ALGORITHMS FOR BLACK-BOX SAFETY VALIDATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Anthony Corso

January 2021

Abstract

Autonomous systems have the potential to improve safety and efficiency in safety-critical domains such as transportation, and medicine. Since human lives are at risk in these applications, we require rigorous safety validation before deployment. Traditional safety validation approaches such as real-world testing and scenario-based testing in simulation are not scalable to complex systems and environments and may miss unforeseen failures. Formal verification techniques also lack the scalability required for large scale autonomy.

The thesis address the safety validation problem with black-box sampling techniques, which assume no knowledge of the design of the autonomous system. The system takes actions in a stochastic environment and failures are discovered by sampling environmental disturbances. The black-box assumption allows for better scalability to complex autonomous systems and sampling can be combined with machine learning to discover unforeseen failures. Previous black-box safety validation approaches have been based on optimization, path-planning, reinforcement learning and importance sampling. Although successful for many safety validation applications, existing algorithms may have poor interpretabiliy, scalability, and efficiency.

Black-box sampling approaches can provide example failure trajectories but do not provide a high-level description of failures, as scenario-based approaches do. We present a new technique for generating failure descriptions in the form of signal temporal logic specifications on the environment disturbances. The specifications are optimized with genetic programming to produce failure examples and can used to gain insight into why a failure occurred.

A key contribution of this thesis is the proposal and analysis of a state-dependent sampling distribution to approximate the distribution over failures. The use of the state of the environment produces a more efficient sampling distribution than baseline importance sampling approaches, but may be limited by the size of the state space. To improve scalability, we propose a decomposition technique for multi-agent validation tasks. Each subproblem is solved independently and the results are combined for better performance than learning from scratch.

During the design of an autonomous system, safety validation is performed repeatedly, requiring a large computational expense. We propose a transfer learning technique that can reduce the number of required samples and lead to better performance. Knowledge from previous validation tasks is transferred to new tasks in the form of value functions that are combined using a learned set of attention weights. Results show improved knowledge transfer between tasks compared to baseline techniques.

The safety validation algorithms presented in this work are tested on two gridworld scenarios and two driving scenarios. A simple gridworld scenario is used to illustrate important safety validation concepts while a gridworld with multiple adversaries is used as a test case for multi-agent validation. A rules-based autonomous driving policy is tested in a crosswalk scenario with a pedestrian and a T-intersection scenario with multiple vehicles. It is shown that the presented algorithms can improve the interpretability, scalability and efficiency of safety validation.

Acknowledgments

This dissertation would not have been possible without the help of many people including my family, my friends, and everyone at Stanford who makes research possible. First, I would like thank my reading committee, Professor Dorsa Sadigh and Professor Marco Pavone, for providing valuable feedback on my research, defense, and dissertation. Their perspective and inquiry helped hone my research direction, and both have provided valuable advice and opportunities outside of my dissertation work. Then, I would like to thank my advisor Professor Mykel Kochenderfer for mentoring me in research and in life. Mykel helped me discover the wonders of machine learning and decision making, taught me how to communicate more clearly and succinctly, and demonstrated how to live a life full of passion, gratitude, and kindness. I am incredibly grateful for everything I have learned from Mykel, and look forward to our continued collaborations as co-workers and friends.

My first three years at Stanford were funded by the Lillie Family Stanford Graduate Fellowship. Without that fellowship I may not have attended Stanford, so I thank the Aeronautics and Astronautics department for nominating me and the Lillie Family for providing the funds. I want to thank the Stanford Center for AI safety for funding my dissertation research, and specifically NVIDIA for providing the funding and research direction. Thank you to Michael Cox and Ahmed Nassar at NVIDIA for always providing feedback on my work and supporting the safety center. Thank you to NASA Ames and Dr. Ritchie Lee for supporting my summer internship at NASA Ames and collaborating on several projects. Lastly, I would like to thank the NASA University Leadership Initiative for supporting my last quarter of research, and Professor Marco Pavone for providing the opportunity.

I have had the opportunity to collaborate with many amazing friends and researchers. I would first like to thank Dr. Ashish Goel, Dr. Paul Tarantino and especially Dr. Siddarth Krishnamoorthy for giving me valuable research experience and advice as I navigated my first several years of graduate school. Thank you to Dr. Jeremy Morton, Professor Freddie Witherden, Mark Koren, Xiaobai Ma, Peter Du, and Professor Katherine Driggs-Campbell, for collaborating on my first research projects as a member of SISL. Thank you to Dr. Ritchie Lee for developing many of the ideas that I built on for my thesis, and for being a valuable research mentor. Thank you to Robert Moss, Dr. Alex Koufos, Dr. Maxime Bouton, Dr. Di Wu, and all of the other SISL members who gave me advice and feedback on my research. Lastly, thank you to Sydney Katz and Chris Strong for helping me develop my next stage of research projects.

Thank you to my roommate Stephen for being a supportive and easy going living partner. Thank you to Krishna, Jasper, Jenny, John, and all of the other friends I have met (and climbed with) at Stanford, as well as my friends from Harvey Mudd and back home in Oregon. Thank you to my girlfriend Emma. She provided an endless source of advice, support, and love during my time in graduate school. Thank you for always having my back, cheering me on, and always being willing to share a laugh or go on an adventure.

Lastly, thank you to my family, to whom I truly owe everything. Thank you for teaching me to be the person I am today and providing me all the support I have needed to pursue my dreams. Thank you to my sister Amie, who can always make me feel better when I am in a bad spot, and whose enthusiasm about the work we do always leads to great conversations. Thank you to my Mom and Dad who have always encouraged me to think for myself and work hard for the things that are important. I love you all.

Contents

List of Tables

List of Figures

List of Algorithms

1 Introduction

With the recent increase in capability of autonomous systems, they are being deployed in safety-critical domains such as autonomous driving, aviation, and medicine where the consequences of operational errors include loss of property or human life. Ensuring the safe operation of safety-critical autonomous systems is a necessary step before deployment, but remains a significant technical challenge due to the complexity of systems and the environments in which they operate. This thesis presents scalable, interpretable, and efficient methods for validating the safety of autonomous systems in simulation. This chapter first discusses the general problem of autonomous system safety and argues that safety validation through black-box sampling plays a crucial role. The next sections goes on to give an overview of the challenges of black-box to safety validation approaches, followed by the contributions and outline of the thesis.

1.1 Safety Validation of Autonomous Systems

Introducing autonomy into safety-critical domains has the potential to improve both safety and efficiency. Driving accidents killed 1.35 million people in 2016 [1], and approximately 94% of accidents between 2005–2007 in the US were due to driver error [2]. Autonomous aircraft collision avoidance systems were put into place after a series of devastating mid-air collisions [3] and have been shown to reduce the (already rare) risk of mid-air collision by at least a factor of 3 [4]. The next generation of aircraft collision avoidance systems will improve safety and function in the higher-density airspaces of the future [5].

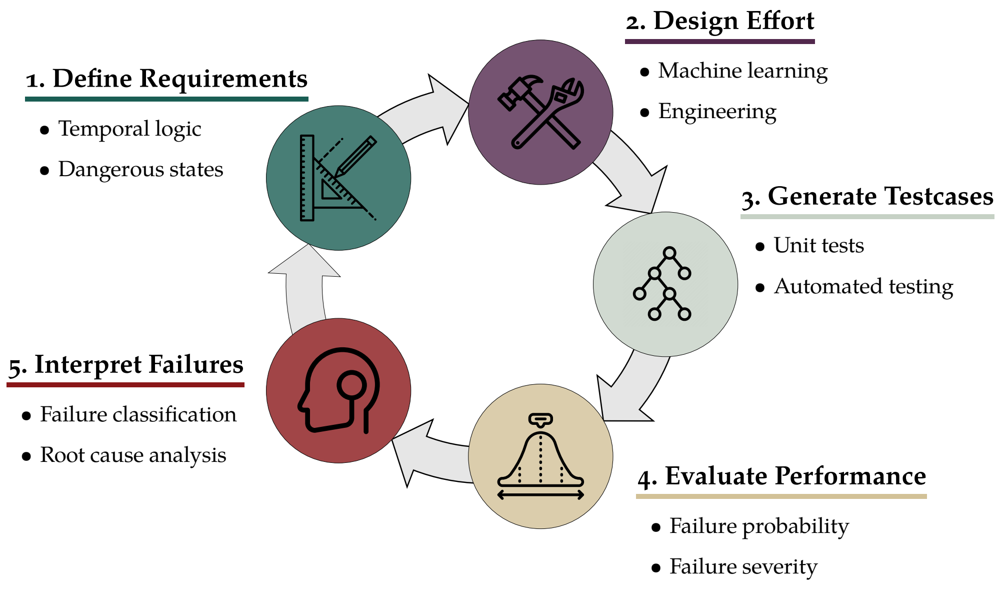


Figure 1.1: The design cycle of an autonomous system.

Although the upside of automation can be large, there are serious risks involved in deploying such systems. On March 18, 2018, a vehicle controlled by an autonomous system struck and killed a pedestrian in Arizona. A report of the incident found that the vehicle observed the pedestrian as an unknown object 6 seconds prior to the collision but failed to classify the pedestrian and gave varying predictions for the future path of travel. It took an additional 4.7 seconds for the autonomous system to determine that emergency braking was required [6]. Another example of risk comes from aircraft collision avoidance systems. Aircraft collision avoidance systems have the potential to induce a small number of mid-air collisions that would not have otherwise occurred [4]. The number of induced collisions is much smaller than the number of prevented collisions, but these induced collisions highlight the possibility of bad outcomes for systems that are improperly designed, tested or deployed.

To understand how to incorporate safety into autonomous system, we must first understand how such systems are designed. The design cycle of autonomous systems is shown in Figure 1.1. The first step in system design is the *definition of requirements* which may include specifications on performance, safety, interpretability, and more. Then, some *design effort* is expended to develop a prototype of the system. Modern and future autonomous system design will usually involve a combination of machine learning and traditional engineering. Once a prototype of the system is available, it must undergo *testing* in the form of unit tests of specific components and integration tests of the larger subsystems. The generation of these test cases can be done from expert knowledge or can be automated. Based on the results of the tests, the system is *evaluated* on any number of performance metrics. Safety metrics may include the probability the system violates its safety specification (referred to as a *failure*) or the severity of observed failures. At this point, if the systems meets all required performance and safety specifications, it can be deployed. If not, then the failures of the system need to be *interpreted* to discover discover broad categories of failures, called *failure modes* and to perform root-cause analysis to discover flaws in the system. Once failures are identified and categorized, the system requirements may need to be updated and the cycle continues until the system is ready for deployment.

Safety can and should be incorporated at all stages of the design cycle. This thesis focuses primarily on *safety validation*, which covers the stages of testing, performance evaluation and failure interpretation, but we will first briefly describe how safety can be included in the stages of requirement definition and design.

The most straightforward way of including safety in the definition of system requirements is to identify what is meant by a failure and stipulate a maximum allowable failure probability or failure severity. Although this type of requirement provides a concrete safety goal, it does not necessarily help the design process more easily produce a safe system. A thorough analysis of the environment and system may allow for a set of specifications that can guarantee safety under a set of assumptions. For autonomous driving, the responsibility-sensitive safety model [7] and the safety force field model [8] both develop a set of rules that restrict the actions of a driving agent to ensure safety. These specifications are not comprehensive enough to define a complete driving policy, but an agent based on machine learning may be trained to satisfy a goal while respecting such specifications [9], [10].

Designing a safe autonomous system with machine learning can be especially challenging. Approaches to improve safety of machine-learned systems include defining objectives that adequately penalize rare but catastrophic failures [11] or are risk-aware [12], learning from safe human demonstrations [13], and training on adversarial examples [14]. For agents that continue to learn in their operational environment, they will need to perform safe exploration which can be done by only allowing exploratory actions that are reversible [15], through active learning [16], or through uncertainty-awareness [17]. For a good overview on the topic of safe reinforcement learning see Garcıa and Fernández [18] and Amodei, Olah, *et al.* [19].

Safety validation is performed on an existing prototype of a system and can have several objectives such as discovering failures of the system, computing the likelihood that the system fails, or understanding which environmental factors might lead to a system failure. There are many techniques that are used for safety validation, each with pros and cons. Below we discuss four categories of validation approaches.

Scenario-based testing Usually done in simulation, scenario-based testing is a form of unit testing where scenarios are developed to challenge specific behaviors or components of an autonomous system. These scenarios can be constructed from real-world data or through expert knowledge, and can be designed to focus on potential weaknesses of the system. Scenario-based testing can be used for regression testing, where the same suite of scenarios is applied to the system each time it changes to make sure no old failure modes have been re-introduced. The downside to scenario-based testing is the possibility of missing important failure cases because they were not included the suite of tests. Important tests may be excluded due to incorrect assumptions about the environment or due to the difficulty of predicting failures caused by the complex interaction of many components.

Real-world testing In real-world testing, a prototype of the system is used in an environment that is as close as possible to the intended deployment environment. For autonomous driving this may be a closed course or an area of a city that has already undergone extensive mapping. Usually, the system will be equipped with extra safety features to mitigate risk. This form of testing is critical before deployment because there may be complications that arise when moving from simulated environments to the real world. The downside to real-world testing is the high cost (in both time and other resources) and the high consequence if the system behaves unsafely.

Formal verification Formal verification techniques can provide mathematical proofs about system safety under a set of assumptions. The system must first be described by a mathematical model such as a finite state machine, computer code, or the parameters and architecture of a neural network. The model is checked against a safety specification usually written in a temporal logic. In *model checking* all inputs to the system are exhaustively checked to discover any outputs that violate the safety specification. If there are no violations, then the system is concluded to be safe. Another approach is *deductive verification* where a proof of safety is constructed through a series of intermediate theorems about the behavior of the system. These proofs are often constructed with automated theorem provers. Formal verification gives the highest degree of confidence in the safety of a system under the assumption that the mathematical model adequately represents the true system. The main drawback to formal verification is that it may not scale well to systems with large or continuous state spaces operating in environments with a large number of inputs to the system.

Black-box sampling and learning Black-box sampling techniques involve repeatedly simulating the system in a stochastic model of the environment to discover failures. The term *black-box* comes from the assumption that we do not know anything about the structure of the system and can only interact with it through inputs and outputs. This assumption allows black-box sampling approaches to be applied to complex systems. In the case where failures are rare, random simulation may be inefficient for finding failure examples or computing the probability of failure. In this case, learning techniques can be applied to iteratively improve the stochastic parameters of the environment to make failures more likely to occur. The downside to black-box sampling techniques and learning is that we will never be fully confident in the safety of the system, and instead must rely on probabilistic statements of safety.

While all of the safety validation approaches are useful in different circumstances, we focus on black-box safety validation due to its favorable scalability and the ease with which it can be combined with powerful machine learning algorithms.

1.2 *Challenges to Black-Box Safety Validation*

There are many challenges for algorithms that use black-box sampling including accurate modeling of the system and the environment, efficiency in the number of simulations required, and interpretability of discovered failures.

Although not discussed in detail in the rest of this thesis, the challenge of accurately modeling an autonomous system interacting with a physical environment is crucial to the success of simulation-based validation strategies. To accurately model the physical world, we need to define the dynamics of a large number of continuous and discrete variables. In general, the dynamics may be stochastic and nonlinear which can pose significant computational challenges. In addition, there may be other decision-making agents (such as humans) in the environment whose decisions we must model. Consider, for example, that we wish to test an autonomous vehicle (AV) navigating through an intersection with human drivers. To do this in simulation, we must accurately model the 3D environment, the dynamics of the AV and the other vehicles, the functioning of all sensors on the AV, the behavior of other drivers and pedestrians, and more. To address these modeling challenges, we can make simplifying assumptions about the environment, learn models from real-world data, or construct models from first principles. Although there are many open questions in the modeling literature, we assume that for any system we wish to validate, there already exists a sufficiently good simulator of the environment and that any source of stochasticity (such as the behavior of other agents) is accurately modeled.

Once a simulator of the system and the environment is defined, the next biggest challenge is efficiently discovering failures of the system due to variations in the environment over time. Suppose that the stochastic elements of the environment can be described by M disturbance variables over T timesteps. If each disturbance can take on K values (assuming a discretization for continuous variables), then the number of possible disturbance trajectories is K^{MT}. In the worst case, we must enumerate a combinatorially large number of possibilities to discover a failure, which becomes intractable for even moderate values of M and T. The problem becomes more challenging when the system or the environment is stochastic, so the same disturbance trajectory may have different outcomes for different trials. To handle stochasticity or to help guide the search for failures, we may wish to consider the state of the environment when choosing disturbances by constructing a policy that maps states to disturbances. If we assume that the environment has N states and that the policy depends only on the current state (Markov assumption), then the total number of possible policies is K^{MN}. In practice, the problem of combinatorial complexity may be overcome through the use of search heuristics based on optimization, path-planning or reinforcement learning, which will be discussed in detail in the next chapter. These techniques have demonstrated good empirical results but offer no guarantees of finding failures without exhaustive search.

For some systems, we may not require safety for all possible disturbances but instead specify a maximum probability of failure assuming a distribution over disturbances. For example, no AV can avoid an accident in all scenarios [7] but we hope to develop AVs with a failure probability of less than 10^{-9} per hour [20]. The benefit of this approach is that estimating the probability of failure is independent of the number of possible disturbance and state trajectories, and only depends on the sampling distribution and number of samples. The number of samples required, however, may still be intractably large. Consider the goal of verifying that the probability of failure of an AV in a given driving scenario is no larger than 10^{-9}. To be confident in this conclusion using Monte Carlo estimation, we would require on the order of 10^9 samples [20]. This number of samples may be much lower than an exhaustive search of disturbance trajectories, but is still infeasible when dealing with computationally expensive driving simulators, and the need to repeat this process for many driving scenarios and versions of the AV. To overcome this challenge, *importance sampling* approaches construct a new sampling distributions where failures are more likely, leading to better sample efficiency.

In order to uncover the flaws in a system, we must first understand under what circumstances the system fails. When performing black-box safety validation we generate failure trajectories that are high dimensional (each with $M \times T$ dimensions) so they may not provide any insight into what caused the system to fail. Even with many failure examples it is not clear how to induce the causal environmental factors that lead to failure, making it difficult to understand what needs to be fixed in the system. Currently, the best approaches for understanding what caused a system to fail is the use of unit tests that test specific component behavior, or failure classification that groups failure examples into similar failure modes.

The algorithms presented in this thesis make progress toward better scalability, efficiency and interpretability for black-box safety validation. The intended goal of these algorithms is to develop more useful safety validation tools to help develop safer autonomy.

1.3 Contributions

This thesis provides a general formulation of the problem of safety validation for black-box autonomous systems. This formulation helps categorize different algorithms for safety validation and provides a framework in which to develop new safety validation algorithms. We address some of the major challenges of black-box safety validation–interpretability, scalability, and sample efficiency–by developing new algorithms.

To address the problem of interpretability, we introduce *interpretable safety validation*, an algorithm that produces human-understandable descriptions of scenarios that lead to failure. Traditional safety validation approaches may return a number of failure examples, but these examples can be high-dimensional and therefore be difficult to interpret and analyze. A simple description of the scenarios that lead to failure allows engineers to quickly identify flaws in the system and to generate a large number of failure examples for further analysis.

Safety validation of multi-agent systems is hindered by the exponential increase in states and actions with the number of agents in an environment. We address the problem of scalability by assuming multi-agent systems can be decomposed into a set of smaller problems that are easier to solve. The solutions to these smaller problems can then be recombined to more efficiently solve the full-scale safety validation problem. Our decomposition approach is shown to increase the efficiency of safety validation tasks in an autonomous driving setting.

When many related systems need to be validated, conventional approaches require the validation to start from scratch for each system, resulting in low sample efficiency. By applying techniques from transfer learning, we demonstrate that results from previous validations can be used to accelerate the process of validation of a new system, event when the systems have significantly different behavior. For autonomous systems that are under continuous development, or for an analysis of many related systems, our approach can provide a large improvement in sample efficiency.

1.4 Outline

The rest of the thesis is organized as follows.

Chapter 2 formulates the problem of safety validation and presents the notation used in the rest of this work. It then provides an overview of the existing techniques for the safety validation of black-box autonomous systems, categorized into approaches based on optimization, path-planning, reinforcement learning, and importance sampling. The motivations, benefits and drawbacks of each category are discussed and specific algorithms are outlined.

Chapter 3 discusses the practical considerations involved in formulating a safety validation problem from a system and simulator. We present four concrete example systems to be used in the remainder of the thesis. The first two examples are gridworld problems with simple dynamics and are used to illustrate the core ideas of safety validation. The second two examples involve autonomous vehicles in complex environments and are used to demonstrate safety validation on a larger scale.

Chapter 4 argues for the importance of interpretable safety validation strategies and proposes a new interpretable safety validation algorithm based on temporal logic and expression optimization algorithms. The algorithm discovers temporal logic descriptions of disturbances that lead to failure. These descriptions are used to understand flaws in the system and to generate examples of high-likelihood failures. The algorithms is demonstrated on two autonomous driving scenarios.

Chapter 5 introduces the task of learning the optimal sampling distribution for estimating the probability of failure of sequential decision-making systems. We derive the optimal distribution in the case of a deterministic Markov simulator and suggest an effective distribution for stochastic or non-Markov systems. The proposed sampling distributions are able to generate a high rate of failures and quickly converge when estimating the probability of failure. These results are demonstrated through experimentation in the gridworld and autonomous driving settings.

Chapter 6 outlines the scalability challenges of computing the optimal sampling distribution for estimating the probability of failure and proposes a decomposition approach to improve scalability. When an environment consists of multiple interacting agents, we can model the problem as a combination of subproblems, each involving only two agents. The optimal sampling distribution can be computed for each subproblem and the optimal sampling distribution for the combined problem can be estimated through a combination of the subproblem solutions and a correction factor to account for multi-agent interactions. We demonstrate that this approach can more efficiently solve the safety validation problem of multi-agent systems through experimentation with a driving scenario with 5 agents.

Chapter 7 describes a new approach for improving sample efficiency of safety validation algorithms when they are applied to multiple systems that share some characteristics. The approach is based on *transfer learning*, where the insights gained during the safety validation of one system are applied to another, insofar as they are applicable. We demonstrate improvements in initial and final performance of learning-based safety validation algorithms and a reduction in the number of training steps required.

Finally, Chapter 8 provides a summary of the contributions of the thesis and discusses some of the remaining challenges for safety validation of black-box autonomous systems. Some possible future directions to address those challenges are presented.

2 *Background*

In order to develop algorithms for safety validation of black-box systems, we need to mathematically formalize the problem. In this chapter, we first define safety validation and the black-box assumption, then formulate safety validation as the interaction between two decision-making agents: the system and the adversary. We define notation and formalize the underlying goals of different safety validation algorithms. With the goals defined, we survey the existing literature on black-box safety validation and categorize different algorithms into four types: optimization, path planning, reinforcement learning and importance sampling.

2.1 *Definitions*

There can be multiple definitions for the terms *safety*, *validation*, and *black box*. Here we define how these terms are used in this thesis.

Safety Validation. A *safety* property specifies that a certain "bad event" will not occur. In contrast, a *liveness* property specifies that a certain "good event" will eventually occur. The safety-liveness distinction is important because a safety property can be shown to be violated with a concrete counterexample (the goal of the algorithms presented in this thesis), while the violation of a liveness property requires formal argumentation [21]. The definition of a "bad event" is domain specific.

Verification is the process of proving that the system meets its requirements while *validation* is the process of ensuring that a system fulfills its intended purpose in its operational environment [22]. Although many of the algorithms presented in this survey can be applied to both verification and validation, we choose the term *validation* to emphasize the focus on testing full-scale system prototypes in simulated operational environments. *Safety validation* is therefore the processes of investigating the adherence of a system to a safety property in its operational domain.

Black-Box Assumption. A system is said to be a *black box* if the system model is not known or is too complex to explicitly reason about. In contrast, a *white-box* system can be described analytically or specified in a formal modeling language, and a *gray-box* system lies in between. Some white-box systems may be treated as a black box if knowledge of their design does not help the validation process. For example, while small neural networks can have properties formally verified by analyzing the network weights [23], large neural networks with millions or billions of parameters are generally too large for such techniques, and they would need to undergo black-box validation. In some cases, both the system and the environment are treated as a black box, which precludes the use of validation algorithms that require the environment state.

2.2 *Problem Formulation*

Suppose we have an autonomous system under test (referred to as the *system*) that takes actions $a \in A$ in an *environment* with state $s \in S$. Actions are selected based on observations $o \in O$ of the environment. The system interacts with the environment over discrete time $t \in \{1, \dots, t_{\max}\}$. We denote a state trajectory up to time t as $\mathbf{s}_{1:t} = [s_1, \dots, s_t]$, an observation trajectory $\mathbf{o}_{1:t} = [o_1, \dots, o_t]$, and an action trajectory $\mathbf{a}_{1:t} = [a_1, \dots, a_t]$. The system may be modeled by a function $\mathcal{M}$ that maps an observation trajectory to an action

$$a_t = \mathcal{M}(\mathbf{o}_{1:t}). \tag{2.1}$$

An *adversary* $\mathcal{A}$ produces disturbances $x \in X \subseteq \mathbb{R}^n$ in the environment with the goal of causing the system to fail. A disturbance trajectory is denoted $\mathbf{x}_{1:t}$ or just $\mathbf{x}$.

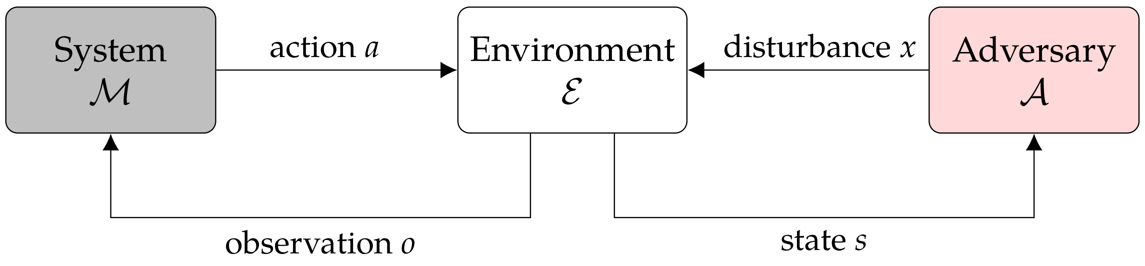


Figure 2.1: Model of the safety validation problem.

Although we present the most general case of disturbances as a trajectory of inputs to the environment, $\mathbf{x}$ may also represent static environment parameters or initial conditions.

The adversary can use full or partial knowledge of the environment state to determine the next disturbance such that

$$x_t = \mathcal{A}(\mathbf{s}_{1:t}). \tag{2.2}$$

Adversaries that use the system state require an environment that is constructed to provide that information. In cases where the state information is unavailable (possibly due to the simulator implementation or privacy concerns), the adversary must produce disturbance trajectories based solely on the outcomes of past trajectories.

The environment state evolves over time and is influenced by the actions of the system and the disturbances from the adversary. The environment is modeled by $\mathcal{E}$, where

$$s_{t+1}, o_{t+1} = \mathcal{E}(\mathbf{a}_{1:t}, \mathbf{x}_{1:t}). \tag{2.3}$$

The interaction between the system, the environment, and the adversary is depicted in Figure 2.1.

Some safety validation tasks require that the environment has a model of the disturbances. Let $p(\mathbf{x})$ be the probability density of the disturbance trajectory $\mathbf{x}$. If the disturbances are independent across time, then it is sufficient to define the density over a single disturbance $p(x)$, or if the disturbances depend only on the state it is sufficient to define $p(x \mid s)$. The disturbance model can be constructed through expert knowledge or learned from data.

In this work, we assume that the environment and the system are fixed, making disturbances the only way to affect the system. Therefore, from the point of view of the adversary, the system and environment can be combined into a single function f that maps disturbance trajectories into state trajectories

$$\mathbf{s} = f(\mathbf{x}). \tag{2.4}$$

Some algorithms require the ability to simulate a disturbance x_t for a single timestep from a state s_t. We denote the simulation step

$$s_{t+1} = f(s_t, x_t). \tag{2.5}$$

The desired operation of the system is described by one or more specifications ψ which are either written in a formal specification language or designed ad hoc. We overload the notation ψ to be the set of state trajectories that satisfy the specification and write $\mathbf{s} \in \psi$ when the state trajectory satisfies the specification and $\mathbf{s} \notin \psi$ when $\mathbf{s}$ violates the specification.

Temporal Logic. Safety specifications are often defined as statements in temporal logic, a category of languages used to reason about the temporal behavior of dynamic systems. Temporal logic statements evaluate to a Boolean for a (potentially infinite) sequence of states, or *path*. Temporal logic statements are generated from a set of propositional variables, logical operators and temporal modal operators. The logical operators include *conjunction* ($\wedge$), *disjunction* ($\vee$), and *negation* ($\neg$), while the temporal modal operators depend on the type of temporal logic.

Linear temporal logic (LTL) [24] includes the temporal modal operators *always* $\Box\psi$ (ψ is true on the entire path), *eventually* $\Diamond\psi$ (ψ is true anywhere on the path), and *until* $\psi_1\mathcal{U}\psi_2$ (ψ_1 holds at least until ψ_2 becomes true on the path). Computational tree logic (CTL) [25] and its more expressive counterpart CTL* [26] reason about futures with branching paths and therefore include the temporal modal operators of *all* $A\psi$ (ψ is true on all possible paths originating from the current state) and *exists* $E\psi$ (ψ is true on at least one path originating from the current state). Metric temporal logic (MTL) [27] allows for the temporal modal operators to be applied to a finite time interval I, denoted with a subscript ($\Box_I\psi$ means ψ always holds in the interval I). Signal temporal logic (STL) [28] extends the set of propositional variables to allow for real-valued signals, which are converted to Boolean statements using the comparison operators ($<, \leq, =, \geq, >$). For a concrete example, consider a real-valued variable d that represents the absolute distance between two vehicles. If we want to encode the safety specifications "the vehicles will not collide in the next 30 seconds" we can write

$$\psi := \Box_{[0,30]}(d > 0). \tag{2.6}$$

2.3 *Safety Validation Tasks*

Safety validation is concerned with finding disturbance trajectories that cause a system to failure and reasoning about the probability of those failures. We have identified four common safety validation tasks which are formally defined below.

Falsification: Find a counterexample to the specification.

$$\mathbf{x} \quad \text{s.t.} \quad f(\mathbf{x}) \notin \psi \tag{2.7}$$

Most likely failure analysis: Find the most likely counterexample.

$$\arg\max_{\mathbf{x}} \quad p(\mathbf{x}) \quad \text{s.t.} \quad f(\mathbf{x}) \notin \psi \tag{2.8}$$

Probability of failure estimation: Compute the probability that the system fails.

$$P_{\text{fail}} = \mathbb{E}_{p(\mathbf{x})}\left[\mathbb{1}\{f(\mathbf{x}) \notin \psi\}\right] \tag{2.9}$$

Note that the tasks are specified in order of increasing difficulty and utility. The first two tasks involve finding individual failure examples, with most-likely failure analysis being more challenging due to the need for maximizing the probability of the failure trajectory. To estimate the probability of failure, we need to discover many failure examples, each with relatively high likelihood. Thus, if we can generate a sampling distribution that produces many high-likelihood failure examples (solving the failure probability task), then we have effectively solved the task of falsification and most likely failure analysis.

2.4 Existing Approaches

The naive approach to finding failures of an autonomous system assumes that each disturbance trajectory leads to a binary outcome: failure or not failure. If that is the case, then the best we can do is search randomly over the space of disturbance trajectory until a failure is discovered. The space of disturbance trajectories scales exponentially with the dimension of the disturbance space and the length of the trajectories, so exhaustive search quickly becomes intractable. Fortunately, we can often gather more information about how close the system was to failure and use that information to guide the search toward failure trajectories. We encode the information as a safety metric $c_{\text{safe}}(\mathbf{s})$ over a state trajectory $\mathbf{s}$, with lower values indicating less safety.

The design of a safety metric is specific to the application and the type of failure. For collision avoidance applications in autonomous driving or aviation, a common choice is the *miss distance* (the closest physical distance between two agents) [29], [30] or the *time to collision* (the time until a collision if no intervention occurs) [31]. In aviation, a safety metric could include the deviation from a desired altitude [32], [33], or the off-center distance when taxiing down a runway [34].

More complex safety specifications such as abiding by traffic laws [35], [36] or other driving rules to avoid at-fault collisions [7], [37], [38] may require temporal logic specifications, in which case the temporal logic *robustness* $\rho(\mathbf{s}, \psi)$ can be used as a safety metric. The robustness is a measure of the degree to which the trajectory $\mathbf{s}$ satisfies the specification ψ. Large values of robustness mean that at no point does the trajectory come close to violating the specification, while low but positive values of robustness mean that the trajectory is close to violating the specification. A robustness value less than 0 means that the specification has been violated and gives an indication of by how much. The robustness for space-time signals can be computed recursively [39]–[41]. Upper and lower bounds on robustness can be computed for incomplete signals [42], which is useful when constructing a trajectory sequentially [42], [43]. The derivative of the robustness with respect to the state trajectory can be computed, which may help derive gradient-driven optimization algorithms [31], [44]. If the state variables that define robustness have large differences in scale then Zhang, Hasuo, *et al.* [45] proposed measuring the robustness of each state variable independently and using a multi-armed bandit algorithm to decide which robustness value to optimize on each iteration.

Once a safety metric is defined, we need to choose how to use it to guide the search for failures. The following sections describe safety validation algorithms that are based on optimization, path-planning, reinforcement learning and importance sampling. Optimization approaches search directly over the space of disturbance trajectories and is the technique of choice if no state information is available. Path-planning approaches maximize exploration in the environment state space to discover disturbance trajectories that lead to unsafe states, but do not naturally handle stochasticity in the simulator. Reinforcement learning approaches learn a policy (mapping states to disturbances) that leads to failures of the system but typically require the system and the environment to be Markov. Importance sampling approaches construct a distribution that makes failures more likely and can be used to estimate the probability of failure. For each category of approach, we provide a summary of algorithms and present one representative algorithm in detail. Lastly, we discuss the pros and cons of each category and summarize under which conditions each approach should be used. For a more detailed discussion of existing safety validation algorithms, refer to our survey paper [46].

2.4.1 Optimization

Optimization problems involve minimizing a cost function with respect to a design variable. For safety validation, the design variable is a disturbance trajectory $\mathbf{x}$ and the cost function $c(\mathbf{x})$ is a function of the safety metric. The optimal disturbance trajectory $\mathbf{x}^*$ is defined as

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} c(\mathbf{x}). \tag{2.10}$$

The cost function is designed such that

$$c(\mathbf{x}) \geq \epsilon \iff f(\mathbf{x}) \in \psi, \tag{2.11}$$

where ϵ is a safety threshold. Therefore, if any $\mathbf{x}$ causes $c(\mathbf{x}) < \epsilon$, then it is a counterexample. If the global minimum $c(\mathbf{x}^*) \geq \epsilon$, then no failures exist, but in practice, we can rarely be certain that we have found the true global minimum.

When the safety validation task is falsification, the cost function can simply be the safety metric induced by the disturbance trajectory

$$c(\mathbf{x}) = c_{\text{safe}}(f(\mathbf{x})). \tag{2.12}$$

To find the most-likely failure, we can modify the safety metric to include the likelihood of the disturbance trajectory $p(\mathbf{x})$. We can consider a piecewise objective that only considers the likelihood when the disturbance trajectory leads to a failure

$$c(\mathbf{x}) = \begin{cases} c_{\text{safe}}(f(\mathbf{x})) & \text{if } c_{\text{safe}}(f(\mathbf{x})) \geq \epsilon \\ -p(\mathbf{x}) & \text{if } c_{\text{safe}}(f(\mathbf{x})) < \epsilon. \end{cases} \tag{2.13}$$

Piecewise objectives may be more difficult to optimize so another option is to define a multiobjective cost function defined as

$$c(\mathbf{x}) = c_{\text{safe}}(f(\mathbf{x})) - \lambda p(\mathbf{x}), \tag{2.14}$$

where $\lambda > 0$ is user-specified. For the appropriate choice of lambda, both objectives should yield the same optimum.

formulating safety validation as an optimization problem is that it allows for the use of many existing optimization algorithms (see Kochenderfer and Wheeler [47] for an overview). Due to the complexity of many autonomous systems and environments, the optimization problem is generally non-convex and can have many local minima. Therefore it is common to use global optimization algorithms that can avoid local minima or include a coverage metric in the cost function that encourages exploration [48]–[50]. Global and local approaches can be combined so that the global optimization algorithm identifies regions of interest in the space of disturbance trajectories and the local algorithm finds the best disturbance trajectory in a region [51]–[54].

There are many optimization algorithms that have been used for safety validation. Genetic algorithms [55], [56] maintain a population of disturbance trajectories that evolves over time due to trajectory mutations and crossover. Bayesian optimization [57]–[62] maintains a probabilistic surrogate model of the cost function and can therefore handle stochastic cost functions. If safety validation is formulated as a graph-traversal problem by discretizing the disturbance space, then ant-colony optimization can be used to find the optimal path [63]. Covariance matrix adaptation evolution strategy [64], simulated annealing [65] and globalized Nelder-Mead [66] are successful global optimization techniques that are commonly used in falsification software [67], [68]. We now give a more detailed description of simulated annealing to demonstrate how optimization is used for safety validation.

Simulated Annealing. An approach to stochastic global optimization known as simulated annealing (SA) uses a random walk around the disturbance space to minimize the cost function c. A temperature parameter β is used to control the amount of stochasticity in the method over time and a transition function $T(\mathbf{x}' \mid \mathbf{x})$ describes the probability distribution over the next disturbance trajectory $\mathbf{x}'$. SA has been an effective optimization algorithm for falsification [65], [69].

The basic approach is presented in algorithm 2.1. It begins by selecting a random starting disturbance trajectory $\mathbf{x}$ (line 2). At each iteration, a new disturbance trajectory $\mathbf{x}'$ is sampled from $T(\mathbf{x}' \mid \mathbf{x})$ (line 4). If the new trajectory is a counterexample, then it is returned (line 6), otherwise it is subjected to an acceptance test with probability $\exp(-\beta(c(\mathbf{x}') - c(\mathbf{x})))$ (line 7). If $\mathbf{x}'$ is accepted, then it replaces $\mathbf{x}$, otherwise $\mathbf{x}$ is left unchanged. The procedure repeats until the computational budget is exhausted.

