement learning. In *International Conference on Machine Learning* (pp. 20–29). PMLR.

Andrychowicz, M., Raichuk, A., Stanczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., Gelly, S., & Bachem, O. (2020). What matters in on-policy reinforcement learning? A large-scale empirical study. *CoRR*, *abs/2006.05990*. URL: https://arxiv.org/abs/2006.05990. arXiv:2006.05990.

Baldoni, R., Coppa, E., Delia, D. C., Demetrescu, C., & Finocchi, I. (2018). A survey of symbolic execution techniques. *ACM Computing Surveys (CSUR)*, *51*, 1–39.

Ball, T., Daniel, J., & Ball, T. (2015). Deconstructing dynamic symbolic execution. In *The 2014 Marktober Summer School on Deop*. IOS Press. URL: https://www.microsoft.com/en-us/research/publication/deconstructing-dynamic-symbolic-execution/.

Barreto, A., Hou, S., Borsa, D., Silver, D., & Precup, D. (2020). Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, *117*, 30079–30087.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., & Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, *29*.

Bellman, R. (1957). A Markovian decision process. *Journal of mathematics and mechanics*, (pp. 679–684).

Cadar, C., Dunbar, D., Engler, D. R. et al. (2008). Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI* (pp. 209–224). volume 8.

Chen, Y.-F., Tsai, W.-L., Wu, W.-C., Yen, D.-D., & Yu, F. (2021). PyCT: A python concolic tester. In *Programming Languages and Systems: 19th Asian Symposium, APLAS 2021, Chicago, IL, USA, October 17–18, 2021, Proceedings 19* (pp. 38–46). Springer.

Dadvar, M., Nayyar, R. K., & Srivastava, S. (2023). Conditional abstraction trees for sample-efficient reinforcement learning. In *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence* (pp. 485–495). PMLR. URL: https://proceedings.mlr.press/v216/dadvar23a.html iSSN: 2640-3498.

Fruit, R., & Lazaric, A. (2017). Exploration-exploitation in mdps with options. In *Artificial intelligence and statistics* (pp. 576–584). PMLR.

Ghosh, D., Singh, A., Rajeswaran, A., Kumar, V., & Levine, S. (2017). Divide-and-conquer reinforcement learning. *arXiv preprint arXiv:1711.09874*, .

Jang, S., & Kim, H.-I. (2022). Entropy-aware model initialization for effective exploration in deep reinforcement learning. *Sensors*, *22*, 5845.

Kim, J., Hyeon Park, J., Cho, D., & Kim, H. J. (2022). Automating reinforcement learning with example-based resets. *IEEE Robotics and Automation Letters*, *7*, 6606–6613.

King, J. C. (1976). Symbolic execution and program testing. *Communications of the ACM*, *19*, 385–394.

Kolter, J. Z., & Ng, A. Y. (2009). Near-bayesian exploration in polynomial time. In *Proceedings of the 26th annual international Conference on Machine Learning* (pp. 513–520).

Kraemer, L., & Banerjee, B. (2014). Reinforcement learning of informed initial policies for decentralized planning. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, *9*, 1–32.

Machado, M. C., Bellemare, M. G., & Bowling, M. (2020). Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 5125–5133). volume 34.

Messikommer, N., Song, Y., & Scaramuzza, D. (2024). Contrastive initial state buffer for reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 2866–2872). IEEE.

Mnih, V. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, .

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning* (pp. 1928–1937). PMLR.

Montgomery, W., Ajay, A., Finn, C., Abbeel, P., & Levine, S. (2017). Reset-free guided policy search: Efficient deep reinforcement learning with stochastic initial states. In *2017 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 3373–3380). IEEE.

Moore, A. W. (1990). *Efficient memory-based learning for robot control*. Ph.D. thesis University of Cambridge, UK. doi:10.1.1.17.2654.

Păsăreanu, C. S. (2020). *Symbolic Execution and Quantitative Reasoning: Applications to Software Safety and Security*. Morgan & Claypool Publishers.

Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning* (pp. 2778–2787). PMLR.

Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., & Dormann, N. (2019). Stable baselines3. https://stable-baselines3.readthedocs.io/.

Raileanu, R., & Rocktäschel, T. (2020). Ride: Rewarding impact-driven exploration for procedurally-generated environments. *arXiv preprint arXiv:2002.12292*, .

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning* (pp. 1889–1897). PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, .

Sharma, A., Gupta, A., Levine, S., Hausman, K., & Finn, C. (2021). Autonomous reinforcement learning via subgoal curricula. *Advances in Neural Information Processing Systems*, *34*, 18474–18486.

Strehl, A. L., & Littman, M. L. (2008). An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*, *74*, 1309–1331.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tarbouriech, J., Garcelon, E., Valko, M., Pirotta, M., & Lazaric, A. (2020). No-regret exploration in goal-oriented reinforcement learning. In *International Conference on Machine Learning* (pp. 9428–9437). PMLR.

Uchendu, I., Xiao, T., Lu, Y., Zhu, B., Yan, M., Simon, J., Bennice, M., Fu, C., Ma, C., Jiao, J. et al. (2023). Jump-start reinforcement learning. In *International Conference on Machine Learning* (pp. 34556–34583). PMLR.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*, 279–292.

Zha, D., Ma, W., Yuan, L., Hu, X., & Liu, J. (2021). Rank the episodes: A simple approach for exploration in procedurally-generated environments. *arXiv preprint arXiv:2101.08152*, .