Inside the Box:

Analysing Cyber-physical Systems, Exploiting Models and Specifications

by

Tanmay Bhaskar Khandait

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved October 2022 by the
Graduate Supervisory Committee:

Giulia Pedrielli, Co-Chair
Georgios Fainekos, Co-Chair
Nakul Gopalan

ARIZONA STATE UNIVERSITY

December 2022

ABSTRACT

The notion of the safety of a system when placed in an environment with humans and other machines has been one of the primary concerns of practitioners while deploying any cyber-physical system (CPS). Such systems, also referred to as safety-critical systems, needs to be exhaustively tested for erroneous behavior. This generates the need for coming up with algorithms that can help ascertain the behavior and safety of the system by generating tests for the system where they are likely to falsify.

In this work, three algorithms have been presented that aim at finding falsifying behaviors in cyber-physical Systems. PART-X intelligently partitions while sampling the input space to provide probabilistic point and region estimates of falsification. PySOAR-C and LS-emiBO aims at finding falsifying behaviors in gray-box systems when some information about the system is available. Specifically, PySOAR-C aims to find falsification while maximising coverage using a two-phase optimization process, while LS-emiBO aims at exploiting the structure of a requirement to find falsifications with lower computational cost compared to the state-of-the-art. This work also shows the efficacy of the algorithms on a wide range of complex cyber-physical systems. The algorithms presented in this thesis are available as python toolboxes.

ACKNOWLEDGMENTS

I would first like to express my sincere gratitude to my mentors and the committee co-chairs, Dr. Georgios Fainekos and Dr. Giulia Pedrielli, and my committee member Dr. Heni Ben Amor for giving me various opportunities to work on the exciting ideas throughout my master's graduate program which culminated into this thesis. They have played a huge role in supporting me, guiding me, and helping me chalk up and explore various ideas some of which have culminated with the work presented in this thesis.

Most of the work presented in this thesis would not have been possible without the help of the team-mates Jacob Anderson, Aniruddh Chandratre, Surdeep Chotaliya, Keyvan Majd, Tomas Hernandez, Mina Jiang, and Quinn Thibeault. I would like to especially thank Aniruddh Chandratre, who was my friend before I joined the lab, and helped me with coming up to speed with the research work, as well as the countless discussion we had on a wide range of topics. I also had great pleasure working with Keyvan Majd, who was always supportive of the work that I did and who was my go-to person to discuss when I had to make any decision related to my future.

The two years of my master's degree would not have been possible without the support of my friends who I made here. From late-night study sessions to hiking some of the toughest trails in and around Arizona, to skiing from the top of the mountains, you all made these two years fun and memorable.

Finally, I would like to express my profound gratitude to my parents for providing me with unfailing support when I found myself in a fix and continuous encouragement through my major and minor successes in my research process, my education, and my life.

This work is supported by the DARPA ARCOS program under contract FA8750-20-C-0507.

TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

"Automation may be a good thing, but don't forget that it began with Frankenstein."

-Anonymous

Today, automation has found its place in tasks as trivial as switching on a light bulb to non-trivial tasks like exploring the unknown parts of the universe. Whether we realize it or not, it has become an indispensable part of our lives. This widespread and rapid automation has been possible, in part, due to the ever-growing scale and complexity of embedded systems and code. Today, it is pretty normal to see a few million lines of code spread across hundreds of microprocessor-based electronic control units (ECUs) in even a low-end car. Features like Adaptive cruise control and emergency braking, which were once considered a luxury, are now becoming an integral part of cars [2].

Any physical system which can be controlled by a computer is considered a cyber-physical system (CPS). For instance, here are two examples of CPS: (i) consider an automatic thermostat installed in a room that can sense temperatures and automatically switch to heating or cooling. In this case, we have sensors that sense the environment, an onboard processor that processes the information and controls the thermostat (the physical system) to heat or cool the room in order to maintain the optimal temperature [3], and (ii) consider an adaptive cruise control in a car, where the sensors sense the environment (distance to the car and speed, etc) and the onboard computer uses this information to accelerate or decelerate the car to maintain a safe distance and follow the speed limit [4]. Both of these are examples of physical

systems that are controlled by computers and thus are cyber-physical systems.

CPS is being increasingly deployed in a variety of fields including manufacturing, healthcare, smart grids, transportation, etc [5]. Safety-critical CPS is a system whose safety is of utmost importance, and failure of which can lead to catastrophic damage to human lives and infrastructure. In the context of safety-critical systems, CPS has found immense application in various fields including healthcare, aerospace, and transportation. Some examples include safety-critical CPS including implantable artificial pancreas and Pacemaker, F16 - Ground Collision Avoidance System (GCAS), and Boeing Maneuvering Characteristics Augmentation System (MCAS), which are described below.

1. **Artificial Pancreas**: The artificial pancreas CPS aims at partially or fully automating the process of delivering insulin to the body. A continuous glucose sensor periodically senses blood glucose levels subcutaneously and the delivery of insulin by an insulin pump is controlled using an onboard closed-loop controller to control the glucose levels to a certain target [6].

2. **Pacemakers**: Pacemakers are often implanted to enable efficient functioning of the heart. The core goal is to deliver timely electrical pulses to the heart to maintain an appropriate heart rate and Atrial-Ventricular synchrony [7].

3. **F16-GCAS**: Put into force in 2014 on the F-16 platform, GCAS is another example of CPS, which takes over the control of an aircraft when it predicts an imminent collision with the ground. Once detected, a fly-up maneuver is initiated to deviate the flight away from danger. Once the danger is mitigated, the control is given back to the user. [8, 9]

4. **Boeing-MCAS**: Boeing 737 MAX was the successor of Boeing 737, where larger fuel-efficient engines were introduced on the wings. This however led to a

change in the aircraft's characteristics, such that accelerating the aircraft would lead to an increasing angle of attack eventually leading to a stall. In order to mitigate the risks, Boeing 737 MAX was fitted with the MCAS system, which took control of the aircraft when it detected an imminent threat, and lowered the angle-of-attack to avoid a stall [10].

The growing adoption of CPS implies increased interaction of humans with them. It is obvious that failure of any of these systems can lead to catastrophic disasters to both human life and infrastructure. This, along with the growing code complexity and increasing interaction with CPS calls for developing frameworks to evaluate the safety of a CPS.

The safety of a system can be ascertained through three methods: (i) verification, (ii) testing, and (iii) falsification [11]. Consider a model $\mathcal{M}$ and a requirement φ (often related to safety). First, verification refers to proving that the model $\mathcal{M}$ is robust against the requirement φ. Second, testing refers to finding parameters and inputs to model $\mathcal{M}$ such that the system is robust against the requirement φ. And third, falsification refers to finding parameters and inputs to model $\mathcal{M}$ that can produce a failure of the system to follow the requirement φ. While the difference between verification and testing is apparent, the difference is subtle when comparing them to falsification. The objective of testing is to find inputs and parameters that can satisfy the requirement from a finite set, while falsification aims to find inputs and parameters that can invalidate the requirement, from a possibly infinite set. Verification and falsification, on the other hand, are complementary to each other [11]. To put it simply, finding falsifying behaviors can show that the system cannot be verified (wrt the definition above).

In this thesis, the focus is on coming up with falsification algorithms and showing their efficacy and efficiency by exhaustively experimenting with various benchmarks.

Based on the information available about the system, a CPS can be viewed as a black-box system when no information is available about the dynamics of the system, or as a gray-box system when some information is available.

In the case of black-box systems, while coming up with inputs that lead to falsifying behavior is useful, often nothing can be said about the system if no such inputs are found. While one can argue in favor of a more exhaustive search, it is important to note that simulating such systems is often an expensive task. Thus, providing probabilistic estimates of a smaller subregion can help a practitioner, even if no falsifying inputs are found. One straightforward application is that the practitioner can utilize subregion with a higher probability of falsification and then run an exhaustive search to find falsifying behaviors. We tackle these kinds of problems using the Part-X algorithm, which is an adaptive branch-and-bound-based level-set estimation algorithm. On the other hand, if we have some knowledge about the structure of the problem, it becomes helpful to use this prior information to find falsifications. The information can be available in two forms. First, knowledge about the hybrid automaton modeling of the system can often help in exploring areas of the state-space that are less accessible and might possibly contain falsification, thus giving an advantage to the search method to find falsifying inputs. With respect to this problem, the thesis utilizes the PySOAR-C algorithm, a modified version of the algorithm proposed in [12, 13] to incorporate hybrid distances. Second, if we know the structure of the requirements, we can utilize this information to exploit and direct the sampling process. With respect to this problem, the thesis utilizes the Large Scale Extended Minimum Bayesian Optimization (LS-emiBO) algorithm that exploits the structure of conjunctive requirements and provides an efficient heuristic to the sample points.

1.1 Research Questions

This thesis seeks to answer three major research questions.

RQ1. In black-box systems, when no information is available, can we find falsifying behaviors, while also providing probabilistic estimates of a point leading to falsification as well as regions that can lead to finding inputs that produce a falsifying behavior?

RQ2. In gray-box systems, when we have some knowledge about the dynamics of the system-under-test, can we come up method that can utilize this information in order to find inputs that lead to falsifying behaviors?

RQ3. If we have some knowledge of the structure of the requirement, can we exploit the structure in order to efficiently come up with faster falsification?

1.2 Contributions

This work outlines and proposes three different algorithms:

1. When no information is available about the underlying system, the PART-X algorithm is able to estimate the $0-$level set of the robustness function against a certain requirement by adaptively branching and intelligently sampling in the search space.

2. When information about the dynamics of the system is known, the PySOAR-C algorithm proposes to find falsifying behaviors with respect to a certain requirement by minimizing robustness and maximizing state coverage.

3. When the structure of the requirement is in the form of a conjunctive requirement, the LS-EMIBO algorithm tends to find falsifying behaviors by sampling over the likely minimum sub-requirements.

In this thesis, the proposed algorithms are tested on a variety of problems, including the benchmarks utilized in the friendly ARCH Competition [14].

PART-X, PySOAR-C, and LS-emiBO are provided as both stand-alone packages as well as add-on packages to the PSY-TaLiRo SBTG tool.

- PART-X is available at `https://gitlab.com/bose1/part-x`.
- PySOAR-C is available at `https://gitlab.com/sbtg/pysoar-c`.
- LS-emiBO is available at `https://gitlab.com/sbtg/LS-emiBO`.

1.3 Structure

The organization of the thesis is as follows. Chapter 2 gives a brief overview of the background of the thesis and introduces some key concepts that will be used extensively. Chapter 3 gives an exhaustive overview of the research happening in the literature along with a comparison with the methods proposed. This is followed by three chapters on three different algorithms where the working is described along with a numerical experiment, such that: Chapter 4 talks about the PART-X, Chapter 5 talks about the PySOAR-C, and Chapter 6 talks about the LS-emiBO algorithm. The thesis then shows the results from using PART-X and LS-emiBO on complex cyber-physical systems from [14] in Chapter 7 followed by the conclusion in Chapter 8.