Algorithm 2.1 Simulated annealing.

```
1: function SimulatedAnnealing($\beta$, $T$, $c$, $\epsilon$)
2:     Sample initial $\mathbf{x}$
3:     loop
4:         $\mathbf{x}' \sim T(\mathbf{x}' \mid \mathbf{x})$
5:         if $c(\mathbf{x}') < \epsilon$
6:             return $\mathbf{x}'$
7:         if UniformRand() $< \exp(-\beta(c(\mathbf{x}') - c(\mathbf{x})))$
8:             $\mathbf{x} \leftarrow \mathbf{x}'$
```

The main design choices of the algorithm are the annealing temperature β and the choice of the transition function T. The annealing temperature can be adjusted based on the number of accepted and rejected disturbance trajectories. A typical choice is to adjust β so that the next point is accepted approximately half of the time [65].

A common choice for transition function is to use a Gaussian distribution around the current point $\mathbf{x}$ with a standard deviation that is adjusted based on the ratio of accepted points [47]. This approach may not work well when the disturbance space has constraints that must be satisfied, such as lower and upper bounds on the possible disturbances [65]. Abbas, Fainekos, *et al.* [65] proposes the use of a *hit and run* approach to transitioning that respects constraints [65]. It follows three steps:

1. Sample a random direction $\mathbf{d}$ in the disturbance trajectory space.
2. Perform a line search in the direction of $\mathbf{d}$ to determine the range of α such that $\mathbf{x} + \alpha\mathbf{d}$ does not violate any constraints.
3. Sample α from this range according to a chosen distribution. The standard deviation of this distribution can be adjusted using the acceptance ratio to improve convergence.

Aerts, Minh, *et al.* [69] improved the hit and run scheme by suggesting that α be chosen for each disturbance dimension separately so that highly constrained dimensions do not restrict the step size of less constrained dimensions [69].

2.4.2 *Path-Planning*

Discovering failures can be framed as a path planning problem through the state space of the environment using the disturbances as control inputs. In path planning, there is an initial state s_0 and a set of failure states S_{fail} that we seek to reach by sequentially constructing a disturbance trajectory $\mathbf{x}$. The benefits to a path planning approach are the use of state information to guide the choice of disturbance and the ability to reuse partial trajectory segments. For a discussion of general path planning algorithms see the overview by LaValle [70].

Path planning algorithms were designed for robotic applications so they often assume that the system is *controllable*, where all states in the state space are reachable using some sequence of control inputs, and the dynamics are *differentiable*. In the safety validation setting, the control inputs are environmental disturbances which may have limited control over the system, especially if the system is robust to disturbances. In this case the reachable set of states may be a small subset of the state space. Additionally, since we assume the system is a black box, we cannot differentiate through the actions of the system or the dynamics of the environment. Therefore, most path-planning algorithms need to be modified to function for black-box safety validation. For example, the multiple shooting method of Zutshi, Deshmukh, *et al.* [71] frames falsification as a graph traversal problem over a discretized state space, where edges connect cells in the state space. Paths through the graph are initially constructed from disconnected segments, but the discretization is refined until the segments can be connected into a concrete trajectory. Las Vegas tree search [43] constructs a tree of disturbances from a predefined set of trajectory segments, and biases the search toward segments with extreme disturbance values. Below, we expand on one of the most common path-planning algorithms: rapidly-exploring random tree, and describe modifications that make it applicable to safety validation.

Rapidly Exploring Random Tree Rapidly-exploring random tree (RRT) is a path planning technique for efficiently finding failure trajectories [72]. A space-filling tree is iteratively constructed by sampling the state space and growing in the direction of unexplored regions. RRT has been applied to the falsification of black-box systems [42], [48], [49], [73]–[78].

Algorithm 2.2 Rapidly-exploring random tree.

```
1: function RRT($s_0$, $S_{\text{fail}}$)
2:     $T \leftarrow$ InitializeTree($s_0$)
3:     loop
4:         $s_{\text{goal}} \leftarrow$ SampleState()
5:         $s_{\text{near}} \leftarrow$ NearestNeighbor($T$, $s_{\text{goal}}$)
6:         $x_{\text{new}} \leftarrow$ GetDisturbance($s_{\text{near}}$, $s_{\text{goal}}$)
7:         $s_{\text{new}} \leftarrow f(s_{\text{near}}, x_{\text{new}})$
8:         AddNode($T$, $s_{\text{near}}$, $s_{\text{new}}$)
9:     return CounterExamples($T$, $S_{\text{fail}}$)
```

The basic approach is presented in algorithm 2.2. On each iteration, a random point in the state space s_{goal} is generated, which acts as the goal state for the next node to be added (line 4). The tree is searched for the node that is closest to the goal state s_{near} (line 5) based on some distance metric d. This node will act as the starting point when attempting to reach the goal. A disturbance x_{new} is generated that drives s_{near} toward s_{goal} (line 6). Since the system is a black-box, we cannot generally determine an x_{new} that causes $s_{\text{near}} = s_{\text{goal}}$ exactly. Instead we can use random sampling of x_{new} or a more advanced optimization procedure to get as close as possible. Lastly, the disturbance x_{new} is simulated, starting from s_{near}, resulting in a new state s_{new} (line 7) which is then added to the tree as a child of s_{near} (line 8). Note that if the simulator cannot be initialized to any state, then the trace can be simulated by starting at the root and simulating the disturbances through the branch containing s_{near}. The algorithm stops when the maximum number of iterations is reached, a suitable falsifying trajectory is found, or tree coverage reaches a specified threshold. Variants of RRT [42], [48], [49], [73]–[75], [77], [78] typically differ in their approach to state space sampling, choice of distance metric for nearest neighbor selection, or by adding additional steps that reconfigure the tree for improved performance.

In contrast to optimization approaches, RRT does not include a safety metric to guide the search. Instead, coverage metrics are often used to determine a stopping condition. Common metrics include dispersion of the tree nodes [48] and star discrepancy [42], [49], [75], which measures how evenly the nodes are distributed. In cases where the reachable set of states is small compared to the state space, coverage metrics may always have a low value. Esposito, Kim, *et al.* [48] suggest using the change in the coverage between iterations to determine when the reachable set has been sufficiently explored. To include a safety metric, Karaman and Frazzoli [79] developed RRT*, which uses a cost function to select the next state s_{goal} from a randomly chosen set of possibilities. Similarly, Kim, Esposito, *et al.* [73] maintains a distribution over s_{goal} that is biased toward regions of low cost.

To address the problem of reachability Kim, Esposito, *et al.* [73] suggest using a dynamics-informed distance metric to determine s_{near}, so that s_{goal} is reachable from s_{near}. They include an additional term to discount states that are repeatedly chosen as s_{near} but fail to get near s_{goal}. Such states are hypothesized to appear on the boundary of the reachable set and may hinder exploration inside the set. Koschi, Pek, *et al.* [78] handles reachbility by proposing a backwards RRT algorithm that starts at failure states and builds a tree backward towards a set of initial condition. On each iteration, the tree is reconstructed to ensure paths exist between the leaves of the tree and the root.

2.4.3 *Reinforcement Learning*

Safety validation can often be formulated as a Markov decision process (MDP) and solved with reinforcement learning algorithms. In the context of safety validation, an MDP [80] is a model for sequential decision making problems defined by a state space S, disturbance space X, a transition function P, a reward function R, and a discount factor γ. The state space S contains all possible states of the environment and system, and the disturbance space X contains the possible disturbances available to the adversary. The transition function is said to be *Markov* if the next state only depends on the current state and disturbance. The agent interacts with the environment over a series of *episodes*. Each episode starts with a random initial condition $s \sim S_0$ and then, at each step the adversary chooses a disturbance x, the MDP transitions to a new state s' with probability $P(s' \mid s, x)$, and receives a reward $R(s, x)$ discounted by a factor $\gamma \in (0, 1]$ for each step. The episode proceeds until a *terminal* state $s \in S_{\text{terminal}}$ is reached or a maximum number of timesteps is exceeded.

The adversary's behavior is controlled by a policy π that maps states to disturbances $x = \pi(s)$, with the goal of trying to maximize the expected sum of discounted rewards

$$\mathbb{E}\left[\sum_t \gamma^t R(s_t, x_t)\right]. \tag{2.15}$$

Since the reward is maximized (which is opposite of a cost function), the reward function is a measure of risk rather than of safety. The use of a policy removes the need to represent full disturbance trajectories and allows for the solution of long time horizon problems [30]. Since the policy is defined for all states, stochasticity in the transition function is handled naturally so the adversary does not need to re-plan at each step. For an overview of MDPs and their solvers see the texts by Kochenderfer [80] or Sutton and Barto [81].

There are several concepts from the MDP literature that are useful for understanding the algorithms in this section. The first is the notion of the value function $V^\pi(s)$, which is the expected discounted sum of future rewards when in the state s and then following the policy π. The value function can be computed as the solution to the Bellman equation

$$V^\pi(s) = \mathbb{E}\left[R\left(s, \pi(s)\right) + \gamma V^\pi(s')\right]. \tag{2.16}$$

The optimal policy π^* is defined as

$$\pi^*(s) = \arg\max_\pi V^\pi(s). \tag{2.17}$$

The action value function $Q^\pi(s, x)$ for a policy is the value of being in state s, applying disturbance x, and then following the policy π. It is defined by the Bellman equation

$$Q^{\pi}(s,x) = \mathbb{E}\left[R(s,x) + \gamma Q^{\pi}\left(s', \pi(s')\right)\right]. \tag{2.18}$$

If the size of the state space and disturbance space is discrete then V^{π} or Q^{π} can be computed exactly with matrix inversion or dynamic programming [80]. If the state or action space is large or continuous (as is often the case for cyber-physical systems), V^{π} and Q^{π} can be estimated through policy *rollouts* (where the policy is used to get a random trajectory of states) and *bootstrapping* (where the value of states in a rollout are estimated with a learned model) [81].

A major challenge in formulating an MDP is designing the reward function. A reward is given at each time step and added up over a trajectory, so for a safety metric to be used to guide the search, it must be computed at each state as $c_{\text{safe}}(s)$, and a failure must be defined by a set of failure states S_{fail}. A reward function that can be used for falsification is

$$R_{\text{Falsification}}(s,x) = \begin{cases} \lambda & \text{if } s \in S_{\text{fail}} \\ -c_{\text{safe}}(s) & \text{otherwise,} \end{cases} \tag{2.19}$$

where λ is a large positive constant. An approach known as adaptive stress testing (AST) [29], [82] defines a reward function that leads to the discovery of the most-likely failure as

$$R_{\text{AST}}(s,x) = \begin{cases} \lambda & \text{if } s \in S_{\text{fail}} \\ \log p(x \mid s) - \beta c_{\text{safe}}(s) & \text{otherwise,} \end{cases} \tag{2.20}$$

where β appropriately weights the safety metric so that it does not dominate the log-probability of the disturbance.

The first reinforcement learning algorithm we consider for safety validation is Monte Carlo tree search (MCTS). MCTS is an online planning algorithm that iteratively constructs a tree of possible disturbances to determine the best disturbance at each step. On each iteration, the algorithm recursively selects a branch to go down until it hits a leaf, then it chooses a new disturbance to add to the tree and estimates its value with a random rollout from that node. Branch selection can be done with the Upper Confidence Tree (UCT) [83] algorithm which optimally trades off between exploiting actions with high value and exploring new paths. For continuous disturbance spaces, progressive widening [84], [85] is used to continually sample new disturbances at all depths in the tree. Lee, Kochenderfer, *et al.* [29] uses MCTS to find failures in a collision avoidance system where the disturbances are the seeds of a the random number generator of the simulator. MCTS has also been used for falsification of a vision-based controller for aircraft taxiing [34], hybrid flight control laws [32], and deep neural networks [86]. Zhang, Ernst, *et al.* [87] combined MCTS with global optimization, using MCTS for exploration of the disturbance trajectory space and global optimization for refinement of the disturbance trajectories.

The second category of reinforcement learning algorithms that have been used for safety validation is deep reinforcement learning (DRL), which uses deep neural networks to represent the value functions, policies or both. If the disturbance space X is discrete (or can easily be discretized), then deep Q-learning (DQN) [88] can be used for falsification [36], [89] (discussed in detail below). For large or continuous disturbance spaces, the policy itself is represented by a neural network (with parameters θ) that takes the state as input and either outputs a disturbance directly (e.g. $x = \pi_\theta(s)$) or outputs parameters of a distribution from which a disturbance can be sampled (e.g. for a normal distribution $[\mu, \sigma^2] = \pi_\theta(s)$ and $x \sim \mathcal{N}(\mu, \sigma^2)$). The policy is optimized to produce higher rewards using the policy gradient method [90]. Policy gradient methods can suffer from high variance and can be unstable during optimization. To improve optimization stability, an approach known as trust region policy optimization (TRPO) [91] restricts the amount a policy can change at each step. TRPO has previously been used for falsification of autonomous vehicles [30], [92], [93]. When a simulator does not provide access to the state on each timestep, policy gradient approaches can use recurrent neural network (RNN) with long-short term memory (LSTM) layers as the policy [94], and update at the end of each episode [92].

Another drawback of policy gradient methods is their inability to learn from off-policy data. Without data reuse, these methods can require a large number of simulations to converge. Newer approaches combine policy gradient methods with value function methods to create the actor-critic paradigm, which can perform well on problems with continuous disturbance spaces while also using previous simulation data to improve sample efficiency. Actor-critic methods [95] use two neural networks, one for the policy (the actor network) and one for the value function (the critic network) and come in several varieties. Advantage actor critic (A2C) was used for falsification by Kuutti, Fallah, *et al.* [96]. Its more scalable counterpart, asynchronous advantage actor critic (A3C), was used by Akazaki, Liu, *et al.* [89]. Behzadan and Munir [97] use another actor-critic method known as deep deterministic policy gradient (DDPG) combined with Ornstein-Uhlenbeck exploration [98].

Deep Q-Learning In DQN [88], the optimal action value function is approximated by a deep neural network with parameters θ, $Q(s, a; \theta) \approx Q^*(s, a)$. The Q-network tries to minimize the loss with respect to a target network, and is able to learn from off-policy data by always estimating the value of the next state using its model. To minimize variance from off-policy updates, DQN uses an experience buffer that stores and randomizes previous state transitions before selecting them to train on. DQN has been successful for reinforcement learning problems with large state spaces such as atari games [88] and is sample efficient compared to many other DRL algorithms.

The algorithm is shown in algorithm 2.3. It starts by initializing the experience buffer B to the empty set (line 2 and the set of model weights θ using Xavier initialization [99] (line 3). Then, until the computational budget is exhausted, we sample a random initial state (line 6) and iterate through an episode. At each step, a disturbance is selected using an ϵ-greedy policy, which means that with probability ϵ, a random disturbance is selected (line 8) and with probability $1 - \epsilon$ the action with the largest value is selected (line 9. The reward and next state are generated by the MDP (lines 10 and 11), and the experience tuple is stored in the experience buffer (line 12). The experience buffer is usually randomized to break correlations between experience samples, kept to fixed size, and prioritized by the temporal difference error of each sample [100]. To update the Q-network, a minibatch of experience tuples is sampled from the experience buffer (line 13) and used to perform an update step (line 14) using stochastic gradient descent

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathbb{E}[L(y(s, x, s'; \theta^-), Q(s, x; \theta)] \tag{2.21}$$

where α is the learning rate and loss function L can be the mean squared error or the Huber loss [101]. The target is

$$y(s, x, s'; \theta^-) = \begin{cases} R(s, x) & \text{if } s' \in S_{\text{terminal}} \\ R(s, x) + \gamma \max\limits_{x'} Q(s', x'; \theta^-) & \text{if } s' \notin S_{\text{terminal}} \end{cases}. \tag{2.22}$$

The target network parameters θ^- are periodically updated to θ after a specified number of training steps (line 15). After training has finished, the algorithm returns the model parameters that represent the optimal action value function. This function can be used as a disturbance policy to produces failure examples.

Algorithm 2.3 Deep Q-learning.

```
1: function DQN($s_0$, $S_{\text{fail}}$)
2:     $B \leftarrow \varnothing$
3:     $\theta \leftarrow$ InitializeWeights()
4:     $\theta^- \leftarrow \theta$
5:     loop
6:         $s \sim S_0$
7:         while $s \notin S_{\text{terminal}}$
8:             With probability $\epsilon$ choose $x_t$ at random
9:             Otherwise, $x_t \leftarrow \max_{x_t} Q(s_t, x_t; \theta)$
10:            $r_t \leftarrow R(s_t, x_t)$
11:            $s_{t+1} \sim P(s' \mid s, x)$
12:            $B \leftarrow B \cup (s_t, x_t, r_t, s_{t+1})$
13:            Sample minibatch $(s_t, x_t, r_t, s_{t+1}) \sim B$
14:            $\theta \leftarrow \theta - \alpha \nabla_\theta \mathbb{E}[L(y(s, x, s'; \theta^-), Q(s, x; \theta)]$
15:            If a fixed number of steps has passed, $\theta^- \leftarrow \theta$
16:     return $\theta$
```

2.4.4 *Importance Sampling*

For many safety-critical applications, it is impossible to design a system that is robust to all possible disturbances. In that case, we may wish to know how likely it is for a system to fail. To estimate the probability of failure, we can sample disturbance trajectories from the distribution with density $p(\mathbf{x})$ and compute the empirical probability of failure. If the probability of failure is very small, i.e. a failure event is rare, then the Monte Carlo approach will require a large number of samples before converging to the true probability of failure [102]. To address this problem, we would like to artificially make failures more likely, and then weight them accordingly, to get an unbiased estimate of the probability of failure with fewer samples. This is the idea behind importance sampling.

Suppose we wish to estimate the probability that a system violates a specification. A failure occurs when the safety evaluation metric $c(\mathbf{x}) = c_{\text{safe}}(f(\mathbf{x}))$ is less than a safety threshold ϵ, represented here as the indicator function $\mathbb{1}\{c(\mathbf{x}) < \epsilon\}$. Similar to the optimization formulation, $c(\mathbf{x})$ is defined such that $c(\mathbf{x}) < \epsilon$ implies $f(\mathbf{x}) \notin \psi$. The probability of failure P_{fail} is the expectation over the probability density $p_{\mathbf{x}}$, i.e.

$$P_{\text{fail}} = \mathbb{E}_p \left[\mathbb{1}\{c(\mathbf{x}) < \epsilon\} \right]. \tag{2.23}$$

To get a Monte Carlo estimate from N samples of $\mathbf{x} \sim p(\mathbf{x})$, we construct the unbiased estimator

$$\tilde{P}_{\text{fail}} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\{c(\mathbf{x}) < \epsilon\}, \tag{2.24}$$

which has variance

$$\text{Var}_p \left[\tilde{P}_{\text{fail}} \right] = \frac{1}{N} P_{\text{fail}} (1 - P_{\text{fail}}). \tag{2.25}$$

Informally, to ensure a good estimate of the probability of failure we want $\text{Var}_p \left[\tilde{P}_{\text{fail}}) \right] \ll P_{\text{fail}}^2$. If P_{fail} is very small, then $\text{Var}_p \left[\tilde{P}_{\text{fail}} \right] \approx \frac{1}{N} P_{\text{fail}}$ so we require $N \gg 1/P_{\text{fail}}$. For safety critical applications we may require $P_{\text{fail}} \sim 10^{-8}$, which would require more than 10^8 samples, making Monte Carlo approaches intractable.

In importance sampling, we choose a proposal distribution $q(\mathbf{x})$ that makes failures more likely but has the property that $q(\mathbf{x}) > 0$ everywhere $p(\mathbf{x})\mathbb{1}\{c(\mathbf{x}_i) < \epsilon\} > 0$ (so all disturbances that lead to failure can be sampled from q). The proposal distribution is sometimes referred to as the biased, sampled, or importance distribution. The importance sampling estimate of the probability of failure is done by taking N samples drawn from q and computing the weighted average

$$\hat{P}_{\text{fail}} = \frac{1}{N}\sum_{i=1}^{N}\frac{p(\mathbf{x}_i)}{q(\mathbf{x}_i)}\mathbb{1}\{c(\mathbf{x}_i) < \epsilon\}. \tag{2.26}$$

The variance of the importance sampling estimate is given by

$$\operatorname{Var}\left[\hat{P}_{\text{fail}}\right] = \frac{1}{N}\mathbb{E}_q\left[\frac{(p(\mathbf{x})\mathbb{1}\{c(\mathbf{x}_i) < \epsilon\} - q(\mathbf{x})P_{\text{fail}})^2}{q(\mathbf{x})}\right]. \tag{2.27}$$

The goal of a good importance sampling distribution is to minimize the variance of the estimator $\hat{P}_{\text{fail}}$ so that fewer samples are needed for a good estimate. From Equation (2.27), we can see that a zero variance estimate can be obtained if we use the optimal importance sampling distribution

$$q^*(\mathbf{x}) = \frac{p(\mathbf{x})\mathbb{1}\{c(\mathbf{x}) < \epsilon\}}{P_{\text{fail}}}. \tag{2.28}$$

Generating this distribution is not possible in practice because $c(\mathbf{x})$ is a black box and the normalization constant P_{fail} is the very quantity we would like to estimate. The algorithms in this section seek to estimate the optimal importance sampling distribution $q^*(\mathbf{x})$.

The choice of a proposal distribution can significantly affect the performance of importance sampling algorithms and a bad choice can lead to estimates with larger variance than the basic Monte Carlo approach. For example, if $q(x)$ is very small, where $p(\mathbf{x})$ is relatively large, then the weight $p(\mathbf{x})/q(\mathbf{x}) \gg 1$ and some samples will dominate the probability estimate (Equation (2.26)). To identify if a bad proposal distribution is chosen consider the size of the weights or compute the effective sample size. When samples with large weights are being drawn, Kim and Kochenderfer [103] suggest limiting the maximum weight by clipping the proposal distribution in regions with large weights. Uesato, Kumar, *et al.* [104] suggest combining the importance sampling estimator with a basic Monte Carlo estimator to minimize the downside of a bad proposal distribution and Neufeld, Gyorgy, *et al.* [105] provide a principled way of choosing the best estimator from several possibilities.

Multilevel splitting [106] is a non-parametric approach to estimating the optimal importance sampling distribution based on Markov chain Monte Carlo sampling and has been applied to the safety validation of autonomous driving policies [107]. Other approaches try to classify the safety of each disturbance trajectory and use that classification to generate an efficient proposal distribution. Huang, Guo, *et al.* [108] builds a proposal distribution centered on the boundary between safe and unsafe failure trajectories. Uesato, Kumar, *et al.* [104] uses an estimate of the probability of failure with a rejection-sampling scheme to create an efficient proposal distribution. We now introduce the cross-entropy method which is the most commonly used importance sampling approach for safety validation.

cross-entropy method The cross-entropy method [109], [110] iteratively learns the optimal importance sampling distribution from a family of distributions $q(\mathbf{x};\theta)$ parameterized by θ. The optimal distribution parameters θ^* are found by minimizing the KL-divergence between a proposal distribution $q(\mathbf{x};\theta)$ and the optimal distribution $q^*(\mathbf{x})$, i.e.

$$\theta^* = \arg\min_{\theta} D_{\mathrm{KL}}(q^*(\mathbf{x}) \parallel q(\mathbf{x};\theta)), \tag{2.29}$$

where D_{KL} calculates the KL-divergence. Substituting the definitions, we can arrive at a stochastic optimization program

$$\theta^* = \arg\min_{\theta} - \int_{\mathbf{x}} q^*(\mathbf{x}) \log q(\mathbf{x}; \theta) d\mathbf{x} \tag{2.30}$$

$$= \arg\max_{\theta} \int_{\mathbf{x}} \frac{\mathbb{1}\{c(\mathbf{x}) < \epsilon\} p(\mathbf{x})}{P_{\text{fail}}} \log q(\mathbf{x}; \theta) d\mathbf{x} \tag{2.31}$$

$$= \arg\max_{\theta} \mathbb{E}_{\varphi} \left[\mathbb{1}\{c(\mathbf{x}) < \epsilon\} \frac{p(\mathbf{x})}{q(\mathbf{x}; \varphi)} \log q(\mathbf{x}; \theta) \right] \tag{2.32}$$

$$\approx \arg\max_{\theta} \frac{1}{N} \sum_{i=1}^{N} \left[\mathbb{1}\{c(\mathbf{x}_i) < \epsilon\} \frac{p(\mathbf{x}_i)}{q(\mathbf{x}_i; \varphi)} \log q(\mathbf{x}_i; \theta) \right], \tag{2.33}$$

where φ is any set of parameters and $\mathbf{x}_i$ are sampled from $q(\mathbf{x}; \varphi)$. Equation (2.33) can be solved analytically when the family of algorithms is in the natural exponential family (i.e. normal, exponential, Poisson, gamma, binomial, and others), and the solution corresponds to the maximum likelihood estimate of the parameters [110].

For an iterative solution to finding θ^*, we start by choosing a set of starting parameters θ_0 so that $q(\mathbf{x}; \theta_0)$ is close to $p(\mathbf{x})$. Then, we iterate $k = 0, 1, \ldots$

1. Set $\varphi = \theta_k$.
2. Draw samples $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ from $q(\mathbf{x}; \varphi)$.
3. Solve Equation (2.33) for θ_{k+1}.

One major challenge to this approach is the rarity of failure events. If all samples have $c(\mathbf{x}) > \epsilon$, then $\hat{P}_{\text{fail}} = 0$ and the algorithm may not converge to the optimal proposal distribution. One solution is to adaptively update the safety threshold ϵ at each iteration. At iteration k, a safety threshold ϵ_k and a rarity parameter ρ is chosen so that the fraction of samples that have $c(\mathbf{x}) < \epsilon_k$ is ρ. The parameter ρ is also known as the quantile level and is often set in the range $\rho = [0.01, 0.2]$ [103], [111]. The entire algorithm is shown in algorithm 2.4. Note that we assume ρN is an integer used for indexing on line 7.

The cross-entropy method has been used to estimate the probability of failure for aircraft collision avoidance systems [103] and autonomous vehicles [111]–[113].

Algorithm 2.4 Cross-entropy method.

```
1:  function CrossEntropy($p, q, \theta_0, \rho, c, \epsilon$)
2:      $\theta \leftarrow \theta_0$
3:      $k \leftarrow 0$
4:      loop
5:          Sample $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ from $q(\mathbf{x}; \theta_k)$
6:          Sort $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ by $c(\mathbf{x}_i)$
7:          $\epsilon_k \leftarrow \max(c(\mathbf{x}_{\rho N}), \epsilon)$
8:          $\theta_{k+1} \leftarrow \arg\max_{\theta} \frac{1}{N} \sum_{i=1}^{N} \left[ \mathbb{1}\{c(\mathbf{x}_i) < \epsilon\} \frac{p(\mathbf{x}_i)}{q(\mathbf{x}_i; \theta_k)} \log q(\mathbf{x}_i; \theta) \right]$
9:          if $\epsilon_k < \epsilon$
10:             break
11:         $k \leftarrow k + 1$
12:     return EstimateProbability($p, q_{\theta_k}$)
```

Typically, probability distributions in the natural exponential family are used [103], [111], [112] so that cross-entropy updates can be performed analytically. Huang, Lam, *et al.* [113] propose a method for using piecewise exponential distributions for more flexibility while retaining the ability to compute updates analytically. Sankaranarayanan and Fainekos [114] discuss piecewise uniform distributions over the disturbance space, and techniques for factoring the space to reduce the number of parameters needed.

2.4.5 Strengths and Limitations

Each category of safety validation algorithm has benefits and drawbacks that depend on safety validation task and the details of the system and environment. We allude to some of these strengths and weaknesses in the preceding sections but here we try to spell them out with clarity

Optimization The primary benefit of optimization-based safety validation is the minimal set of restrictions placed on the simulation of the system and the environment. The formulation we provide does not need access to the state of the environment and only needs to return the value of a safety metric of a given disturbance trajectory. The simulation state may be unavailable for logistical or privacy reasons so optimization-based approaches would be a good choice in those cases. If, however, the state is available and would be useful for finding failures then optimization approaches may not function as well as path planning or reinforcement learning approaches. If an environment has stochasticity beyond the disturbances, then optimization techniques such as Bayesian optimization can be used to account for it. The biggest drawback to optimization strategies is the need to optimize over the entire disturbance trajectory. When solving long time horizon problems, the disturbance trajectory can be high dimensional and consequently difficult to optimize over.

Path Planning Path planning algorithms rely heavily on the environment state to discover failures. A strength of path planning approaches is their ability to efficiently search over large state spaces by reusing trajectory segments. Path planning algorithms often provide a natural way to compute state space coverage which can be used to determine when sufficient testing has been done. A drawback to path planning algorithms is their inability to naturally handle stochasticity. Most path-planning algorithms rely on the ability to deterministically replay trajectory segments or initialize a simulator into a predefined state, which may be challenging for some simulators. Additionally if the problem has a long time horizon then prohibitively large trees may be required to find failure trajectories.

Reinforcement Learning Reinforcement learning algorithms are similar to path-planning approaches because they also rely on the environment state to function (unless specifically formulated otherwise as in Koren, Corso, *et al.* [82]). Monte Carlo tree search has a similar pros and cons to the path planning algorithms because it is similar in structure. Deep reinforcement learning algorithms, however, learn a policy that maps states to disturbances. The space of possible policies may be easier to optimize than the space of disturbance trajectories and policies can be applied to long time horizon problems. Uncontrolled stochasticity in the environment is naturally handled by DRL algorithms, which are designed to function in stochastic environments. The downside to DRL algorithms is that they may be sample inefficient and require complex training procedures compared to optimization and path planning approaches.

Importance Sampling Importance sampling approaches require finding many failure examples to learn a distribution over failures. Therefore, failure examples can, in principle, be found using any of the three previous approaches. The most common importance sampling approaches such as multilevel splitting and the cross entropy method function most similarly to optimization-based techniques because they search directly over the space of disturbance trajectories and do not require state information. These techniques therefore carry the same pros and cons as optimization techniques. As we will see in Chapter 5, however, we can construct importance sampling techniques that follow the framework of reinforcement learning instead.

2.5 *Discussion*

This chapter introduced our model for safety validation of black-box autonomous systems. We assume that a black-box system takes actions in an environment that can be influenced by an adversary through stochastic disturbances. The system and environment evolve over time, and the goal of the adversary is to find sequences of disturbances that cause the system to violate a safety specification, which is often defined using temporal logic.

We identified three common safety validation tasks. In falsification, we try to find *any* sequence of disturbances that lead to failure. In most likely failure analysis we try to find the failure example that has the largest probability according to our disturbance model. In probability of failure estimation, we compute the probability that the system fails under the disturbance model.

With the safety validation tasks defined, we survey the existing literature on black-box safety validation and identify approaches based on optimization, path planning, reinforcement learning and importance sampling. The techniques differ in how they use safety metrics to guide the search, how they deal with stochasticity in the simulator, whether or not they require the use of the environment state, and if they search for full trajectories, construct them iteratively, or solve for a disturbance policy.

In the following chapter we discuss some of the practical considerations for constructing a safety validation problem from a system and environment. We introduce four sample systems that will be used to test various safety validation algorithms in later chapters.

3 Sample Systems

In this chapter we describe how to formulate the safety validation problem as a Markov decision process. Given a system described as a policy and an environment modeled as an MDP, we construct an adversarial MDP whose solution uncovers failures of the system. We then give detailed examples of four MDPs and their adversarial versions which will be used in later chapters for safety validation experiments. First we describe a simple gridworld with success and failure states. The agent tries to reach a success state while disturbances perturb the transition of the agent. Then we describe a gridworld that has two agents: a system and an adversary. The system tries to arrive at a success state and fails if the adversary collides with it. The next two adversarial MDPs involve the safety validation of a rule-based autonomous driving policy. In the first MDP, an autonomous vehicle tries to avoid a crossing pedestrian where the disturbances are the pedestrian motion and sensor noise. In the second MDP, an autonomous vehicle tries to make an unprotected left turn onto a highway with other driving agents with disturbances applied to their accelerations and turning behavior.

3.1 A Model for Adversarial Testing

Suppose the system we wish to validate is described by a behavioral π that operates in a simulated environment represented by a Markov decision process (S, A, P, R, γ) which we call the *base MDP*. The system takes actions $a = \pi(s)$ and the environment transitions to the next state s' with probability $P(s' \mid s, a, x)$ where the disturbance $x \sim p(x \mid s)$. If x includes all stochasticity in the MDP then $P(s' \mid s, a, x) = 1$.

To formulate the safety validation problem we construct an *adversarial MDP*. The solution to the adversarial MDP is an adversarial policy $\pi_{\text{adv}}(x)$ that can be used to generate disturbances that cause failures of the system. The adversarial MDP shares the same state space and transition model as the original MDP but the action space becomes the set of disturbances X. The adversarial reward function R_{adv} and discount factor γ_{adv} depend upon the specific safety validation task and MDP solver. The adversarial MDP is therefore described by $(S, \pi, X, P, p, R_{\text{adv}}, \gamma_{\text{adv}})$.

The description of a failure is encoded in the adversarial reward function R_{adv} and depends on the specific MDP. A common way to describe a failure is to identify a set of failure states S_{fail} and give a reward when one of those states is reached. As discussed in Section 2.4.3, we can include a safety metric to help guide the adversary toward failures and we can include the disturbance probability when performing a most likely failure analysis. Another choice for the adversarial reward is the negative of the base reward $R_{\text{adv}}(s) = -R(s)$ so the adversary minimizes the system's reward.

Describing a safety validation problem as an MDP does not limit us to reinforcement learning solution techniques. Approaches that search directly over disturbance trajectories (such as optimization and importance sampling approaches) can be used by defining a *playback policy* that generates a disturbance trajectory $\mathbf{x}$ regardless of the state trajectory. If the episode length of the MDP is not fixed then it is important to also define a fallback policy that is used in case the timestep exceeds the length of the disturbance trajectory and the MDP has not yet terminated. The disturbance distribution $p(x \mid s)$ is often a good choice for the fallback policy. Path planning approaches, where trajectories are built from smaller segments, can be used as long as the MDP can be initialized into an arbitrary state. If not, but the segments are branches of a tree (as in RRT or MCTS) and the MDP is deterministic given the disturbance, then each node in the tree can be reached by replaying the same set of disturbances from the initial state. For techniques that need a cost function, the negative discounted sum of rewards can be used

$$c(\mathbf{x}) = -\sum_t \gamma^t R(s_t, x_t). \tag{3.1}$$

So far we have made the Markov assumption when formulating the safety validation problem, but many real-world problems do not have this property. For example, any good driving policy should retain some historical information about other drivers to help predict their future behavior and make more informed decisions. We can relax the Markov assumption by assuming that the system policy, the transition function, the reward functions, and the disturbance model are also a function of hidden variables that depend upon the state history. Therefore, non-Markov systems cannot be fully described by the state and consequently cannot be initialized into arbitrary states. Many safety validation algorithms still work when the Markov assumption is relaxed such as those that search over entire disturbance trajectories (e.g. optimization and importance sampling). Tree-based path planning and reinforcement learning algorithms can also work for non-Markov systems as long as they are deterministic. In the cases where a safety validation does rely on the Markov assumption we will make that clear and suggest alternative approaches when the assumption is violated.

The following sections describe several MDPs and the adversarial versions that will be used for experimentation in later chapters. These MDPs serve as good examples for demonstrating the process of describing an adversarial MDP and will be used to test various safety validation algorithms in later chapters. Each adversarial MDP has an associated Julia package for those interested in testing their own safety validation algorithms.

3.2 *Gridworld Problems*

As is done in many treatments of algorithms for sequential decision making, we start with the gridworld. The gridworld is a simple MDP with a discrete state and action space that can capture the fundamental principles of sequential decision making. As such, we use it as a testbed for safety validation.

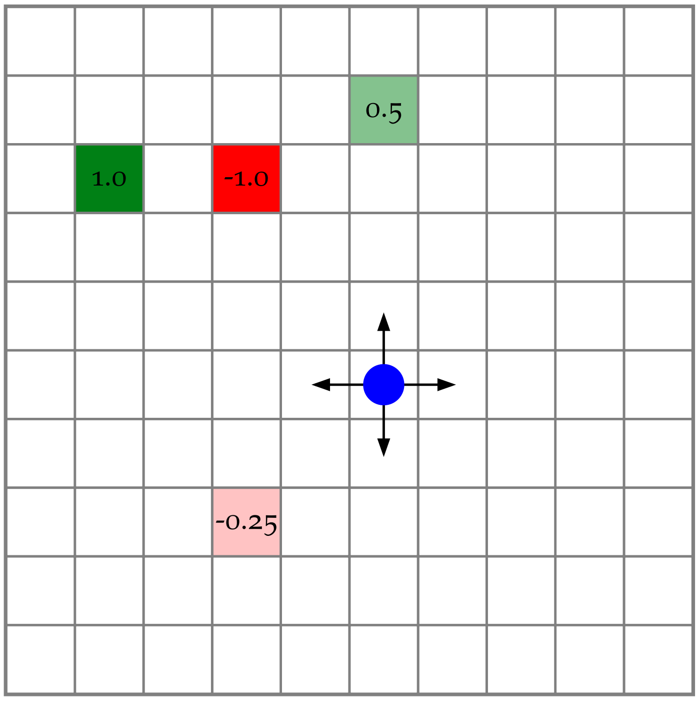


Figure 3.1: Simple Gridworld MDP.

3.2.1 *Simple Gridworld*

As pictured in Figure 3.1, the gridworld is defined on an $N_x \times N_y$ grid of states, each with a different reward. The state of the environment is the grid location of an agent and the possible actions of the agent are $[up, down, left, right]$. For each episode, the agent is initialized to a random non-terminal state and on each step, can transition to any adjacent state that is not out of bounds of the gridworld. The transition function depends upon a parameter $p_{\text{success}} \in [0, 1]$ such that the agent goes in the direction of its action with probability p_{success} or another random direction is selected with probability $1 - p_{\text{success}}$. If the transition direction results in an invalid state then the agent remains in its current state. Once the agent arrives in a state with nonzero rewards it will transition (regardless of its action) to a terminal state, ending the episode. The optimal agent policy $\pi(s)$ can be computed using dynamic programming [80] to any desired level of accuracy.

The adversarial version of the simple gridworld is described by the failure condition, disturbance space, and disturbance model. We define a failure as any time the agent reaches a terminal state with negative reward. The disturbances are the stochastic transitions of the agent so the disturbance space is the same as the action space of the base MDP: [*up*, *down*, *left*, *right*]. The probability of a disturbance is in state s is

$$p(x \mid s) = \begin{cases} p_{\text{success}} & \text{if } x = \pi(s) \\ (1 - p_{\text{success}})/3 & \text{if } x \neq \pi(s) \end{cases} \tag{3.2}$$

The reward function and discount factor are task and solver dependent and will be discussed in the description of the safety validation experiments.

The implementation of both the base and adversarial MDP can be done using the `SimpleGridworld` MDP distributed as part of the POMDPModelTools.jl[1] package. The base MDP can be implemented by defining the grid size, the value and location of the rewards, the states to terminate from, and the probability of a successful transition. The adversarial MDP can then be constructed with the same size and terminal states, but the rewards are now set to 1 for each failure state in the base MDP, and 0 for all others. The probability of transition success is set to 1 so that the actions of the adversarial MDP completely control the agent.

3.2.2 *Gridworld with Adversary*

In order to increase the difficulty of the safety validation problem we introduce a new gridworld variation that includes an adversarial agent (shown in Figure 3.2). Two agents move on an $N_x \times N_y$ gridworld: the ego agent (in blue) tries to reach states with high reward, while the adversary (in orange) tries to collide with the ego agent. The state of the environment is the grid location of each agent and the action space is [*up*, *down*, *left*, *right*, *stay*, *up-right*, *up-left*, *down-right*, *down-left*], which control the transitions of the ego agent. For each episode the agents are initialized to a random non-terminal position. On each iteration both agents select an action and then transition to the corresponding cell with probability p_{success}, or to another

[1]https://github.com/JuliaPOMDP/POMDPModelTools.jl

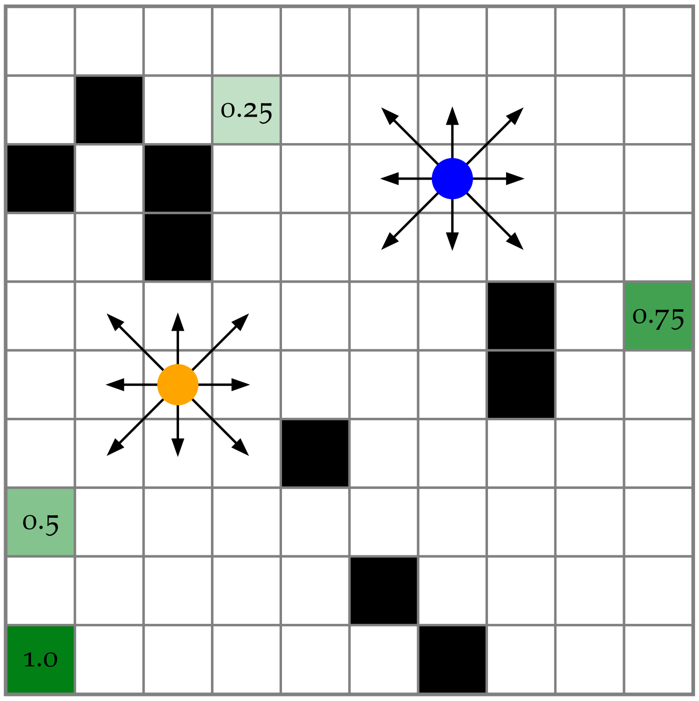


Figure 3.2: Gridworld with Adversray MDP.

adjacent cell with probability $1 - p_{\text{success}}$. Some states are marked as impassible (shown in black), so if the agent would transition to an impassible state or out of bounds of the gridworld then it stays in its current state. The episode terminates when the ego agent reaches a terminal state (any state with positive reward) or the two agents overlap in the same state. The state space of this MDP is the square of the simple gridworld, but for modestly sized grids, dynamic programming is tractable for solving for the optimal ego policy.

For the adversarial MDP, we define a failure of the ego agent as a collision between the ego agent and the adversary. The disturbances are the actions of the adversarial agent and so the disturbance space is the same as the action space in the base MDP. The probability of each disturbance can be user specified. If we assume that the adversary moves entirely at random, then the probability of each disturbance would be $p(x \mid s) = 1/9$.

Both the base MDP and the adversarial MDP can be implemented using the julia package AdversarialGridworld.jl.[2] A gridworld with an adversary is defined by specifying the grid size, the location of terminal states, the rewards, and the location of impassible states and the probability of successful transitions p_{success}. The MDP can be switched between base mode and adversarial mode. In base mode, the actions of the MDP correspond to the actions of the ego agent and the reward function returns positive reward when the ego agent reaches a reward state and negative reward for a collision. In adversarial mode, the actions of the MDP correspond to the adversary and the reward function gives a positive reward for collisions. In both modes, the agent that is not controlled by the actions of the MDP must have a policy associated with it.

3.3 *Autonomous Driving*

To complement the simplicity of the gridworld MDPs, we introduce two MDPs to model autonomous driving behavior in complex environments. Autonomous driving is an especially important domain for safety validation due to the recent development and deployment of autonomous vehicles [20]. Safety validation of an autonomous driving policy is challenging due to the size of the state and disturbance spaces, the duration of the diving episodes, and the rarity of failure events.

To model driving scenarios in simulation we build off of the julia package AutomotiveSimulator.jl[3] which provides methods for constructing road geometries, vehicles, pedestrians and behavior models. To facilitate the construction of adversarial MDPs for driving scenarios, we developed the package AdversarialDriving.jl.[4] Adversarial driving scenarios are constructed from a set of agents where each agent is defined by

- An *initialization function* that generates the initial state of the agent. The function may be deterministic or randomized, and is called each time the MDP is reset.

[2]https://github.com/sisl/AdversarialGridworld.jl
[3]https://github.com/sisl/AutomotiveSimulator.jl
[4]https://github.com/sisl/AdversarialDriving.jl

- A *behavior model* that determines the actions of the agent. For the system under test, the behavior model implements the autonomous driving policy we wish to test. For adversaries, the model implements their behavior and disturbances.
- A *disturbance model* that is a probability distribution over disturbances which can be used to sample disturbances and compute their probability.

The MDP is constructed with an agent representing the system (called the ego vehicle), a vector of agents that act as the adversaries (either other vehicles or pedestrians), a road geometry, and the simulation timestep. The state and action spaces are dynamically constructed based on the number and type of agents in the scene. We assume that vehicles stay in their lane during the simulation and can therefore specify a vehicle's state using four state variables (r, v, ℓ, b) where r is the position along the lane, v is the speed of the vehicle in the direction of the lane, ℓ is an integer representing which lane the vehicle is in, and b is a Boolean that indicates if the vehicle's turn signal is on. Pedestrians can move in any direction so their state is described by two position and two velocity variables (r_x, r_y, v_x, v_y). At each step, vehicles can apply an acceleration in the direction of the direction of the lane and pedestrians can apply a 2D acceleration. The position and velocity of each agent (in each direction) is updated by the kinematic relation

$$r_{t+\Delta t} = r_t + v_t \Delta t + \frac{1}{2} a_t \Delta t^2 \tag{3.3}$$

$$v_{t+\Delta t} = v_t + a_t \Delta t \tag{3.4}$$

where Δt is the simulation timestep. Additionally, vehicles that have the opportunity to make a turn can toggle their intention to turn and whether or not their turn signal is on. The policies we use to control agents in the scene are Markov, so the simulation can be initialized into an arbitrary state.

The driving policy that we use for vehicles is based on the intelligent driver model (IDM), a rule-based algorithm that applies longitudinal acceleration to reach a desired velocity while avoiding rear-end collisions. The IDM algorithm is shown

Table 3.1: Default IDM parameter values.

Description	**Variable**	**Default Value**
Speed tracking constant	k	$1.0\,\mathrm{s}^{-1}$
Acceleration exponent	δ	4.0
Desired time headway	T_{des}	$1.5\,\mathrm{s}$
Minimum allowed gap	r_{min}	$5.0\,\mathrm{m}$
Desired velocity	v_{des}	$15.0\,\mathrm{m/s}$
Maximum acceleration	a_{max}	$3.0\,\mathrm{m/s^2}$
Comfortable deceleration	d_{comf}	$2.0\,\mathrm{m/s^2}$
Maximum deceleration	d_{max}	$9.0\,\mathrm{m/s^2}$

in algorithm 3.1. It takes as input the current separation between the ego vehicle and the agent ahead of it Δs, the velocity of the ego v_{ego}, and the velocity of the lead vehicle v_{lead}. If there is no lead vehicle then $\Delta s = \infty$ and $v_{\text{lead}} = \text{NaN}$. In the presence of a leading vehicle (line 3), the IDM computes a desired following distance (line 4) and then computes an acceleration to match the desired velocity and following distance (line 5). If there is no leading vehicle (line 6) then the acceleration is computed to gradually approach the desired driving velocity (line 7). The behavior of the IDM is governed by a set of driving parameters which are shown in Table 3.1 with their default values.

Algorithm 3.1 Intelligent Driver Model.

```
1: function IDMAcceleration($\Delta r$, $v_{\text{ego}}$, $v_{\text{lead}}$)
2:     $\Delta v \leftarrow v_{\text{lead}} - v_{\text{ego}}$
3:     if $v_{\text{lead}} \neq \text{NaN}$
4:         $r_{\text{des}} = r_{\text{min}} + v_{\text{ego}}\Delta T_{\text{des}} - \frac{v_{\text{ego}}\Delta v}{2\sqrt{a_{\text{max}} d_{\text{comf}}}}$
5:         $a \leftarrow a_{\text{max}}\left(1 - \left(\frac{v_{\text{ego}}}{v_{\text{des}}}\right)^{\delta} - \left(\frac{r_{\text{des}}}{\Delta r}\right)^{2}\right)$
6:     else
7:         $a \leftarrow k\Delta v$
8:     return clamp($a$, $-d_{\text{max}}$, $a_{\text{max}}$)
```

The intelligent driving model works well for traffic on a single roadway, but is not designed to handle intersections. To allow the autonomous vehicle to navigate intersections we developed a rule-based intersection navigation algorithm that uses the IDM (shown in algorithm 3.2). The algorithm takes as input the ego vehicle *veh* that is being controlled, the *scene* that contains the other agents on the road, and the *roadway* which describes the road geometry. We first determine the velocity and relative position of any agent in the same lane as the ego vehicle (line 3). If there is no leading vehicle, then the velocity will be $v_{\text{lead}} = \text{NaN}$ and the relative position will be infinite. We then check if the ego vehicle has right of way in the intersection (line 4), and if it does, the vehicle proceeds with the IDM acceleration (line 5). The right of way rules are 1) vehicles going straight have priority over vehicles that are turning and 2) vehicles turning right have priority over vehicles turning left. If a vehicle has a turn signal on, them it is assumed to be turning and vice versa.

If the vehicle does not have right of way then the distance to the intersection (line 7 and the time it will take to cross it (line 8) are computed and used to determine if the intersection is occupied. The intersection is considered occupied if for any other agent on the road, that agent is already in the intersection or, based on its current velocity, will be in the intersection during the planned crossing time of the ego vehicle. If the intersection is occupied and the lead vehicle has already passed the intersection then we use the stationary boundary of the intersection as the reference point for the IDM (line 10), and otherwise we use the leading vehicle still (line 12).

The intersection navigation algorithm relies on several assumptions to make it functional. For sensing, it assumes that the ego vehicle can noisily observe all other agents in the scene (no occlusions), and has perfect knowledge of the road geometry (such as the location of any intersections). For the behavioral modeling of other agents, it assumes that an agent will turn on its turn signal if and only if it is turning, and that other agents drive with a constant velocity when approaching the intersection. Any violation of these assumptions could lead to a collision. The goal of this driving policy, however, is not to build an industrial autonomous driving planner, but to construct a driving policy that exhibits complex driving behavior while avoiding most collisions. Weaknesses in the algorithm provide potential failure modes that we can investigate with the safety validation algorithms developed in

Algorithm 3.2 Intersection navigation algorithm.

```
1: function IntersectionIDMAcceleration(veh, scene, roadway)
2:     $v \leftarrow$ Velocity(veh)
3:     $v_{\text{lead}}, \Delta s_{\text{lead}} \leftarrow$ LeadingVehicle(veh, scene)
4:     if HasRightOfWay(veh, scene, roadway)
5:         return IDMAcceleration($\Delta s_{\text{lead}}, v, v_{\text{lead}}$)
6:     else
7:         $\Delta s_{\text{int}} \leftarrow$ DistanceToIntersection(veh, roadway)
8:         $\Delta t_{\text{cross}} \leftarrow$ TimeToCrossIntersection(veh, roadway)
9:         if IntersectionOccupied(scene, roadway, $\Delta t_{\text{cross}}$) and $\Delta s_{\text{int}} < \Delta s_{\text{lead}}$
10:            return IDMAcceleration($\Delta s_{\text{int}}, v, 0$)
11:        else
12:            return IDMAcceleration($\Delta s_{\text{lead}}, v, v_{\text{lead}}$)
```

later chapters.

The simulation environment is lower fidelity than simulators used in industry. For example, the motion of the agents is controlled by point-mass dynamics instead of the complex 3D dynamics of a real vehicle or pedestrian. The environment is not rendered photo-realistically, so advanced perception systems cannot be used. The driving policy does not have machine-learned components and is assumed to be Markov. Despite these differences we have designed the autonomous driving scenarios to have similar state spaces, actions spaces, number of interacting agents and rarity of failure events to those of an industrial simulator and driving policy. Due to the this matching, we believe that safety validation algorithms that work for the following scenarios will be applicable to real-world autonomous driving simulators.

3.3.1 *Pedestrian in Crosswalk*

In the pedestrian crosswalk MDP shown in Figure 3.3, an autonomous vehicle approaches a crosswalk where a pedestrian is attempting to cross. The vehicle makes noisy observations of the pedestrian's position and velocity, and uses those observations with the intersection navigation algorithm (algorithm 3.2) to determine its acceleration. Since there is only a single lane for the vehicle, the state space of the

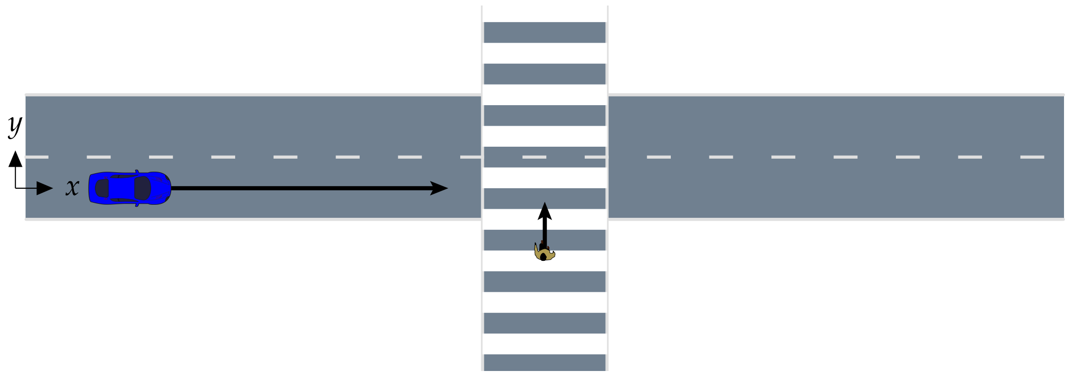


Figure 3.3: Crosswalk MDP where the ego vehicle approaches a crosswalk while a pedestrian tries to cross.

vehicle can be reduced to the longitudinal position and velocity (r_x^{veh}, v_x^{veh}) while the state of the pedestrian is the lateral and longitudinal position and velocity (r_x^{ped}, r_y^{ped}, v_x^{ped}, v_y^{ped}). The action space of the vehicle is the range of accelerations between $[-d_{\text{max}}, a_{\text{max}}]$. At the start of each episode, both agents are initialized randomly with the vehicle driving in the positive x-direction and the pedestrian is on the right side of the vehicle. At each step, both agents determine their accelerations and their position and velocity are updated according to the kinematic relations in Equations (3.3) and (3.4). The episode ends if there is a collision between the vehicle and the pedestrian or if the vehicle makes it to the end of the roadway. Since the driving policy is rule-based and does not require any learning, we have no need to explicitly specify a reward for the base MDP.

Failures of this scenario are defined by any collision between the pedestrian and vehicle. The disturbances are the lateral and longitudinal accelerations of the pedestrian (δa_x δa_y), and the lateral and longitudinal noise in the measurement of the pedestrian's position and velocity (δn_x, δn_y, δn_v). All five disturbances are continuous but if we are using a safety validation solver that requires a discrete disturbance space, then we can discretize them. If we assume that the disturbances are independent then we can model each with a zero-mean Gaussian distribution. If we want to model correlated disturbances to better caption human-like behavior, we can model them as Gaussian processes with kernel function $\mathbf{k}$ and mean function $\boldsymbol{\mu}$.

Table 3.2: Continuous disturbance space for adversarial pedestrian in crosswalk.

Disturbance	**Independent**	**Correlated**
Lateral Acceleration	$\delta a_x \sim \mathcal{N}(0, \sigma^2_{a_x})$	$\boldsymbol{\delta}\mathbf{a}_x \sim \mathcal{N}(\boldsymbol{\mu}_{a_x}(\mathbf{t}), \mathbf{k}_{a_x}(\mathbf{t}, \mathbf{t}))$
Longitudinal Acceleration	$\delta a_y \sim \mathcal{N}(0, \sigma^2_{a_y})$	$\boldsymbol{\delta}\mathbf{a}_y \sim \mathcal{N}(\boldsymbol{\mu}_{a_y}(\mathbf{t}), \mathbf{k}_{a_y}(\mathbf{t}, \mathbf{t}))$
Lateral Position Noise	$\delta n_x \sim \mathcal{N}(0, \sigma^2_{r_x})$	$\boldsymbol{\delta}\mathbf{n}_x \sim \mathcal{N}(\boldsymbol{\mu}_{r_x}(\mathbf{t}), \mathbf{k}_{r_x}(\mathbf{t}, \mathbf{t}))$
Longitudinal Position Noise	$\delta n_y \sim \mathcal{N}(0, \sigma^2_{r_y})$	$\boldsymbol{\delta}\mathbf{n}_y \sim \mathcal{N}(\boldsymbol{\mu}_{r_y}(\mathbf{t}), \mathbf{k}_{r_y}(\mathbf{t}, \mathbf{t}))$
Velocity Noise	$\delta n_v \sim \mathcal{N}(0, \sigma^2_{v})$	$\boldsymbol{\delta}\mathbf{n}_v \sim \mathcal{N}(\boldsymbol{\mu}_{v}(\mathbf{t}), \mathbf{k}_{v}(\mathbf{t}, \mathbf{t}))$

These options are shown in Table 3.2.

3.3.2 *T-Intersection*

In the T-intersection MDP (shown in Figure 3.4), the ego vehicle tries to make an unprotected left turn onto a two-lane through-street with other vehicles at the intersection. The state of the ith vehicle is (r_i, v_i, ℓ_i, b_i) where r_i and v_i are the position and velocity along lane ℓ_i, and b_i is a Boolean indicating if the vehicle's turn signal is on. The intersection is composed of 6 lanes: $\ell = 1$ is the lane that goes straight from left to right, $\ell = 2$ starts as lane 1 but turns right, $\ell = 3$ is the lane that goes straight from right to left, $\ell = 4$ starts as lane 3 but turns left, $\ell = 5$ turns left onto the through street, and $\ell = 6$ turns right onto the through street. Vehicles that are on the through-street can change their turn intention by changing their lane prior to the intersection. For example, a vehicle going straight in lane 1 can choose to turn right by switching to lane 2 before the turn. Each lane has at most one other lane that a vehicle could switch to without changing its position, so changing lanes can be considered an operation that toggles *turn intention*.

At the start of each episode, the agents are randomly initialized into a configuration that would lead to no collisions in the absence of disturbances. On each step, each vehicle on the road makes observations of the other agents and uses algorithm 3.2 to determine their acceleration to safely travel through the intersection. The episode terminates when there is a collision between any two vehicles or when the ego vehicle successfully makes the left turn and reaches the end of the roadway.

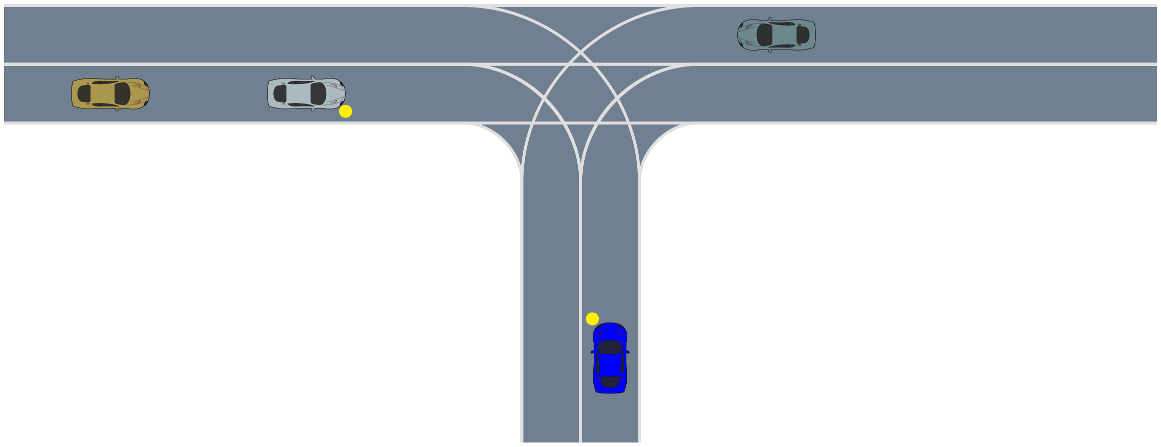


Figure 3.4: T-Intersection MDP where the ego vehicle tries to take unprotected left turn.

Table 3.3: Continuous disturbance space for adversarial vehicles in T-intersection.

Disturbance	**Distribution**
Acceleration	$\delta a \sim \mathcal{N}(0, \sigma_a^2)$
Toggle turn intent	$\delta \ell \sim \text{Bern}(\rho_I)$
Toggle turn signal	$\delta b \sim \text{Bern}(\rho_S)$

In the adversarial MDP, a failure is any collision between the ego vehicle and another vehicle on the road. For each vehicle (other than the ego), the disturbances are perturbations to the vehicle's acceleration δa, toggling turn intention $\delta \ell$, or toggling the turn signal δb. If we treat the acceleration disturbance as continuous, then the disturbances can be modeled by the distributions shown in Table 3.3. If we require discrete disturbances then we can choose 7 representative disturbances and their associate probabilities as shown in Table 3.4.

Table 3.4: Discrete disturbance space for adversarial vehicles in T-intersection.

Disturbance	**Acceleration**	**MC Probability**
No disturbance	$0\,\mathrm{m/s^2}$	0.976
Medium slowdown	$-1.5\,\mathrm{m/s^2}$	1×10^{-2}
Major slowdown	$-3\,\mathrm{m/s^2}$	1×10^{-3}
Medium speedup	$1.5\,\mathrm{m/s^2}$	1×10^{-2}
Major speedup	$3\,\mathrm{m/s^2}$	1×10^{-3}
Toggle blinker	N/A	1×10^{-3}
Toggle turn intent	N/A	1×10^{-3}

3.4 *Discussion*

In this chapter we introduced the idea of an adversarial MDP to model the safety validation of an autonomous policy for a base MDP. Both MDPs share the same state space and transition model, but the actions of the base MDP are the actions of the system while the actions of the adversarial MDP are the disturbances. The adversarial MDP is a flexible model that can be used in conjunction with safety validation algorithms based on optimization, path-planning, reinforcement learning, and importance sampling.

We then introduced four safety validation problems that are used to test the safety validation algorithms developed in the remainder of this thesis. The first scenario involves the safety validation of a simple gridworld policy where the disturbances are the stochastic transitions of the agent. The second scenario is also a gridworld problem, but with another adversarial agent that causes collisions. The next two systems model autonomous driving scenarios. In the first, the ego vehicle approaches a crosswalk with a crossing pedestrian and must avoid the pedestrian under disturbances to the pedestrian motion and sensor noise. The second driving scenario is a T-intersection where the ego vehicle makes an unprotected left turn onto a through-street with other drivers on the road. The disturbances are the behavior of other vehicles on the road and include perturbations to the acceleration, toggling of turn intention and toggling the turn signal.

The following chapter begins the development of safety validation algorithms. We introduce a technique for interpretable safety validation and demonstrate it with the autonomous driving scenarios.

4 Interpretable Safety Validation

In this chapter, we develop an algorithm for interpretable safety validation that generates failure descriptions in the form of signal temporal logic expressions. A failure description is any statement that defines a set of disturbance trajectories that cause the system to fail and is useful for understanding failure modes, producing failure examples. Failure descriptions are found using expression optimization and evaluated by sampling satisfying trajectories. We introduce a technique for sampling satisfying trajectories of STL specifications by applying a set of linear constraints to the disturbance model which can be done efficiently for independent disturbances and disturbances that are jointly distributed as a Gaussian process. Through experiments in two autonomous driving scenarios, we demonstrate that our interpretable safety validation algorithm can find failure descriptions, and produce a diverse set of failure examples. Failure descriptions may also be used to analyze the sensitivity of the safety of the system with respect to important disturbance parameters.

4.1 Motivation

One of the drawbacks to black-box safety validation techniques approaches is that they do not produce high-level descriptions of failures and instead only produce high-dimensional failure examples. In scenario-based testing, by contrast, each scenario involves semantically similar disturbance trajectories which can provide clues to the source of an error. For example, suppose a black-box safety validation algorithm discovered a failure of an autonomous vehicle driving on the highway.

It would not be clear from this one example if the failure was due to perception, planning, control or something else. If, however, we found that the vehicle failed in a scenario designed to test its response to partially occluded stop signs, we would be justified in investigating the perception system. This work tries to bring a greater degree of interpretability to black box safety validation algorithms.

Our approach seeks to find a *failure description* φ such that any disturbance trajectory that satisfies the description will cause the system to fail. For falsification, we want to find any such φ where

$$\mathbf{x} \in \varphi \implies f(\mathbf{x}) \notin \psi. \tag{4.1}$$

For most likely failure analysis we want to find the φ that produces failures with high likelihood

$$\max_{\varphi} \quad \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}\mid\mathbf{x}\in\varphi)}[p(\mathbf{x})] \quad \text{s.t.} \quad \mathbf{x} \in E \tag{4.2}$$

where $p(\mathbf{x} \mid \mathbf{x} \in \varphi)$ is the probability density of disturbance trajectories that satisfy the description.

A failure description φ contains more information than a single failure example. For example, a single failure description can produce many failure examples by sampling from $p(\mathbf{x} \mid \mathbf{x} \in \varphi)$. It also identifies logical features of a disturbance trajectory that are sufficient to cause a failure. As in scenario-based testing, those features may provide clues to the source of the error in the system. Lastly, if the features are low dimensional, they can be used to perform a sensitivity analysis of the safety of the system with respect to the disturbance variables.

4.2 STL Expression Optimization

This section provides the necessary technical background to understand our approach for STL expression optimization. We first introduce context free grammars as a way to compactly represent signal temporal logic and describe how to sample valid STL expressions. We then describe genetic programming, a technique used

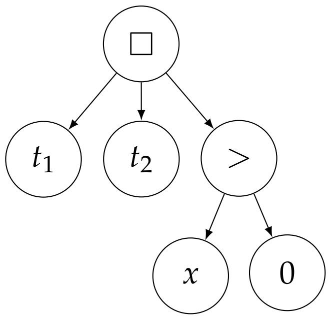


Figure 4.1: Expression tree for $\square_{[t_1,t_2]} x > 0$.

to optimize tree-like structures such as STL expressions. Lastly, we introduce the failure description cost function that is used to find optimal failure descriptions.

4.2.1 *Context-Free Grammars*

Logical expressions can be represented by a tree of symbols and operators. For example, the STL expression

$$\square_{[t_1,t_2]} x > 0 \tag{4.3}$$

can be represented by the tree shown in Figure 4.1. The semantics of a formal language specify a set of constraints on space of possible expression trees. These constraints can be compactly represented using a context-free grammar, which is a collection of production rules of the form

$$\mathbb{T} \mapsto \alpha, \tag{4.4}$$

where $\mathbb{T}$ is a type and α is an expression that consists of types and symbols. A terminal production rule is one where α consists only of symbols.

For example, the context-free grammar for a simplified version of STL for a single disturbance variable x is given by

$$\begin{aligned}
\mathbb{B} &\mapsto \mathbb{B} \wedge \mathbb{B} \mid \mathbb{B} \vee \mathbb{B} \mid \neg\mathbb{B} \mid \Box_{[\mathbb{T},\mathbb{T}]}(\mathbb{S}) \mid \Diamond_{[\mathbb{T},\mathbb{T}]}(\mathbb{S}) \\
\mathbb{T} &\mapsto 0 : t_{\max} \\
\mathbb{S} &\mapsto \mathbb{S} \wedge \mathbb{S} \mid \mathbb{S} \vee \mathbb{S} \mid \neg\mathbb{S} \mid x \leq \mathbb{X} \mid x \geq \mathbb{X} \mid x = \mathbb{X} \\
\mathbb{X} &\mapsto [x_{\min}, x_{\max}]
\end{aligned} \tag{4.5}$$

The type $\mathbb{B}$ is used for Boolean scalars, $\mathbb{S}$ is for Boolean series, $\mathbb{T}$ is for time, and $\mathbb{X}$ is for values of x. The logical operators apply element-wise on series. The symbol $|$ separates rules on the same line and : indicates a range of symbols. In this work, time is discrete and the variable x may be discrete or continuous. When x is continuous, $\mathbb{X}$ is sampled uniformly at random in the range $[x_{\min}, x_{\max}]$.

To generate an expression from a context-free grammar, we start by declaring the type of the desired expression and choose a production rule for that type. For example, if we want a valid STL, expression we start with the type $\mathbb{B}$ and select a rule from the first line of Equation (4.5). Then, for each type in the expression, we expand it using a compatible production rule. The process recurses until all types have been expanded by terminal production rules and the expression contains only symbols. To construct Equation (4.3) we applied the production rules

$$\mathbb{B} \mapsto \Box_{[\mathbb{T},\mathbb{T}]}(\mathbb{S}) \tag{4.6}$$
$$\mathbb{T} \mapsto t_1 \tag{4.7}$$
$$\mathbb{T} \mapsto t_2 \tag{4.8}$$
$$\mathbb{S} \mapsto x \leq \mathbb{X} \tag{4.9}$$
$$\mathbb{X} \mapsto 0. \tag{4.10}$$

The selection of production rules can be done according to any distribution but it is common to choose them uniformly at random which favors shorter expressions [47]. To implement grammars and expression sampling we use the package ExprRules.jl.[1]

[1]https://github.com/sisl/ExprRules.jl

4.2.2 *Genetic Programming*

Genetic programming [47], [115] is a population-based optimization technique for expression trees that mimics the biological process of evolution. The algorithm (algorithm 4.1) takes as input a grammar $\mathcal{G}$, an expression cost function c_φ, and a population size M. To start, a population of M expressions (or *individuals*) is sampled from the grammar $\mathcal{G}$ (line 2) using a uniform distribution over production rules. On each iteration, the cost of each expression is computed (line 4) and used to select parents for reproduction (line 5). The parents reproduce through a crossover operation to create a new population of M children (line 5). Then, some individuals are mutated to add new subtrees into the population (line 7). Once the computational budget is exhausted, the individual with the lowest cost is returned (line 8). We use the implementation of genetic programming from the ExprOptimiation.jl[2] package.

Algorithm 4.1 Genetic programming.

```
1: function GeneticProgramming($\mathcal{G}$, $c_\varphi$, $M$)
2:     $\boldsymbol{\varphi} \leftarrow [\varphi_1, \dots, \varphi_M]$ with $\varphi_i \sim \mathcal{G}$
3:     loop
4:         $\mathbf{c}_\varphi \leftarrow [c_\varphi(\varphi_1), \dots, c_\varphi(\varphi_M)]$
5:         $(\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2) \leftarrow$ Select$(\boldsymbol{\varphi}, \mathbf{c})$
6:         $\boldsymbol{\varphi} \leftarrow$ Crossover$(\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2))$
7:         $\boldsymbol{\varphi} \leftarrow$ Mutate$(\boldsymbol{\varphi})$
8:     return $\min_\varphi[c_\varphi(\varphi_1), \dots, c_\varphi(\varphi_M)]$
```

Selection. The selection procedure is designed to favor individuals with high *fitness*. The fitness of an individual φ_i is inversely related to its cost and can be computed as $\max \mathbf{c}_\varphi - c_\varphi(\varphi_i)$ where $\max \mathbf{c}_\varphi$ returns the largest cost in the population and $c_\varphi(\varphi_i)$ is the cost of individual i. Selection can be done using:

- *Truncation selection*: Parents are sampled uniformly at random from the M fittest individuals.

[2]https://github.com/sisl/ExprOptimization.jl

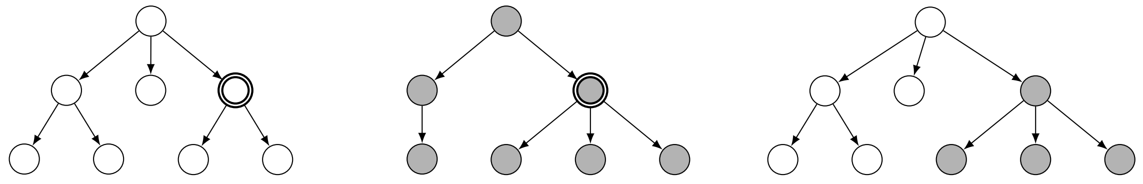

(a) Parent 1 (b) Parent 2 (c) Child

Figure 4.2: Crossover operation that creates a child from two parents by replacing the highlighted subtree of parent one with the highlighted subtree of parent 2.

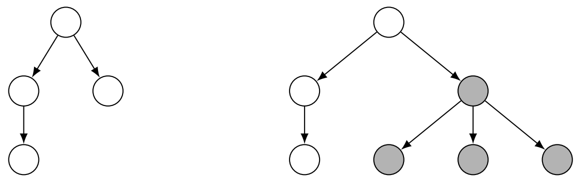

(a) Before Mutation. (b) After Mutation.

Figure 4.3: Mutation operation replaces a subtree with a randomly generated tree.

- *Tournament selection*: Each parent is the fittest individual of a randomly sampled subpopulation.
- *Fitness-proportionate selection*: Parents are chosen with probability proportional to their fitness.

Crossover. Crossover is way of constructing a new tree that has attributes of two parent trees. The procedure is shown in Figure 4.2. A random node is selected in the first parent and replaced by a random subtree of the second parent. Care must be taken to ensure type-compatibility when making the exchange.

Mutation. Trees experience random mutations in order to introduce new features into the population. Mutation is demonstrated in Figure 4.3. A random node in the tree is selected and replaced with a new expression of the same type sampled from the grammar.

4.2.3 *Failure Description Cost Functions*

A cost function for finding STL expressions that describe failure trajectories is shown in algorithm 4.2 which takes as input an STL expression φ and returns the average safety metric of N disturbance trajectories that satisfy φ. We first check if the description is satisfiable (line 2) in case we sample a logically inconsistent expression such as $\square(x \geq 1 \wedge x \leq -1)$. In practice, we assume that an expression is not satisfiable if the sampling procedure fails to produce valid samples, in which case we return the maximal cost value $c_{\max}$ (line 7). Otherwise, we obtain N disturbance trajectories that satisfy φ (line 4) and use them to estimate the expected value of the safety metric from the average. Lastly, we add a penalty term (with parameter λ) for the number of nodes in the tree to ensure that expressions remain parsimonious (line 5).

If the specification φ is a failure description then trajectories that satisfy it should have a low value of the safety metric and lead to low cost, while the opposite is true if φ has nothing to do with failures of the system. Therefore when c_φ is used for optimization it should produce descriptions that frequently produce failure trajectories. If the probability is incorporated into $c(\mathbf{x})$ (as in Equation (2.13)), then the optimization procedure will produce a failure description of the most likely failure modes. The most challenging part of computing the cost, however, is the need to sample trajectories that satisfy a given specification, which is the topic of the next section.

Algorithm 4.2 Failure description cost function.

```
1: function $c_\varphi(\varphi)$
2:     if IsSatisfiable($\varphi$)
3:         for $i$ in $1 : N$
4:             Sample $\mathbf{x}_i$ from $p(\mathbf{x})$ s.t. $\mathbf{x} \in \varphi$
5:         return $\frac{1}{N}\sum_{i=1}^{N} c(\mathbf{x}_i) + \lambda$NodeCount($\varphi$)
6:     else
7:         return $c_{\max}$
```

4.3 STL Satisfaction

A crucial piece of using expression optimization to find failure descriptions is the ability to sample a disturbance trajectory that satisfies a given specification φ. The naive approach is to sample from the distribution $p(\mathbf{x})$ until $\in \varphi$. This approach, known as *rejection sampling*, can sample exactly from the distribution $p(\mathbf{x} \mid \mathbf{x} \in \varphi)$ but can be inefficient if the specification φ represents a rare event. To make the sampling tractable, we split the problem into two parts: sampling a set of linear constraints that are sufficient for STL satisfaction, and sampling a time series that satisfies those constraints. Our approach no longer samples exactly from $p(\mathbf{x} \mid \mathbf{x} \in \varphi)$, but approximates it while producing trajectories that satisfy the desired specification. In this section, we first describe how we sample a set of sufficient linear constraints, and then describe how to sample constrained trajectories from several useful distributions. Lastly, we describe a limited grammar that allows for sampling state-dependent specifications.

4.3.1 Sufficient Linear Constraints

To build intuition of how linear constraints can enforce STL satisfaction, consider the STL expression

$$(\square_{[0,10]}\mathbf{x} \geq 0) \wedge (\lozenge_{[5,10]}\mathbf{x} \geq 2). \tag{4.11}$$

Any disturbance trajectory that satisfies the constraints $\mathbf{x} \geq 0 \ \forall t \in [0, 10]$ and $x_t \geq 2$ for some $t \in [5, 10]$ will also satisfy the specification. To make the constraints concrete, we must sample a time $t \in [5, 10]$, say $t = 7$, to get

$$\begin{aligned} x_t &\geq 0 \quad \forall t \in [0, 10] \\ x_7 &\geq 2. \end{aligned} \tag{4.12}$$

We systematize this process by defining constraints for each possible expression type and recursively applying them until a given expression is guaranteed to be satisfied.