Chapter 2

BACKGROUND

2.1 System-under-test and Falsification

System-under-test (SUT) can be defined as a complete system, a cyber-physical system in our case, which is the target of a performance measurement or a benchmarking test. Based on the information available about the internal dynamics of the SUT, it can be classified into three types of models: (i) a white box model, where we have complete information about the system, (ii) a black-box model, where we do not have any information about the internal working of the model except the inputs and corresponding output of the system, and (iii) a gray-box model, where we have partial information about the system. The falsification task, as discussed in chapter 1, refers to finding inputs to the SUT such that the corresponding output violates a certain requirement. This thesis introduces algorithms that deal with the falsification task for both black-box and gray-box models.

This chapter gives a brief overview of black-box models and how a specification is written and evaluated to obtain a degree of satisfaction with the requirement. Since gray-box models can have different interpretations depending on the information available, we discuss the gray-box models in the context of algorithms in their respective chapters. The chapter then gives a brief overview of the PSY-TaLiRo toolbox followed by a discussion on Search-Based Test Generation (SBTG) methods.

2.1.1 Black-box Models

Formally, the SUT in the form of a black-box model can be represented as an input/output function such that, given a black-box model $\mathcal{M}_{BB}$, it can be represented as:

$$\mathcal{M}_{BB} : \big(\boldsymbol{u} : ([0,T] \to \mathbb{R}^d)\big) \to (\boldsymbol{v} : ([0,T] \to \mathbb{R}^n)), \tag{2.1}$$

where $\boldsymbol{u} : ([0,T] \to \mathbb{R}^d)$ is a real-valued input signal. Upon simulation of the $\mathcal{M}_{BB}$, a real-valued output signal $\boldsymbol{v} : ([0,T] \to \mathbb{R}^n)$, which could have a different dimensionality compared to the input signal. The input signal $\boldsymbol{u}$ is sampled from a bounded convex region $\boldsymbol{S} \subseteq \boldsymbol{R}^d$, i.e., $\boldsymbol{u} \in \boldsymbol{S}$, which in this work is considered to be a hypercube. Throughout the thesis, the inputs and outputs refer to timed-trajectory signals or just signals.

2.1.2 Requirement and Metrics of Distance

In order to express the requirement against which the system is being tested, Signal Temporal Logic [15] is used throughout. The syntax of an STL formula is given by

$$\varphi ::= \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \,\mathcal{U}_{\mathcal{I}}\, \varphi \mid \Box_{\mathcal{I}}\, \varphi \mid \Diamond_{\mathcal{I}}\varphi \tag{2.2}$$

In the given syntax, the logical connectives and the temporal operators have their classical Boolean interpretation and equivalences. $\mathcal{I}$ represents the interval of time over which the associated temporal operator is checked for. Qualitatively, $\boldsymbol{x} \models \varphi$ signifies that the requirement is satisfied with respect to a signal $\boldsymbol{x}$, while $\boldsymbol{x} \not\models \varphi$ signifies the violation. Quantitatively, the degree of satisfaction or violation is referred to as the robustness [16] of the system against a certain specification. Fundamentally, the robustness is a distance metric between a point $y \in Y$ and a set $S \subset Y$ [17]. Given

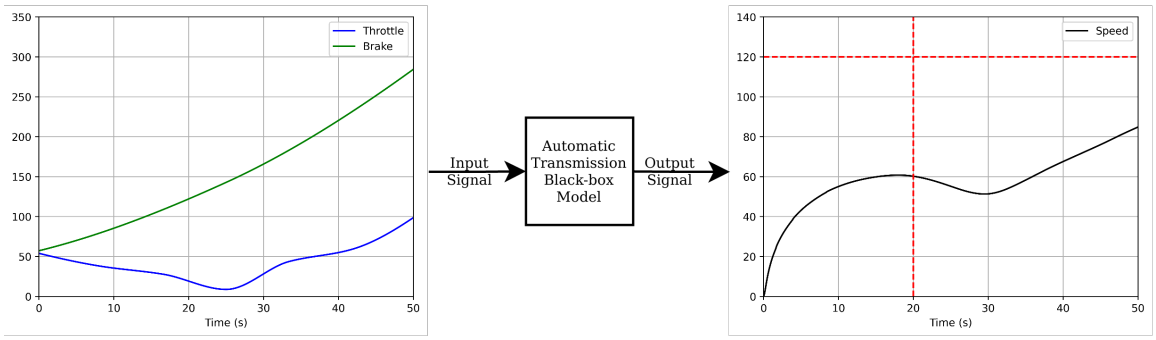
a certain model $\mathcal{M}$ and a requirement φ, the robustness of a signal $\boldsymbol{u}$ can be denoted as $\rho_\varphi : ([0, T] \to \mathbb{R}^n) \to \mathcal{R} \cup \{-\infty, \infty\}$, where robustness is a real-valued quantity. The signal $\boldsymbol{u}$ is said to violate a requirement φ when the robustness $\rho_\varphi \leqslant 0$. Otherwise, the specification is satisfied [11]. Thus the task of finding falsifying inputs to a model, at least in the black-box setting, reduces to solving

$$\boldsymbol{u} = \arg\min_{\boldsymbol{u}} \rho_\varphi(\mathcal{M}(\boldsymbol{u})) \tag{2.3}$$

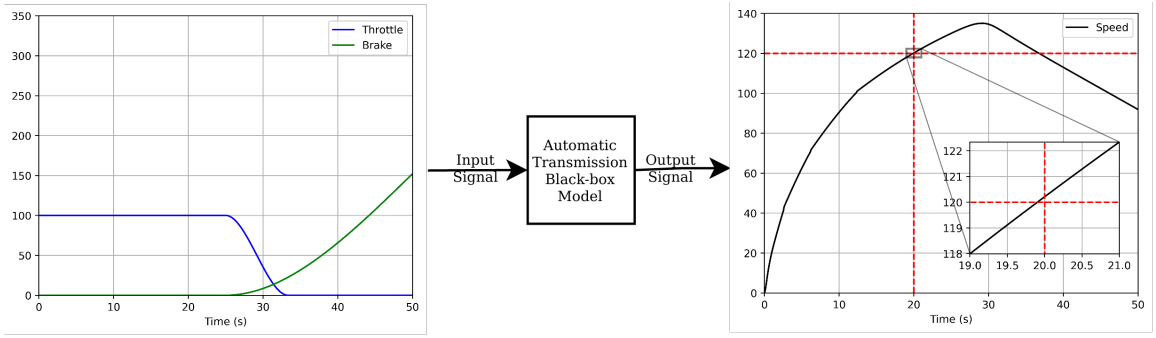
In practice, there are multiple off-the-shelf libraries that can be used to calculate the robustness when provided the timed trajectory and the requirement. In this work, RTAMT [18] and TLTk [19] were used, which are available as a part of the Search Based Generation Tool (SBTG) PSY-TaLiRo. To read more about how the robustness is calculated with respect to different logical and temporal operators, the reader is directed towards [16, 15, 17].

Example

Consider a black-box model of an automatic transmission controller which takes as input the throttle and brake at a certain time t and returns the current speed of the car. The requirement of the system is such that the in the first 20 seconds, the speed should never exceed $120kmph$. This can be defined in STL as $\varphi = \square_{[0,20]}(\text{speed} <= 120)$. The task of falsification in this case would be to find an input signal from time $t = 0s$ to $t = 50s$, such that the speed goes over $120kmph$ in the first 20 seconds. On observing the signals in fig. 2.1, it can be observed that on giving the input signal to the black-box model as shown in fig. 2.1a, the corresponding output signal output does not go above the $120kmph$ mark during the first $20s$. The behavior after $20s$ does not affect the value of robustness. This is a non-falsifying behavior since the requirement is satisfied. On the other hand, in fig. 2.1b, the output signal denoting



(a) An input which satisfies the requirement: $\rho_\varphi(\mathcal{M}(\boldsymbol{x})) = 59.250$



(b) An input which falsifies the requirement: $\rho_\varphi(\mathcal{M}(\boldsymbol{x})) = -0.213$

Figure 2.1: Examples of two input signals and their corresponding output. Figure 2.1a is a non-falsifying input such that it produces an output that satisfies the requirement $\varphi = \square_{[0,20]}(\text{speed} <= 120)$. However, fig. 2.1b is a falsifying input because the corresponding output signal falsifies the requirement.

the speed corresponding to the input signal exceeds $120kmph$ right before $20s$. This is a case when the input signal led the system to produce a falsifying behavior. This can also be validated by noticing the corresponding robustness values: the falsifying output signal has a non-positive robustness while non-falsifying output signal has a positive robustness value.

2.2 SBTG Methods

Search-based Test Generation (SBTG) Methods are a class of methods that aim at searching for falsifying inputs. Concretely, given a SUT $\mathcal{M}$ and a requirement φ, the aim is to find an input signal $\boldsymbol{u}$ such that the robustness is $\leqslant 0$ as defined in eq. (2.3).

A variety of methods for SBTG have been developed to solve eq. (2.3) that range from tree exploration to black-box optimization based methods and related variations of these techniques. More recently, reinforcement learning methods are also being explored for SBTG. The basic idea behind SBTG is to find a falsifying inputs in as less simulation as possible. If no falsifications are found, the algorithm is terminated after an allotted budget of evaluations is exhausted. However, these methods lack in reasoning about the falsification if no falsifications are found. We discuss this in detail in chapter 5.

2.2.1 PSY-TaLiRo

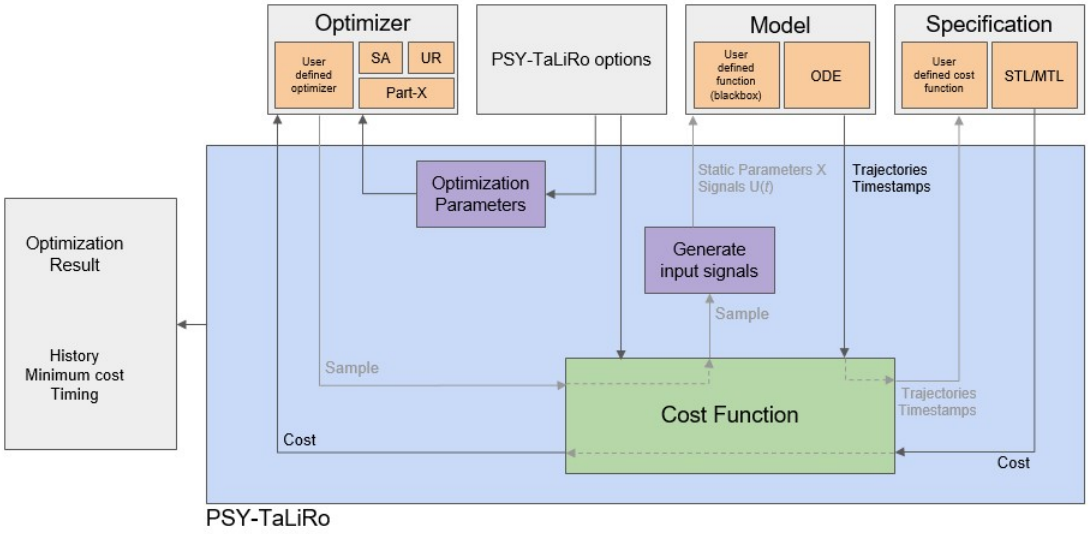


Figure 2.2: PSY-TaLiRo Architecture

PSY-TaLiRo [20], the python version of S-TaLiRo [21], is an open-source toolbox for temporal logic robustness-guided falsification of Cyber-Physical Systems (CPS). The architecture[1] of PSY-TaLiRo is visually shown in Figure 2.2. The user de-

[1] `https://sbtg.gitlab.io/psy-taliro/architecture.html`

fines the model (black-box or gray-box), specification in STL and the optimizer. The optimizer then computes sample points using which the system is simulated. The generated trajectories from the model are sent back to evaluate their robustness againt the STL specification. The optimizer then produces the next sample to repeat the process. RTAMT [18] and TLTk [19] are used for parsing and computing the robustness against a certain specification. This completely modular toolbox helps in the generation of test cases for falsification of a SUT using a common interface for temporal logic monitors. While the toolbox provides inbuilt optimizers (DA, Uniform Random etc.), one can also pass custom optimizers to PSY-TaLiRo. The tool box is publicly available on-line under General Public License (GPL) [2]. In this thesis, the algorithms are integrated with PSY-TaLiRo to develop end-to-end python toolboxes.

[2] `https://gitlab.com/sbtg/psy-taliro`

Chapter 3

REVIEW

3.1 Introduction

In this chapter, the initial part exhaustively discusses the methods proposed in the falsification community and then discusses the relationship to the work algorithms in this thesis.

As mentioned before, cyber-physical systems (CPS) are usually hybrid systems that are an intermix of continuous and discrete dynamics [22]. These hybrid systems can be treated as black-box systems when we do not have any information about the system-under-test (SUT). On the other hand, if we have some prior knowledge about the dynamics or the structure of the requirement, we can utilize such knowledge to test these gray-box systems. In both cases, our goal is to find falsifying behaviors of the SUT.

The robustness of the continuous state of a CPS against a certain specification can be quantitatively measured using Signal Temporal Logic (STL) [16]. To measure the robustness of a certain specification from the discrete state, if known, is computed in order to give information about both the continuous and discrete state present in the system [17]. In both cases, the system is considered to be falsified against a certain requirement when the robustness is less than or equal to zero. Thus, the task of finding falsifying behaviors in a CPS corresponding to a certain requirement is equivalent to solving global optimization problems, where the objective is to minimize the robustness of the system corresponding to that requirement. However, it is important to note that when looked at from the perspective of optimization, the surfaces are

notoriously known to have highly non-linear, non-convex, and often discontinuous optimization surfaces. Several methods were proposed to find falsifying behaviors, and the following is an attempt to categorize the research that has taken place in the falsification community.

3.2 Global Optimization Methods

The first set of approaches solves the global optimization problem using an optimization procedure. The tricky optimization surface can yield various local minima which might not exhibit a falsifying behavior, and thus it becomes important that there is a balance between exploration and exploitation. Various algorithms have been proposed that used meta-heuristic algorithms like Ant Colony Optimization (ACO) [23, 21], Monte-Carlo Approaches [24, 21], Nelder-Meads [15], and Tabu Search [25] to successfully solve the problem. However, these algorithms can only provide a range of test points where falsifying behaviors can be seen, but not comment upon the falsifiability if no falsifying behaviors are found. A combination of global and local search algorithms has also been used to find falsifying behavior. The two phase-method for general global optimization [26] was modified to find the falsifying behaviors in [27]. This approach searches for various segments of trajectories that could produce falsifying behaviors in the global search phase. These segments of trajectories were then joined using a derivative-based continuous local search. The peculiarity of this proposed approach was in its ability to find signals of arbitrary length that could produce falsifying behaviors. In [28], the global search phase iteratively divides regions based on a classifier and then samples in these subregions using biased sampling and singularity sampling. If no falsifications are found in the global phase, local search methods search for falsifying inputs using the CMA-ES algorithm. Several gradient-based methods were also proposed that could solve the global optimization problem.

[29] proposed an algorithm where they find points in the system trajectory that affect the robustness corresponding to a certain specification. They then proposed to find falsifying behaviors by providing theoretical proofs of descent direction that could reduce the robustness of the trajectory. As we discussed before, SA offers the advantage of balancing exploration and exploitation efficiently, however, the tricky optimization surface can lead to slow convergence. Finding gradients of the robustness can also be a challenging task due to the inaccessibility of information about the SUT. [30] proposed a method where they find approximated descent direction by linearizing the system and show that these descent directions can closely approximate the computed descent directions without the need for information about the structure of the problem. These descent directions along with the points of interest (found using SA) are used to form a mixture of global and local searches. [31] was an improvement over [30] where they computed the local descent direction in the local search phase. Similarly, [32] proposed an approach where the time-varying inputs to the signal were parameterized in both space and time, and the gradient descent (GD) was employed to change the amplitude and the switching time of the input signal to find falsifying behaviors.

3.3 Meta-Model Based Approaches

The second set of approaches solves the global optimization problem by employing meta-models. [33] aimed at solving the problem of parameter mining where the goal was to find parameters (either in models or requirements), in order to ascertain the robust functioning of the system. In other words, this corresponded to maximizing the robustness (in expectation). They successfully used the GP-UCB algorithm where they fit a surrogate model over the true points and then sample points from a grid using the GP-UCB strategy and demonstrated its usefulness on three different CPS

benchmarks. However, GP-UCB suffered from the problem of exploration due to the presence of only the covariance term. [34] solved this problem by introducing a normalization term to propose the GP-ACB method. While [33, 34] solved problems not related to the scope of the thesis, these works were included as they were the initial approaches to purely use gaussian processes as meta-models in the context of CPS. Coming back to the falsification procedure, GP-UCB along with domain estimation was used as a strategy to find falsifying behaviors [35]. The problem is posed in a way such that the satisfiability of an antecedent requirement implies satisfaction of safety constraint. GP-UCB was used to guide the search toward the antecedent constraint in order to find normal and falsifying behaviors. Another approach was to construct the probabilistic semantics of the specification, thus identifying parts of the specification that falsify with higher probability [36]. GP-UCB was then used in order to find falsifying behaviors.

Bayesian Optimization (BO) methods have also been widely used in finding falsifying behaviors of SUT. [37] introduced model reduction techniques and coupled them with BO in order to find falsifying behaviors on a wide range of industrial problems. [38, 39] are some other works on similar lines. [40] proposed an approach to use partitioning to identify the falsifying level sets. The partitioning and sampling parameters are provided as inputs, and the algorithm iteratively uses conformal regression to obtain an estimate of the maximum and minimum robustness with respect to a certain specification within a subregion. Based on such predictions, a subregion can be deemed unsafe, safe, or unknown. If a region is deemed unsafe, it is further branched. The proposed method allows deriving a probabilistic guarantee over a subregion when it is classified as safe or unsafe. [12, 13] came up with an approach that utilized a combination of global and local sampling algorithms in order to find falsifying behaviors. In the global search phase, the algorithm fused the expected improvement

acquisition function and the crowding distance function in order to generate a point proposal that was not seen before and further away from the points already sampled. Using this proposal, an iterative trust region-based method was employed to guide the search toward falsifying behaviors in the local phase. An adaptive restart mechanism is also provided in the local phase in order to ensure that the evaluation of the systems does not waste the simulation budget. Another approach was also developed that exploited the structure in conjunctive requirements. In conjunctive requirements, sub-requirements can be generated by taking all the components joined with a conjunction operator, and falsifying even one of these sub-requirements corresponds to falsifying the entire requirement. [41] proposed an approach called the minBO where the pairwise Expected Improvement (EI) acquisition function was devised to look at pairs of sub-requirements and samples a point with respect to the pair with minimum EI. This helps the algorithm in looking at components from a finer point of view and avoids the masking effect of certain sub-requirements. In another related work, a method was proposed to estimate the confidence measure of the system not falsifying [42]. The method first collects the samples and their computed robustness values and then models a surrogate over this data. They then estimate the points with a high probability of lower robustness values using a global search algorithm and run a local search from these points. If no falsification is found, the method still returns an estimate of the confidence measure.

3.4 Tree Exploration Methods

The third set of approaches uses tree exploration methods to find falsifying behaviors. Motion Planning algorithms often utilize tree exploration to sample trajectories in the context of robotic applications [43]. Computation of falsifying behaviors can also be considered as a task of finding trajectories from a certain set to an unsafe set.

This notion has been used in a variety of papers to find falsifying behaviors. Motion Planning algorithm was combined with discrete search to create a multi-layered algorithm to find falsifying behavior in [44]. Algorithms based on Rapidly-exploring Random Tree (RRT) with coverage-based guided sampling were proposed as a tool to find falsifying behaviors in [45, 46]. Monte-Carlo Tree Search (MCTS) was interleaved with hill climbing optimization, where MCTS along with hill-climbing was first used to find promising trajectories and another hill climbing optimization was performed on these trajectories to find falsifying behaviors [47]. FalStar [48] uses a tree-exploration-based method in a rather different way. It uses the adaptive Las Vegas Search tree (aLVTS) to develop an input signal gradually by first defining and exploring coarse signal and then moving towards finer-grained trajectories, thus exploiting the time-causal dependencies for a given SUT. Something peculiar about [48] is that it considers the structure of the problem into account, but neither relies on any insight into the model nor uses any sophisticated optimization algorithms. Another algorithm that exploits the structure of the requirement is the ForeSee tool [49], where the authors define a new robustness measure called the QB robustness and combine it with MCTS and hill-climbing to find falsifications. The QB robustness measure combines quantitative robustness and classical Boolean satisfaction, thus eliminating the need to compute minimums and maximums typically used during robustness calculation. In [50], the authors propose latent action Monte Carlo Tree Search (LA-MCTS) which drives the simulation budget by iteratively splitting the input space to identify safe and unsafe regions. Various approaches that use Reinforcement Learning (RL) have been proposed as well to find falsifying behaviors [51, 52].

3.5 Model Abstraction Methods

The fourth set of approaches deals with using model abstraction paired with an optimization algorithm to find falsifying behaviors. Model abstraction and approximation have been studied and shown to produce accurate approximations, which can be used for model checking and verification [53]. A common theme amongst such methods is to create an abstraction and find counterexamples. If these counterexamples turn out to be spurious, the abstract models are refined, and if not, they are reported back to the user. [54] constructs an abstract graph that approximates the underlying system. Counterexamples are found on this abstract graph and checked on the true system to verify if these samples are spurious or not. If they are spurious, the graph is refined and the process is repeated again. In the case that no counterexamples are found on the abstract graph, a new abstract graph is created, and the process is repeated again. ARIsTEO [55] on the other hand approximates the model by creating a surrogate of true inputs and outputs. Falsifying behaviors on this abstraction are then found, and if none or spurious falsifying behaviors are found, the surrogate model is refined. The surrogate modeling is done using the System Identification toolbox [56] in MATLAB. Another automated testing tool based on active automata learning and model checking is FalCAuN [39], where the model-under-test is converted into a black-box Mealy machine. FalCAuN learns this mealy machine and runs optimization procedures to find falsifying behaviors.