The general process of sampling sufficient constraints is outlined in algorithm 4.3. The function takes as input an expression φ and an assignment b which is a scalar or a vector depending on the type of φ. The assignment consists of the Booleans True (T) and False (F), or a symbol Arbitrary (A), which means the output of φ is unspecified. If φ is a direct comparison such as $x \leq 0$ (line 2), then we call a function ParseBounds to obtain a set of lower bound $\boldsymbol{\ell}$ and an upper bound $\mathbf{u}$ constraints (line 3). If the expression consists of more than just a simple comparison, then the we initialize the lower and upper bounds to negative and positive infinity so there are no constraints (line 5). Then, decompose the expression into its child expressions and sample a set of assignments for each that would combine to give b (line 6). Each child expression φ_i and its corresponding assignment b_i are passed recursively to SampleConstraints to get a set of sufficient constraints (line 7). All the constraints of the children are combined (line 8) and then returned (line 9).

The function ParseBounds converts comparisons into lower and upper bound constraint vectors $(\boldsymbol{\ell}, \mathbf{u})$. Consider the comparison $\varphi = (x \leq \alpha)$ with assignment $b = [\mathrm{T}, \mathrm{F}, \mathrm{A}]$. In the case that φ is True, then we set the upper bound to α and the lower bound to $-\infty$. When φ is False we do the opposite and set the lower bound to α and the upper bound to ∞. When φ is Arbitrary, we impose no constraint and set the lower bound to $-\infty$ and the upper bound to ∞. The resulting bounds are

$$\boldsymbol{\ell} = [-\infty, \alpha, -\infty] \tag{4.13}$$

$$\mathbf{u} = [\alpha, \infty, \infty]. \tag{4.14}$$

We perform a similar procedure for the $\varphi = (x \geq \alpha)$ expression. For equality comparison $\varphi = (x = \alpha)$, if φ is True, then we set both the lower and upper bounds to α. If equality is False then we need a different mechanism to represent the constraint. In our implementation of interpretable validation,[3] we store any values of x that are not allowed at a given time and update the sampling distribution accordingly.

The function UpdateBounds combines sets of lower and upper bound constraints

[3]https://github.com/sisl/InterpretableValidation.jl

$(\boldsymbol{\ell}_1, \mathbf{u}_1)$ and $(\boldsymbol{\ell}_2, \mathbf{u}_2)$. The constraints are combined so the more restrictive constraint is always used:

$$\boldsymbol{\ell} = \max(\boldsymbol{\ell}_1, \boldsymbol{\ell}_2) \tag{4.15}$$

$$\mathbf{u} = \min(\mathbf{u}_1, \mathbf{u}_2). \tag{4.16}$$

If at any point $\boldsymbol{\ell} > \mathbf{u}$ then the sampling procedure has failed and must start over. Sampling failure is guaranteed to occur when φ is logically inconsistent, but may also occur for an unfortunate assignment of child expressions.

The function SampleChildrenAssignments can be summarized by Table 4.1. The input is an expression φ (composed of subexpressions φ_i) and an assignment b. If the assignment is Arbitrary then the assignment of all subexpressions is also Arbitrary. For Boolean assignments, the corresponding subexpression assignments are shown in the second column. In the case where there is more than one possible assignment of the children that would lead to the assignment of the parent, we sample uniformly at random from the possibilities.

Algorithm 4.3 Sampling constraints for STL satisfaction

```
1: function SampleConstraints($\varphi$, $b$)
2:     if $\varphi$ is a comparison
3:         $\boldsymbol{\ell}, \mathbf{u} \leftarrow$ ParseBounds($\varphi, b$)
4:     else
5:         $\boldsymbol{\ell}, \mathbf{u} \leftarrow [-\infty, \dots, -\infty], [\infty, \dots, \infty]$
6:         for ($\varphi_i$, $b_i$) in SampleChildrenAssignments($\varphi$, $b$)
7:             $\boldsymbol{\ell}_i, \mathbf{u}_i \leftarrow$ SampleConstraints($\varphi_i, b_i$)
8:             $\boldsymbol{\ell}, \mathbf{u} \leftarrow$ UpdateBounds($\boldsymbol{\ell}, \mathbf{u}, \boldsymbol{\ell}_i, \mathbf{u}_i$)
9:     return $\boldsymbol{\ell}, \mathbf{u}$
```

The result of algorithm 4.3 is a set of lower and upper bounds for the disturbance trajectory $\mathbf{x}$. In the section, we discuss how to sample constrained disturbance trajectories for a variety of distributions.

Table 4.1: Representation of the SampleChildrenAssignments function.

Input (φ, b)	**Output**
$(\varphi_1 \wedge \varphi_2, \mathrm{T})$	$(\varphi_1 = \mathrm{T}, \varphi_2 = \mathrm{T})$
$(\varphi_1 \wedge \varphi_2, \mathrm{F})$	$(\varphi_1 = \mathrm{F}, \varphi_2 = \mathrm{A})$ or $(\varphi_1 = \mathrm{A}, \varphi_2 = \mathrm{F})$
$(\varphi_1 \vee \varphi_2, \mathrm{T})$	$(\varphi_1 = \mathrm{T}, \varphi_2 = \mathrm{A})$ or $(\varphi_1 = \mathrm{A}, \varphi_2 = \mathrm{T})$
$(\varphi_1 \vee \varphi_2, \mathrm{F})$	$(\varphi_1 = \mathrm{F}, \varphi_2 = \mathrm{F})$
$(\neg\varphi_1, \mathrm{T})$	$\varphi_1 = \mathrm{F}$
$(\neg\varphi_1, \mathrm{F})$	$\varphi_1 = \mathrm{T}$
$(\Box_{[t_1,t_2]}\varphi_1, \mathrm{T})$	$\varphi_1 = [\ \mathrm{A},\ \ldots,\ \underbrace{\mathrm{T},\ \ldots,\ \mathrm{T}}_{[t_1,t_2]},\ \mathrm{A},\ \ldots\ ]$
$(\Box_{[t_1,t_2]}\varphi_1, \mathrm{F})$	$\varphi_1 = [\ \mathrm{A},\ \ldots,\ \underset{t\in[t_1,t_2]}{\underset{\uparrow}{\mathrm{F}}},\ \mathrm{A},\ \ldots\ ]$
$(\Diamond_{[t_1,t_2]}\varphi_1, \mathrm{T})$	$\varphi_1 = [\ \mathrm{A},\ ...,\ \underset{t\in[t_1,t_2]}{\underset{\uparrow}{\mathrm{T}}},\ \mathrm{A},\ \ldots\ ]$
$(\Diamond_{[t_1,t_2]}\varphi_1, \mathrm{F})$	$\varphi_1 = [\ \mathrm{A},\ \ldots,\ \underbrace{\mathrm{F},\ \ldots,\ \mathrm{F}}_{[t_1,t_2]},\ \mathrm{A},\ \ldots\ ]$

4.3.2 *Sampling Linearly Constrained Trajectories*

Given a set of lower bound and upper bound constraints $(\boldsymbol{\ell}, \mathbf{u})$ that encode the satisfaction of an STL expression, we wish to sample disturbance trajectories $\mathbf{x}$ from $p(\mathbf{x})$ that satisfy those constraints, i.e.

$$\begin{aligned} &\mathbf{x} \sim p(\mathbf{x}) \\ &\text{s.t.} \quad \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}. \end{aligned} \tag{4.17}$$

where that the comparisons are done for each timestep x_t. If the disturbance space $X \subset \mathbb{R}^n$, then the lower and upper bound at each timestep will have n components and the comparisons are again performed component-wise.

The naive approach is to use rejection sampling, where we take samples from p until we find one that satisfies the constraints. Since $\boldsymbol{\ell}$ and $\mathbf{u}$ are likely to encode rare events then rejection sampling can be inefficient. Instead, we focus on the subset of distributions that have efficient algorithms for sampling constrained disturbance trajectories. We first discuss trajectories where each time step is independent, in which case the application of constraints is usually straightforward. An independence assumption may be valid when the disturbances are sensor noise but may not be valid when modeling more complex disturbances such as the motion of adversarial agents in a driving scenario. For disturbance trajectories that are correlated, we propose using Gaussian process models and discuss efficient strategies for sampling from them with linearly-constrained.

Independent Samples If the disturbances are independent over time then the probability distribution decomposes to

$$p(\mathbf{x}) = \prod_{t=1}^{t_{\max}} p(x_t) \tag{4.18}$$

When the components of x_t (denoted $x_{t,i}$ for the ith component) are also independent, it decomposes further to

$$\prod_{t=1}^{t_{\max}} p(x_t) = \prod_{t=1}^{t_{\max}} \prod_{i=1}^{n} p(x_{t,i}) \tag{4.19}$$

If the probability model of the ith component is uniform, then $x_{t,i}$ is distributed as

$$x_{t,i} \sim \mathcal{U}(l_{t,i}, u_{t,i}) \tag{4.20}$$

where $\mathcal{U}(\ell, u)$ is the uniform distribution between ℓ and u. Similarly, if the probability model of the ith component is normal with mean $\mu_{t,i}$ and standard deviation $\sigma_{t,i}$, then $x_{t,i}$ is distributed as

$$x_{t,i} \sim \mathcal{N}(\ell_{t,i}, u_{t,i}, \mu_{t,i}, {\sigma_{t,i}}^2) \tag{4.21}$$

where $\mathcal{N}(\ell, u, \mu, \sigma^2)$ is the Gaussian distribution truncated between ℓ and u with mean μ and variance σ^2. The truncated normal distribution can be efficiently sampled from efficiently using the cumulative distribution function of a Gaussian.

Gaussian Process A Gaussian process [116] is a stochastic process where any finite set of sample points $\mathbf{t} = [1, \dots, t_{\max}]$ have values $\mathbf{x} = [x_1, \dots, x_{t_{\max}}]$ that are distributed according to a multivariate normal distribution of the form

$$\mathbf{x} \sim \mathcal{N}\big(\boldsymbol{\mu}(\mathbf{t}), \mathbf{K}(\mathbf{t}, \mathbf{t})\big) \tag{4.22}$$

where $\mu_t = \mu(t)$ for mean function μ and $K_{t,t'} = k(t, t')$ for kernel function k. A common choice for k is the squared exponential kernel with covariance σ^2 and characteristic length l given by

$$k(t, t') = \sigma^2 \exp\left(-\frac{(t - t')^2}{2l^2}\right) \tag{4.23}$$

Suppose some values $\mathbf{x}^o$ are observed at points $\mathbf{t}^o$, and we wish to sample new values $\mathbf{x}^*$ at $\mathbf{t}^*$. The conditional distribution of the new sample points given the observed points is multivariate normal

$$\mathbf{x}^* \mid \mathbf{x}^o \sim \mathcal{N}\big(\boldsymbol{\mu}(\mathbf{t}^*) + \tilde{\mathbf{K}}(\mathbf{t}^o, \mathbf{t}^*)(\mathbf{x}^o - \boldsymbol{\mu}(\mathbf{t}^o)), \boldsymbol{\Sigma}(\mathbf{t}^o, \mathbf{t}^*, \mathbf{t}^*)\big) \tag{4.24}$$

where

$$\boldsymbol{\Sigma}(\mathbf{t}^o, \mathbf{t}^*, \mathbf{t}') = \mathbf{K}(\mathbf{t}^*, \mathbf{t}') - \tilde{\mathbf{K}}(\mathbf{t}^o, \mathbf{t}^*)\mathbf{K}(\mathbf{t}^o, \mathbf{t}') \tag{4.25}$$

$$\tilde{\mathbf{K}}(\mathbf{t}^o, \mathbf{t}^*) = \mathbf{K}(\mathbf{t}^*, \mathbf{t}^o)\mathbf{K}(\mathbf{t}^o, \mathbf{t}^o)^{-1} \tag{4.26}$$

Suppose we know that another set of points at $\mathbf{t}^c$ have constrained values such that $\ell_t \leq x_t \leq u_t$ for all $t \in \mathbf{t}^c$. The conditional distribution on the sample points, given the observed points and the constraints is given by the compound distribution [117]

$$\begin{aligned}\mathbf{x}^* \mid \mathbf{x}^c, \mathbf{x}^o \sim \mathcal{N}\big(&\boldsymbol{\mu}(\mathbf{t}^*) + \mathbf{A}(\mathbf{x}^c - \boldsymbol{\mu}(\mathbf{t}^c)) + \mathbf{B}(\mathbf{x}^o - \boldsymbol{\mu}(\mathbf{t}^o)),\\ &\boldsymbol{\Sigma}(\mathbf{t}^o, \mathbf{t}^*, \mathbf{t}^c) - \mathbf{A}\boldsymbol{\Sigma}(\mathbf{t}^o, \mathbf{t}^c, \mathbf{t}^*)\big)\end{aligned} \tag{4.27}$$

$$\mathbf{x}^c \mid \mathbf{x}^o \sim \mathcal{N}(\boldsymbol{\ell}, \mathbf{u}, \big(\boldsymbol{\mu}(\mathbf{t}^c) + \tilde{\mathbf{K}}(\mathbf{t}^o, \mathbf{t}^c)(\mathbf{x}^o - \boldsymbol{\mu}(\mathbf{t}^o)), \boldsymbol{\Sigma}(\mathbf{t}^o, \mathbf{t}^c, \mathbf{t}^c)\big) \tag{4.28}$$

where

$$\mathbf{A} = \boldsymbol{\Sigma}(\mathbf{t}^o, \mathbf{t}^*, \mathbf{t}^c)\boldsymbol{\Sigma}(\mathbf{t}^o, \mathbf{t}^c, \mathbf{t}^c)^{-1} \tag{4.29}$$

$$\mathbf{B} = \tilde{\mathbf{K}}(\mathbf{t}^o, \mathbf{t}^*) - \mathbf{A}\tilde{\mathbf{K}}(\mathbf{t}^o, \mathbf{t}^c) \tag{4.30}$$

To sample from a linearly constrained Gaussian process, we first sample points $\mathbf{x}^c$ from the truncated multivariate normal distribution in Equation (4.28) and then sample $\mathbf{x}^*$ from Equation (4.27) [117]. Sampling from a truncated multivariate distribution can be done efficiently with the minimax tilting approach [118], where rejection sampling is applied to an efficient importance sampling distribution generated by transforming the mean and variance of a multivariate Gaussian.

4.3.3 *State-Dependent Specifications*

Up to this point we have assumed that the failure descriptions would only include the disturbance variables and time. As discussed in Section 2.4.3, however, including the environment state in the optimization procedure can be beneficial. Additionally, developing specifications that depend on the state of the environment may produce more interpretable failure descriptions. Unfortunately, it is challenging to sample disturbance trajectories that have an arbitrary dependence on the environment state (whose dynamics are a black box). We therefore consider a restricted set of STL specifications that allows for the sampling disturbance trajectories.

The restricted grammar is shown in Equation (4.31). We consider expressions that are conjunctions of statements of the form $\square(\mathbb{K} \implies \mathbb{V})$. The operator $\varphi_1 \implies \varphi_2$ means that any time φ_1 is TRUE then φ_2 is true as well. The type $\mathbb{K}$ contains expressions that are formed from logical relationships between comparisons of state variables, while $\mathbb{V}$ is the same for disturbance variables.

$$\begin{aligned}
\mathbb{B} &\mapsto \mathbb{B} \wedge \mathbb{B} \mid \square(\mathbb{K} \implies \mathbb{V}) \\
\mathbb{K} &\mapsto \mathbb{K} \wedge \mathbb{K} \mid \mathbb{K} \vee \mathbb{K} \mid \neg\mathbb{K} \mid s \leq \mathbb{S} \mid s \geq \mathbb{S} \mid s = \mathbb{S} \\
\mathbb{S} &\mapsto [s_{\min}, s_{\max}] \\
\mathbb{V} &\mapsto \mathbb{V} \wedge \mathbb{V} \mid \mathbb{V} \vee \mathbb{V} \mid \neg\mathbb{S} \mid x \leq \mathbb{X} \mid x \geq \mathbb{X} \mid x = \mathbb{X} \\
\mathbb{X} &\mapsto [x_{\min}, x_{\max}]
\end{aligned} \tag{4.31}$$

Expressions sampled from Equation (4.31) create a reactive policy for the disturbances, governed by which clauses $\mathbb{K}$ are TRUE. For example, consider the expression

$$\square(s \geq 0 \implies x < 1) \wedge \square(s \geq 1 \implies x < 0). \tag{4.32}$$

If the state is $s = 0.5$, then the first clause is satisfied and the disturbance must be $x < 1$. If the state is $s = 1.5$, then both clauses are satisfied and the disturbance must be $x < 0$. Lastly, if $s = -0.5$, then neither clause is satisfied and the disturbance is unrestricted.

To sample disturbance trajectories that satisfy this specification, we start by observing the state s and using it to determine which clauses are satisfied. For all such clauses we construct a specification φ that is the conjunction of all related constraints on the disturbance variables. We can then use algorithm 4.3 to sample the needed constraints and use them to produce a disturbance x. The disturbance is applied, the environment transitions to the next state and the process repeats.

4.4 Experiments

This section describes experiments that apply interpretable validation to the most likely failure analysis of an autonomous driving policy. We use the two driving scenarios described in Chapter 3: a T-intersection scenario where the ego vehicle makes an unprotected left turn, and a crosswalk scenario where the ego vehicle has to avoid a crossing pedestrian. The T-intersection scenario uses discrete disturbances and relatively simple failure modes. The crosswalk scenario has continuous disturbances (some of which are correlated in time) so the failure trajectories are higher dimensional.

In all of the experiments, the STL optimization was performed with genetic programming with a population of 3,000 individuals over 30 generations. Individuals were selected for the next generation through reproduction with probability 0.3, crossover with probability 0.3 and mutation with probability 0.4.

To compute the cost of each specification, we use a function similar to Equation (2.13), given by

$$c(\mathbf{x}) = \begin{cases} d(\mathbf{x}) + \beta & \text{no failure} \\ -\log p(\mathbf{x}) / \textsc{length}(\mathbf{x}) & \text{failure.} \end{cases} \tag{4.33}$$

where $d(\mathbf{x})$ is the point of closest approach between the ego vehicle and any adversarial agent in the scene with disturbance trajectory $\mathbf{x}$ and $\beta = 10^7$ is a large penalty for not finding a failure. The probability is divided by the length of the trajectory to better compare the likelihood of disturbances in trajectories of different lengths. The cost of each specification was computed using the average of $N = 10$ trajectories.

Specifications that failed to produce satisfying trajectories (i.e. were not satisfiable), had a cost of $c_{\max} = 10^9$ and we use a node count penalty of $\lambda = 0.01$.

We sample expressions from the STL grammar (Equation (4.5)) for both the T-intersection and crosswalk scenarios, and additionally use the state-dependent grammar (Equation (4.31)) for the crosswalk scenario. When the disturbance variables are continuous, we sample comparisons uniformly at random within 5 standard deviations of the mean (i.e. for $x \leq \alpha$, we sample $\alpha \in [\bar{x} - 5\sigma + \bar{x} + 5\sigma]$).

The performance of our algorithm is evaluated on three metrics: 1) The rate of failures generated (evaluated from 1,000 trials), 2) the likelihood of the failure trajectories (evaluated from 10 failure trajectories), and 3) the interpretability of the STL expression. Additionally, we show that failure descriptions can be used to generate similar failure examples and used in a low-dimensional sensitivity analysis of the system. We compare our approach to Monte Carlo (MC) sampling and two versions of the cross entropy method (CEM) (algorithm 2.4):

- Monte Carlo sampling: Disturbance trajectories are sampled from the true distribution $p(\mathbf{x})$.
- Cross entropy method (iid): We apply the cross entropy method and assume that the disturbances are independent and identically distributed (iid) at each time. We therefore optimize the parameters of a single distribution per disturbance component.
- Cross entropy method (trajectory): We apply the cross entropy method (still assuming independence at each timestep) but use a different distribution for each time, allowing the CEM to produce correlated trajectories.

The cross entropy methods use the same number of individuals and iterations as the genetic programming technique.

4.4.1 T-Intersection

The first scenario we analyze is the T-intersection scenario with a single adversarial agent. The action space is discrete and each action has probability shown in Table 3.4.

Three initial conditions of the left-turn (LT) scenario were investigated. Each initial condition is represented by the tuple $(s_{\text{ego}}, v_{\text{ego}}, s_{\text{adv}}, v_{\text{adv}})$ where s is the distance from the center of the intersection and v is the vehicle velocity. The initial conditions and corresponding nominal behavior of the ego vehicle are given below.

- LT1: $(15\,\text{m}, 9\,\text{m/s}, 29\,\text{m}, 10\,\text{m/s})$. The nominal behavior is for the ego vehicle to take the left turn before the other vehicle arrives at the intersection.
- LT2: $(15\,\text{m}, 9\,\text{m/s}, 29\,\text{m}, 20\,\text{m/s})$. The nominal behavior is for the ego vehicle to wait for other vehicle to pass through the intersection before turning left.
- LT3: $(19\,\text{m}, 9\,\text{m/s}, 43\,\text{m}, 29\,\text{m/s})$. The nominal behavior is the same as in LT2.

The results for the three initial conditions are shown in Table 4.2. Trajectories that were sampled from the optimal STL expression produced at least an order of magnitude more failures than the rollouts from the IS distribution. This means that once the optimal expression is computed, it is very effective at generating failures. Additionally, we see that the average disturbance probability is much higher for the interpretable failures than for the IS failures.

For LT1, we see that the optimal STL expression enforces a major acceleration in the first two timesteps of the simulation. This acceleration allows the adversarial vehicle to just reach the ego vehicle as it completes its left turn as shown in Figure 4.4. The ego vehicle initiated the turn based on the initial velocity of the other vehicle and did not predict such a significant speedup. For LT2, the generated expression has the adversarial vehicle toggle its turn signal but continue straight through the intersection as shown in Figure 4.5. The ego vehicle expects the adversarial vehicle to turn, so it initiates the left turn and causes a collision. Lastly, for LT3, the generated expression has the adversary either turn on its turn signal or perform a major deceleration in the first two timesteps of the simulation (Figure 4.6) leading to a similar failure as in LT2. From these experiments, we conclude that our approach is able to find interpretable expressions that reliable produce likely failures of the SUT.

One benefit to learning a failure description is the ability to produce many failure examples that have a shared temporal property. For example, we took the third

Table 4.2: Comparing interpretable validation (IV) to importance sampling (IS) for three different initial conditions of the left-turn scenario.

Method	**Expression**	**Failure Rate**	**Likelihood**
IV (LT1)	$\square_{[0,.36]} a_{\text{maj}}$	0.968 ± 0.003	0.78 ± 0.05
IS (LT1)	-	0.008 ± 0.008	0.11 ± 0.10
IV (LT2)	$\square_{[0,0]} B$	0.994 ± 0.001	0.82 ± 0.05
IS (LT2)	-	0.092 ± 0.036	0.15 ± 0.130
IV (LT3)	$\square_{[0,.36]} B \vee d_{\text{maj}}$	0.164 ± 0.004	0.75 ± 0.03
IS (LT3)	-	0.002 ± 0.001	0.15 ± 0.13

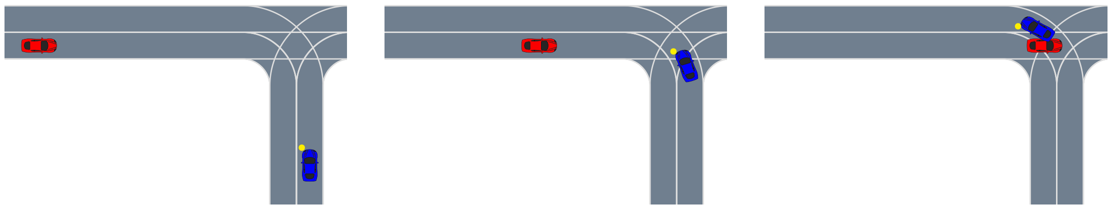


Figure 4.4: Collision for LT1 at $t = (0\,\text{s}, 1.08\,\text{s}, 1.98\,\text{s})$.

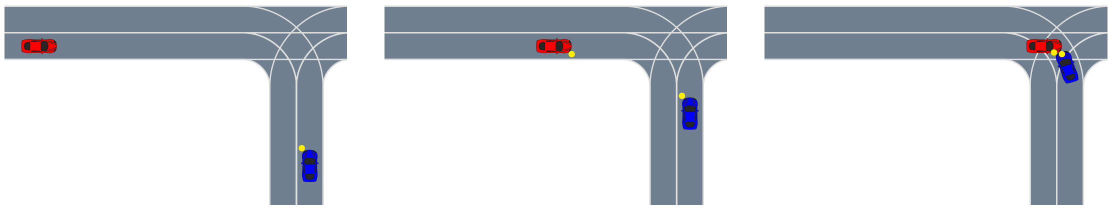


Figure 4.5: Collision for LT2 at $t = (0\,\text{s}, 0.72\,\text{s}, 1.26\,\text{s})$.

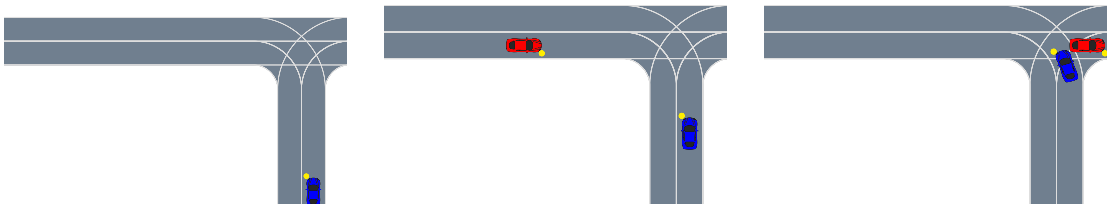


Figure 4.6: Collision for LT3 at $t = (0\,\text{s}, 0.90\,\text{s}, 1.62\,\text{s})$.

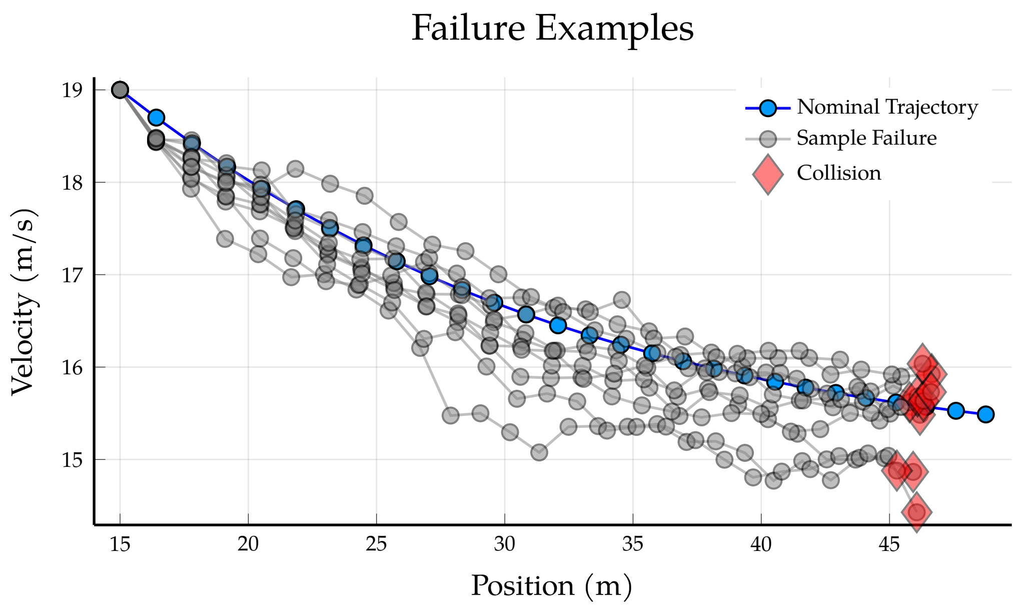


Figure 4.7: Sample failures that satisfy $\square_{[0,0.36]} B \vee d_{\text{maj}}$

failure description

$$\square_{[0,.36]} B \vee d_{\text{maj}} \tag{4.34}$$

and use it to sample 10 failure trajectores. We plot the velocity vs. the position of the adversary for the failure examples in Figure 4.7. We see that in the first two timesteps there is always a signification slowdown, but the trajectories then evolve stochastically before ending in a failure.

4.4.2 *Pedestrian in Crosswalk*

In the first set of experiments involving the pedestrian in a crosswalk we use the full STL grammar (Equation (4.5)) to sample expressions. The pedestrian accelerations in each direction were modeled as zero-mean Gaussian processes with a squared exponential kernel with parameters $\ell = 4$ and $\sigma^2 = 0.5$. The noise was modeled iid as zero mean Gaussians with parameters $\sigma_{r_x} = \sigma_{r_y} = 0.2$ and $\sigma_v = 0.5$. The initial condition for the vehicle was $(r_x^{\text{veh}}, v_x^{\text{veh}}) = (9.23\,\text{m}, 18.72\,\text{m/s})$ and for the pedestrian $(r_x^{\text{ped}}, r_y^{\text{ped}}, v_x^{\text{ped}}, v_y^{\text{ped}}) = (25\,\text{m}, -3.83\,\text{m}, 0.83\,\text{m/s}, 0\,\text{m/s})$.

Table 4.3: Comparison of interpretable validation with STL grammar to baselines for pedestrian scenario.

Method	**Failure Rate**	**Log-Likelihood**
Monte Carlo	0.003 ± 0.002	2.365 ± 0.235
Cross entropy method (IID)	0.865 ± 0.011	$-9.317 \times 10^4 \pm 2.824 \times 10^3$
Cross entropy method (Trajectory)	0.989 ± 0.003	-11.31 ± 1.980
Interpretable validation	0.960 ± 0.006	1.835 ± 0.218

The interpretable validation algorithm produced the failure description

$$\Box_{[1,3]} \left((1.68 \leq \delta a_y \leq 2.69) \land (\delta a_x \geq 0.34) \land (\delta n_y \geq 0.13) \right). \tag{4.35}$$

In words, there is a positive lateral and longitudinal acceleration of the pedestrian early in the trajectory, combined with a positive bias in the noise. The vehicle initially does not think the pedestrian will reach the road before it passes, but the sudden change in pedestrian velocity leads to this being a miscalculation and a collision occurs. Ten example failure trajectories were generated from the failure description and plotted in Figure 4.8.

The failure rate and log-likelihood results for the failure description and the baseline are shown in Table 4.3. We notice that although the cross entropy methods can produce a similarly high rate of failures as the failure description, the log probability of those failures is significantly smaller. The failure description produces failure trajectories nearly 100% of the time while retaining a log-likelihood similar to failures found through Monte Carlo sampling.

We can investigate which combination of acceleration and noise is critical for causing failures of this sort by doing a sensitivity analysis with respect to these to variables. Figure 4.9 shows the failure rate over a range of values on the lower bounds of δa_y and δn_v. There seems to be a critical threshold for δa_y around $1.5\,\mathrm{m/s^2}$ where collision becomes inevitable. At lower values of δa_y, higher values of noise are required to cause a collision. This suggests the need to improve the trajectory prediction of the autonomous vehicle or it needs to behave more conservatively to

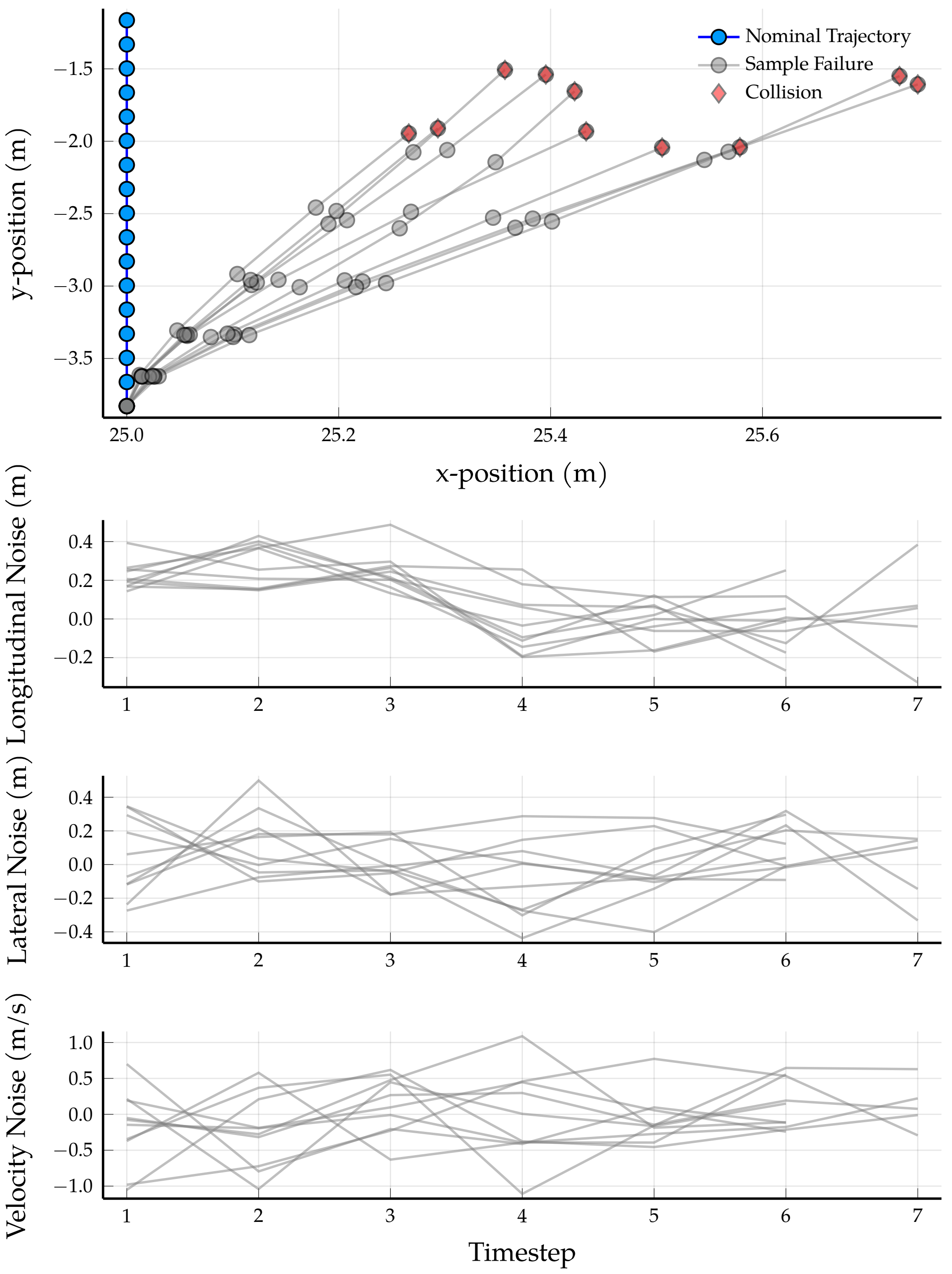


Figure 4.8: Sample failure trajectories from pedestrian STL failure description.

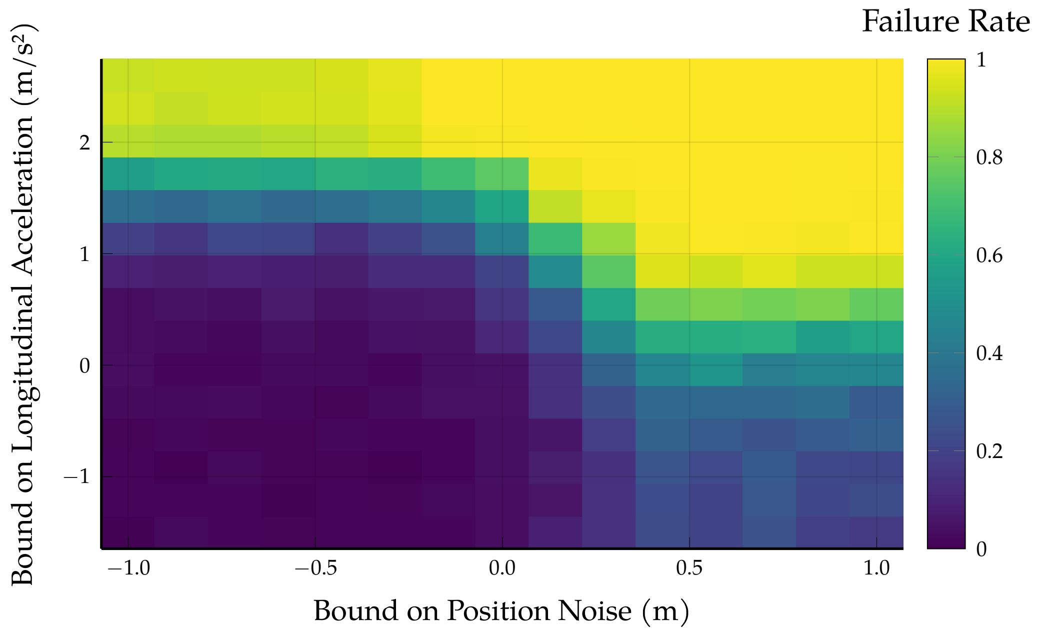


Figure 4.9: System sensitivity analysis for longitudinal acceleration bound and velocity noise upper bound.

account for this type of pedestrian behavior.

The second set of experiments with the pedestrian use the state-dependent STL grammar (Equation (4.31)) to sample expressions. For the state-dependent setting it is much easier if the disturbances are not correlated over time. We therefore modeled all of the disturbances as zero-mean Gaussians with standard deviations given by $\sigma_{a_x} = \sigma_{a_y} = 1$ and $\sigma_{r_x} = \sigma_{r_y} = \sigma_v = 0.5$. The initial condition for the vehicle was $(r_x^{\text{veh}}, v_x^{\text{veh}}) = (9.64\,\text{m}, 9.34\,\text{m/s})$ and for the pedestrian $(r_x^{\text{ped}}, r_y^{\text{ped}}, v_x^{\text{ped}}, v_y^{\text{ped}})$ $= (25\,\text{m}, -6.99\,\text{m}, 1.94\,\text{m/s})$.