Research within the falsification community has also fostered a number of tools to find falsifying behaviors. The friendly ARCH Competition [14] witnesses the participation of various tools ([21, 15, 55, 39, 49, 48, 51]) on wide range benchmarks and requirements with a wide range of difficulties.

3.6 Comparison

With respect to our algorithm Part-X, while the aim is to find falsifying behaviors, we also want to provide probabilistic estimates of finding falsifying behavior, both in terms of points as well as regions. This problem is essentially trying to estimate the level set of the objective function (robustness in our case). Within the falsification community, the work closely resembles the second set of approaches mentioned in section 3.3. [40] as discussed above attempts to find probabilistic bounds on sub-regions using conformal regression. However, they are unable to produce estimates of the probability of a certain point falsifying, which we believe is important to the practitioner. In [42], the method provides probabilistic point estimates of a finding non-falsifying behaviors, however, they lack in providing information at the subregion level, which is also of high importance to a practitioner. With respect to our algorithms related to gray-box testing, PySOAR-C and LS-emiBO are introduced. PySOAR-C is based on the work proposed in [12, 13] to incorporate hybrid distances and show that the algorithm can quickly produce falsification in a complex system. LS-emiBO, on the other hand, exploits the information available in the requirement, specifically the conjunctive requirements. LS-emiBO looks at the individual components of the conjunctive requirement in order to sample the next promising point. The closest work that compares with LS-emiBO is the work in [41]. However, the difference is from the fact that while minBO samples point based on maximizing the expected improvement from the component with minimum robustness, the work in LS-emiBO considers a weighted version of the expected improvement, which yields faster falsification with less computational resources.

Chapter 4

PARTITIONING WITH X-DISTRIBUTED SAMPLING (PART-X)

4.1 Introduction

This chapter discusses PART-X, a partitioning-based algorithm that relies on local gaussian processes for generating surrogates over the subsets of input space and searching for falsifying behavior. The critical contributions of the algorithm are:

1. Using local surrogate models instead of a single global surrogate model allows the calculation of point estimates within the subregions. This helps in calculating estimates of both the best and the worst inputs to the system, as well as the volume of the falsifying input sets.

2. Presence of a branching criterion to branch the (sub)region allows for keeping the growth of the associated tree in check.

3. Ability to integrate PART-X with other Test Generation Methods in order to find falsifications in regions with a higher probability of falsification.

Complete paper for the PART-X is available at [57].

Given a model $\mathcal{M}$ and a requirement φ, the robustness can be defined as $f(\boldsymbol{x}) = \rho_{\varphi}(\mathcal{M}(\boldsymbol{x})) : \mathbb{R}^d \to \mathbb{R}$. PART-X aims at minimizing this function while also returning an estimate of the level set associated with robustness function $f(\boldsymbol{x}) \leqslant 0$ and the corresponding falsification volume $\mathcal{L}_0$. Using the local surrogate models, the PART-X algorithm also produces a model that can predict the robustness at each input in the input space. Thus, PART-X can help in estimating the probability of the existence of

a falsifying input when no falsifications are found as well as the associated normalized falsification volume. In the remainder of this chapter, the algorithm is explained.

Figure 4.1 provides an overview of the algorithm. The core idea of the algorithm is to sequentially sample points (step 1) and adaptively branch the active subregion using estimates from the surrogate model (step 2), particularly the Gaussian Process Regressor. The Gaussian Process Regressor (discusssed in section 4.3) provides the sampling distribution, which can be used in classifying the region into either positive class, negative class, or undecided (step 3). If a region is classified into positive or negative class, it can be reclassified into either positive, negative or undecided. If a region is classified as undecided or reclassified, the region is branched further in the next iteration, provided the volume after being branched is greater than the minimum branchable volume (step 4). If not, it is not further branched. The algorithm terminates when the budget is exhausted or all the leaf nodes reach the minimum branchable volume.

4.2 Tree Notation and Definitions

Through various iterations in the PART-X algorithm, the most basic structure is the node of a PART-X tree. The i^{th} node of the tree at j^{th} level obtained during the k^{th} iteration of the algorithm having a class $\gamma \in \{+, -, r^+, r^-, r, u\}$ is denoted by $\sigma^{\gamma}_{i,j,k}$. In the context of the algorithm, $\sigma^{\gamma}_{i,j,k}$ denotes a region of the input space with volume $\geqslant 0$ along with the samples that belong to the subregion and their corresponding robustness values. This also means we indirectly have access to the surrogate model in the form of a gaussian process for every subregion. In general, we will refer to $\sigma^{\gamma}_{i,j,k}$ as a **subregion**. At the k^{th} iterations of the algorithm, there will be $N_k = (N_k^+ + N_k^{r+} + N_k^- + N_k^{r-} + N_k^r + N_k^u)$ number of new subregions. During each iteration k, the leaf nodes can assume the following classes:

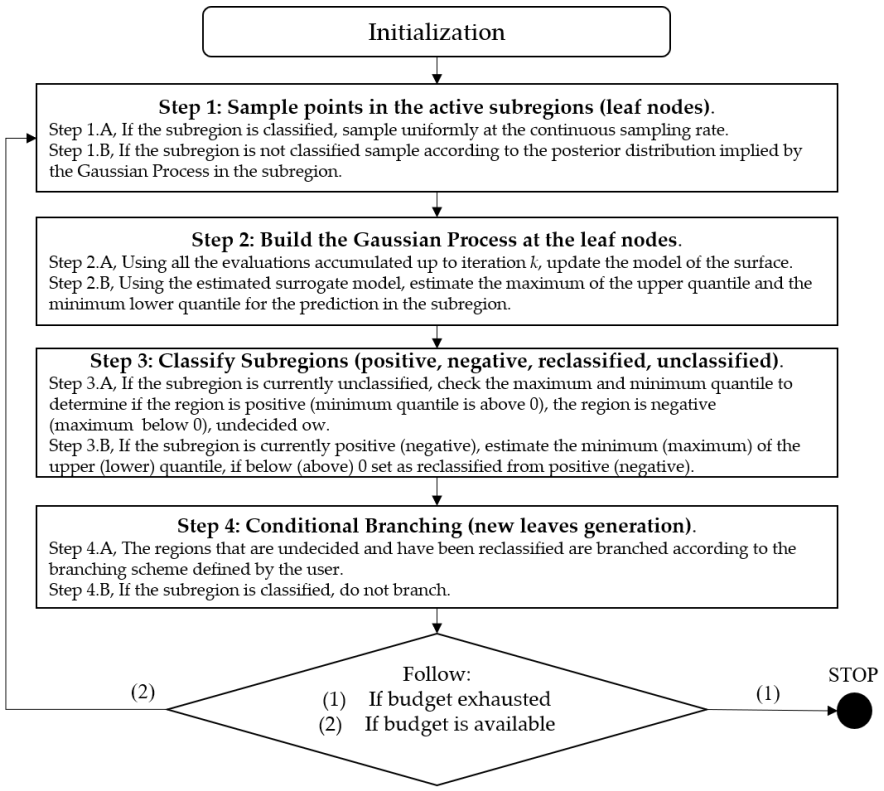


Figure 4.1: Overview of PART-X.

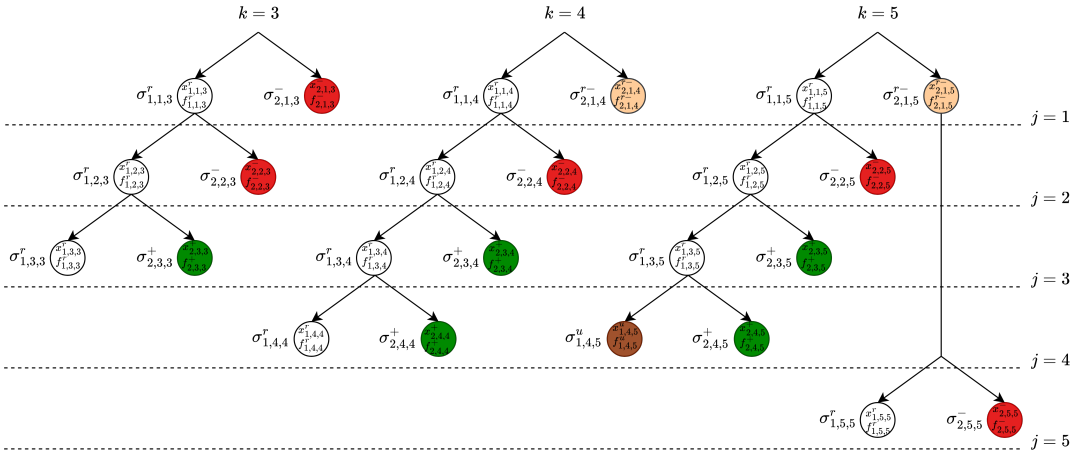


Figure 4.2: An example of partitioning tree generated by PART-X at iteration $k = 3$. Subregions in red are classified as violating, while those in green are satisfying, the orange region is being reclassified from violating to remaining (it would be light green if reclassified from satisfying to remaining). In figure i, j, k indexes refer to the index of leaf i located at the h_j^{th} level of the tree, at the k^{th} iteration of the algorithm. Each leaf is a set $\sigma^{\gamma}_{i,j,k}$ with locations sampled up to iteration k, and the associated robustness values. Thus, each subregion has an associated Gaussian process.

1. New satisfying region ($\sigma^{+}_{j,k}$): The set of new subregions classified as satisfying/positive regions is denoted as $\sigma^{+}_{j,k} = \bigcup_i \sigma^{+}_{i,j,k} \forall j = \{1, 2, \ldots, k\}$ at iteration k formed by the $N^{+}_{j,k}$ new subregions. In fig. 4.2, the leaf of the tree $\sigma^{r}_{1,3,3}$ at iteration $k = 3$ is branched in the subsequent iteration to obtain a satisfying region $\sigma^{+}_{2,4,4}$ (shown in dark green color).

2. New Positive Reclassified Regions ($\sigma^{r+}_{j,k}$): The set of new regions reclassified from positive are denoted as $\sigma^{r+}_{j,k} = \bigcup_i \sigma^{r+}_{i,j,k} \forall j = \{1, 2, \ldots, k\}$ at iteration k formed by the $N^{r+}_{j,k}$ new subregions.

3. New Violating Regions $\sigma^{-}_{j,k}$: The set of new subregions classified as violating is denoted as $\sigma^{-}_{j,k} = \bigcup_i \sigma^{-}_{i,j,k} \forall j = \{1, 2, \ldots, k\}$ at iteration k formed by the $N^{-}_{j,k}$ new subregions. In fig. 4.2, the leaf of the tree $\sigma^{r-}_{2,1,4}$ at iteration $k = 4$ is branched in the subsequent iteration to obtain a violating region $\sigma^{-}_{2,5,5}$ (shown in dark red color).

4. New Negative Reclassified Regions ($\sigma^{r-}_{j,k}$): The set of new subregions reclassified from violating are denoted as $\sigma^{r-}_{j,k} = \bigcup_i \sigma^{r-}_{i,j,k} \forall j = \{1, 2, \ldots, k\}$ at iteration k formed by the $N^{r-}_{j,k}$ new subregions. In fig. 4.2, the leaf of the tree $\sigma^{-}_{2,1,3}$ at iteration $k = 3$ is reclassified from violating (shown in orange color in the subsequent iteration). Notice that if a region is reclassified (either from violating or satisfying), it is branched further (look at $\sigma^{-}_{2,1,4}$ and $\sigma^{-}_{2,1,5}$).

5. New Remaining Regions ($\sigma^{r}_{j,k}$): The set of new subregions classified as remaining is denoted as $\sigma^{r}_{j,k} = \bigcup_i \sigma^{r}_{i,j,k} \forall j = \{1, 2, \ldots, k\}$ at iteration k formed by the $N^{rk}_{j,k}$ new subregions. These subregions are neither satisfying nor violating the property. In fig. 4.2, all those regions that are branched in the subsequent iteration are the new remaining regions (for eg: $\sigma^{r}_{1,3,3}$ at iteration $k = 3$ is

branched in iteration $k = 4$, where the new branched region $\sigma^r_{1,4,4}$ is classified as remaining).

6. New Unclassified Regions ($\sigma^u_{j,k}$): The set of new subregions classified as unclassified are denoted as $\sigma^r_{j,k} = \bigcup_i \sigma^r_{i,j,k} \forall j = \{1, 2, \ldots, k\}$ at iteration k formed by the $N^{rk}_{j,k}$ new subregions. Similar to the remaining regions, these subregions have not been classified into satisfying or violating. However, these subregions are also not branchable in any direction since they reached the minimum branchable threshold. In fig. 4.2, $\sigma^r_{1,4,4}$ at iteration $k = 4$ cannot be branched further and is unclassified in the subsequent iteration.

Following this discussion, it can be easily inferred that: (i) the root of the tree represents the entire search space; (ii) the union of areas denoted by all leaf nodes during any iteration will be the entire search space, and (iii) the intersection of areas denoted by all leaf nodes during any iteration will be the null space. Let us define the set of classified subregions at level j and at iteration k as $\Theta^+_{j,k}, \Theta^{r+}_{j,k}, \Theta^-_{j,k}, \Theta^{r-}_{j,k}, \Theta^r_{j,k}, \Theta^u_{j,k}$ for satisfying, reclassified from satisfying, violating, reclassified from violating, remaining, and unclassified. If we drop the index j, the result will lead to the notation for the same sets at each iteration k.

4.3 Modelling the Robustness as a Gaussian Process

As shown in fig. 4.2, every subregion has an associated gaussian process. Gaussian Processes are statistical methods used for regression or classification. They can be defined as a collection of random variables of which every finite subset has a Gaussian distribution [58]. Given a set of noise-free observations for the function $f(\boldsymbol{x})$, where $\boldsymbol{x} \in S$, a gaussian process $Y(\boldsymbol{x})$ will interpolate the true function at the evaluated points. The gaussian process produces a conditional density $P(Y(\boldsymbol{x}_0)|\boldsymbol{X}_n)$, where $\boldsymbol{X}_n$

is a set of observations $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$. Given the constant process mean μ, constant process variance τ^2, and the correlation matrix R, the gaussian process

$$\begin{aligned} Y(\boldsymbol{x}) &= \mu + Z(\boldsymbol{x}) \qquad\qquad , \\ \text{where} \quad Z(\boldsymbol{x}) &\sim \mathcal{GP}(0, \tau^2 R) \end{aligned}$$

If we are given n points, under the gaussian correlation assumption, the correlation matrix $R_{hm} = \Pi_{l=1}^{l=d} \exp(\theta_l(\boldsymbol{x}_{hl} - \boldsymbol{x}_{ml})^2)$ where $h, m = 1, 2, \ldots, n$ and d = dimensionality. The smoothing intensity of the predictors in different dimensions is controlled by the d-dimensional vector θ. The parameters μ and τ^2 can be estimated using MLE to get the followings estimates [59, 60]:

$$\widehat{\mu} = \frac{\mathbf{1}_n^T R^{-1} f(\boldsymbol{X}_n)}{\mathbf{1}_n^T R^{-1} \mathbf{1}_n} \tag{4.1}$$

$$\tau^2 = \frac{(f(\boldsymbol{X}_n) - \mathbf{1}_n \widehat{\mu}_g)^T R^{-1} (f(\boldsymbol{X}_n) - \mathbf{1}_n \widehat{\mu}_g)}{n} \tag{4.2}$$

And, the Best Linear Unbiased Predictor $\widehat{Y}(\boldsymbol{x})$ and the associated model variance to the predictor $s^2(\boldsymbol{x})$ is [59, 60]:

$$\widehat{Y}(\boldsymbol{x}) = \widehat{\mu} + \boldsymbol{r}^T R^{-1} (f(\boldsymbol{X}_n) - \mathbf{1}_n \widehat{\mu}) \tag{4.3}$$

$$s^2(\boldsymbol{x}) = \tau^2 \left(1 - \boldsymbol{r}^T R^{-1} \boldsymbol{r} + \frac{(1 - \mathbf{1}_n^T R^{-1} \boldsymbol{r})^2}{\mathbf{1}_n^T R^{-1} \boldsymbol{r}} \right), \tag{4.4}$$

where $\boldsymbol{r}$ is the n-dimensional vector that contains the guassian correlation between locations $\boldsymbol{x}_0 \in S$ and the n elements of $\boldsymbol{X}_n$, i.e., $\boldsymbol{r}_i(\boldsymbol{x}_0) = \Pi_{l=1}^{l=d} \exp\left(-\theta_l \left(x_{0l} - x_{hl}^2\right)\right)$.

In the context of PART-X, we use the model in eq. (4.1) as a meta-model for the unknown robustness function. Given the inputs in every subregion and their associated evaluations $\{\boldsymbol{x}_i, f_{ii=1}^n\}$, we can predict the robustness at $\widehat{Y}_{i,j,k}^{\gamma}(\boldsymbol{x}_{n+1})$ and the associated model variance $s^2(\boldsymbol{x}_{n+1})$ at a new, unsampled location $\boldsymbol{x}_{n+1}$ in the associated subregion $\sigma_{i,j,k}^{\gamma}$

Algorithm 1 Sequential subregion sampling with Bayesian optimization (`SampleBO`)

Input: Subregion $\sigma_{ijk}^{\gamma} \subset \mathbb{R}^d$, objective function $f(\boldsymbol{x})$, initialization budget n^0, total budget n_{BO}, n_{jk} locations sampled so far $\left(\boldsymbol{x}_{ijk}^{\gamma}, \boldsymbol{f}_{ijk}^{\gamma}\right)$;

Output: best location and value $\boldsymbol{x}_{ijk}^{*} \in \sigma_{ijk}^{\gamma}$, $f\left(\boldsymbol{x}_{ijk}^{*}\right)$, final Gaussian process model $\left(\widehat{Y}_{ijk}^{\gamma}(\boldsymbol{x}), s_{ijk}^{2,\gamma}(\boldsymbol{x})\right)$.;

Step 1: Compute the initial required evaluation budget:
if $n_{jk} \geqslant n_0$ **then**
    Use n_{jk} sampled points within the subregion as initializing points for the Gaussian process estimation; $t \leftarrow 0$;
else
    Sample $n_0 - n_{jk}$ points using a Latin Hypercube design. Return $\boldsymbol{x}_{\text{train}} \in \sigma_{ijk}^{\gamma}$; $f(\boldsymbol{x}), \forall \boldsymbol{x} \in \boldsymbol{x}_{\text{train}}$; $t \leftarrow n_0 - n_{jk}$;
end if
while $t < n_{\text{BO}}$ **do**
    Step 2.1: Estimate the GP using the training data $\{\boldsymbol{x}_{\text{train}}, \boldsymbol{y}_{\text{train}}\}$, return $\left(\widehat{Y}_{ijk}^{\gamma}(\boldsymbol{x}), s_{ijk}^{2,\gamma}(\boldsymbol{x})\right)$ for all $\boldsymbol{x} \in \sigma_{ijk}^{\gamma}$;
    Step 2.2: Select the next location $\boldsymbol{x}_{\text{EI}}^{*} \leftarrow \arg\max_{\boldsymbol{x} \in \mathbb{X}} \text{EI}(\boldsymbol{x})$; Evaluate and store $f(\boldsymbol{x}_{\text{EI}}^{*})$.
    Step 2.3: $t \leftarrow t + 1$
end while

4.3.1 Sequential Conditional Branching with Classification

PART-X starts by looking at the entire space, S, and then keeps branching adaptively until the allotted budget is exhausted or all the leaf nodes have been classified as either satisfying, violating, or unbranchable. The first step is to sample points in the subregion and then classify the new subregions. The branching of subregions that meet the branching conditions are then branched and the process is repeated again.

Sampling

PART-X samples differently depending upon the class of the subregion. Primarily, there are two types of sampling mechanisms dependent on if the subregion $\sigma_{i,j,k}^{\gamma}$ has $\gamma \in \{r^+, r^-, r, u\}$ or $\gamma \in \{+, -\}$. If the subregion belongs to classes in the first group, each subregion is required to have n_0 points in the subregions before the estimation

of the Gaussian process. Then, n_{BO} points are sampled using bayesian optimization. The core idea is to bias the search toward finding falsifying inputs. Formally, consider the inputs and their evaluations in region $\sigma^{\gamma}_{i,j,k}$ where $\gamma \in \{r^+, r^-, r, u\}$, $\{\boldsymbol{x}_n, f(\boldsymbol{x}_n)\}$. By choosing an appropriate gaussian process prior, we can estimate the predictor and its associated variances by using eq. (4.3) and eq. (4.4). Using these, we can build an acquisition function, which is an inexpensive function that can be evaluated at any given point such that it shows what the evaluation function f will be for a given observation $\boldsymbol{x}$ for a minimization problem. In this thesis, the Expected Improvement (EI)[59] acquisition function and a new point is generated by finding the point which maximizes the EI, namely [59]:

$$\begin{aligned}\boldsymbol{x}_{t+1} &\in \arg\max_{\boldsymbol{x}\in S} \mathrm{EI}(\boldsymbol{x}) \\ &= \max\left\{0, \left[f^* - \widehat{Y}\left(\boldsymbol{x}\right)\right] \Phi\left(\frac{f^* - \widehat{Y}\left(\boldsymbol{x}\right)}{\widehat{s}\left(\boldsymbol{x}\right)}\right) + \widehat{s}\left(\boldsymbol{x}\right) \phi\left(\frac{f^* - \widehat{Y}\left(\boldsymbol{x}\right)}{\widehat{s}\left(\boldsymbol{x}\right)}\right)\right\}, \end{aligned} \tag{4.5}$$

where f^* is the best function value sampled so far, $\widehat{Y}(\boldsymbol{x})$ is the estimated predictor, $\widehat{s}(\boldsymbol{x}^2)$ is the estimated model variance associated with the predictor, Φ is the normal distribution function, and ϕ the normal probability density function. Once the point is sampled, the gaussian process is updated and another point is sampled. This process is repeated until n_{BO} evaluations are made. This step is shown in algorithm 1.