Interpretable validation produced the STL expression

$$\begin{aligned} r_x^{\text{ped}} \geq 20.23 &\implies \delta a_y \leq 1.67 \\ r_x^{\text{veh}} \geq 16.51 &\implies \delta n_y \geq 1.59 \\ v^{\text{veh}} \geq 0.26 &\implies \delta a_y \geq 1.10 \\ (r_x^{\text{veh}} \leq 27.87) \vee (r_x^{\text{veh}} \geq 15.61) &\implies (\delta n_v \leq 0.20) \wedge (\delta a_x \leq 1.91) \end{aligned} , \qquad (4.36)$$

Table 4.4: Comparison of interpretable validation with state-dependent grammar to baselines for pedestrian scenario.

Method	**Failure Rate**	**Log-Likelihood**
Monte Carlo	0.0 ± 0.0	-
Cross entropy method (IID)	0.184 ± 0.012	-7.743 ± 0.264
Cross entropy method (Trajectory)	0.566 ± 0.016	-3.088 ± 0.003
Interpretable validation	0.997 ± 0.002	-8.500 ± 0.166

with failure rate and log-likelihood results shown in Table 4.4. We see that the failure description produces failure trajectories almost every time, significantly more than either cross entropy approach. The log-likelihood of the failures from the failure description have lower log-likelihood than from the CEM, but the CEM lacks interpretability.

We again notice that some of the constraints are not likely to be applied but the bounds on the acceleration in the y direction and noise on the y position are strict. In this case, failure occurs because the vehicle observes that the pedestrian is in the road, and slows just the right amount for the accelerating pedestrian to collide with the side of the vehicle. If the vehicle had not slowed down it would have missed the pedestrian entirely.

We can again investigate the sensitivity of this failure mode with respect to the restrictive bounds (shown in Figure 4.11). We see that there is only a small area where this type of failure can occur. If we could guarantee that the longitudinal noise was always less than $0.5\,\mathrm{m}$ then it would not be possible for this type of failure to occur if the pedestrian cannot accelerate by more than $1.5\,\mathrm{m/s^2}$.

Through these experiments we see that interpretable safety validation provides a technique for generating failure descriptions that can produce a diverse set of high-likelihood failure examples. Additionally, we can use the low-dimensionality of the failure descriptions to perform a sensitivity analyses on the failure modes.

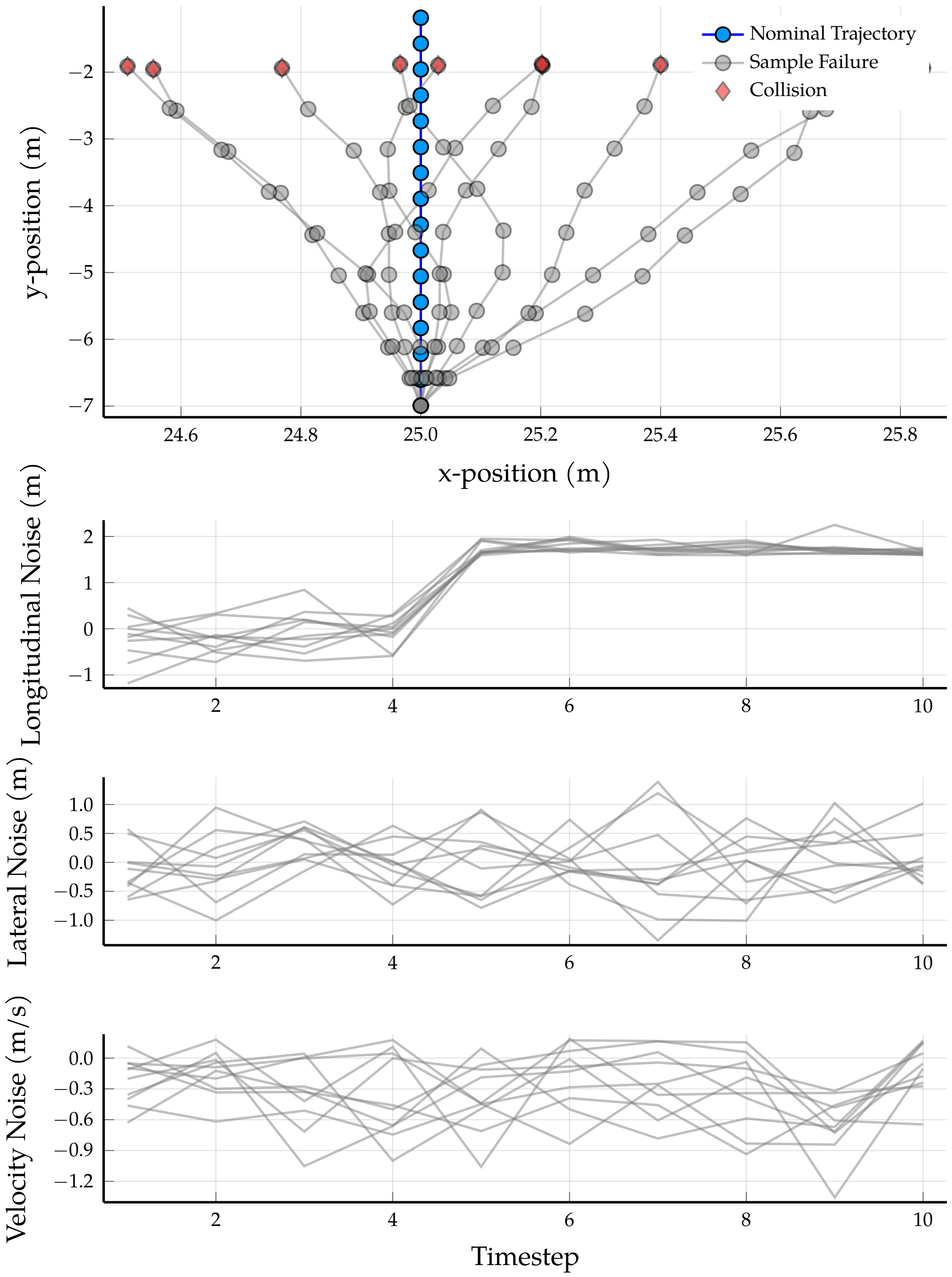


Figure 4.10: Sample failure trajectories from pedestrian state-dependent failure description.

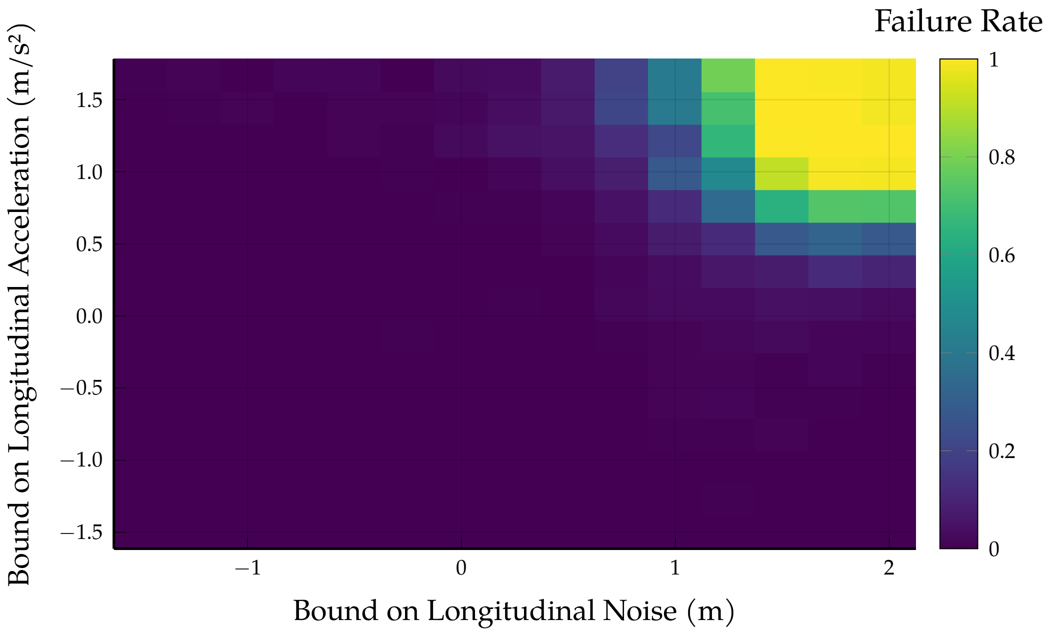


Figure 4.11: System sensitivity analysis for longitudinal acceleration bound and longitudinal position noise bound.

4.5 *Discussion*

In this section, we developed a technique for finding STL expressions that describe failure trajectories. The expressions are optimized using genetic programming and evaluated by computing the cost of satisfying trajectories. We introduced a constraint-based technique for sampling satisfying trajectories which can be used for simple disturbance models. Through some autonomous driving experiments, we demonstrated that interpretable validation can be used to find failure descriptions that reliable produce failure trajectories and provide some insight into the possible changes required of the system for it to perform safety.

We find that interpretable failure descriptions place minimal constraints on the disturbance trajectories, so many of the disturbances are selected according to the true model of the environment. Since rare event are only selected when they are needed to cause a failure, trajectories sampled from the failure description tend to have likelihood. Techniques that generally increase the probability of rare events (such as the iid cross entropy method) cause rare disturbances to be chosen with regularity, even when they are not required for causing a failure. This leads to failure trajectories with low log-probability.

This work on finding interpretable failure descriptions is related to the broader task of interpretable machine learning, where models classes are chosen so as to be understood by a human. Interpretable models must choose an output representation that is sufficiently expressive to compactly convey the desired information while having direct mappings to words and concepts. Decision trees use a branching set of rules to classify data [119] or represent a reinforcement learning policy [120]. Simple mathematical expressions can be used for regression [121], governing equation discovery [122], and reinforcement learning [123]. Signal temporal logic (STL) has been used for logical clustering of high-dimensional trajectory data data [124], [125] because it describes logical relationships between temporal variables and has a natural-language description that is easily understood.

Most similar to the present work that of Lee, Kochenderfer, *et al.* [124] where they generated STL descriptions of failure modes using grammar-based decision trees for classification. They were able to find interpretable features of the time series that differentiated failure modes, but did not generate specifications that were sufficient for describing them.

In the next chapter we approach the safety validation problem from an importance sampling perspective and try to construct the distribution over failures. The proposal distribution allows us to sample a diverse set of the most likely failures of a system and can be learned using reinforcement learning techniques.

5 Distribution Over Failures

In this chapter, we develop a technique for approximating the distribution over failure trajectories. We construct a state-dependent disturbance policy that, when rolled out, causes the system to fail. The distribution over failures allows us to sample a diverse set of likely failures as well as efficiently compute the probability of failure. We start by showing that a sampling distribution that is proportional to the probability of failure is an efficient proposal distribution, and is exactly the distribution over failures when the environment is deterministic. We then describe how to compute the probability of failure using a Bellman equation and approximate it using neural networks and local linear interpolation. Through experiments in the gridworld and T-intersection environments, we show that even when the estimate of the probability of failure has a large amount of error, we still obtain an efficient proposal distribution.

5.1 Motivation

In the analysis of safety-critical autonomous systems, the tools of falsification and most-likely failure analysis may not be sufficient to certify the safety of system. They can merely demonstrate that the system is unsafe by finding failure examples. For complex cyber-physical systems, some failures will almost surely exist, and certification will depend on how likely it is for a system to fail. For this reason, we need to develop effective techniques to estimate the probability of failure of an autonomous system, which means creating a distribution over disturbances that causes the most likely failures to occur.

In addition to the goal of efficient estimation, having a distribution over failures is helpful in the development of safety-critical autonomous systems. If an autonomous system is developed through machine learning, then its ability to navigate dangerous scenarios will likely depend on how many dangerous examples it was trained on. A distribution that produces a diverse set of dangerous scenarios could be used to improve the safety of such systems.

The difference between the three safety validation tasks is further illustrated in Figure 5.1. In Figure 5.1a we show the expert policy of an agent navigating a simple gridworld with stochastic transitions. When we apply falsification (Figure 5.1b), we find a failure trajectory that is extremely unlikely because falsification algorithms do not consider the probability of the disturbance. We can partially remedy the situation by performing a most-likely failure analysis to arrive at the trajectory shown in Figure 5.1c. Although this failure is the one most likely to occur, it cannot be used to estimate the probability that the system fails and only provides a single example that could be used for improving the system performance. Instead, if we can compute the distribution over failures, that is, sample

$$\mathbf{x} \sim p(\mathbf{x} \mid f(\mathbf{x}) \notin \psi), \tag{5.1}$$

then we produce many different failure examples, all with relatively high likelihood. Figure 5.1d shows samples from the distribution over failures which was computed using techniques developed in the rest of this chapter.

Estimating the distribution over failures is synonymous with finding an efficient proposal distribution for estimating the probability of failure. Importance sampling approaches for estimating the optimal proposal distribution (as discussed in Section 2.4.4) are limited by the fact that they produce samples of the entire disturbance trajectory **x** and do not have a way of incorporating the environment state. A distribution that does not depend on the environment state will perform poorly in settings where the initial state of the environment or the state transitions are stochastic. To handle these settings, we want to develop a sampling distribution that depends upon the environment state and approximates the distribution over failures.

In the rest of the chapter, we propose a stochastic disturbance policy $q(x \mid s)$ that, when applied to a safety validation problem, approximately generates failures from the distribution over failures. If the safety validation problem is deterministic, we find that the optimal disturbance policy is $q^*(x \mid s) \propto p(x \mid s)P_{\text{fail}}(s')$ where $P_{\text{fail}}(s')$ is the probability that the system fails from the state s' reached by applying disturbance x in state s. We use this optimal policy as a motivation for a general policy that can be used in the non-deterministic case. We show that this distribution lowers the variance of an estimator of the probability of failure compared to a basic Monte Carlo approach.

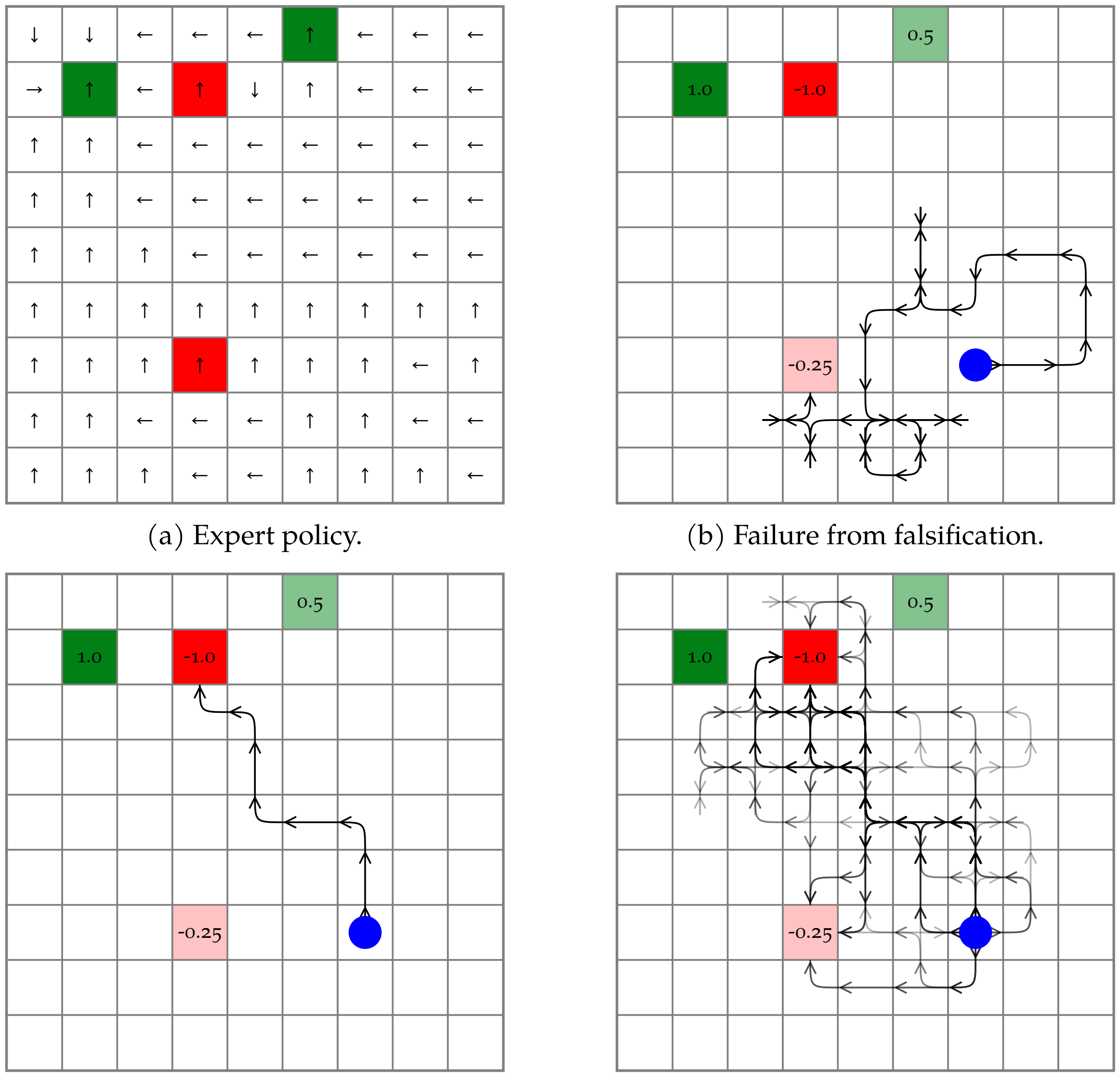
5.2 State-Dependent Importance Sampling

This section describes the problem of constructing a state-dependent importance sampling distribution for efficiently estimating the probability of failure of a system. We first describe the sampling problem and then propose a state-dependent disturbance distribution that produces low-variance estimates of the probability of failure. Through an analysis of variance, we show that our proposed distribution is optimal when the system and environment are deterministic given the disturbances.

5.2.1 Problem Setup

Suppose we wish to estimate the probability of failure P_{fail} of a system in an environment with states $s \in S$, subject to disturbances $x \in X$. A trajectory is $\boldsymbol{\tau} = [s_0, x_1, s_1, \ldots x_{t_{\max}}, s_{t_{\max}}]$ and has probability $p(\boldsymbol{\tau})$, which models the likelihood of state transitions and disturbances. A trajectory is a failure if the sequence of states violates the safety specification ψ and we denote a failure trajectory of both states and disturbances as $\boldsymbol{\tau} \notin \psi$.

For importance sampling, we wish to develop a proposal distribution $q(\boldsymbol{\tau})$ that estimates the probability of failure with the minimal number of samples. An unbiased estimator of the probability of failure can be computed from N independent



(a) Expert policy.

(b) Failure from falsification.

(c) Most-likely failure.

(d) Samples from failure distribution.

Figure 5.1: Comparison of safety validation tasks.

samples $\boldsymbol{\tau}_i \sim q(\cdot)$ as

$$\hat{P}_{\text{fail}} = \frac{1}{N} \sum_{i=1}^{N} \frac{p(\boldsymbol{\tau}_i)\mathbb{1}\{\boldsymbol{\tau}_i \notin \psi\}}{q(\boldsymbol{\tau}_i)}, \tag{5.2}$$

where $\mathbb{1}$ is the indicator functions.

The variance of $\hat{P}_{\text{fail}}$ is given by

$$\text{Var}_q\left[\hat{P}_{\text{fail}}\right] = \frac{1}{N}\left(\mathbb{E}_q\left[\left(\frac{p(\boldsymbol{\tau})\mathbb{1}\{\boldsymbol{\tau} \notin \psi\}}{q(\boldsymbol{\tau})}\right)^2\right] - \mathbb{E}_q\left[\frac{p(\boldsymbol{\tau})\mathbb{1}\{\boldsymbol{\tau} \notin \psi\}}{q(\boldsymbol{\tau})}\right]^2\right) \tag{5.3}$$

$$= \frac{1}{N}\left(\mathbb{E}_p\left[\frac{p(\boldsymbol{\tau})\mathbb{1}\{\boldsymbol{\tau} \notin \psi\}}{q(\boldsymbol{\tau})}\right] - P_{\text{fail}}^2\right) \tag{5.4}$$

where we have used the fact that the estimator is unbiased so $\mathbb{E}_q\left[\hat{P}_{\text{fail}}\right] = P_{\text{fail}}$. From importance sampling theory (and by inspection of Equation (5.3)), we know that the minimum variance proposal distribution is given by

$$q^*(\boldsymbol{\tau}) = \frac{p(\boldsymbol{\tau})\mathbb{1}\{\boldsymbol{\tau} \notin \psi\}}{P_{\text{fail}}}. \tag{5.5}$$

In safety validation of sequential decision-making problems, we cannot directly sample trajectories $\boldsymbol{\tau}$ of the system. Instead, the adversary influences the trajectories through the choice of disturbance. We must therefore define a distribution over disturbances that depends only on the trajectory at earlier timesteps and not future ones. We construct a distribution over the disturbance at time t with the form

$$q(x_t, \mid \boldsymbol{\tau}_{0:t-1}) \tag{5.6}$$

where $\boldsymbol{\tau}_{0:t} = [s_0, x_1, s_1, \dots, x_t, s_t]$. The full distribution over trajectories is constructed as

$$q(\boldsymbol{\tau}) = p(s_0) \prod_{t=1}^{t_{\max}} p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t) q(x_t, \mid \boldsymbol{\tau}_{0:t-1}). \tag{5.7}$$

5.2.2 *Proposal Distribution*

We now present a proposal distribution that reduces the variance of $\hat{P}_{\text{fail}}$ compared to Monte Carlo sampling and is optimal in the deterministic setting. Define the disturbance policy

$$q^*(x_t, \mid \boldsymbol{\tau}_{0:t-1}) = \frac{p(x_t \mid \boldsymbol{\tau}_{0:t-1}) P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1}, x_t)}{P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1})}, \tag{5.8}$$

where $P_{\text{fail}}(\boldsymbol{\tau}_{0:t})$ is the probability that the system fails under the distribution $p(\boldsymbol{\tau})$ given the sequence $\boldsymbol{\tau}_{0:t}$. The full proposal distribution over trajectories is given by

$$q^*(\boldsymbol{\tau}) = p(s_0) \prod_{t=1}^{t_{\max}} \frac{p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t) p(x_t \mid \boldsymbol{\tau}_{0:t-1}) P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1}, x_t)}{P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1})} \tag{5.9}$$

$$= p(\boldsymbol{\tau}) \prod_{t=1}^{t_{\max}} \frac{P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1}, x_t)}{P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1})}. \tag{5.10}$$

At first, it may seem circular to define a proposal distribution that depends upon the very quantity we are trying to estimate (the probability of failure). Our goal is not just to estimate the probability of failure, but to construct a distribution that we can sample failures from. Additionally, we will show through experimentation that an efficient proposal distribution can be constructed with even a rough estimate of the probability of failure at each state. For the following analysis we assume we know the probability of failure exactly and that the states and disturbances are discrete. A similar analysis may be performed for continuous variables by appropriately converting sums to integrals.

Proposition. *The disturbance policy $q^*(x_t, \mid \boldsymbol{\tau}_{0:t-1})$ induces a valid probability distribution over trajectories.*

Proof. To show this, we need to show that $q^*(x_t, \mid \boldsymbol{\tau}_{0:t-1})$ is a valid probability distribution and $q^*(\boldsymbol{\tau})$ will be valid by construction from Equation (5.7). First we note that $P_{\text{fail}}(\cdot) \in [0, 1]$ and $p(x_t \mid \boldsymbol{\tau}_{0:t-1}) > 0$ so $q^*(x_t, \mid \boldsymbol{\tau}_{0:t-1})$ is non-negative for all disturbances x_t. Then we can show proper normalization by using a recursive definition of the probability of failure, given by

$$P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1}) = \sum_{x_t} p(x_t \mid \boldsymbol{\tau}_{0:t-1}) P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1}, x_t). \tag{5.11}$$

Thus

$$\begin{aligned}
\sum_{x_t} q^*(x_t, \mid \boldsymbol{\tau}_{0:t-1}) &= \sum_{x_t} \frac{p(x_t \mid \boldsymbol{\tau}_{0:t-1}) P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1}, x_t)}{P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1})} && (5.12)\\
&= \frac{P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1})}{P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1})} && (5.13)\\
&= 1 && (5.14)
\end{aligned}$$

□

If we use the disturbance policy to sample trajectories $\boldsymbol{\tau}$ and use them to estimate the probability of failure, we need to understand the variance of the estimate. A good proposal distribution will produce estimates that have lower variance than estimates computed from the true distribution $p(\boldsymbol{\tau})$.

Proposition. *The variance of the failure probability estimate (Equation (5.2)) under the proposal distribution q^* is*

$$Var_{q^*}\left[\hat{P}_{\text{fail}}\right] = \frac{P_{\text{fail}}^2}{N}\left(\mathbb{E}_{p_{\tau \mid \mathbb{1}\{\tau \notin \psi\}}}\left[\prod_{t=0}^{t_{\max}} \frac{p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t, \boldsymbol{\tau} \notin \psi)}{p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t)}\right] - 1\right), \tag{5.15}$$

Proof. To show this, we can start by substituting Equation (5.10) into Equation (5.4) to get

$$\text{Var}_{q^*}\left[\hat{P}_{\text{fail}}\right] = \frac{1}{N}\left(\mathbb{E}_p\left[\mathbb{1}\{\boldsymbol{\tau} \notin \psi\} \prod_{t=1}^{t_{\max}} \frac{P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1})}{P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1}, x_t)}\right] - P_{\text{fail}}^2\right) \tag{5.16}$$

Then, applying Bayes' rule we have

$$P_{\text{fail}}(\boldsymbol{\tau}_{0:t}) = \frac{p(\boldsymbol{\tau}_{0:t} \mid \boldsymbol{\tau} \notin \psi) P_{\text{fail}}}{p(\boldsymbol{\tau}_{0:t})}. \tag{5.17}$$

When substituted into the ratio term in Equation (5.16) we get

$$\prod_{t=1}^{t_{\max}} \frac{P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1})}{P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1}, x_t)} = \prod_{t=1}^{t_{\max}} \frac{p(\boldsymbol{\tau}_{0:t-1} \mid \boldsymbol{\tau} \notin \psi) p(\boldsymbol{\tau}_{0:t-1}, x_t)}{p(\boldsymbol{\tau}_{0:t-1}, x_t \mid \boldsymbol{\tau} \notin \psi) p(\boldsymbol{\tau}_{0:t-1})} \tag{5.18}$$

$$= \prod_{t=1}^{t_{\max}} \frac{p(x_t \mid \boldsymbol{\tau}_{0:t-1})}{p(x_t \mid \boldsymbol{\tau}_{0:t-1}, \boldsymbol{\tau} \notin \psi)} \tag{5.19}$$

$$= \prod_{t=1}^{t_{\max}} \frac{p(x_t, s_t \mid \boldsymbol{\tau}_{0:t-1}) p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t, \boldsymbol{\tau} \notin \psi)}{p(x_t, s_t \mid \boldsymbol{\tau}_{0:t-1}, \boldsymbol{\tau} \notin \psi) p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t)} \tag{5.20}$$

$$= \frac{p(\boldsymbol{\tau})}{p(\boldsymbol{\tau} \mid \boldsymbol{\tau} \notin \psi)} \prod_{t=0}^{t_{\max}} \frac{p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t, \boldsymbol{\tau} \notin \psi)}{p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t)} \tag{5.21}$$

$$= P_{\text{fail}} \prod_{t=0}^{t_{\max}} \frac{p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t, \boldsymbol{\tau} \notin \psi)}{p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t)} \tag{5.22}$$

where we have used the fact $p(\boldsymbol{\tau} \mid \boldsymbol{\tau} \notin \psi) = p(\boldsymbol{\tau})/P_{\text{fail}}$ to get Equation (5.22). Finally, we get the desired result by plugging Equation (5.22) into Equation (5.16) and applying the identity

$$\mathbb{E}_p \left[\mathbb{1}\{\boldsymbol{\tau} \notin \psi\} f(\boldsymbol{\tau})\right] = \mathbb{E}_p \left[\mathbb{1}\{\boldsymbol{\tau} \notin \psi\}\right] \mathbb{E}_{p_{\boldsymbol{\tau} \mid \mathbb{1}\{\boldsymbol{\tau} \notin \psi\}}} \left[f(\boldsymbol{\tau})\right] \tag{5.23}$$

$$= P_{\text{fail}} \mathbb{E}_{p_{\boldsymbol{\tau} \mid \mathbb{1}\{\boldsymbol{\tau} \notin \psi\}}} \left[f(\boldsymbol{\tau})\right]. \tag{5.24}$$

□

The variance of the estimator from q^* differs from the Monte Carlo estimator by the term

$$\mathbb{E}_{p_{\tau \mid \mathbb{1}\{\boldsymbol{\tau} \notin \psi\}}} \left[\prod_{t=1}^{t_{\max}} \frac{p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t, \boldsymbol{\tau} \notin \psi)}{p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t)} \right]. \tag{5.25}$$

We note that this term is always greater than 1 and is maximized when the choice of disturbance has no influence on the next state of the environment. In that case, $P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1}) = P_{\text{fail}}(\boldsymbol{\tau}_{0:t-1}, x_t)$ and we can see from Equation (5.16), that the variance increases to the Monte Carlo estimate. The less stochasitic the environment is given the state, the smaller the variance of the estimator. In the case where each state transition is deterministic given the disturbance, we have $p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t, \boldsymbol{\tau} \notin \psi) =$

$p(s_t \mid \boldsymbol{\tau}_{0:t-1}, x_t)$, and the variance goes to zero, indicating that q^* is the optimal importance sampling distribution.

Assuming the environment does not have too much uncontrolled stochasticity, we can approximate the distribution over failures by sampling disturbances according to q^*. To compute q^* we need $p(x \mid s)$, which is given, and $P_{\text{fail}}(\cdot)$. The following section describes how to efficiently compute the probability of failure as a function of the state when we can make the Markov assumption.

5.3 *Estimating the Probability of Failure*

An efficient proposal distribution can computed using an estimate of the probability of failure for each trajectory segment. Estimating the probability of failure over the distribution of trajectories could quickly become intractable due to the dimensionality of the space of trajectories. The computation can be significantly simplified if we assume that the system and environment are Markov so the probability of failure only depends on the current state and the choice of the next disturbance. The Markov version of all the equations in the previous section can be obtained by letting $\boldsymbol{\tau}_{0,t} = s_t$.

With the Markov assumption, the probability of failure satisfies the Bellman equation

$$P_{\text{fail}}(s) = \begin{cases} 1 & \text{if } s \in S_{\text{fail}} \\ 0 & s\text{if } \in S_{\text{term}} \text{ and } s \notin S_{\text{fail}} \\ \sum_x p(x \mid s) P_{\text{fail}}(s, x) & \text{otherwise} \end{cases} \tag{5.26}$$

where S_{fail} is the set of failure states, S_{term} is the set of terminal states, and

$$P_{\text{fail}}(s, x) = \sum_{s'} p(s' \mid s, x) P_{\text{fail}}(s'). \tag{5.27}$$

Equation (5.26) is similar in structure to the Bellman equations that show up in the theory of sequential decision making (i.e. Equations (2.16) and (2.18)). As such, we can apply many of the same techniques used in reinforcement learning to compute the probability of failure. We now introduce exact and approximate value iteration and Q-learning with neural network function approximation.

5.3.1 *Value Iteration*

When the state space of the environment is discrete or easily discretized, we can use exact or approximate value iteration [80] to compute the probability of failure. Value iteration is a dynamic programming technique that iteratively solves a Bellman equation over a set of states. In exact value iteration, $P_{\text{fail}}(s)$ is represented by a table with entries at every state. Exact value iteration is only feasible when the state space is discrete and relatively small. For larger state spaces, approximate value iteration represents $P_{\text{fail}}(s)$ by a function. In local approximation value iteration the function is usually constructed from piecewise continuous polynomials that can exactly satisfy the estimated value at a set of sampled states. In global approximation value iteration there is a single function that maps states to values (such as a neural network), so we cannot generally specify the value of given state.

A general value iteration procedure is outlined in algorithm 5.1. It takes as input the set of states $\mathbf{s}$ (either the complete state space or a finite number of sample states), the set of failure states S_{fail} and the set of terminal states S_{terminal}. The algorithm starts by initializing the estimate of the probability of failure at each state (line 2), and then iterates until convergence. For each state, we apply the Bellman update operator to compute the target value y (lines 5 to 10). When computing the term on line 10, we usually estimate $P_{\text{fail}}(s, x)$ by sampling a set of next states s' and using an average of the approximation $P_{\text{fail}}(s')$. Lastly, we update the estimate of the probability of failure using the target value. In exact and local approximation value iteration the update simply assigns the value y to the corresponding state. In global approximation value iteration, we typically minimize $L(P(s), y)$ for some loss function L. Once the algorithm converges the estimate of the probability of failure is returned (line 12).

Exact value iteration is guaranteed to converge to the true value function which

Algorithm 5.1 Value Iteration for computing the probability of failure.

```
 1: function ValueIteration($S$, $S_{\text{fail}}$, $S_{\text{terminal}}$)
 2:     Initialize($P_{\text{fail}}(s)$)
 3:     loop
 4:         for each state $s \in \mathbf{s}$
 5:             if $s \in S_{\text{fail}}$
 6:                 $y \leftarrow 1$
 7:             else if $s \in S_{\text{terminal}}$ and $s \notin S_{\text{fail}}$
 8:                 $y \leftarrow 0$
 9:             else
10:                 $y \leftarrow \sum_x p(x \mid s) P_{\text{fail}}(s, x)$
11:             update($P_{\text{fail}}(s)$, $y$)
12:     return $P_{\text{fail}}(s)$
```

makes it an appealing algorithm to use, but it has limits in scalability and the types of environments it can work with. Explicitly looping over the set of state excludes environments with large or continuous state spaces. Even a coarse discretization becomes intractable because the number of grid points scales exponentially with the number of state dimensions. Additionally, value iteration requires the ability to initialize a system and environment into a particular state. This requirement may not be feasible more many real-world simulators and it precludes the analysis of non-Markovian systems. We now introduce deep Q-Learning for failure probability estimation, which relaxes these assumptions.

5.3.2 *Deep Q-Learning*

The DQN algorithm [88] can easily be adapted to compute the probability of failure. The Q-network takes as input the state and outputs the probability of a failure for each disturbance, estimating $P_{\text{fail}}(s, x)$. As in value iteration techniques, we start be assigning failure states a value of 1 and non-failure terminal states a value of 0. We use the same algorithm as algorithm 2.3 but alter the target (Equation (2.22)) to be

$$y(s,x,s';\theta^-) = \begin{cases} 1 & \text{if } s' \in S_{\text{fail}} \\ 0 & \text{if } s' \notin S_{\text{fail}} \text{ and } s' \in S_{\text{terminal}} \\ \sum_{x'} p(x' \mid s) P_{\text{fail}}(s', x') & \text{otherwise} \end{cases}. \tag{5.28}$$

Similar to traditional DQN, we collect new experience samples with an ϵ-greedy approach. We sample a disturbance according to our current estimate of q^* with probability $1-\epsilon$, and with probability ϵ, we sample a disturbance uniformly at random.

In addition to changing the target and rollout policy, there are a few other changes that are necessary for training a good predictor of the probability of failure. First, we need to ensure that the network output is always between 0 and 1 to be a valid probability value. The easiest way to force this constraint is to have the last layer of the network be a sigmoid activation given by

$$\sigma(z) = \frac{1}{1+e^{-z}}, \tag{5.29}$$

which maps the real numbers to the range $(0,1)$.

When estimating the probability of failure, we must also be careful in our choice of loss function. DQN usually employs the Huber loss or the mean squared error loss but these loss functions do not handle differences in small values very well. Even if the network has many orders of magnitude error in a prediction of the probability of failure, the mean squared error of that prediction may still be small. We therefore employ the mean squared log error (MSLE) given by

$$L_{\text{MSLE}}(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(\log(y+\epsilon) - \log(\hat{y}+\epsilon))^2 \tag{5.30}$$

where $\hat{y}$ is the prediction of y and ϵ is a small value added for numerical stability. Informally, the MSLE penalizes the relative error instead of the absolute error.

In traditional DQN with prioritized experience replay, the experience samples are prioritized according to their temporal difference error $|y - \hat{y}|$. Again, this value can be small even when the relative error is large. We therefore make the priority of each sample equal to the difference in the log values $|\log(y + \epsilon) - \log(y \hat{+} \epsilon)|$. This alteration allows the network to focus on samples that have large relative error.

DQN has no explicit dependence on the size of the state space because it gathers data by sampling episodes from a simulator. It can therefore scale to problems with large state spaces and never requires the simulator to be initialized into a particular state, making it more easily applied to large and inflexible simulators or non-Markovian systems. Some drawbacks to DQN are the lack of formal convergence guarantees and the requirement of a discrete disturbance space.

5.4 Experiments

This section demonstrates our technique for approximating the distribution over failures of a Markovian system using the state-dependent proposal distribution

$$q^*(x_t, \mid s_{t-1}) = \frac{p(x_t \mid s_{t-1}) P_{\text{fail}}(s_{t-1}, x_t)}{P_{\text{fail}}(s_{t-1})}. \tag{5.31}$$

When the ground truth is unavailable, we use the following metrics to evaluate the performance of our proposal distribution:

- *Failure rate*: The fraction of failures found when using the proposal distribution

$$\frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\left\{\boldsymbol{\tau}_i \notin \psi\right\}, \quad \boldsymbol{\tau}_i \sim q^*(\boldsymbol{\tau}) \tag{5.32}$$

  The true distribution over failures would always produce failures, but due to stochasticity in initial conditions and errors in the estimate of the probability of failure, the failure rate is usually less than 1. We want a proposal distribution that returns as many failures as possible. In our experiments, failure rate is estimated with 1,000 samples.

- *Failure trajectory log-likelihood*: The average log-likelihood of failure trajectories produced with the proposal distribution:

$$\frac{1}{N}\sum_{i=1}^{N}\log p(\boldsymbol{\tau}_i), \quad \boldsymbol{\tau}_i \sim q^*(\boldsymbol{\tau} \mid \boldsymbol{\tau} \notin \psi). \tag{5.33}$$

For most-likely failure analysis and estimating the probability of failure, we want a distribution that produces high likelihood failures. In our experiments the log-likelihood of failures is estimated from 100 failure trajectories.

- *Probability of failure estimate*: The best indicator of an efficient proposal distribution is the ability to estimate the probability of failure with minimal samples. We report the estimate of the probability of failure against the number of samples and 99% confidence bounds. The confidence bounds are computed from a Beta distribution with the same mean and variance as the estimate.

We compare our proposal distribution to three baselines:

- *Monte Carlo sampling*: Disturbance trajectories are sampled from the true distribution $p(x \mid s)$.
- *Uniform sampling*: Disturbance trajectories are sampled from a uniform distribution over disturbances.
- *Cross entropy method*: We apply the cross entropy method to learn the optimal distribution over disturbances without reference to the state. The distribution was estimated over 100 iterations, each with 1,000 samples and a minimum of 100 elite samples.

The first set of experiments use the simple gridworld scenario. We first show that the even under a significant amount of error in the estimate of the probability a failure, Equation (5.31) performs well. We then shown that DQN can effectively model the probability of failure even when the probability of failure is very small. The second set of experiments uses the T-intersection scenario with a single adversary. We show that both local approximation value iteration and DQN outperform baseline methods.

5.4.1 *Simple Gridworld*

The first set of experiments involves the simple gridworld scenario because it can be solved using exact value iteration. Access to the ground truth allows us to investigate the effect of errors in the estimate of the probability of failure and the use of approximate algorithms such as DQN. Using the reward distribution shown in Figure 3.1, we train an agent to expertly navigate the gridworld subject to a successful transition probability of $p_{\text{success}} = 0.999$. We then construct an adversarial gridworld that controls the transitions of the agent and use it for the following experiments.

To investigate the effect of errors in the estimate of the probability of failure we perturb the ground truth $P_{\text{fail}}(s)$ and observe its effect on our performance metrics. To compute the noisy probability of failure estimate $\tilde{P}_{\text{fail}}(s)$ at state s we independently sample $\delta \sim \mathcal{U}(-\delta_{\max}, \delta_{\max})$, and then multiply

$$\tilde{P}_{\text{fail}}(s) = 10^{\delta} P_{\text{fail}}(s). \tag{5.34}$$

The parameter $\delta_{\max}$ specifies the maximum magnitude of error in the estimate of the probability of failure. When $P_{\text{fail}}(s)$ is exactly 0, we first initialize it to the smallest nonzero entry in the table and then apply the noise.

The results of experiments with $\delta_{\max} \in [1, 2, 3]$ are shown in Table 5.1 and the estimates of the probability of failure are plotted in Figure 5.2. The failure rate of all noisy proposal distributions stays at 1 but the log-likelihood begins to degrade significantly with $\delta_{\max} = 3$. The estimate of the probability of failure is accurate and relatively low-variance for $\delta_{\max} = 1$ and $\delta_{\max} = 2$. For $\delta_{\max} = 3$, the estimate is ultimately low, likely due to the prevalence of low log-likelihood failure trajectories. Although only a toy example, these experiments show that a proposal distribution based on $P_{\text{fail}}(s)$ may be rather robust to estimation errors.

Despite the robustness to errors in the estimation of the probability of failure, it is beneficial to construct the best estimate possible. We now investigate the use of DQN as an approximate technique for estimating the probability of failure and compare it to the ground truth.

We train a neural network that has two hidden layers (64 and 32 hidden units

Table 5.1: Performance with noisy estimates of the probability of failure.

Noise Level	**Failure Rate**	**Log-Likelihood**
No Noise	1.0 ±0.0	−10.894 ±0.49
$\delta_{\max} = 1$	1.0 ±0.0	−10.493 ±0.421
$\delta_{\max} = 2$	1.0 ±0.0	−12.254 ±0.551
$\delta_{\max} = 3$	1.0 ±0.0	−181.049 ±22.592

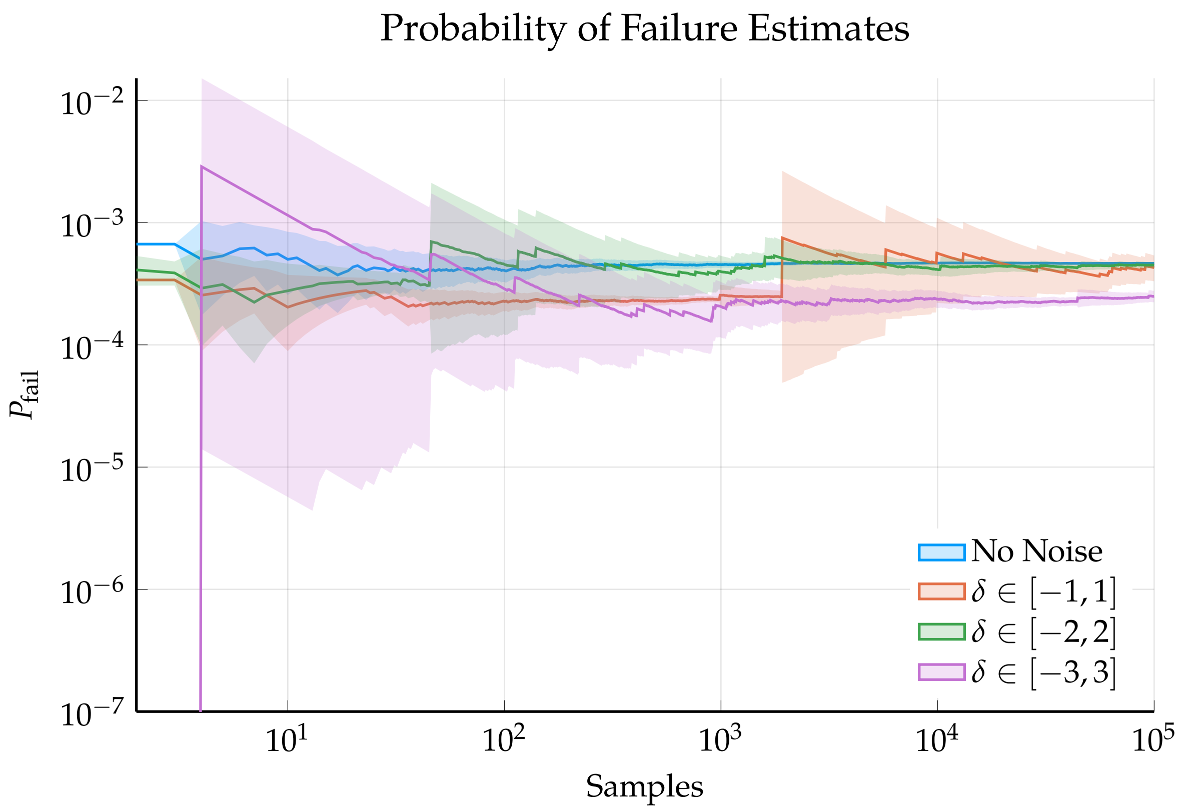


Figure 5.2: P_{fail} estimation with varying degrees of noise.

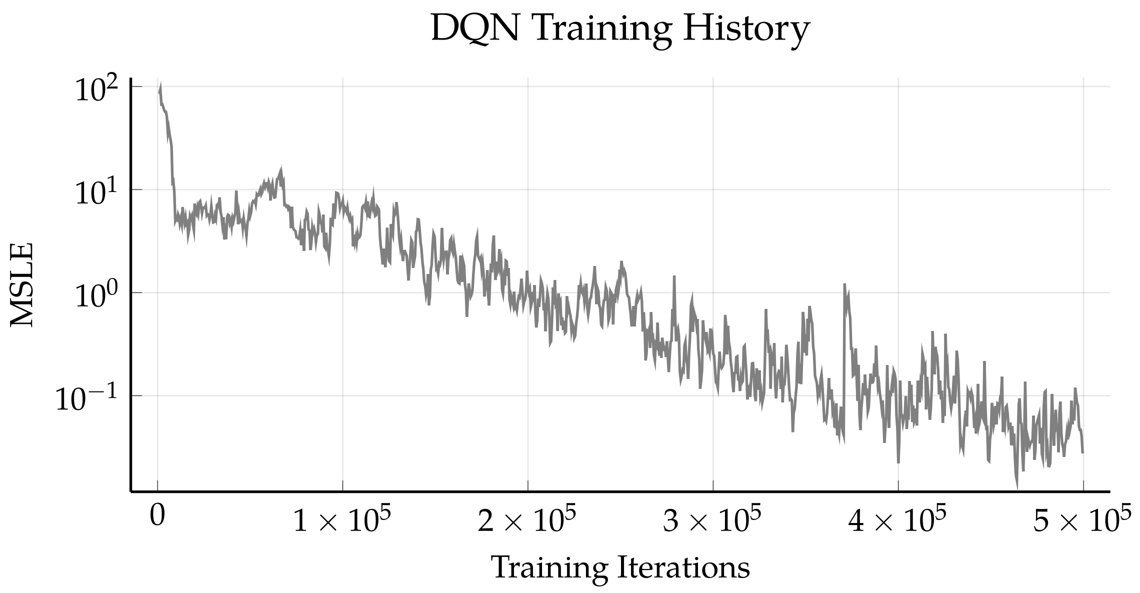
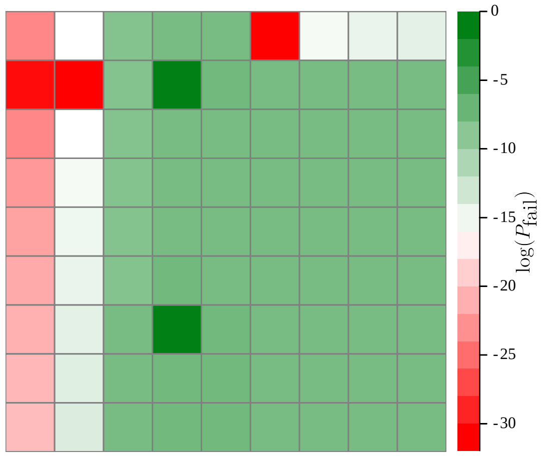


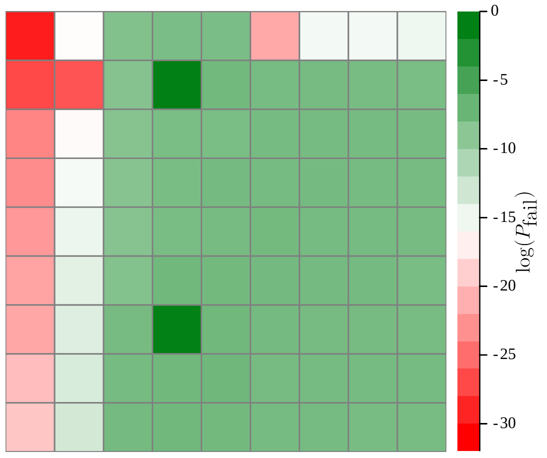
Figure 5.3: MSLE between neural network and ground truth during training.

and relu activations) and a sigmoid activation for each output node. The network was trained over 5×10^5 iterations with the ADAM optimizer and a learning rate of 1×10^{-3}. The neural network represents the function $P_{\text{fail}}(s, x)$ so to compare it to the ground truth from exact value iteration we must compute $P_{\text{fail}}(s) = \sum_x p(x \mid s) P_{\text{fail}}(s, x)$. To evaluate the quality of the neural network approximation, we computed the MSLE between the estimate and the ground truth, averaged over all of the states in the gridworld (shown in Figure 5.3). From the MSLE, we see that the neural network is able to recover $P_{\text{fail}}(s)$ with remarkably low relative error. To visualize the similarity, we plot the log of the probability of failure at each state in the gridworld in Figure 5.4. The neural network recreates the probability of failure accurately over many orders of magnitude.

We now compare the proposal distribution $q^*(x \mid s)$ (with $P_{\text{fail}}(s)$ computed through exact value iteration and DQN) to the Monte Carlo and cross entropy method baselines. The failure rate and log-likelihood of failures is reported in Table 5.2. The cross entropy method is able to significantly increase the rate of



(a) P_{fail} ground truth.



(b) P_{fail} from DQN.

Figure 5.4: Comparison of P_{fail} between DQN and exact value iteration.

discovered failures, but the average log-likelihood of those failures is much smaller than the most commonly observed failures. Using q^* with both DQN estimation and exact value iteration maximizes the rate of failures discovered while retaining a relatively high likelihood of failure trajectories.

Figure 5.5 shows how each technique performs when used to estimate the probability of failure. Monte Carlo sampling ultimately gives an accurate estimate but does not observe a failure for the first 2,000 samples. The cross entropy method observes samples early on, but due to their low log-probability they are give small importance sampling weights and therefore create an erroneous estimate. The DQN and value iteration approaches both quickly converge to a good estimate of the probability of failure with value iteration performing slightly better. The reason the value iteration estimate has any variance at all is because the agent is initialized to a random place on the grid for each sample.

5.4.2 T-Intersection Scenario

We now turn to the more complex problem of discovering failures in the autonomous driving policy of the ego vehicle in the T-intersection scenario. The problem is more challenging due to the longer time horizon and the continuous state space. In

Table 5.2: Simple gridworld failure results.

Method	**Failure Rate**	**Log Likelihood**
Monte Carlo	0.001 ±0.001	−8.013 ±0.0
Cross entropy method	0.255 ±0.014	−36.431 ±2.050
DQN	0.980 ±0.004	−10.654 ±0.510
Value iteration	1.0 ±0.0	−11.375 ±0.604

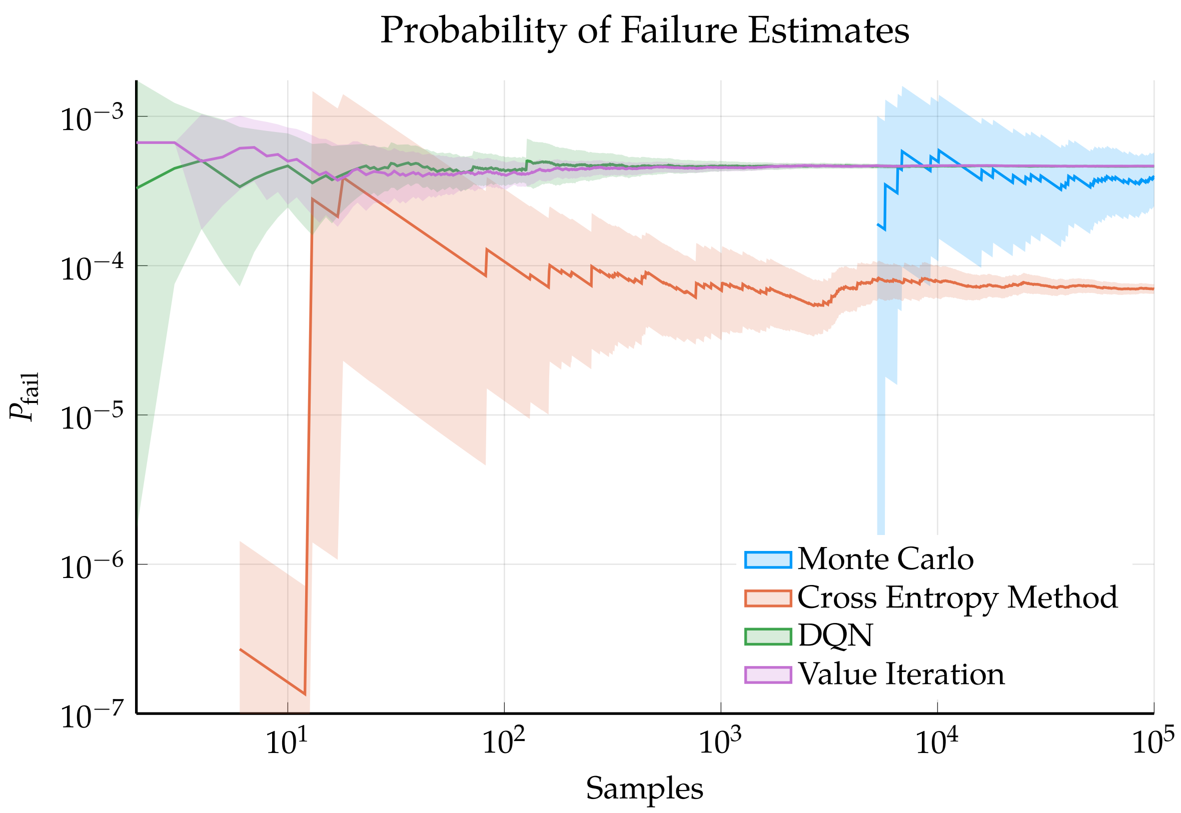


Figure 5.5: Comparison of techniques for P_{fail} estimation for the simple gridworld.

each episode, the agents are initialized with random positions in the range [5 m, 35 m] (with respect to the start of their lane) and random velocities in the range [10 m/s, 20 m/s]. The adversarial agent is initialized on the left with a random turn intention and match turn signal. Any initial condition that had a collision without any disturbance was not used in the analysis.

In addition to all the baselines and DQN, we use local approximation value iteration to estimate the probability of failure. We discretize the state space into 30 position points and 10 velocity points for each vehicle, resulting in a total of 360,000 states. To focus the sample points on the most interesting regions of the state space (i.e. the positions before and immediately after the intersection and the reachable range of velocities), we uniformly discretize the adversary position between [0 m, 75 m] an the ego vehicle between [15 m, 75 m], and the velocity between [0 m/s, 20 m/s]. Interpolation was done using bilinear interpolation over the grid of states.

The failure rates and average failure log-likelihood is shown in Table 5.3. The uniform sampling distribution is able to increase the failure rate but at the cost of low-likelihood failures. The cross entropy method does not perform much better than basic Monte Carlo sampling. Both techniques that use q^* have significantly higher failure rates and relatively high log-likelihoods. When $P_{\text{fail}}(s)$ is estimated using DQN, we get a higher failure rate and log-likelihood compared to local approximation value iteration. Likewise, in the estimate of the probability of failure (Figure 5.6) the DQN approach seems to have less variance than local approximation value iteration.

Local approximation value iteration may perform worse than DQN due to the requirement of such a coarse discretization of the state space. Small features in the landscape of $P_{\text{fail}}(s)$ that are captured by a neural network may be missed using a coarse grid and linear interpolation, leading to a higher variance proposal distribution.

Table 5.3: Two car T-intersection results.

Method	**Failure Rate**	**Log Likelihood**
Monte Carlo	0.0 ±0.0	−9.352 ±0.372
Uniform sampling	0.087 ±0.009	−97.703 ±2.822
Cross entropy method	0.007 ±0.003	−10.416 ±0.313
DQN	0.365 ±0.015	−9.562 ±0.16
Local Approximation VI	0.214 ±0.013	−11.45 ±0.40

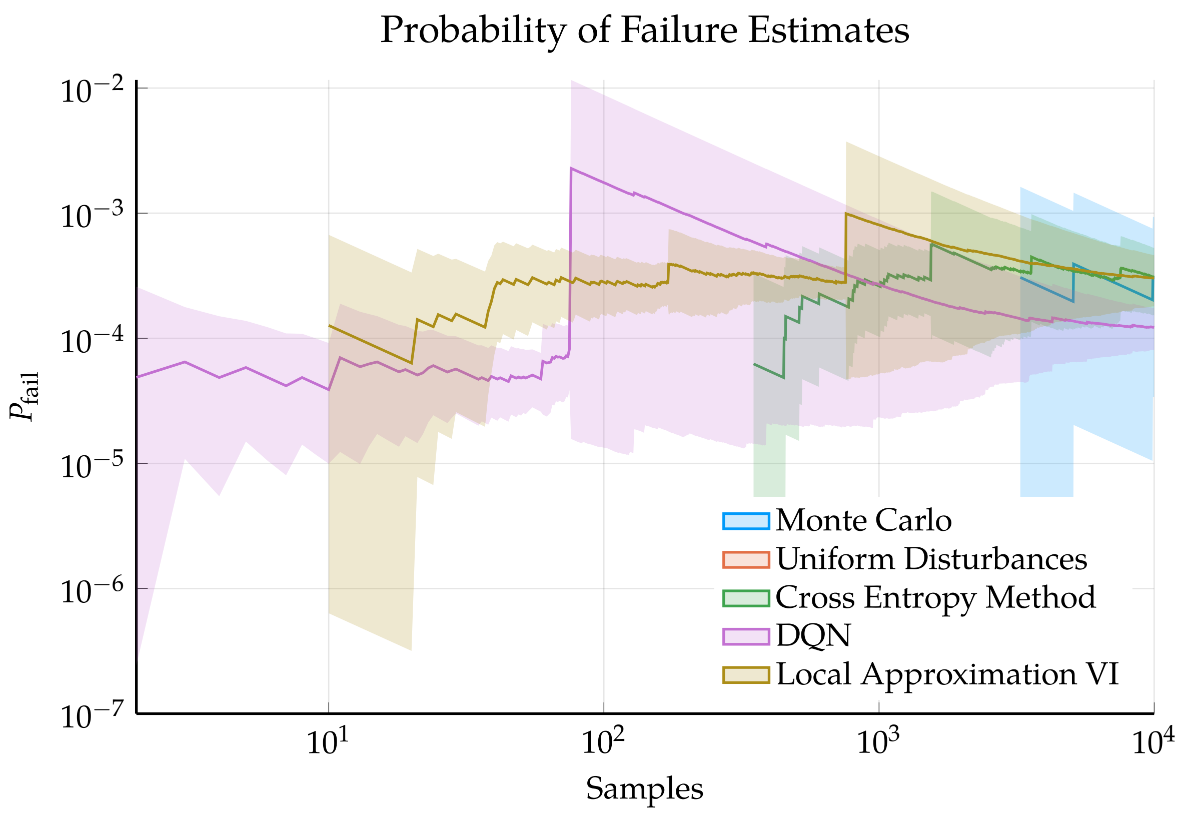


Figure 5.6: Comparison of techniques for P_{fail} estimation for the 2-car T-intersection scenario.

5.5 Discussion

In this chapter, we argue that safety validation algorithms that optimize to find individual failure trajectories are insufficient for obtaining a diverse set of likely failures and for estimating the failure probability. One solution is to approximate the distribution over failures and use it as a stochastic policy for choosing disturbances. We showed that a distribution that is proportional to the probability of failure can be an efficient proposal distribution, and is optimal when applied to a deterministic environment. This insight reduces the problem of approximating the distribution over failure to estimating the probability of failure for each state and disturbance pair.

When we invoke the Markov assumption, we show that the probability of failure is the solution to a Bellman equation which can be solved exactly or approximately depending on the size of the state and disturbance space. We showed that a modified version of DQN can also approximate the probability of failure even when those probabilities are extremely small. Through experiments with the gridworld and T-intersection scenarios, we demonstrate that our proposal distribution is able to find more failures, find failures with high log-likelihood, and better estimate the probability of failure than baseline techniques.

The work presented in this chapter is related to several other approaches. Adaptive stress testing (AST) [29], [30], [92], [93] also frames the safety validation problem as a Markov decision process and uses reinforcement learning to find likely failures. AST optimizes for the most-likely failure and has been shown converge to individual failure modes [93]. The field of statistical model checking [126] deals with estimating the probability of failure of a system but relies on basic sampling techniques with an emphasis on hypothesis testing for making probabilistic safety claims.

There is a large amount of literature where importance sampling has been applied to estimate the probability of failure (some of it is covered in Section 2.4.4). Most relevant to this chapter is the work by Chryssanthacopoulos, Kochenderfer, *et al.* [127], where they apply a similar state-based importance sampling distribution to estimate the probability of failure of an aircraft collision avoidance system. The probability of failure is roughly estimated using local approximation value iteration and used to construct an importance sampling distribution. They do not, however, analyze the variance of the estimator or propose scalable techniques for estimating the probability of failure. Uesato, Kumar, *et al.* [104] also use an estimate of the probability of failure to choose initial conditions that cause an agent to failure. They do not extend their work to sequential decision making, but we note that they derived the optimal sampling distribution for a single step stochastic simulator to be proportional to the $\sqrt{P_{\text{fail}}}$. This insight does not, however, appear to be directly applicable to deriving the optimal state-dependent sampling distribution for sequential problems.

One of the biggest challenges to applying this technique to large problems is the scalability to larger state and disturbance spaces. Value iteration approaches explicitly iterate over states, while the quality of a neural network estimate will depend on having experience samples that sufficiently cover the state space. In the next chapter we introduce a scene decomposition technique that can improve the scalability of this technique.

6 Scalability Through Problem Decomposition

In this chapter we introduce a technique for improving the scalability of safety validation algorithms for multi-agent systems. We first describe the problem of exponential growth of states and disturbances with respect to the number of adversarial entities. We then show how multi-agent safety validation problems can be decomposed into subproblems each involving the system interacting with a subset of the adversaries. The solution to these subproblems can be recombined using an attention network and a base network to correct for multi-agent interactions. We demonstrate the efficacy of scene decomposition by approximating the distribution over failures of a multi-agent gridworld and T-intersection scenario.

6.1 Motivation

Many of the most exciting use cases for autonomy require the system to interact with large number of other human and non-human agents. An autonomous driving policy must reason about a potentially large number other vehicles, pedestrians, cyclists, and traffic control systems, each with a wide variety of possible behaviors. Similarly, autonomous aircraft will soon have to navigate and avoid collisions in an increasingly dense airspace.

Many of the challenges for safety validation are exacerbated when the environment contains multiple interacting agents. The dimensionality of both the disturbance and state space grows with each new agent in the environment. The search over disturbance trajectories, state-dependent policies, or the state space itself can quickly become intractable. Failures may be more or less likely with more agents, but the fraction of the disturbance trajectory space that leads to a failure will likely decrease due to the dramatic increase in the size of the disturbance trajectory space.

As an example, consider the 5×5 adversarial gridworld problem depicted in Figure 6.1a where an ego agent (blue) tries to avoid two adversarial agents (orange and red). Each agent has a 2D position and four possible actions so the state space and disturbance space both grow exponentially with the number of agents. In this case, a failure occurs when all agents overlap on the same state, so failures become less likely with the number agents. Value iteration is tractable on this problem with two adversaries (which is why we include it as a test case), but becomes challenging with 3 agents and intractable with more.

The curse of dimensionality is a well known problem in multi-agent systems and is often solved by decomposing the problem into simpler problems that are tractable and then recombining the solutions. In this chapter we develop a technique for decomposing and recombining safety validation problems. The basic idea is to identify a natural decomposition of the state space that reduces the full problem into a set of subproblems that can be solved individually. For example, the gridworld with two adversaries can be decomposed into two gridworlds, each with a single adversary (as shown in Figures 6.1b and 6.1c). Then, to solve the full problem we learn how to combine the subproblem solutions instead of solving the problem from scratch. The solution to the full problem may not be a simple combination of the subproblem solutions so we simultaneously learn a correction factor.

For this procedure to be successful, however, we need the subproblem solutions to confer some benefit to the solution of the full problem. In the case of the gridworld example, the subproblem solutions will suggest that it is beneficial to move toward the ego to cause failure. This insight will likely aid the adversary when solving the full problem.

In the next section we formalize the decomposition and highlight some potential efficiencies. We then demonstrate that combining the subproblems can be cast as a transfer learning problem and we apply a known transfer learning algorithm to

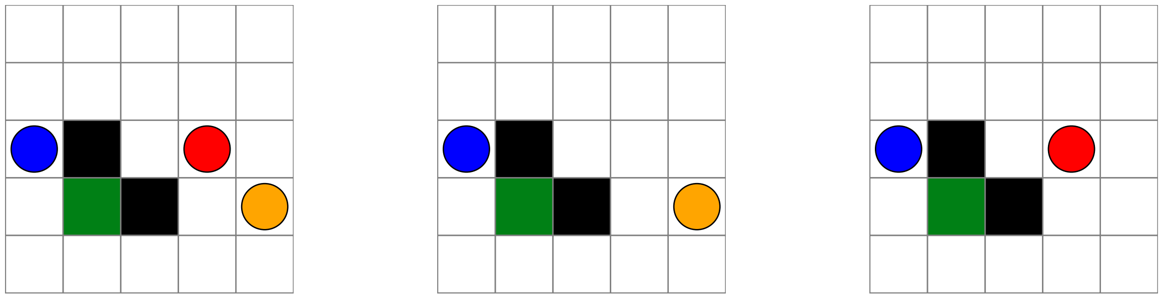

(a) Full problem. (b) Subproblem 1 (c) Subproblem 2.

Figure 6.1: Example decomposition of the gridworld with two adversaries.

solve it.

6.2 Decomposition and Fusion

In this section we describe the process of problem decomposition and identify situations where subproblems are identical. We then introduce the attend, adapt, and transfer algorithm that automatically learns how to combine the subproblem solutions while learning a correction factor.

6.2.1 Problem Decomposition

The goal of state-space decomposition is to identify a set of subspaces that each represent a similar, but smaller, version of the full problem. We denote the state space of the ith subproblem $S^{(i)}$ and the disturbance space $X^{(i)}$. The state $s^{(i)} \in S^{(i)}$ contains the components of s that are associate with the ith subproblem and similarly for the disturbance $x^{(i)}$. A crucial constraint on the choice of decomposition is the ability to simulate transitions of the subproblems

$$s_{t+1}^{(i)} \sim P(s^{(i)} \mid s_t^{(i)}, x_t^{(i)}). \tag{6.1}$$

In the gridworld example, if we choose a decomposition where the state space consists only of the horizontal component of each agent, it is not clear how we would choose a transition model that is faithful to the original problem. Instead, if we choose a decomposition where the state space is the position of the ego and one adversary, and the disturbance space is the disturbance space of that adversary then we have reduced the problem a single-adversary gridworld which we know how to simulate. Indeed, in multi-agent problems a natural decomposition usually consists of the state spaces from a subset of of the agents.

In some cases, multiple subproblems will reduce to the same problem and only one solution will be required. For example, suppose we wish to solve for the probability of failure at each state in a gridworld with N adversarial agents. If we decompose it into N gridworlds (each with the ego and one adversary), then each subproblem is identical (assuming the ego agent cannot distinguish adversaries). Noticing these overlaps can save a significant amount of computational effort when computing subproblem solutions. The next section describes how to combine subproblem solutions once they are computed.

6.2.2 Solution Fusion

We now assume that we have decomposed the safety validation problem into M subproblems and obtained a solution K for each. We assume that solution takes the form of a policy or a value function that solves the associated subproblem. For falsification and most-likely failure analysis tasks, K should produce the lowest cost failure trajectories. When approximating the distribution over failures K may estimate the probability of failure at each state or be the distribution over failures of the subproblem. Then, the goal is to use the M subproblems to learn the solution to the full problem

$$K_{\text{full}} = \mathcal{L}(K_1, \dots, K_M) \tag{6.2}$$

with some learning algorithm $\mathcal{L}$. Fortunately, this problem formulation is the same as transfer learning, which has many existing solution techniques [128].

In safety validation, the full problem may contain failure modes that arise from the complex interaction of multiple agents and are therefore not present in the

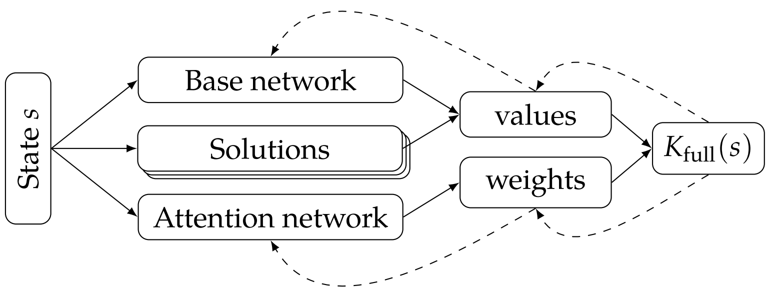


Figure 6.2: The A2T network.

subproblems. The solutions to the subproblems may not contain the needed information for finding failures in the full problem and may even make it more difficult to find failures (an effect known as negative transfer). From these considerations we selected the attend, adapt, and transfer algorithm (A2T) [129]. A2T combines solutions using a learned set of attention weights that can minimize negative transfer, and simultaneously learns a new solution from scratch to account for the complexities of the full problem.

With A2T, the full solution at state s is given by

$$K_{\text{full}}(s) = w_0(s)K_{\text{base}}(s) + \sum_{i=1}^{M} w_i(s)K_i(s^{(i)}) \tag{6.3}$$

where $w(s)$ is a state-dependent attention weight and K_{base} is the solution learned from scratch. Equation (6.3) can be encoded as the network architecture shown in Figure 6.2. The state is passed to the base network, the set of subproblem solutions, and an attention network. The base network outputs $K_{\text{base}}(s)$ while the subproblem solutions output $K_i(s^{(i)}$. The attention network outputs the attention weights $w_i(s)$ which are combined with K_{base} and K_i to get the full solution $K_{\text{full}}(s)$. The parameters in the base network and the attention network can be optimized through backpropagation (as indicated by the dashed lines), using any typical learning algorithm such as DQN or policy gradient methods.

6.3 Experiments

In this section we perform experiments to investigate the benefit of problem decomposition for approximating the distribution over failures. As discussed in Chapter 5, to approximate the distribution over failures we need to estimate the probability of failure at each state. The solutions $K_i(s)$ will represent the probability of failure for each disturbance from state s.

In the first experiment we use the gridworld scenario with two adversaries because it can be solved with exact value iteration. Having the ground truth allows us to compare estimates of the probability of failure with and without the subproblem solutions. In the second experiment we solve for the distribution over failures in the T-intersection with 4 adversaries and show it outperforms our baseline approaches. Lastly, we demonstrate that the attention network may also lend a degree of interpretability to the safety validation results.

6.3.1 Adversarial Gridworld Example

The goal of this experiment is to compare the estimates of the probability of failure between a basic neural network and an A2T network that uses estimates of the probability of failure from decomposed subproblems. To make the comparison, we use value iteration to obtain the ground truth $P_{\text{fail}}(s)$ of a 5×5 gridworld with 2 adversaries and an expert ego policy.

As previously described, the gridworld can be decomposed into two identical gridworlds each with a single adversary (Figure 6.1). We solve for the probability of failure for each state and disturbance of the single-adversary problem $P_{\text{fail}}^{\text{sub}}(s^{(i)}, x^{(i)})$ using exact value iteration. The A2T estimate of the probability of failure is then given by

$$\begin{aligned} P_{\text{fail}}^{\text{A2T}}(s, x) = w_0(s) P_{\text{fail}}^{\text{base}}(s, x) + \sum_{i=1}^{2} w_i(s) P_{\text{fail}}^{\text{sub}}(s^{(i)}, x^{(i)}) \\ + w_3(s) P_{\text{fail}}^{\text{sub}}(s^{(1)}, x^{(1)}) P_{\text{fail}}^{\text{sub}}(s^{(2)}, x^{(2)}) \end{aligned} \tag{6.4}$$

where we included the last term to help model the conjunctive nature of the failure mode of the full problem.

The attention network has a single hidden layer with 32 units and has a softmax layer at the output to normalize the weights. The base network has two hidden layers with 64 and 32 units and relu activations. The base network has sigmoid activations on the last layer to ensure the output is bounded in $(0, 1)$. The A2T network is compared against a neural network with same architecture as the base network. Both networks were trained using our version of DQN for P_{fail} estimation (see Chapter 5). We trained for 1×10^5 steps with the ADAM optimizer and a learning rate of 1×10^{-3}.

During training we periodically computed the mean square log error between the estimate and the ground truth and plotted the results in Figure 6.3. We first notice that the A2T network reaches its lowest error more quickly than the basic network, but more importantly, remains close to this error for the rest of training, likely because it is near a local minimum. The basic network initially reduces in error but eventually goes unstable and does not converge to a minimum. The use of subproblem solutions speeds up the training process and appears to improve the training stability as well. We now perform some experiments on the T-intersection problem to more fully demonstrate the problem decomposition technique.

6.3.2 T-Intersection Scenario

We now solve for the failure distribution of the T-intersection autonomous driving scenario with five interacting agents. If we were to discretize the state space of the full problem (using the same discretization as in Section 5.4.2) we would end up with over 1×10^{14} states and 2,401 actions. Instead we apply problem decomposition to construct subproblems that each consist of one of the adversaries and the ego vehicle. The result is two unique subproblems, one with the adversary coming from the left, and one with adversary coming from the right. We solve both of those problems using DQN and the same architecture described in Section 5.4.2. We combine the solutions with an A2T architecture with a base and attention network

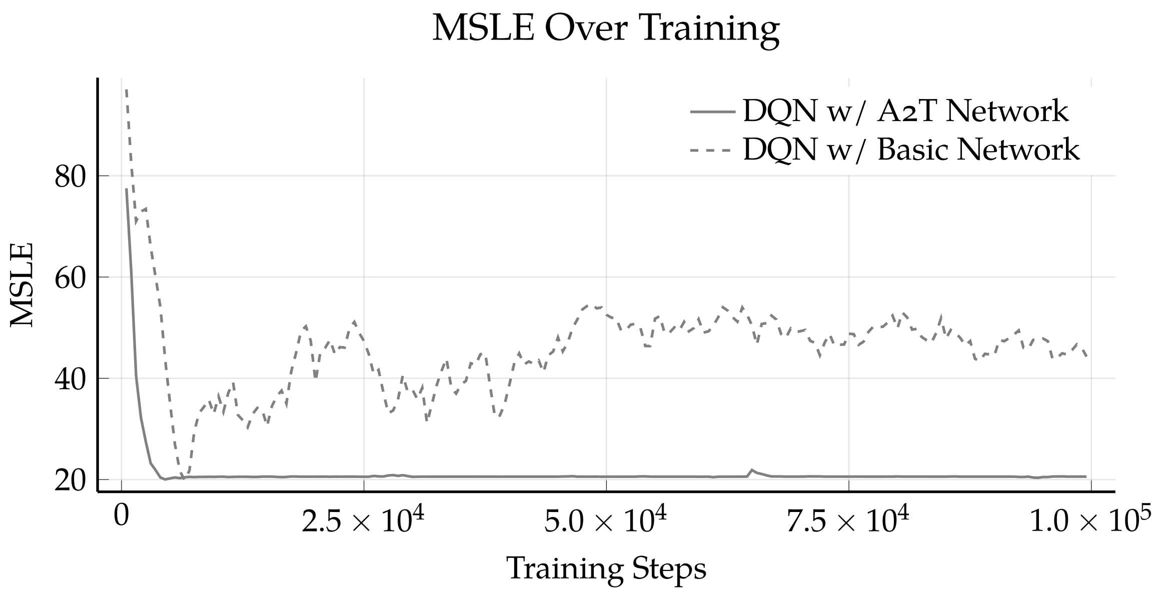


Figure 6.3: Training comparison between A2T and basic network.

each with two hidden layers of 128 and 32 units and relu activations. To handle the explosion in actions, we make the simplifying assumption that only one adversary can have a disturbance at a time which reduces the action space to 25 actions.