If the subregion belongs to the class $\gamma = \{-, +\}$, an overall budget of n_c evaluations is distributed across the set of violating subregions Θ^-_{jk} and satisfying subregions Θ^+_{jk}. The distribution of budgets across the subregions is calculated by the gaussian process predictor $\widehat{Y}^{\gamma}_{i,j,k}$ using the metric:

$$I^{\gamma}_{ijk} = \frac{1}{\upsilon\left(\sigma^{\gamma}_{i,j,k}\right)} \int_{x_0 \in \sigma^{\gamma}_{i,j,k}} \left(\int_{-\infty}^{0} f^{\gamma}_{i,j,k}\left(\widehat{y}^{\gamma}_{i,j,k}(x_0)\right) dy \right) dx_0. \tag{4.6}$$

where $\upsilon\left(\sigma^{\gamma}_{i,j,k}\right)$ is a constant used to normalize in order to keep the metric in the range $[0, 1]$. The metric described in Equation (4.6) corresponds to sampling from a

Algorithm 2 Gaussian process based min-max quantiles estimation (`MCstep`)

Input: subregion $\sigma_{ijk}^{\gamma} \subset \mathbb{R}^d$, objective function $f(\boldsymbol{x})$, Gaussian process model $\left(\widehat{Y}_{ijk}^{\gamma}(\boldsymbol{x}), s_{ijk}^{2,\gamma}(\boldsymbol{x})\right)$. Number of Monte Carlo iterations R, number of evaluations per iteration M;

Output: Estimates for the minimum and maximum δ_{C}-quantiles of the minimum and maximum function value $\hat{\bar{Q}}_j\left(\delta_{\mathrm{C}}\right), \mathrm{Var}\left(\hat{\bar{Q}}_j\left(\delta_{\mathrm{C}}\right)\right); \widehat{\underline{\mathrm{Q}}}_j\left(\delta_{\mathrm{C}}\right), \mathrm{Var}\left(\widehat{\underline{\mathrm{Q}}}_j\left(\delta_{\mathrm{C}}\right)\right)$;

for $r = 1, \ldots, R$ **do**

  for $m = 1, \ldots, M$ **do**

    Sample $\boldsymbol{x}_{mr}$, evaluate $\left(\widehat{Y}_{ijk}^{\gamma}(\boldsymbol{x}_{mr}), s_{ijk}^{2,\gamma}(\boldsymbol{x}_{mr})\right)$;

  end for

  Minimum and maximum quantile:

$$\begin{aligned}
\bar{q}_r\left(\delta_{\mathrm{C}}\right) &= \max_{m=1,\ldots,M}\left(\widehat{Y}_{ijk}^{\gamma}\left(\boldsymbol{x}_{r,m}\right) + Z_{1-\frac{\delta_{\mathrm{C}}}{2}}\sqrt{s_{ijk}^{2,\gamma}\left(\boldsymbol{x}_{r,m}\right)}\right), \\
\underline{\mathrm{q}}_r\left(\delta_{\mathrm{C}}\right) &= \min_{m=1,\ldots,M}\left(\widehat{Y}_{ijk}^{\gamma}\left(\boldsymbol{x}_{r,m}\right) - Z_{1-\frac{\delta_{\mathrm{C}}}{2}}\sqrt{s_{ijk}^{2,\gamma}\left(\boldsymbol{x}_{r,m}\right)}\right)
\end{aligned}$$

end for

Minimum and maximum δ_{C}-quantile:

$$\begin{aligned}
\hat{\bar{Q}}_j\left(\delta_{\mathrm{C}}\right) &= \tfrac{1}{R}\sum_{i=1}^{R}\bar{q}_r\left(\delta_{\mathrm{C}}\right), \mathrm{Var}\left(\hat{\bar{Q}}_j\left(\delta_{\mathrm{C}}\right)\right) = \frac{\mathrm{Var}(\bar{q}_r(\delta_{\mathrm{C}}))}{R}, \\
\widehat{\underline{\mathrm{Q}}}_j\left(\delta_{\mathrm{C}}\right) &= \tfrac{1}{R}\sum_{i=1}^{R}\underline{\mathrm{q}}_r\left(\delta_{\mathrm{C}}\right), \mathrm{Var}\left(\widehat{\underline{\mathrm{Q}}}_j\left(\delta_{\mathrm{C}}\right)\right) = \frac{\mathrm{Var}(\underline{\mathrm{q}}_r(\delta_{\mathrm{C}}))}{R}
\end{aligned}$$

subregion $\sigma_{i,j,k}^{\gamma}$ proportionally to the cumulative distribution $\leqslant 0$.

Classification Scheme

Once we have sampled the subregions corresponding to the trees based on their classes, the next step is to estimate the δ_c-quantiles for the minimum and maximum function values. This is done using a Monte Carlo estimation method as outlined in algorithm 2. Gaussian Processes can help to estimate the function values and the associated variances for any location within a subregion and these point estimates follow the normal distribution, the mean being the predictor and variance being the model variance. Finding the quantiles of M locations helps in obtaining the point

estimates for $(\overline{q}_r(\delta_{\mathrm{C}}), \underline{q}_r(\delta_{\mathrm{C}}))$. Repeating this process R times provides estimates of the mean and variance of the same quantiles. δ_{C} refers to the significance at which the quantile metric is estimated.

Table 4.1: Classification of a subregion based on the lower and upper bound of quantiles as defined in algorithm 3

Current Class	Lower δ_{C} Bound	Upper δ_{C} Bound	New Class	Comments
$\gamma = r$	> 0	> 0	$\gamma = +$	Not branched Further
$\gamma = r$	< 0	< 0	$\gamma = -$	Not branched Further
$\gamma = r$	$\leqslant 0$	$\geqslant 0$	$\gamma = r$	Branched Further
$\gamma = +$	$\leqslant 0$	$\leqslant 0 \vee \geqslant 0$	$\gamma = r^+$	Treated as $\gamma = r$
$\gamma = -$	$\leqslant 0 \vee \geqslant 0$	$\geqslant 0$	$\gamma = r^-$	Treated as $\gamma = r$

Once we have the minimum and maximum quantile estimates, one can now use these to classify a subregion. This process is outlined in algorithm 3, as well as the classification update is shown in table 4.1. If the region is currently remaining and the estimated lower bound of a minimum quantile > 0 (upper bound of a maximum quantile < 0), then the region is classified as satisfying (violating). If the region is currently satisfying (violating) and the estimated lower bound of minimum quantile $\leqslant 0$ (upper bound of a maximum quantile $\geqslant 0$), then the region assumes $\gamma = r^+$ ($\gamma = r^-$) class. The regions with $\gamma \in \{r^+, r^-, r\}$ are branched further in a random direction, provided it does go lower than the minimum branchable volume.

As we described above, the parameters R and M help in obtaining the estimates of the mean and variance of quantiles. We call the $R \times M$ as the grid size, and the choice of parameters mainly stems from the computational resources available at hand. Later, through numerical experiments, we show that PART-X is in fact robust against the grid size.

The PART-X continues to adaptively branch the input space until the budget is exhausted or all the leaves are classified as unbranchable. Finally, once the algorithm terminates, the falsification volume is computed and returned. The PART-X algorithm keeps track of the tree which can be accessed by the user to extract fal-

Algorithm 3 Classification of a subregion (`Classify`)

Input: subregion $\sigma_{ijk}^{\gamma} \subset \mathbb{R}^d$, current subregion type $\gamma \in (+, -, r)$, Gaussian process model $\left(\widehat{Y}_{ijk}^{\gamma}(\boldsymbol{x}), s_{ijk}^{2,\gamma}(\boldsymbol{x})\right)$;
Output: Return region type $(r+, r-, +, -, r, u)$;

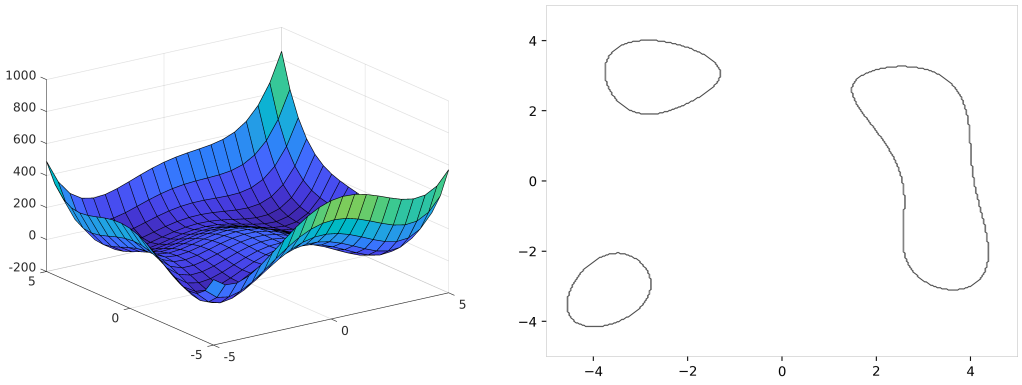
if $v(\sigma_j) \leqslant \prod_{d=1}^{D} \delta_v \cdot X_d$ **then**
  $\gamma = u$;
else if $\gamma = +$ **then**
  if $\underline{\widehat{\mathrm{Q}}}_j(\delta_{\mathrm{C}}) - Z_{1-\delta_{\mathrm{C}}/2}\mathrm{Var}\left(\underline{\widehat{\mathrm{Q}}}_j(\delta_{\mathrm{C}})\right) \leqslant 0$ **then**
    $\gamma = r+$;
  else
    $\gamma = +$;
  end if
else if $\gamma = -$ **then**
  if $\widehat{\widehat{Q}}_j(\delta_{\mathrm{C}}) + Z_{1-\delta_{\mathrm{C}}/2}\mathrm{Var}\left(\widehat{\widehat{Q}}_j(\delta_{\mathrm{C}})\right) \geqslant 0$ **then**
    $\gamma = r-$;
  else
    $\gamma = -$;
  end if
else if $\gamma = r$ **then**
  if $\widehat{\widehat{Q}}_j(\delta_{\mathrm{C}}) + Z_{1-\delta_{\mathrm{C}}/2}\mathrm{Var}\left(\widehat{\widehat{Q}}_j(\delta_{\mathrm{C}})\right) < 0$ **then**
    $\gamma = -$;
  else if $\underline{\widehat{\mathrm{Q}}}_j(\delta_{\mathrm{C}}) - Z_{1-\delta_{\mathrm{C}}/2}\mathrm{Var}\left(\underline{\widehat{\mathrm{Q}}}_j(\delta_{\mathrm{C}})\right) > 0$ **then**
    $\gamma = +$;
  else
    $\gamma = r$;
  end if
end if

sifying regions and other crucial information. The complete algorithm is shown in algorithm 4.

4.4 Numerical Experiments

4.4.1 Detailed Example: Himmelblau Function

Let us discuss the Part-X algorithm to estimate 0-level set of the Himmelblau Non-Linear Function (defined in eq. (4.9)). The robustness landscape and its 0-level set are shown in fig. 4.3. Let us walk through the PART-X algorithm. The input region first receives the initial samples and then computes $n_b o$ for both the region



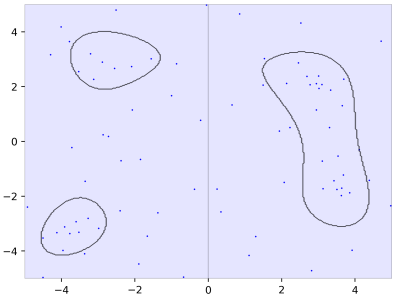
(a) Robustness Landscape of Himmelblau's Function (b) 0-level set of Himmelblau's Function.

Figure 4.3: Himmelblau's Function as Described in eq. (4.9)

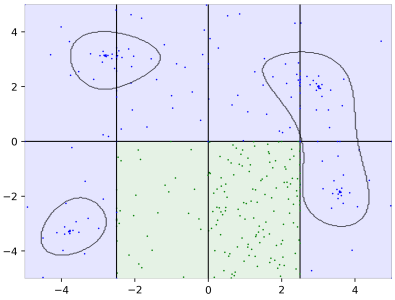
at iteration 1 (see fig. 4.4a), and the regions are classified as remaining. The same process repeats in the subsequent iteration. Now, at iteration 3, some of the regions have been classified as satisfying (see fig. 4.4b). Through the next iterations, these regions are not reclassified. As we discussed before, these do not branch as well. Stepping through iterations 7 and 8, one can notice that regions are being classified into satisfying and violating such that regions where the values of the function $\leqslant 0$ (see fig. 4.3b), have been classified into violating and other have turned satisfying. The areas on the boundary are still remaining, which when branched further will produce branches such that it follows the actual 0-level set. However, the algorithm is terminated at iteration 8 since the budget has been exhausted.

4.4.2 Non-Linear Non-Convex Optimization Examples

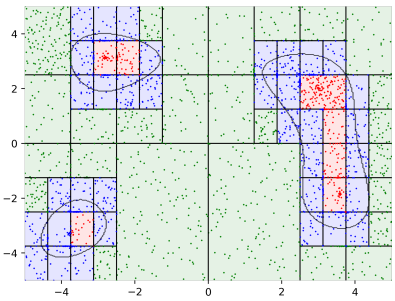
The Part-X is tested on three different non-linear and non-convex optimization examples with the intention to find the 0-level set and find the volume of the region that is inside this level set. The three functions are the Rosenbrock, Goldstein, and Himmelblau's function, which is defined as follows:



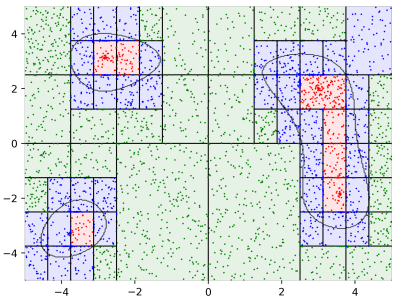
(a) Iteration $k = 1$, the S is branched into two subregions: $\sigma^r_{1,1}, \sigma^r_{2,1}$. Each subregion receives $n_0 + n_{\text{BO}}$ samples.



(b) teration $k = 3$, the remaining regions are branched resulting into 8 subregions: $\sigma^r_{1,3}, \ldots, \sigma^r_{8,3}$. Each subregion receives $\max\{n_0 - n_{jk}, 0\} + n_{\text{BO}}$ samples.



(c) Iteration $k = 7$, the remaining regions (blue) are branched resulting into various subregions. Each subregion receives $\max\{n_0 - n_{jk}, 0\} + n_{\text{BO}}$ samples.



(d) Iteration $k = 8$, the remaining regions (blue) are branched resulting into various subregions. Each subregion receives $\max\{n_0 - n_{jk}, 0\} + n_{\text{BO}}$ samples.

Figure 4.4: Various iterations of the PART-X algorithms. We start with the initial regions as remaining and iteratively branch and classify the remaining regions. One can also notice that with more samples, the reclassification of classified regions takes place (see top-left region in fig. 4.4c and fig. 4.4d).

- Rosenbrock's function $(-1 \leqslant x_i \leqslant 1, i = 1, \ldots d)$:

$$f(\boldsymbol{x}) = \sum_{i=1}^{d-1} \left\{ 100 \left(x_{i+1} - x_i^2 \right)^2 + (1 - x_i)^2 \right\} - 20. \tag{4.7}$$

  For $d = 2$ (2D), it has a symmetric level set in the region: $S = [-1, 1] \times [-1, 1]$.

- Goldstein-Price function $(-1 \leqslant x, y \leqslant 1)$:

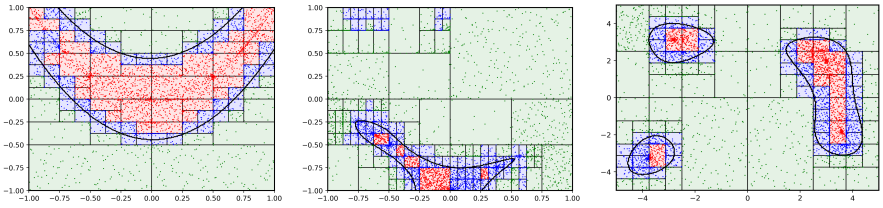
$$f(x, y) = \left\{ 1 + (x + y + 1)^2 \left(19 - 14x + 3x^2 - 14y + 6xy + 3y^2 \right) \right\}$$
$$\left\{ 30 + (2x - 3y)^2 \left(18 - 32x + 12x^2 + 48y - 36xy + 27y^2 \right) \right\} - 50. \tag{4.8}$$

The Goldstein function has a smaller zero level set $\mathcal{L}_0$ volume compared to the Rosenbrock function.

- Himmelblau's function $(-5 \leqslant x, y \leqslant 5)$

$$f(x, y) = \left(x^2 + y - 11\right)^2 + \left(x + y^2 - 7\right)^2 - 40. \tag{4.9}$$

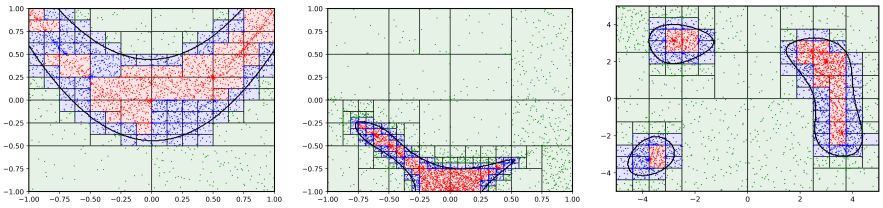
The level set $\mathcal{L}_0$ for the Himmelblau is disconnected.



(a) Grid Size $R \times M = 10 \times 100$, Rosenbrock function.

(b) Grid Size $R \times M = 10 \times 100$, Goldstein function.

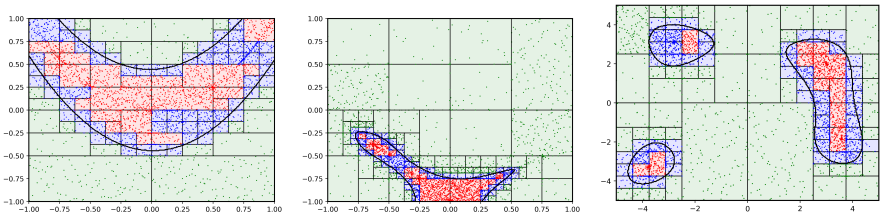
(c) Grid Size $R \times M = 10 \times 100$, Himmelblau's function.



(d) Grid Size $R \times M = 10 \times 500$, Rosenbrock function.

(e) Grid Size $R \times M = 10 \times 500$, Goldstein function.

(f) Grid Size $R \times M = 10 \times 500$, Himmelblau's function.



(g) Grid Size $R \times M = 20 \times 500$, Rosenbrock function.

(h) Grid Size $R \times M = 20 \times 500$, Goldstein function.

(i) Grid Size $R \times M = 20 \times 500$, Himmelblau's function.

Figure 4.5: Level Set Estimation by PART-X on Non-Linear, Non-Convex Functions. Solid lines represent the true 0-level set of the function. Satisfying (with positive) regions are denoted by green areas, Violating (with negative) regions are denoted by red areas, and the Remaining regions are denoted by blue areas.

PART-X was run with an initialization budget $n_0 = 10$, per-subregion budget of unclassified subregions $n_{\text{BO}} = 10$, classified subregions budget $n_c = 100$, maximum

budget $T = 5000$, number of Monte Carlo iterations $R = 10$, number of evaluations per iterations $M = 100$, number of cuts $B = 2$, and classification percentile $\delta_{\text{C}} = 0.05$. Also, we used $\delta_v = 0.001$ to identify dimensions that should not be branched. PART-X was run for 50 microreplication with these settings.

Figure 4.5a - Figure 4.5i shows the partitioning obtained from PART-X for the three test functions function from a randomly chosen macro-replication. The true 0-level set is denoted by the solid black lines, while the red areas are the violating regions (a region where the function has negative values), green areas are the satisfying regions (a region where the function has negative values) and the blue areas denote the regions classified as remaining. From all plots in fig. 4.5, it can be observed that the PART-X algorithm correctly identifies the satisfying and violating regions. Also, it can be seen that even with lowers values of the grid size ($R \times M$), the performance of the algorithm is unaffected. As we pointed out before, the PART-X is robust against the values of the grid size and thus it solely depends on the choice of the user. Also, on observing the sampled locations, a pattern can be seen where sampling is biased toward the 0-level in comparison to other areas.

Table 4.2 shows the results for falsification volume. For PART-X, we estimate the falsification volume using: (1) the sum of violating and remaining regions, and (2) the estimate at δ_q-quantile using Gaussian process prediction. The mean and standard errors of the falsification volumes are over 50 macro-replications. The Monte Carlo estimates of the falsification volume are generated using the same budget, i.e, 50 macro-replications with 5000 budget each, and are treated as the oracle for the true 0-level set. Thus, the closer the estimates are to Monte Carlo estimates, the better the estimates. In table 4.2, the estimates produced using gaussian processes are more accurate and robust with respect δ_c-quantile because of the particularly high density of sampling in the remaining regions producing low model variance at the

Table 4.2: Falsification volume obtained with various grid sizes. We estimate the falsifying volume as the volume of the remaining and violating hyperboxes (P-X). P-X (δ_q) algorithm variants use the Gaussian processes δ_q-quantiles to estimate the falsifying volume. Monte Carlo uses the same number of evaluations (5000×50) to perform a one-shot estimation of the falsifying volume, so no standard error is provided.