The A2T network to approximate the probability of failure was trained over 3×10^5 steps with a learning rate of 1×10^{-3}. The resulting proposal distribution was compared to the same baselines as Section 5.4. For each episode the initial condition of each adversary was sampled with the position in $[0\,\mathrm{m}, 45\,\mathrm{m}]$ and the velocity in $[5\,\mathrm{m}, 20\,\mathrm{m/s}]$. The ego vehicle was initialized with a position in $[10\,\mathrm{m}, 35\,\mathrm{m}]$ and velocity in $[10\,\mathrm{m}, 20\,\mathrm{m/s}]$. Any initial condition that had a collision without any disturbance (due to overlapping initialization for example) was skipped in the analysis. Additionally when computing the probability of failure we removed samples that had too large of an importance sample weight to reduce the variance of the estimates.

The failure rates on the log-likelihood of the failures are reported in Table 6.1 and the probability of failure is plotted in Figure 6.4. First, we note that the probability of failure was sufficiently low that we never observed a failure event for Monte Carlo sampling in the allotted trials. Uniform sampling of the disturbances found a similar number of failures as the cross entropy method but had significantly lower log-likelihood. As such uniform sampling dramatically underestimates the probability of samples and is below the axis of the plot. The cross entropy method was an improvement but still had a low prediction until many failures were observed. The estimate of the probability of failure using A2T performed the best across all categories and produced a good estimate of the probability of failure from only a few samples.

Two example failures are shown in Figures 6.5 and 6.6 where the blue circles indicated the associated attention weight. In the first collision, the ego vehicle calculates that it can make a turn in front of the leading vehicle on the right. That vehicle, however, speeds up unexpectedly and causes a collision. During the first half of the trajectory, the adversary is computing the probability of failure based on both agents on the right, likely meaning that both agents have a possibility of causing a collision. By the time collision is imminent, the adversary has focused on the vehicle that causes the collision. The attention weights might be some indication of which adversaries are most likely to cause a collision. The second collision starts out similar to thee first, but by timestep 8 the attention network is focused on the trailing left adversary which is the vehicle ultimately responsible for the collision. When the collision is imminent, the attention goes back to a vehicle on the right. It may be the case that the vehicle on the right could also cause a collision so the network has conflated the two estimates.

Finding failures in a complex driving scenario with multiple interacting agents is challenging due to the large state and action spaces. We have successfully applied problem decomposition and recombination with A2T to discover the most likely failure modes estimate the probability of failure.

Table 6.1: 5-car results.

Method	**Failure Rate**	**Log Likelihood**
Monte Carlo	0.0 ±0.0	-
Uniform sampling	0.031 ±0.005	−173.01 ±42.93
Cross entropy method	0.035 ±0.006	−82.672 ±7.342
DQN + A2T	0.081 ±0.009	−19.938 ±7.317

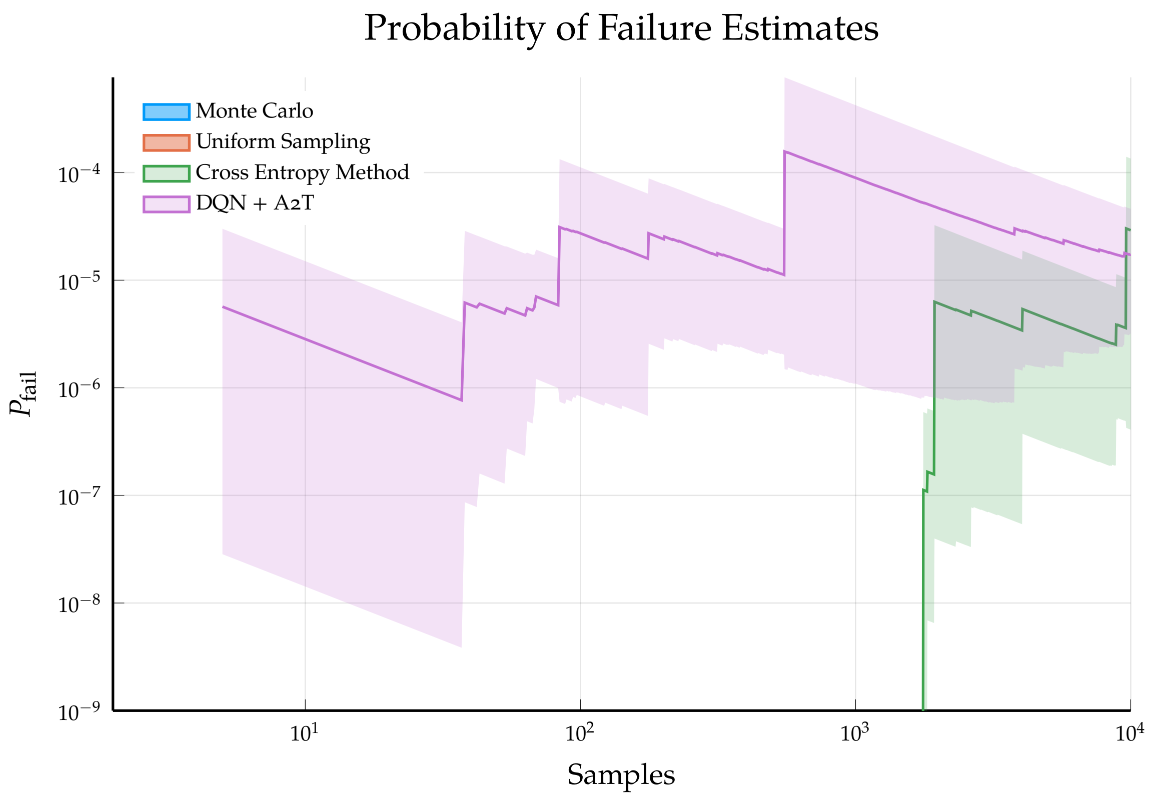


Figure 6.4: Comparison of techniques for P_{fail} estimation for the 5-car T-intersection scenario.

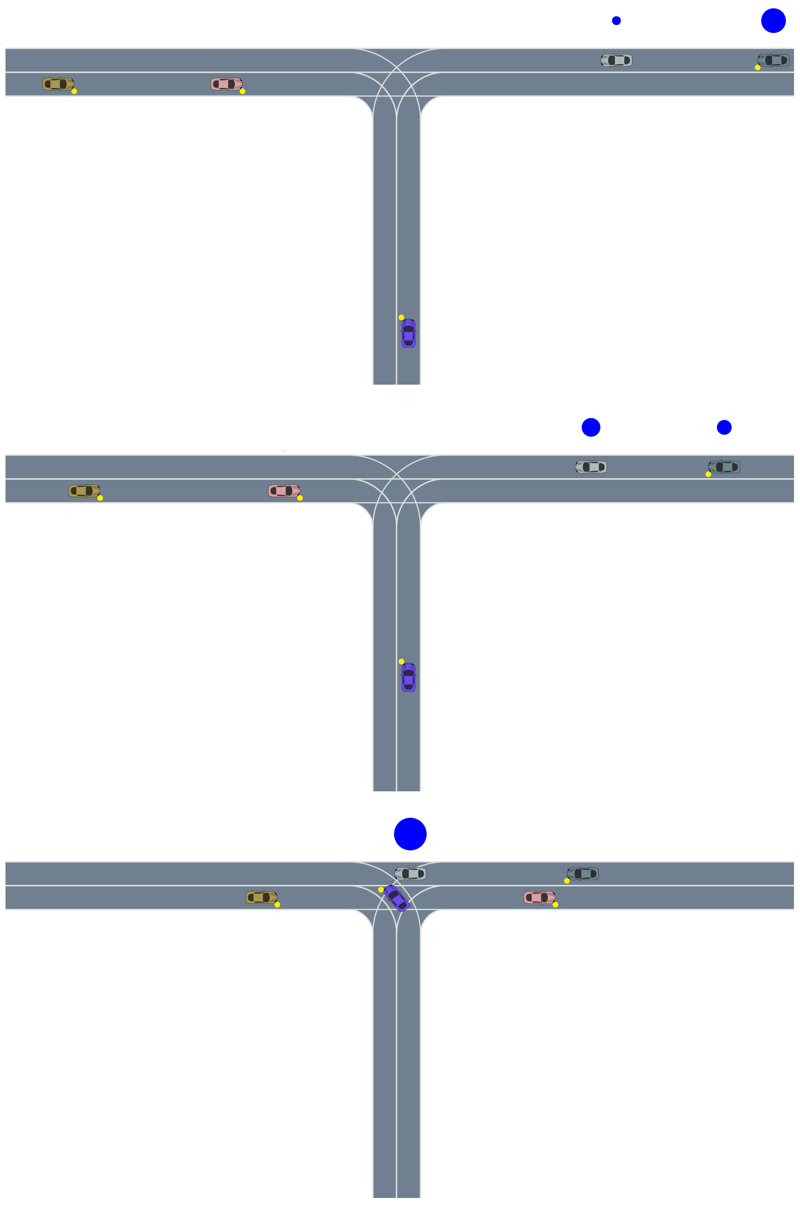


Figure 6.5: Collision in 5-car scenario at timesteps $t = [1, 4, 18]$.

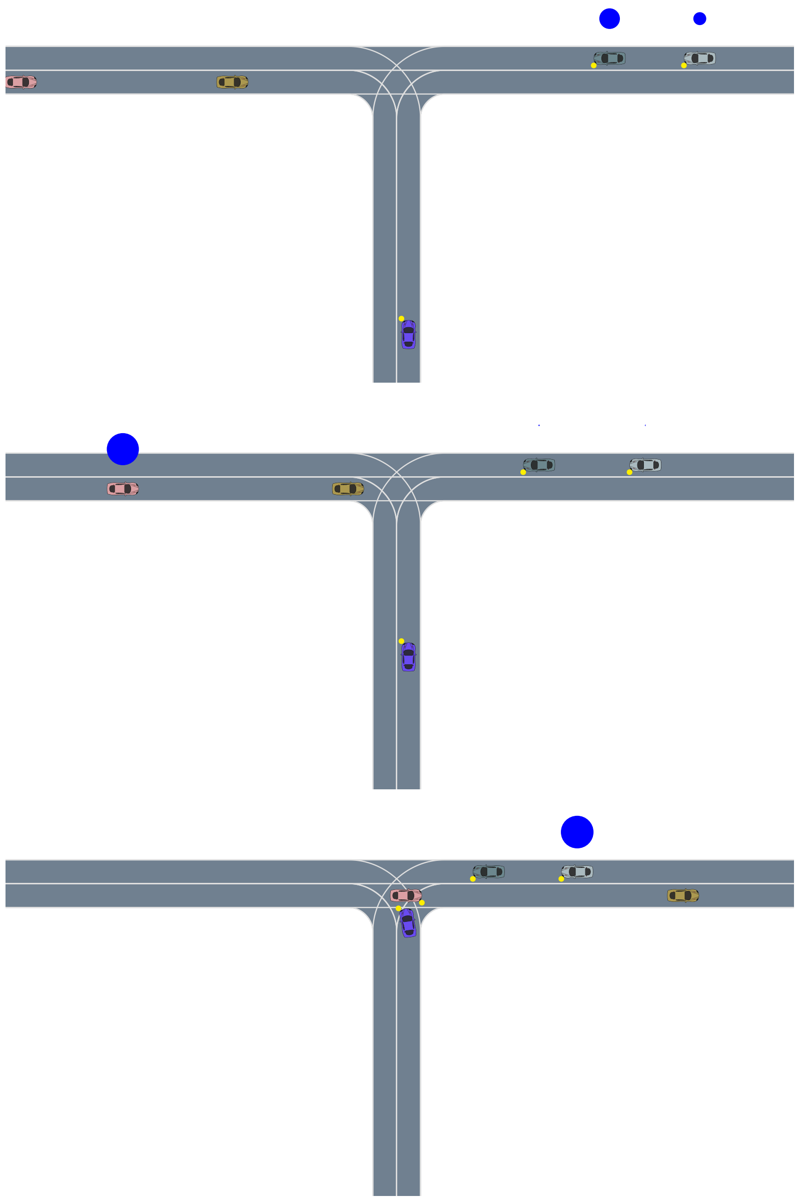


Figure 6.6: Collision in 5-car scenario at timesteps $t = [1, 8, 27]$.

6.4 Discussion

Multi-agent safety validation problems pose a challenge for safety validation algorithms due to the large state and actions paces. In this chapter we made a step toward the scalability of safety validation algorithms to multi-agent settings by applying problem decomposition. We discussed a technique for decomposing a multi-agent system into tractable subproblems and then combining the solutions using the attend, adapt and transfer architecture.

Using the ground truth of simple multi-agent gridworld problem, we showed that estimates of the probability of failure of the 3-agent system were improved by referencing the probability of failure of the 2-agent system. We then applied the decomposition approach to the T-intersection scenario with 4 adversaries. By decomposing the problem into two unique subproblems and recombining their solutions, we were able to approximate the distribution over failures and outperform baseline approaches.

Decomposition of the Q-function has been effective for developing reinforcement learning agents. The technique proposed by [90] creates a learning agent composed of individual subagents with different reward functions. Each subagent learns a Q function based on its reward and provides this information to a centralized arbiter. The arbiter fuses the utility estimates together, and selects the action that maximizes the global utility (a similar approach was used by [130]). The Q-decomposition approach was used by Bouton, Julian, *et al.* [131] to learn a safe driving policy. In that work, they perform a similar pairwise decomposition of driving agents. Each subproblem was solved for the optimal value function and the results were fused with simple aggregation by taking the max or mean. To account for multi-agent interactions, they learned an additive correction factor, parameterized by a deep neural network. Utility decomposition has also been successful for developing collision avoidance strategies with multiple agents [132].

The main distinction between Q-decomposition and our approach is that the joint problem has an action space that is the joint action space of the subproblems. When combining subproblem solutions, we need to select which agents act as well as what actions they choose, while the approach of Sutton, McAllester, *et al.* [90] and Bouton, Karlsson, *et al.* [10] fuses a set of proposed actions into a single action for the joint agent. Additionally, the A2T approach is more flexible because the combination of subproblem solutions is learned and state dependent.

In the next chapter we turn away from solving individual safety validation problems and consider the problem of iterative safety validation. We demonstrate that we can gain significant performance and efficiency improvement by storing and reusing safety validation knowledge across systems.

7 Iterative Safety Validation

In this chapter, we consider the problem of validating many related systems sequentially. Most safety validation algorithms must start from scratch each time the system under test changes, but we demonstrate performance and efficiency benefits to sharing knowledge between safety validation tasks. We apply transfer learning algorithms to share knowledge between related tasks and perform experiments using the gridworld and crosswalk scenarios.

7.1 Motivation

Designing and certifying safety-critical autonomy generally involves assessing a sequence of closely related systems. Despite the similarity between systems, however, existing safety validation approaches generally start from scratch for each system. Since validation can be computationally expensive, repeated safety validation imposes a large computational burden, but also an opportunity to improve efficiency.

To improve the efficiency of safety validation across related systems, we use knowledge from the validation of previous systems to inform the validation of the next system. We formulate iterative safety validation as a transfer learning problem by modeling each safety validation task as a Markov decision process. The previously solved safety validation problems are used as the set of source tasks, and we transfer knowledge to future tasks in the form of action value functions. We use a set of state-dependent attention weights to automatically learn which previous solutions are applicable to the current problem.

We propose a new variation of the attend, adapt, and transfer (A2T) algorithm [129] that transforms the state and action value spaces for each source task to increase knowledge transfer between dissimilar tasks and compare it to basic A2T and a fine-tuning baseline. We evaluate the initial performance, the final performance, and the number of training steps required to reach the same performance as a no-transfer algorithm. We consider four iterative safety validation tasks in grid-world and autonomous driving scenarios and demonstrate that transfer learning has the potential to significantly improve the performance and sample efficiency of safety validation algorithms.

7.2 *Transfer Learning*

Transfer learning is concerned with using knowledge gained by solving one task to improve the learning process in another related task [128]. In reinforcement learning, each task is an MDP and each solution is a policy. When tasks have different state and action spaces, we require task mappings that relate the states and actions between tasks. Task mappings may be provided by a human [133] or learned from data [134]. If the state and action spaces are the same, then tasks can share a variety of low-level information such as experience samples (s, a, s', r), action value functions, policies, or models of the environment. High-level information, such as a set of options, shaping rewards or feature encodings, may also be transferred to improve learning on a new task. One challenge in transfer learning is *negative transfer* where knowledge from one or more source tasks impairs the performance on the current task. Negative transfer can be mitigated using an attention mechanism, or a human oracle that decides which tasks are relevant [128].

Transfer learning algorithms can be evaluated against a no-transfer alternative in a variety of ways (Figure 7.1). *Jumpstart* is the amount of improvement before any training has occurred, *final performance* is the difference between the best performances achieved, *total reward* is the area under the training curve, and *steps to threshold* is the difference between the number of training steps required to reach a specified threshold.

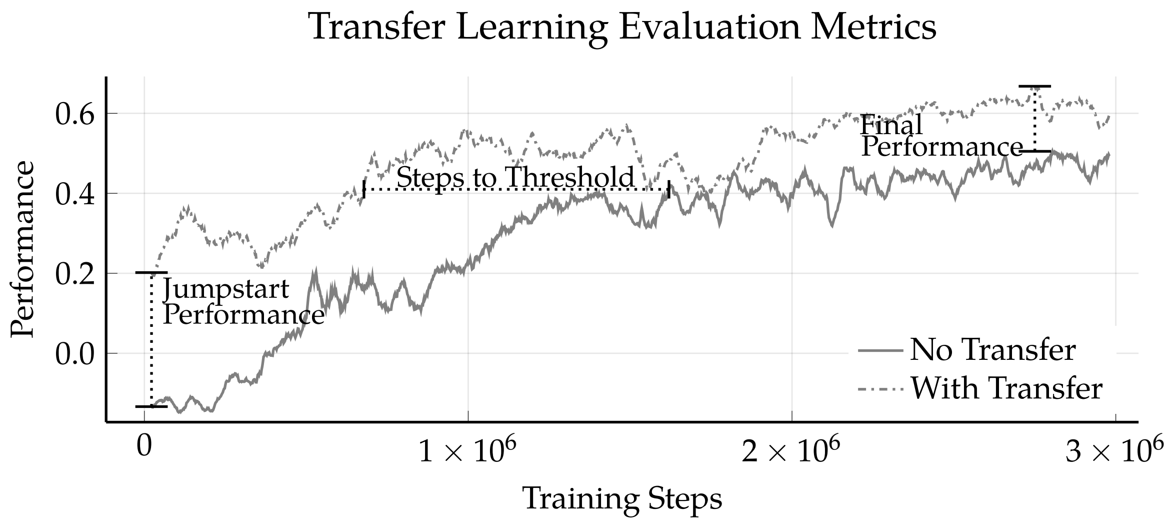


Figure 7.1: Metrics for evaluating transfer learning algorithms.

In the domain of safety validation, tasks may differ in a variety of ways, which we illustrate with an autonomous driving example. If validation is performed on two different road geometries (e.g. highway-driving and an intersection), then the state space and disturbance space may be different. If we fix the road geometry, then the transition model may vary if we validate different driving policies. The reward function will vary if the disturbance model changes or we alter the set of failure states. In this work, we consider systems with different behavior operating in similar environments. We, therefore, assume that the state space, disturbance space, and reward function remain fixed while the transition model varies between tasks.

7.3 *Proposed Approach*

This section describes our approach for improving iterative safety validation using transfer learning. We first show how to formulate iterative safety validation as a sequence of tasks and specify two ways that knowledge transfer may occur. We then introduce A2T as our choice of transfer learning algorithm and propose a modification to it.

7.3.1 Problem Formulation

Suppose we are performing safety validation on a sequence of related systems and must validate each system before observing the next one. We model this problem as solving a sequence of MDPs (or tasks) $[T_1, T_2, \ldots]$ where the ith task is given by $T_i = (\mathcal{S}, \mathcal{X}, P_i, R, \gamma)$. Due to variations in system behavior, the transition model P_i is unique to each task, while $\mathcal{S}$, $\mathcal{X}$, R, and γ are shared across tasks. We wish to develop a learning algorithm L that solves task i given the $i-1$ previous solutions $[K_1, K_2, \ldots, K_{i-1}]$ such that

$$K_i = L(T_i; K_{1:i-1}). \tag{7.1}$$

The previous solutions may take the form of value functions or policies. The learning procedure is iterative because the new solution K_i can be added to the set of previous solutions when solving the next task T_{i+1}. Since all previous solutions are used, L must avoid negative transfer by learning which source task solutions are applicable to the current task.

In the context of safety validation, we hypothesize two qualitatively distinct ways that the tasks will be related. The first case is that of a *learning system* which is improving its performance from task to task. Each task is therefore more challenging for the adversary since failures that were present in previous tasks may no longer exist or may only occur due to a narrower range of disturbances. In this context, the most recent solutions are likely to provide the most relevant information and large parts of those solutions may be directly applicable to the new task. The second case is that of *comparable systems* where the systems have a similar level of competency but exhibit different behavior. Disturbance trajectories that lead to failure for one system may not cause failure in any other systems, although they might share similar conceptual failure modes. In this setting, direct transfer of solutions may be ineffective, and we need to rely on other types of knowledge.

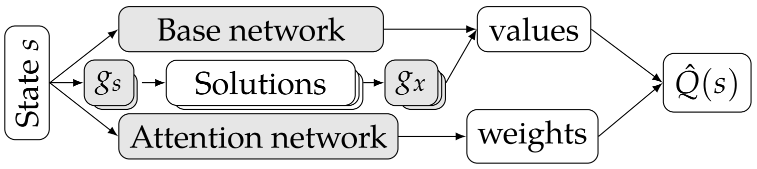


Figure 7.2: Architecture of A2T with state and action transformation.

7.3.2 *Choice of Learning Algorithm*

To accelerate safety validation we use the attend, adapt, and transfer (A2T) learning algorithm [129] with a minor modification. A2T accelerates learning on a new task by combining the solutions of k previous tasks using a learned set of state-dependent attention weights. To avoid negative transfer, A2T simultaneously learns a solution from scratch so there are a total of $k + 1$ solutions and as many attention weights. A2T can be used to estimate optimal policies or optimal value functions and we found it most straightforward to estimate the optimal action value function. When Q^* is estimated by a neural network, called a Q-network, the input is a vector representing the state and the output is a vector representing the values of a discrete set of disturbances. To make this clear, if $\mathcal{X} \subseteq \mathbb{R}^m$, then $\hat{Q}(s) \in \mathbb{R}^m$, is a vector that represents the estimates of the optimal values for each disturbance. The action value function is estimated by the expression

$$\hat{Q}(s) = w_0(s)\hat{Q}_{\text{base}}(s) + \sum_{i=1}^{k} w_i(s)\hat{Q}_i(s) \tag{7.2}$$

where $\hat{Q}_i$ comes from the ith source task, $\hat{Q}_{\text{base}}$ is learned from scratch and $w_i(s)$ is the ith attention weight, normalized so $\sum_{i=0}^{m} w_i(s) = 1$.

To handle substantially different system behaviors, we propose a modification of A2T where we include a learned transformation of the state and action value function for each of the k previous solutions. If the state space $\mathcal{S} \subseteq \mathbb{R}^n$, then we define a *state transformation* as a function $g_s : \mathbb{R}^n \to \mathbb{R}^n$ and an *action value transformation* as a function $g_x : \mathbb{R}^m \to \mathbb{R}^m$. Applied to the A2T algorithm, the action value estimate with state and action space transformations is given by

$$\hat{Q}(s) = w_0(s)\hat{Q}_{\text{base}}(s) + \sum_{i=1}^{k} w_i(s) g_x^{(i)}(\hat{Q}_i(g_s^{(i)}(s))) \tag{7.3}$$

where $g_s^{(i)}$ and $g_x^{(i)}$ are the state and action value transformations for the ith source solution.

The A2T algorithm with state and action value transformations can be encoded as the network architecture shown in Figure 7.2. The state is used as input to the base network, the source solutions, and the attention network. The base network is a Q-network that learns from scratch. The k source solutions are the Q-networks that represent the optimal solutions of the source tasks. The source solutions are preceded by a state transformation and followed by an action value transformation. These transformations have the same output dimension as input dimension and are initialized to the identity transformation (with a small amount of noise for breaking symmetry). The attention network has $k + 1$ output units with a softmax layer for normalization. The Q values of each source solution and the base network are weighted by the corresponding attention weight and summed together to get a final estimate. The network is trained using the DQN algorithm, which updates the parameters in the base network, the attention network, and the state and action value transformations. If the source solutions are not differentiable with respect to the state (as in the case of a lookup table) then we would apply gradient free optimization [47] to the state transformation.

Motivation. Here we provide some intuition for our choice of transfer learning algorithm and the reason for the state and action transformations. Suppose we are solving a sequence of tasks (Figure 7.3), each is a gridworld where an agent (blue circle) moves between adjacent squares attempting to achieve high reward (green squares) while avoiding states with low reward (red squares). Given the optimal action value function for a source MDP (Figure 7.3a), we wish to transfer it to two related tasks. In the first transfer problem (Figure 7.3b), the reward distribution is locally similar to the source MDP, and only differs in one state. In this case, a policy

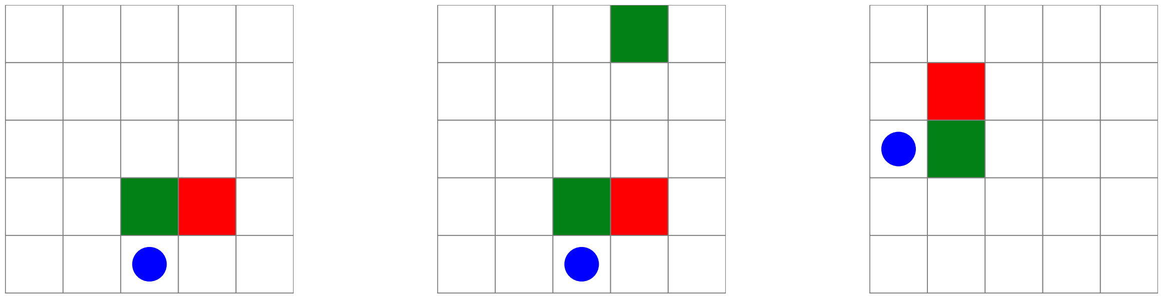

(a) Source MDP (b) Locally similar (c) Transformed

Figure 7.3: Knowledge transfer scenarios.

that works well in the source task will also work well in the new task, especially when the agent is in a state where the local reward landscape matches up (as depicted). A2T will work well in this setting because it can quickly learn attention weights that favor the source policy in most states, and only require our baseline solution when the agent is near the top right corner. In the second transfer problem (Figure 7.3c), A2T is likely to behave poorly because there are no states in which the source policy can be directly applied to achieve high reward. If, however, we could transform the state space by reflecting it across the diagonal and rotate the actions of the agent by 90, then we could directly apply the source policy. This is the motivation for applying transformations both before and after the source solutions.

7.4 Experiments

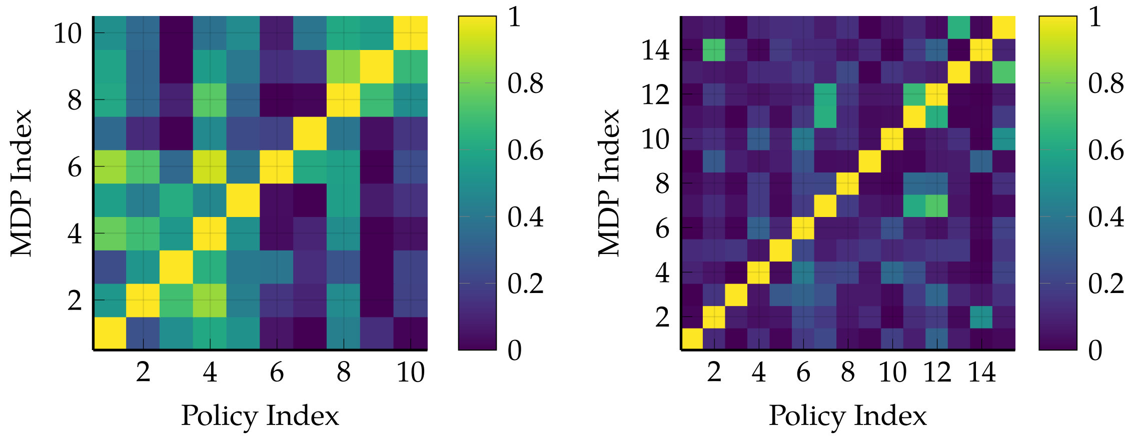
This section describes the iterative most likely failure analysis of the gridworld with an adversary and the crosswalk scenarios. We first describe how both scenarios are modified for iterative safety validation. Each scenario has two transfer learning problems, one for validating a learning system that improves over time, and the other for validating a set of comparable systems with different behaviors. We then describe the experimental setup and the results.

7.4.1 *Task Setup*

The first safety validation problem we consider is the gridworld scenario with a single adversary (see Section 3.2.2). We design two sets of tasks that correspond to the learning system and comparable systems settings. For the learning system, the ego agent is trained using DQN against an adversarial agent that behaves randomly. Over 10^6 training steps, 10 versions of the system policy were stored, each with an increasing level of performance. Each safety validation task has the adversary validate an increasingly capable version of the learning system. The performance of the optimal policy on each task is measured on every other task and plotted in Figure 7.4a to observe the baseline transfer between tasks. We note that each policy works best with the MDP it was trained on but also has a modest amount of transfer to nearby tasks.

For the comparable systems setting, each task has a different distribution of reward locations, reward values and location of walls. The system learns an optimal policy using dynamic programming [80], assuming the adversary behaves randomly. Each system is therefore equally competent, but some configurations of the gridworld are more challenging than others. A similar comparison plot of the optimal policy for each tasks applied to all others is shown in Figure 7.4b. In this setting, the transfer between tasks is minimal due to the diverse set of behaviors of the system.

The second safety validation problem we model is a modified version of the crosswalk scenario where the vehicle has a blindspot as shown in Figure 7.5. We design two sets of tasks corresponding to a learning system setting and a comparable systems setting. Since the autonomous vehicle does not use machine learning, we simulate an improvement by progressively shrinking the blind spot of the vehicle. The blind spot remains in the same direction (20 from the horizontal), but reduces in width from 30 to 6 over 10 iterations. The vehicle therefore has a decreasing rate of failures over the tasks but there is some overlap in failure modes between adjacent tasks (see Figure 7.6a for comparison of tasks and optimal policies). For the comparable systems setting, each system has a blind spot sampled uniformly at



(a) Learning system. (b) Comparable systems.

Figure 7.4: Comparison of policies and tasks for the gridworld scenario.

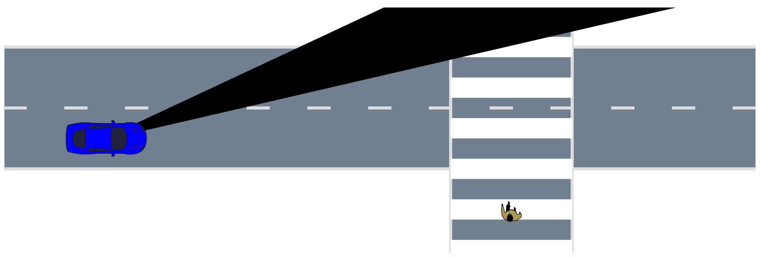
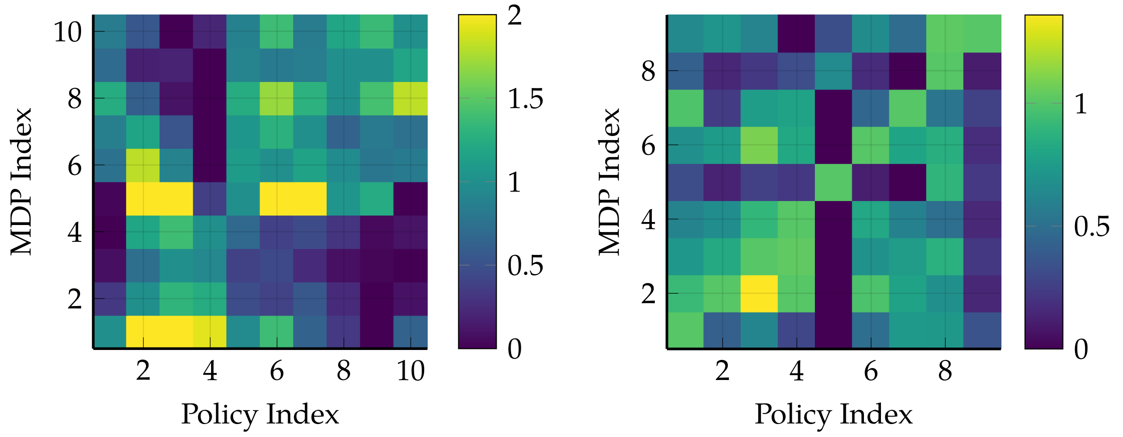


Figure 7.5: Crosswalk scenario with AV blindpsot.



(a) Learning system. (b) Comparable systems.

Figure 7.6: Comparison of policies and tasks for the driving scenario.

random with a direction in the range $[-30, 30]$ and an angular width in the range $[3, 9]$. From a population of 30 tasks, we selected 9 tasks that differed substantially, to make the transfer problem as challenging as possible within our setting. Two tasks differed if the optimal safety validation policy of one performs poorly on the other. We show the comparison of task performance in Figure 7.6b.

7.4.2 *Experimental Setup*

The experimental procedure is as follows. For each task in a set of tasks, we solve for an optimal Q-network from scratch using DQN with prioritized replay, double Q-learning [135], and the Huber loss. We construct a learning curve during training by periodically storing the evaluation of the Q-network. For the second task onward, we then solve it using the same learning algorithm with the following Q-network architectures:

- **Fine-tune**: Train the last layer of the previous Q-network.
- **A2T**: A2T architecture with previous Q-networks as the source solutions.

Table 7.1: Network architectures and hyperparameters.

Parameter	**Value**
Base network	3 hidden layers, [64, 32, 16] units
Attention network	1 hidden layer, 16 units
Activation function	relu
State/Action transform	Linear
Training steps	3×10^6
Batch size	64
Learning rate α	4×10^{-5} (GW), 5×10^{-5} (AD)
Target update frequency	2,000 (GW), 3,000 (AD)
Evaluation frequency	2,000
No. evaluation episodes	300
Exploration policy	ϵ-greedy with $\epsilon \in [1, 0.1]$

- **A2T+SAVT**: A2T architecture augmented with linear state and action value transformations.

The networks are initialized with Xavier initialization [99], while the transformations were initialized to the identity matrix with uniform random noise in the range [-1×10^{-3}, 1×10^{-3}] added to the parameters to break symmetry. Additional information on network architecture and hyperparameters is shown in Table 7.1.

We filter each learning curve using a moving-average filter with a width of 20 evaluations steps to help remove the noise due to finite sample evaluation and any outlier evaluation points. For each learning curve we identify the *near-optimal* performance as $\mu - \sigma$, where μ and σ are the mean and standard deviation of the performance in a window with a width of 100 evaluation steps around the point of maximum performance. We use near-optimal performance because it is a more stable measure of how fast the learning took place than the point of maximum performance.

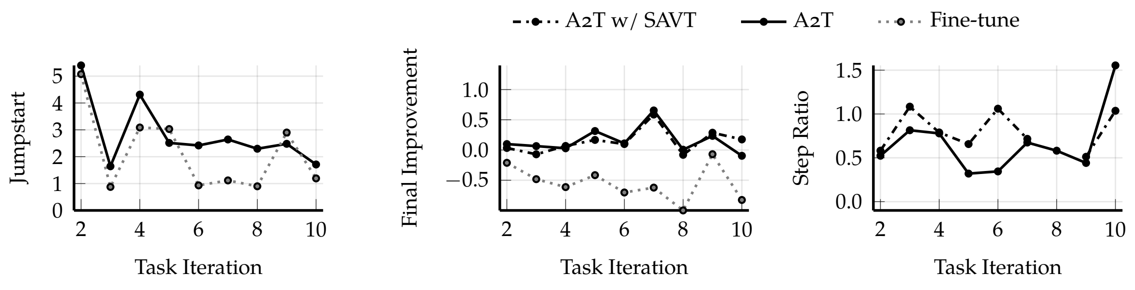
The jumpstart is the difference in initial performance between a transfer and no-transfer learning algorithms. It can be computed from the first entries in the learning curves. When reporting jumpstart, we only include fine-tuning and A2T because A2T+SAVT has the same outputs as A2T until the transformations deviate from identity. The final performance is the difference in near-optimal performance between the transfer and no-transfer learning algorithms. The steps to threshold metric measures how many training steps are required for a transfer learning algorithm to reach the near-optimal performance of the no-transfer algorithm.

The metrics are normalized with reference to the learning curve of the no-transfer algorithm because the initial and near-optimal performance varies between tasks. Let y be the performance (initial or final) of a transfer learning algorithm and y_{ref} be the performance of the no-transfer learning algorithm, then we report the fractional difference in performance $(y - y_{\text{ref}})/|y_{\text{ref}}|$. Let t be the number of training steps required to reach a threshold for a transfer learning algorithm and t_{ref} be the same quantity for the no-transfer learning algorithm, then we report the ratio t/t_{ref}.

We use the number of training steps rather than the wall clock time because we assume that the cost of running the simulator is much larger than the cost of updating the parameters of the model. This is a good assumption for high-fidelity simulators that are often used for validating safety-critical systems. We also assume that the training time of previous source tasks is a sunk cost and it is not included in our efficiency metric. This assumption is valid in the case of iterative safety validation because the new version of the system must be validated regardless of the approach used. When using A2T in a real-world setting, we would not solve each task from scratch and therefore the source solutions would take the form of A2T networks. A practitioner may wish to compress the A2T network [136] into a traditional architecture before it is used as a source solution. We chose to use the networks trained from scratch for ease of implementation and to isolate the effects of transfer learning from other issues.

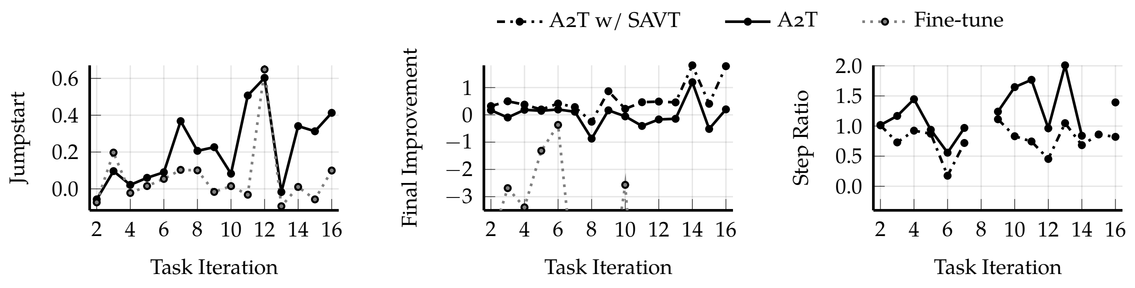
7.4.3 Results

We solved each of the four safety validation tasks using 3 transfer learning algorithms and report the evaluation metrics against the task index in Figures 7.7 to 7.10. The discussion of the results is grouped by evaluation metric.



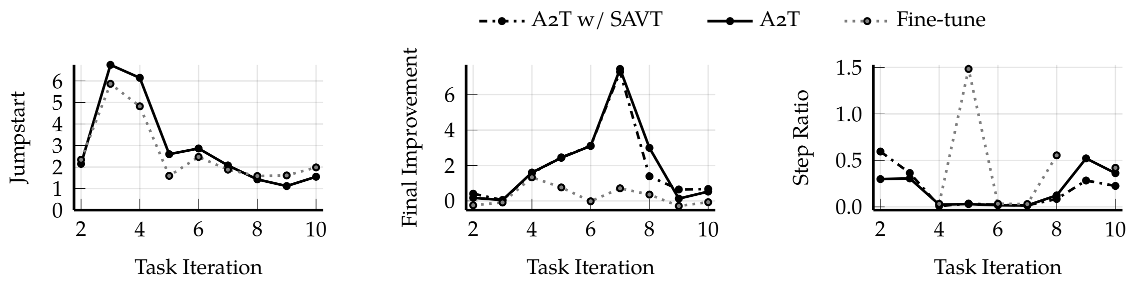
(a) Jumpstart improvement. (b) Final improvement. (c) Step ratio to threshold.

Figure 7.7: Evaluation metrics for the gridworld scenario with a learning system.



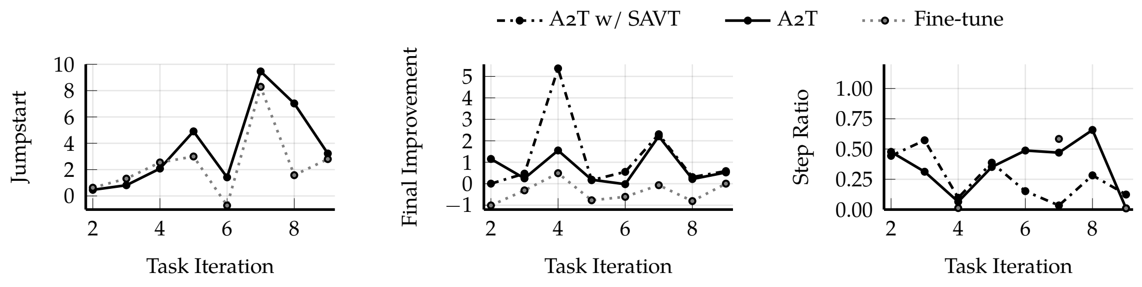
(a) Jumpstart improvement. (b) Final improvement. (c) Step ratio to threshold.

Figure 7.8: Evaluation metrics for the gridworld scenario with comparable systems.



(a) Jumpstart improvement. (b) Final improvement. (c) Step ratio to threshold.

Figure 7.9: Evaluation metrics for the autonomous driving scenario with a learning system.



(a) Jumpstart improvement. (b) Final improvement. (c) Step ratio to threshold.

Figure 7.10: Evaluation metrics for the autonomous driving scenario with a comparable systems.

Jumpstart. Figures 7.7a, 7.8a, 7.9a and 7.10a show the jumpstart of the fine-tune and A2T architectures. We see that across all four safety validation problems, the transfer learning algorithms contributed a significant increase in initial performance. For most tasks, the A2T architecture had slightly better jumpstart than simply reusing the previous solution, especially in Figure 7.8a. The gridworld with comparable systems had substantially different failure modes between tasks and therefore had the least benefit in jumpstart. The safety validation problems with learning systems had the jumpstart decrease with the number of tasks observed, likely due to the increase in difficulty of the tasks. For the safety validation problems involving comparable systems, however, the jumpstart tended to increase with the number of source tasks. We hypothesize that with more source tasks, we are more likely to have a task that closely matches the current tasks, and can therefore immediately have reasonable performance.

Final Performance. Figures 7.7b, 7.8b, 7.9b and 7.10b show the final performance of each transfer learning algorithm. The final safety validation performance can be significantly improved by both A2T approaches, but not through fine-tuning. In all but Figure 7.9b, the fine-tuning approach was not able to match the no-transfer performance given the same number of iterations. This lack of performance could mean that the Q-network for one task is not learning a set of features that is useful for solving other tasks, so updating only the final layer does not provide enough capacity to solve the problem. The A2T networks, however, are able to achieve significantly improved final performance, which generally increases with the number of source tasks. The A2T network with state and action value transformations outperforms the basic A2T network in both safety validation problems with comparable systems, which are the problems it was designed for. Both of the safety validation problems with a learning system show the maximal gain in final performance in the middle of the sequence of tasks. A lower gain in early tasks may be due to those tasks being easy to solve, while a lower gain in the later tasks may be due to only having a few failure modes to exploit. The middle tasks may be challenging to solve but may have a diversity of failure modes that the previous policies can help identify. More experimentation is needed to fully understand these trends.

Steps to Threshold. Figures 7.7c, 7.8c, 7.9c and 7.10c show the number of training steps required to reach the near-optimal performance of the no-transfer algorithm. In some cases, near-optimal performance is never reached so those data points are omitted from the plots. We observe that the number of training steps can be reduced by both A2T networks, but in different conditions. The basic A2T network performs well when validating a learning system because parts of previous solutions can be used directly. In Figure 7.7c, the number of training steps on some tasks could be reduced by 50% and in Figure 7.9c the number of training steps is reduced by more than an order of magnitude in some cases.

The A2T network with state and action transformations performs slightly worse than the basic A2T network in Figure 7.7c and has similar performance in Figure 7.9c, but significantly outperforms the A2T network for many tasks in the comparable systems setting. In Figure 7.8c, the basic A2T network requires more steps than the no-transfer algorithm, which negates the utility of the more complex architecture, while the A2T+SAVT network was able to reduce the number of training steps by up to 50% on some tasks. We note that generally, the fine-tune approach is unable to achieve the same performance as learning from scratch but when it does reach near-optimal performance, it requires fewer training steps than learning from scratch.

Summary. From our experiments we conclude that transfer learning can be an effective strategy for improving performance and efficiency of safety validation algorithms. Transfer through fine-tuning can give a significant increase in jumpstart but often fails to reach the level of performance of a Q-network trained from scratch. The A2T networks also provides an increase in jumpstart as well as an increase in final performance. The use of a small attention network allows for quick adaptation to new domains as evidenced by the reduction in the number of training steps required to reach near-optimal performance. When the tasks differ significantly from each other, however, the basic A2T network may take longer than the no-transfer algorithm to reach near-optimal performance. We fix this problem by introducing state and action value transformations for each source solution and demonstrate improved training efficiency over the no-transfer algorithm.

7.5 Discussion

The validation of safety-critical autonomous systems is crucial for their safe deployment. Existing algorithms for validation often start from scratch each time the system changes. The nature of system design implies that safety validation will be performed iteratively on related systems, and should therefore benefit from past experience. We formulate iterative safety validation as a transfer learning problem and demonstrate improvements in both efficiency and performance of transfer learning algorithms compared to a no-transfer baseline. We augmented the attend, adapt, and transfer algorithm with state and action value transformations to allow for more transfer between disparate tasks. We evaluated jumpstart, final performance, and steps to threshold metrics on four iterative safety validation problems in gridworld and autonomous driving domains. Future work will include exploring the failure modes discovered by each algorithm to gain insights into how transfer is occurring. These insights may help us understand under what conditions we can expect performance and efficiency improvements.

Our work on iterative safety validation is related to some previous work on safety validation. Uesato, Kumar, *et al.* [104] use previous versions of a system to train a failure classifier that predicts which initial conditions of a system will lead to failure, but their approach is not applicable to sequential decision making problems of the type we consider. Wang, Nair, *et al.* [137] alternately train an agent and perform safety validation on it to improve robustness. The safety validation algorithm warm starts with the parameters from the previous iteration to improve efficiency. In fact, any parametric safety validation algorithm [30], [89], [103] could simply reuse parameters from previous tasks and then fine-tune them for better performance. We demonstrated in our experiments, however, that a fine-tuning approach only seems effective when the systems are very similar, but has poor performance when systems exhibit different behavior.

In the final chapter of the thesis we summarize the contributions of this work and discuss some directions for future work on each of the topics covered.

8 Conclusion

This chapter summarizes the safety validation techniques developed in this thesis, highlights the contributions, and presents some possible areas of future work.

8.1 Summary

Before safety-critical autonomous systems can be deployed they must undergo rigorous safety validation, but this remains a challenge due to the complexity of the systems and their operational environments. Scenario-based testing can be used to construct a suite of challenging scenarios that an autonomous system must succeed in, but may miss unforeseen or emergent failures. Formal verification can be used to prove the correct operation of system components but cannot scale to complex systems or stochastic environments. Real-world testing is required to to show that implementation and integration has not introduced new failure modes, but is expensive to perform and can be dangerous if the system is not already very safe.

Black-box sampling approaches address many of the drawbacks to traditional safety validation techniques. The autonomous system is treated as a black box that takes actions in a simulated stochastic environment. Stochastic disturbances in the environment are controlled by an adversary with the goal of causing the system to fail. The adversary relies on sampling to discover failures and can therefore work with complex systems and environments. Machine learning is used to guide the search toward the discovery of unforeseen and complex failures. Black-box sampling has emerged as a promising tool for validating modern autonomous systems but still suffers from several drawbacks including interpretability, scalability and computational expense.

Traditional testing techniques such as unit testing or scenario generation usually target specific behaviors or components of the autonomous system. When a failure is found, it is often clear which narrow set of environmental factors led to it. In black-box sampling, however, there is usually a large number of stochastic variables being controlled over time, so when a failure is discovered it is not always clear what caused it. We solve this problem by searching for failure descriptions, which are low-dimensional, interpretable descriptions of failures, instead of example failure trajectories. Failure descriptions take the form of signal temporal logic expressions that are optimized using genetic programming. Failure descriptions provide insight into the possible causes of a failure and we can use them to produce many failure examples. Since the failure description usually has a limited number of parameters, we can perform a sensitivity analysis on the safety of the system with respect to disturbance parameters.

Black-box sampling techniques can be used to estimate the probability of failure of a system and thereby provide probabilistic guarantees of safety. The traditional technique of estimating the probability of rare events is importance sampling, which iteratively learns the optimal distribution over disturbance trajectories. Learning a distribution over a high-dimensional trajectory, however, is computationally challenging and may requires many samples. To mitigate this problem, we learn a policy that maps environment states to disturbances that cause the system to fail. The policy can be learned using reinforcement learning algorithms and results in a sampling distribution that is effective for estimating the probability of failure.

The use of the environment state for discovering failures can improve performance, but limits the scalability safety validation algorithms when the state space is large, such as in multi-agent settings. To improve scalability for multi-agent systems, we decompose the problem into smaller safety validation problems between the system and each other agent. Each subproblem is solved and the results are combined using a neural network trained on rollouts of the full problem. We show improved rates of discovered failures and better estimation of the probability of failure using decomposition.

During the development of autonomous systems, safety validation is often performed many times on closely related systems. The system may be improving over time as it is developed, or several competing systems may be compared. Despite our described improvements in scalability, black-box safety validation still requires many samples to reliably discover failures, so frequent safety validation imposes a large computation burden. To reduce this expense, we transfer knowledge in the form of value functions from the safety validation of previous systems to later systems. The value functions are combined with a learned set of attention weights that allows for more efficient safety validation on new systems.

8.2 Contributions

This work attempts to address several of the existing challenging for validating safety-critical autonomous systems including scalability, interpretability, and efficiency. In pursuit of these goals we made the following contributions:

A model for black-box safety validation of autonomous systems. In Chapter 2 we presented a general model of black-box safety validation that describes a system taking actions in a stochastic environment with disturbances controlled by an adversary. This model allowed for the identification of three safety validation tasks and the classification of existing safety validation algorithms based on optimization, path-planning, reinforcement learning and importance sampling.

A technique for generating interpretable failure descriptions. In Chapter 4 we developed a technique that can discover failure descriptions of an autonomous system in the from of a signal temporal logic specification on the disturbances. Expressions were evaluated on their ability to produce failure examples and optimized using genetic programming. Failure descriptions provide insight into why the system failed and can produce many failure examples for further analysis or training. The failure descriptions often had a small number of parameters and could be used to perform a sensitivity analysis on the safety of the system.

A state-dependent importance sampling distribution for approximating the distribution over failures. In Chapter 5 we proposed a state-dependent sampling distribution that could be used efficiently estimate the probability of failure. The proposed distribution chooses disturbances proportional to the probability that the disturbance will lead to failure, which is shown to be optimal for deterministic environments. We present several techniques for estimating the probability of failure and show that the resulting policy is robust to errors in that estimate.

A problem decomposition technique to improve the scalability of safety validation algorithms for multi-agent systems. In Chapter 6 we propose a decomposition approach to scale state-dependent safety validation algorithms to multi-agent systems with large state spaces. When the problem is decomposed into pairwise interactions between the system and each adversary it may be tractable to solve. The solutions to these subproblems are combined with a learned set of weights and help solve the full problem efficiently.

A transfer learning approach for improving the efficiency and performance of safety validation of multiple related systems. In Chapter 7 we present a transfer-learning technique to transfer knowledge in the form of value functions from previous safety validation tasks to new tasks. The transfer allows for better initial and final performance of the safety validation algorithm as well as better training efficiency.

8.3 Future Work

The algorithms presented in this thesis take a step toward scalable and interpretable safety validation for black-box autonomous systems but further research is required before applying these approaches confidently to safety-critical applications. This section outlines some possible future research directions that can expand the capability and utility of safety validation algorithms.

Applications to industrial driving policies and real world data distributions. The safety validation algorithms discussed in this thesis were tested on relatively simple 2D driving simulators with rule-based autonomy. It remains an open question how these algorithms will work when used with a more realistic driving environment and more advanced policies. Additionally, we use simplistic models of environment disturbances but it is possible to construct probabilistic behavior models from real world data [138]. Future work should combine realistic simulators with disturbance models generated from real-world data. Large scale simulators that render 3D graphics and vehicle dynamics are much more costly to simulate so data efficiency will be of critical importance. The simulators may also be very complex which could limit how agents in the scene are controlled, but also provide a vast number possible parameters to use as disturbances. More work needs to be done to identify which disturbances are the most useful for causing failures.

Adversarial testing with perceptual inputs. Current work emphasizes the use of disturbances that represent high level perturbations to the system (such as the absolute noise in the detection of a pedestrian's position). Many autonomous systems use perceptual inputs for mapping their environment when are subject to adversarial examples under perturbations to pixel values [139]. Recent work [34] combines black box adversarial testing with formal verification to find sequences of image perturbations that cause a system to fail. The limitation of this work is the requirement for small images and small neural networks. Future work should investigate how to scale up the formal verification procedure, or develop other techniques to construct sequences of adversarial images to find failures.

Policy-gradient methods for estimating the distribution over failures. Our proposed technique in Chapter 5 constructs the distribution over failures by estimating the probability of failure at each state. We run into problems when the disturbance space is continuous or when the probability of failure is difficult to represent using a neural network. In reinforcement learning, continuous actions can be used by representing the policy instead of the value function. By analogy, we could investigate the use of neural network policies to directly approximate the distribution over failures. Challenges would include the choice of cost function and network architecture to best represent the distribution over failures.

Continual learning for safety validation algorithms. Although we have demonstrated performance and efficiency improvements by applying transfer learning to iterative safety validation, our approach is limited if the number of systems grows too large. Since we explicitly store and reference the safety validation policies from previous tasks, the complexity of the algorithm increases over time. The field of *continual learning* tries to address these problems by developing algorithms that can retain competency on previous tasks without explicitly storing previous versions of a policy. Future work on iterative safety validation will include the application of continual learning algorithms to create adversaries that do not forget how to induce previously discovered failure modes.

Bibliography

[1] World Health Organization (WHO), "Global status report on alcohol and health 2018," Tech. Rep., 2019.

[2] National Highway Traffic Safety Administration (NHTSA), "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," Tech. Rep., 2015.

[3] Federal Aviation Administration (FAA), "Introduction to TCAS II: Version 7.1," Tech. Rep., 2011.

[4] T Arino, K Carpenter, S Chabert, H Hutchinson, T Miquel, B Raynaud, K Rigotti, and E Vallauri, "Studies on the safety of acas ii in europe," *Eurocontrol, Technical Rep. ACASA/WP-1.8 D*, vol. 210, 2002.

[5] M. J. Kochenderfer, J. E. Holland, and J. P. Chryssanthacopoulos, "Next-generation airborne collision avoidance system," *Lincoln Laboratory Journal*, vol. 19, no. 1, pp. 17–33, 2012.

[6] National Transportation Safety Board (NTSB), "Preliminary report, highway, hwy18mh010," Tech. Rep., 2018. [Online]. Available: `https://www.ntsb.gov/investigations/AccidentReports/Reports/HWY18MH010-prelim.pdf`.

[7] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a Formal Model of Safe and Scalable Self-driving Cars," *arXiv e-prints*, arXiv:1708.06374, arXiv:1708.06374, Aug. 2017. arXiv: `1708.06374 [cs.RO]`.

[8] D. Nistér, H.-L. Lee, J. Ng, and Y. Wang, "The safety force field," 2019.

[9] D. Sadigh, E. S. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia, "A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications," in *IEEE Conference on Decision and Control*, 2014.

[10] M. Bouton, J. Karlsson, A. Nakhaei, K. Fujimura, M. J. Kochenderfer, and J. Tumova, "Reinforcement Learning with Probabilistic Guarantees for Autonomous Driving," *arXiv e-prints*, arXiv:1904.07189, arXiv:1904.07189, Apr. 2019. arXiv: `1904.07189 [cs.RO]`.

[11] T. M. Moldovan and P. Abbeel, "Risk aversion in markov decision processes via near optimal chernoff bounds," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 3131–3139. [Online]. Available: `http://papers.nips.cc/paper/4514-risk-aversion-in-markov-decision-processes-via-near-optimal-chernoff-bounds.pdf`.

[12] A. Tamar, Y. Glassner, and S. Mannor, "Policy gradients beyond expectations: Conditional value-at-risk," *arXiv preprint arXiv:1404.3862*, 2014.

[13] P. Abbeel and A. Y. Ng, "Exploration and apprenticeship learning in reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*, 2005.

[14] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[15] T. Mihai Moldovan and P. Abbeel, "Safe Exploration in Markov Decision Processes," *arXiv e-prints*, arXiv:1205.4810, arXiv:1205.4810, May 2012. arXiv: `1205.4810 [cs.LG]`.

[16] J. Garcıa, D. Acera, and F. Fernández, "Safe reinforcement learning through probabilistic policy reuse," *RLDM 2013*, p. 14, 2013.

[17] Y. Sui, A. Gotovos, J. Burdick, and A. Krause, "Safe exploration for optimization with gaussian processes," in *International Conference on Machine Learning*, 2015.

[18] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[19] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete Problems in AI Safety," *arXiv e-prints*, arXiv:1606.06565, Jun. 2016. arXiv: `1606.06565 [cs.AI]`.

[20] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.

[21] B. Alpern and F. B. Schneider, "Recognizing safety and liveness," *Distributed computing*, vol. 2, no. 3, pp. 117–126, 1987.

[22] S. R. Hirshorn, L. D. Voss, and L. K. Bromley, *NASA Systems Engineering Handbook*. NASA Special Publication, 2017.

[23] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification (CAV)*, 2017.

[24] A. Pnueli, "The temporal logic of programs," in *Foundations of Computer Science, 1977*, 1977.

[25] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Workshop on Logic of Programs*, 1981.

[26] E. A. Emerson and J. Y. Halpern, ""sometimes" and "not never" revisited: On branching versus linear time temporal logic," *Journal of the ACM (JACM)*, vol. 33, no. 1, pp. 151–178, 1986.

[27] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, pp. 255–299, 1990.

[28] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Springer, 2004, pp. 152–166.

[29] R. Lee, M. J. Kochenderfer, O. J. Mengshoel, G. P. Brat, and M. P. Owen, "Adaptive stress testing of airborne collision avoidance systems," in *Digital Avionics Systems Conference (DASC)*, 2015.

[30] M. Koren, S. Alsaif, R. Lee, and M. J. Kochenderfer, "Adaptive stress testing for autonomous vehicles," in *IEEE Intelligent Vehicles Symposium (IV)*, 2018.

[31] K. Leung, N. Aréchiga, and M. Pavone, "Backpropagation for parametric stl," in *IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[32] R. Delmas, T. Loquen, J. Boada-Bauxell, and M. Carton, "An evaluation of Monte Carlo tree search for property falsification on hybrid flight control laws," in *International Workshop on Numerical Software Verification*, 2019.

[33] G. Ernst, P. Arcaini, A. Donze, G. Fainekos, L. Mathesen, G. Pedrielli, S. Yaghoubi, Y. Yamagata, and Z. Zhang, "Arch-comp 2019 category report: Falsification," *EPiC Series in Computing*, vol. 61, pp. 129–140, 2019.

[34] K. D. Julian, R. Lee, and M. J. Kochenderfer, "Validation of image-based neural network controllers through adaptive stress testing," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2020.

[35] H. Kress-Gazit and G. J. Pappas, "Automatically synthesizing a planning and control subsystem for the darpa urban challenge," in *International Conference on Automation Science and Engineering (CASE)*, 2008.

[36] X. Qin, N. Aréchiga, A. Best, and J. Deshmukh, "Automatic Testing and Falsification with Dynamically Constrained Reinforcement Learning," *arXiv e-prints*, arXiv:1910.13645, arXiv:1910.13645, Oct. 2019. arXiv: `1910.13645 [cs.LG]`.

[37] M. Hekmatnejad, S. Yaghoubi, A. Dokhanchi, H. B. Amor, A. Shrivastava, L. Karam, and G. Fainekos, "Encoding and monitoring responsibility sensitive safety rules for automated vehicles in signal temporal logic," in *IEEE International Conference on Formal Methods and Models for System Design*, 2019.

[38] M. Hekmatnejad, B. Hoxha, and G. Fainekos, "Search-based test-case generation by monitoring responsibility safety rules," *arXiv preprint arXiv:2005.00326*, 2020.

[39] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[40] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2010.

[41] H. Yang, "Dynamic programming algorithm for computing temporal logic robustness," Master's thesis, Arizona State University, 2013.

[42] T. Dreossi, T. Dang, A. Donzé, J. Kapinski, X. Jin, and J. V. Deshmukh, "Efficient guiding strategies for testing of temporal properties of hybrid systems," in *NASA Formal Methods Symposium (NFM)*, 2015.

[43] G. Ernst, S. Sedwards, Z. Zhang, and I. Hasuo, "Fast falsification of hybrid systems using probabilistically adaptive input," in *International Conference on Quantitative Evaluation of Systems (QEST)*, 2019.

[44] Y. V. Pant, H. Abbas, and R. Mangharam, "Smooth operator: Control using the smooth robustness of temporal logic," in *IEEE Conference on Control Technology and Applications (CCTA)*, 2017.

[45] Z. Zhang, I. Hasuo, and P. Arcaini, "Multi-armed bandits for boolean connectives in hybrid system falsification," in *Computer Aided Verification (CAV)*, 2019.

[46] A. Corso, R. J. Moss, M. Koren, R. Lee, and M. J. Kochenderfer, "A Survey of Algorithms for Black-Box Safety Validation," *arXiv e-prints*, arXiv:2005.02979, arXiv:2005.02979, May 2020. arXiv: `2005.02979 [cs.LG]`.

[47] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization*. MIT Press, 2019.

[48] J. M. Esposito, J. Kim, and V. Kumar, "Adaptive RRTs for validating hybrid robotic control systems," in *Algorithmic Foundations of Robotics VI*, Springer, 2004, pp. 107–121.

[49] T. Nahhal and T. Dang, "Test coverage for continuous and hybrid systems," in *Computer Aided Verification*, 2007, ISBN: 978-3-540-73368-3.

[50] A. Dokhanchi, A. Zutshi, R. T. Sriniva, S. Sankaranarayanan, and G. Fainekos, "Requirements driven falsification with coverage metrics," in *International Conference on Embedded Software (EMSOFT)*, 2015.

[51] J. Deshmukh, X. Jin, J. Kapinski, and O. Maler, "Stochastic local search for falsification of hybrid systems," in *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2015.

[52] A. Adimoolam, T. Dang, A. Donzé, J. Kapinski, and X. Jin, "Classification and coverage-based falsification for embedded control systems," in *International Conference on Computer Aided Verification (CAV)*, 2017.

[53] S. Yaghoubi and G. Fainekos, "Gray-box adversarial testing for control systems with machine learning components," in *ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2019.

[54] L. Mathesen, S. Yaghoubi, G. Pedrielli, and G. Fainekos, "Falsification of cyber-physical systems with robustness uncertainty quantification through stochastic optimization with adaptive restart," English (US), in *International Conference on Automation Science and Engineering (CASE)*, Aug. 2019.

[55] Q. Zhao, B. H. Krogh, and P. Hubbard, "Generating test inputs for embedded control systems," *IEEE Control Systems Magazine*, vol. 23, no. 4, pp. 49–57, 2003.

[56] X. Zou, R. Alexander, and J. McDermid, "Safety validation of sense and avoid algorithms using simulation and evolutionary search," in *International Conference on Computer Safety, Reliability, and Security (SafeComp)*, 2014.

[57] T. Akazaki, Y. Kumazawa, and I. Hasuo, "Causality-aided falsification," *Electronic Proceedings in Theoretical Computer Science*, vol. 257, 2017.

[58] S. Silvetti, A. Policriti, and L. Bortolussi, "An active learning approach to the falsification of black box cyber-physical systems," in *International Conference on Integrated Formal Methods (iFM)*, 2017.

[59] J. Deshmukh, M. Horvat, X. Jin, R. Majumdar, and V. S. Prabhu, "Testing cyber-physical systems through bayesian optimization," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, Sep. 2017, ISSN: 1539-9087.

[60] G. E. Mullins, P. G. Stankiewicz, R. C. Hawthorne, and S. K. Gupta, "Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles," *Journal of Systems and Software*, vol. 137, pp. 197–215, 2018.

[61] Y. Abeysirigoonawardena, F. Shkurti, and G. Dudek, "Generating adversarial driving scenarios in high-fidelity simulators," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[62] X. Yang, M. Egorov, A. Evans, S. Munn, and P. Wei, "Stress testing of uas traffic management decision making systems," in *AIAA AVIATION Forum*, 2020.

[63] Y. S. R. Annapureddy and G. E. Fainekos, "Ant colonies for temporal logic falsification of hybrid systems," in *Annual Conference on IEEE Industrial Electronics Society (IECON)*, 2010.

[64] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *IEEE International Conference on Evolutionary Computation*, 1996.

[65] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta, "Probabilistic temporal logic falsification of cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, pp. 1–30, 2013.

[66] M. A. Luersen and R. Le Riche, "Globalized nelder–mead method for engineering optimization," *Computers & Structures*, vol. 82, no. 23-26, pp. 2251–2260, 2004.

[67] Y. Annapureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2011.

[68] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *International Conference on Computer Aided Verification (CAV)*, 2010.

[69] A. Aerts, B. T. Minh, M. R. Mousavi, and M. A. Reniers, "Temporal logic falsification of cyber-physical systems: An input-signal-space optimization approach," in *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2018.

[70] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[71] A. Zutshi, J. V. Deshmukh, S. Sankaranarayanan, and J. Kapinski, "Multiple shooting, CEGAR-based falsification for hybrid systems," in *International Conference on Embedded Software (ICESS)*, 2014.

[72] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Tech. Rep., 1998.

[73] J. Kim, J. M. Esposito, and V. Kumar, "An RRT-based algorithm for testing and validating multi-robot controllers," Moore School of Electrical Engineering GRASP Lab, Tech. Rep., 2005.

[74] M. S. Branicky, M. M. Curtiss, J. Levine, and S. Morgan, "Sampling-based planning, control and verification of hybrid systems," *IEEE Proceedings - Control Theory and Applications*, vol. 153, no. 5, pp. 575–590, 2006.

[75] T. Dang, A. Donzé, O. Maler, and N. Shalev, "Sensitive state-space exploration," in *IEEE Conference on Decision and Control (CDC)*, 2008.

[76] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Hybrid systems: From verification to falsification by combining motion planning and discrete search," *Formal Methods in System Design*, vol. 34, no. 2, pp. 157–182, 2009.

[77] C. E. Tuncali and G. Fainekos, "Rapidly-exploring random trees for testing automated vehicles," in *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019.

[78] M. Koschi, C. Pek, S. Maierhofer, and M. Althoff, "Computationally efficient safety falsification of adaptive cruise control systems," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2019.

[79] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[80] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[81] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[82] M. Koren, A. Corso, and M. J. Kochenderfer, "The adaptive stress testing formulation," in *Workshop on Safe Autonomy, Robotics: Science and Systems*, 2019.

[83] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *European Conference on Machine Learning (ECML)*, 2006.

[84] R. Coulom, "Computing 'ELO ratings' of move patterns in the game of go," *ICGA Journal*, vol. 30, no. 4, pp. 198–208, 2007.

[85] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. V. D. Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive strategies for Monte Carlo tree search," *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.

[86] M. Wicker, X. Huang, and M. Kwiatkowska, "Feature-guided black-box safety testing of deep neural networks," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2018.

[87] Z. Zhang, G. Ernst, S. Sedwards, P. Arcaini, and I. Hasuo, "Two-layered falsification of hybrid systems guided by Monte Carlo tree search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2894–2905, 2018.

[88] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[89] T. Akazaki, S. Liu, Y. Yamagata, Y. Duan, and J. Hao, "Falsification of cyber-physical systems using deep reinforcement learning," in *International Symposium on Formal Methods (FM)*, 2018, ISBN: 978-3-319-95582-7.

[90] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems (NIPS)*, 2000.

[91] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning (ICML)*, 2015.

[92] M. Koren and M. J. Kochenderfer, "Efficient autonomy validation in simulation with adaptive stress testing," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2019.

[93] A. Corso, P. Du, K. Driggs-Campbell, and M. J. Kochenderfer, "Adaptive stress testing with reward augmentation for autonomous vehicle validation," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2019.

[94] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[95] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2016.

[96] S. Kuutti, S. Fallah, and R. Bowden, "Training Adversarial Agents to Exploit Weaknesses in Deep Control Policies," *arXiv e-prints*, arXiv:2002.12078, arXiv:2002.12078, Feb. 2020. arXiv: `2002.12078 [cs.LG]`.

[97] V. Behzadan and A. Munir, "Adversarial reinforcement learning framework for benchmarking collision avoidance mechanisms in autonomous vehicles," *IEEE Intelligent Transportation Systems Magazine*, 2019.

[98] T. P. Lillicrap, J. J. Hunt, A. e. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv e-prints*, arXiv:1509.02971, arXiv:1509.02971, Sep. 2015. arXiv: `1509.02971 [cs.LG]`.

[99] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

[100] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations*, 2016.

[101] P. J. Huber, "Robust estimation of a location parameter," in *Breakthroughs in Statistics*, Springer, 1992, pp. 492–518.

[102] G. Hahn, "Sample sizes for Monte Carlo simulation," *IEEE Transactions on Systems, Man, and Cybernetics*, 1972.

[103] Y. Kim and M. J. Kochenderfer, "Improving aircraft collision risk estimation using the cross-entropy method," *Journal of Air Transportation*, vol. 24, no. 2, pp. 55–62, 2016.

[104] J. Uesato, A. Kumar, C. Szepesvári, T. Erez, A. Ruderman, K. Anderson, K. D. Dvijotham, N. Heess, and P. Kohli, "Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures," in *International Conference on Learning Representations*, 2019.

[105] J. Neufeld, A. Gyorgy, C. Szepesvari, and D. Schuurmans, "Adaptive monte carlo via bandit allocation," in *International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, Jun. 2014.

[106] H. Kahn and T. E. Harris, "Estimation of particle transmission by random sampling," *National Bureau of Standards Applied Mathematics Series*, vol. 12, pp. 27–30, 1951.

[107] J. Norden, M. O'Kelly, and A. Sinha, "Efficient Black-box Assessment of Autonomous Vehicle Safety," *arXiv e-prints*, arXiv:1912.03618, arXiv:1912.03618, Dec. 2019. arXiv: `1912.03618 [cs.LG]`.

[108] Z. Huang, Y. Guo, M. Arief, H. Lam, and D. Zhao, "A versatile approach to evaluating and testing automated vehicles based on kernel methods," in *American Control Conference (ACC)*, 2018.

[109] R. Y. Rubinstein and D. P. Kroese, *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte Carlo Simulation and Machine Learning*. Springer Science & Business Media, 2013.

[110] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, no. 1, pp. 19–67, 2005.

[111] M. O'Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi, "Scalable end-to-end autonomous vehicle testing via rare-event simulation," in *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[112] D. Zhao, H. Lam, H. Peng, S. Bao, D. J. LeBlanc, K. Nobukawa, and C. S. Pan, "Accelerated evaluation of automated vehicles safety in lane-change scenarios based on importance sampling techniques," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 3, pp. 595–607, 2016.

[113] Z. Huang, H. Lam, D. J. LeBlanc, and D. Zhao, "Accelerated evaluation of automated vehicles using piecewise mixture models," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 9, pp. 2845–2855, 2017.

[114] S. Sankaranarayanan and G. Fainekos, "Falsification of temporal properties of hybrid systems using the cross-entropy method," in *ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2012.

[115] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992, vol. 1.

[116] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, 3. MIT press Cambridge, MA, 2006, vol. 2.

[117] C. Jidling, N. Wahlström, A. Wills, and T. B. Schön, "Linearly constrained Gaussian processes," in *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[118] Z. I. Botev, "The normal law under linear restrictions: Simulation and estimation via minimax tilting," *Journal of the Royal Statistical Society: Series B*, vol. 79, no. 1, pp. 125–148, 2017.

[119] L. Breiman, *Classification and Regression Trees*. Routledge, 2017.

[120] I. D. J. Rodriguez, T. Killian, S.-H. Son, and M. Gombolay, "Interpretable reinforcement learning via differentiable decision trees," *arXiv*, no. 1903.09338, 2019.

[121] H. Schielzeth, "Simple means to improve the interpretability of regression coefficients," *Methods in Ecology and Evolution*, vol. 1, no. 2, pp. 103–113, 2010.

[122] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[123] D. Hein, A. Hentschel, T. Runkler, and S. Udluft, "Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies," *Engineering Applications of Artificial Intelligence*, vol. 65, pp. 87–98, 2017.

[124] R. Lee, M. J. Kochenderfer, O. J. Mengshoel, and J. Silbermann, "Interpretable categorization of heterogeneous time series data," in *SIAM International Conference on Data Mining*, 2018.

[125] M. Vazquez-Chanlatte, J. V. Deshmukh, X. Jin, and S. A. Seshia, "Logical clustering and learning for time-series data," in *International Conference on Computer Aided Verification*, 2017.

[126] G. Agha and K. Palmskog, "A survey of statistical model checking," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 28, no. 1, pp. 1–39, 2018.

[127] J. P. Chryssanthacopoulos, M. J. Kochenderfer, and R. E. Williams, "Improved Monte Carlo sampling for conflict probability estimation," in *AIAA Non-Deterministic Approaches Conference*, 2010.

[128] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey.," *Journal of Machine Learning Research*, vol. 10, no. 7, pp. 1633–1685, 2009.

[129] J. Rajendran, A. S. Lakshminarayanan, M. M. Khapra, P. Prasanna, and B. Ravindran, "Attend, adapt and transfer: Attentive deep architecture for adaptive transfer from multiple sources in the same domain," in *International Conference on Learning Representations (ICLR)*, 2017.

[130] J. K. Rosenblatt, "Optimal selection of uncertain actions by maximizing expected utility," *Autonomous Robots*, vol. 9, no. 1, pp. 17–25, 2000.

[131] M. Bouton, K. D. Julian, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer, "Decomposition methods with deep corrections for reinforcement learning," *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, vol. 33, no. 3, pp. 330–352, 2019.

[132] J. P. Chryssanthacopoulos and M. J. Kochenderfer, "Decomposition methods for optimized collision avoidance with multiple threats," *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 2, pp. 398–405, 2012.

[133] M. E. Taylor, P. Stone, and Y. Liu, "Transfer learning via inter-task mappings for temporal difference learning," *Journal of Machine Learning Research*, vol. 8, no. 9, pp. 2125–2167, 2007.

[134] M. E. Taylor, G. Kuhlmann, and P. Stone, "Autonomous transfer for reinforcement learning.," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.

[135] H. v. Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[136] K. D. Julian, M. J. Kochenderfer, and M. P. Owen, “Deep neural network compression for aircraft collision avoidance systems,” *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 3, pp. 598–608, 2019.

[137] X. Wang, S. Nair, and M. Althoff, “Falsification-Based Robust Adversarial Reinforcement Learning,” *arXiv e-prints*, arXiv:2007.00691, arXiv:2007.00691, Jul. 2020. arXiv: `2007.00691 [cs.RO]`.

[138] D. Ellis, E. Sommerlade, and I. Reid, “Modelling pedestrian trajectory patterns with gaussian processes,” in *International Conference on Computer Vision (ICCV)*, 2009.

[139] C. Sitawarin, A. Nitin Bhagoji, A. Mosenia, M. Chiang, and P. Mittal, “DARTS: Deceiving Autonomous Cars with Toxic Signs,” *arXiv e-prints*, arXiv:1802.06430, arXiv:1802.06430, Feb. 2018. arXiv: `1802.06430 [cs.CR]`.