Grid Size	Algorithm	Rosenbrock		Goldstein		Himmelblau	
		Mean	std err	Mean	std err	Mean	std err
$R \times M = 10 \times 100$	P-X	2.866	7.627E-04	1.410	2.705E-03	34.313	1.836E-02
	P-X ($\delta_q = 0.5$)	1.629	3.900E-08	0.305	1.536E-08	17.647	1.806E-05
	P-X ($\delta_q = 0.95$)	1.633	4.180E-08	0.307	2.278E-08	17.756	1.724E-05
	P-X ($\delta_q = 0.99$)	1.635	4.462E-08	0.308	2.604E-08	17.803	1.747E-05
$R \times M = 10 \times 500$	P-X	2.798	6.475E-04	0.700	2.198E-03	34.555	2.369E-02
	P-X ($\delta_q = 0.5$)	1.628	6.494E-09	0.305	1.582E-09	17.652	4.340E-06
	P-X ($\delta_q = 0.95$)	1.634	9.389E-09	0.307	1.598E-09	17.776	5.417E-06
	P-X ($\delta_q = 0.99$)	1.636	1.179E-08	0.308	4.269E-09	17.829	5.855E-06
$R \times M = 20 \times 500$	P-X	2.799	8.035E-04	0.655	3.256E-04	34.500	2.459E-02
	P-X ($\delta_q = 0.5$)	1.629	4.159E-09	0.305	4.154E-10	17.651	2.477E-06
	P-X ($\delta_q = 0.95$)	1.634	5.169E-09	0.307	7.526E-10	17.780	2.822E-06
	P-X ($\delta_q = 0.99$)	1.637	8.092E-09	0.308	1.090E-09	17.835	3.241E-06
	Monte Carlo	1.626		0.302		17.030	

corresponding level set. On comparing the estimates across all the grid sizes, it can be observed that results are consistent with each other as well as the Monte Carlo estimate with a single exception. The higher estimate of P-X falsification volume mean on the Goldstein function for the grid size $R \times M = 10 \times 100$ appears to be an exception, however, it can be verified from fig. 4.5b that some of the partitions in $[-1, 0] \times [0.75, 1]$ are not classified yet which impacts the P-X estimate.

Algorithm 4 Partitioning with Continued X-distributed Sampling

Input: Input space S, function $f(\boldsymbol{x})$, initialization budget n_0, Bayesian optimization budget, n_{BO}, and unclassified subregions budget n_c, total number of evaluations T. Define branching operator $\mathcal{P} : A \to (A_i)_i : \bigcup_i A_i = A, \bigcap_i A_i = \varnothing$. Number of Monte Carlo iterations R, number of evaluations per iteration M; number of cuts per dimension per subregion B, classification percentile δ_{C}, δ_v;

Set the iteration index $k \leftarrow 1$, Initialize the sets $\widehat{\Theta^-}_k = \widehat{\Theta^+}_k = \widehat{\Theta^{r,+}}_k = \widehat{\Theta^{r,-}}_k = \widehat{\Theta^u}_k = \varnothing$, $\Theta^r_k \leftarrow S$;

Output: $\widehat{\Theta^-}, \widehat{\Theta^+}, \widehat{\Theta^{r,+}}, \widehat{\Theta^{r,-}}, \widehat{\Theta^r}, \widehat{\Theta^u}$;

while $T_k \geqslant 0$ **do**

  Branching

  for $\sigma^{\gamma}_{ijk} \in \left(\widehat{\Theta^{r,+}}_k \cup \widehat{\Theta^{r,-}}_k \cup \widehat{\Theta^r}_k\right)$ **do**

    Return $\left(\sigma^r_{i,j+1,k}\right)_{i=1}^{B} = \mathcal{P}\left(\sigma^{\gamma}_{ijk}\right)$;

  end for

  Count $N_k^{\bar{C}} \leftarrow$ number of non-classified leaves of the partitioning tree at iteration k; $N_k^C \leftarrow$ number of classified leaves of the partitioning tree at iteration k;

  if $T_k \geqslant n_{\text{BO}} \cdot N_k^{\bar{C}} + \sum_{\sigma^{\gamma}_{ijk}:\gamma\in\{r+,r-,r\}} \max\left(n_{jk} - n_0, 0\right)$ **then**

    for All unclassified subregions $\sigma^r_{i,j,k}, \forall j, i(j)$ **do**

      Execute $\texttt{SampleBO}\left(\sigma^r_{i,j,k}, n_{\text{BO}}, n_0, n_{jk}\right)$;

      Return the quantiles for the minimum and maximum function value executing $\texttt{MCstep}\left(R, M, \sigma^r_{i,j,k}, \widehat{Y}^{\gamma}_{ijk}, s^{2,\gamma}_{ijk}\right)$;

      Update subregions type:
      $\gamma \leftarrow \texttt{Classify}\left(\sigma^r_{i,j,k}, \widehat{Y}^{\gamma}_{ijk}, s^{2,\gamma}_{ijk}\right), \widehat{\Theta^{\gamma}}_k \leftarrow \widehat{\Theta^{\gamma}}_k \bigcup \sigma_j, \widehat{\Theta^r}_k \leftarrow \widehat{\Theta^r}_k \backslash \sigma_j$;

      $N^{\gamma}_{jk} \leftarrow N^{\gamma}_{jk} + 1$;

      $T_k \leftarrow T_k - n_{\text{BO}} - \max\left(n_{jk} - n_0, 0\right)$;

    end for

    for $j = 1, \ldots, N_k^+ \cup N_k^-$ **do**

      Allocate n_c across the subregions proportional to the metric in eqn. (4.6);

      Return the quantiles for the minimum and maximum function value executing $\texttt{MCstep}\left(R, M, \sigma^{\gamma}_{i,j,k}, \widehat{Y}^{\gamma}_{ijk}, s^{2,\gamma}_{ijk}\right)$;

      Update subregions type:
      $\gamma \leftarrow \texttt{Classify}\left(\sigma^r_{i,j,k}, \widehat{Y}^{\gamma}_{ijk}, s^{2,\gamma}_{ijk}\right), \widehat{\Theta^{\gamma}}_k \leftarrow \widehat{\Theta^{\gamma}}_k \bigcup \sigma^{\gamma}_{ijk}, \widehat{\Theta^r}_k \leftarrow \widehat{\Theta^r}_k \backslash \sigma^{\gamma}_{ijk}$;

    end for

    $T_k \leftarrow T_k - n_c$;

  else

    Allocate T_k to the subregions proportionally to the volume;

    Evaluate the function at the sampled point and update the Gaussian processes;

    $T_k \leftarrow 0$;

  end if

  $k \leftarrow k + 1$;

end while

Return $\mathcal{V}^f = \frac{v\left(\widehat{\Theta^-}_{k-1}\right)}{v(S)}$

Chapter 5

STOCHASTIC OPTIMIZATION WITH ADAPTIVE RESTART FOR COVERAGE (PYSOAR-C)

5.1 Introduction

In this chapter, the Stochastic Optimization with Adaptive Restart for Coverage (PySOAR-C) is discussed in the context of the gray-box system, wherein we exploit the information available from the dynamics of the system-under-test (SUT) in the form of hybrid automata. A hybrid automata is an automata that consists of both a continuous and a discrete component. First, let us establish the background with respect to hybrid systems and their robustness computation. Then, the PySOAR-C is described followed by preliminary results on a small-scale example.

5.2 Hybrid Systems

Hybrid systems can be represented as Hybrid Automaton (HA) (as defined in [17]).

Definition 1 (Hybrid Automaton) *A Hybrid Automata* Ψ *consists of components* $\langle V, \boldsymbol{L}, \mathcal{T}, \Theta, \boldsymbol{D}, \boldsymbol{I}, \ell_0 \rangle$*, where* $V = \{x_1, x_2, \ldots, x_n\}$ *is the set of continuous variables,* $\boldsymbol{L}$ *is a finite set of locations (modes),* $\mathcal{T}$ *is a set of (discrete) transitions such that for each* $\tau : \langle \ell_1 \longrightarrow \ell_2, g_\tau \rangle \in \mathcal{T}$*, we move from location* $\ell_1 \in \boldsymbol{L}$ *to location* $\ell_2 \in \boldsymbol{L}$ *and the relation* g_τ *over* $V \cup V^{'}$ *is satisfied,* $H_0 = \{\ell_0\} \times \Theta$ *is the set of initial conditions with* $\ell_0 \in \boldsymbol{L}$ *to a vector field* $\boldsymbol{D}(\ell)$*; and* $\boldsymbol{I}$ *is a mapping of each* $\ell \in \boldsymbol{L}$ *to a location invariant set* $\boldsymbol{I}(\ell) \subseteq \mathbb{R}^n$*.*

The locations and continuous state spaces together form the hybrid state space $\mathbb{H} = \boldsymbol{L} \times \mathbb{R}^n$. Following definition 1, the timed trajectory of a hybrid automaton can be defined as an infinite sequence of states $\langle \ell, \overrightarrow{x} \rangle \in \mathbb{H} = \boldsymbol{L} \times \mathbb{R}^n$ represented as $\langle \ell_0, \overrightarrow{x}_0 \rangle, \langle \ell_1, \overrightarrow{x}_1 \rangle, \langle \ell_2, \overrightarrow{x}_2 \rangle, \ldots$ such that starting from initial location ℓ_0 and $\overrightarrow{x}_0 \in \Theta$, we move to state $\langle \ell_i, \overrightarrow{x}_i \rangle$ such that we either make a discrete transition from ℓ_i to ℓ_{i+1} or evolve from $\overrightarrow{x}_i$ to $\overrightarrow{x}_{i+1}$ under the continuous state dynamics $\boldsymbol{D}(\ell_i)$.

Thus, a gray-box model where we have information about the underlying hybrid automaton can be represented as an input/output function:

$$\mathcal{M}_{GB} : ([0, T] \rightarrow \mathbb{R}^d) \rightarrow ([0, T] \rightarrow \mathbb{H} = \boldsymbol{L} \times \mathbb{R}^n), \tag{5.1}$$

where symbols have the same meaning as in definition 1 and eq. (2.1).

Unlike the black-box models we discussed earlier, the hybrid automaton returns a timed trajectory of hybrid states. To incorporate this information quantitatively, a different metric is needed to define the distance between two locations in a hybrid trajectory. A generalized quasi-distance δ metric given in [17] can be used. Given two hybrid locations $h = \langle \ell, \overrightarrow{x} \rangle$ and $h^{'} = \langle \ell^{'}, \overrightarrow{x}^{'} \rangle$, the quasi-distance metric is defined as:

$$\delta(h, h^{'}) = \begin{cases} \langle 0, d(x, x') \rangle & \text{if } \ell = \ell^{'} \\ \langle \pi(\ell, \ell^{'}), \min_{\ell'' \in \partial\mathcal{N}_\pi(\ell, \ell^{'})} \mathbf{dist}_d(\overrightarrow{x}^{'}, \boldsymbol{G}^t(\ell, \ell'')) \rangle & \text{otherwise} \end{cases} \tag{5.2}$$

where π is the shortest path metric, d is a metric on $\mathbb{R}^n$, and $\partial\mathcal{N}_\pi(\ell, \ell^{'})$ is the "boundary" of all locations which are closer to location $\ell^{'}$ than location ℓ and may be visited from ℓ within one transition if the associated guard set $\boldsymbol{G}^t$, which may be changing with time, is satisfied. Therefore, if the two states $h, h^{'}$ are in the same location, the distance computation reduces to computing the distance between two points $\overrightarrow{x}, \overrightarrow{x}^{'}$ in the continuous state-space. Else, the distance is the path between the two states weighted by their distance to the closest guard set that can enable the transition from the first state to another and reduces the path distance.

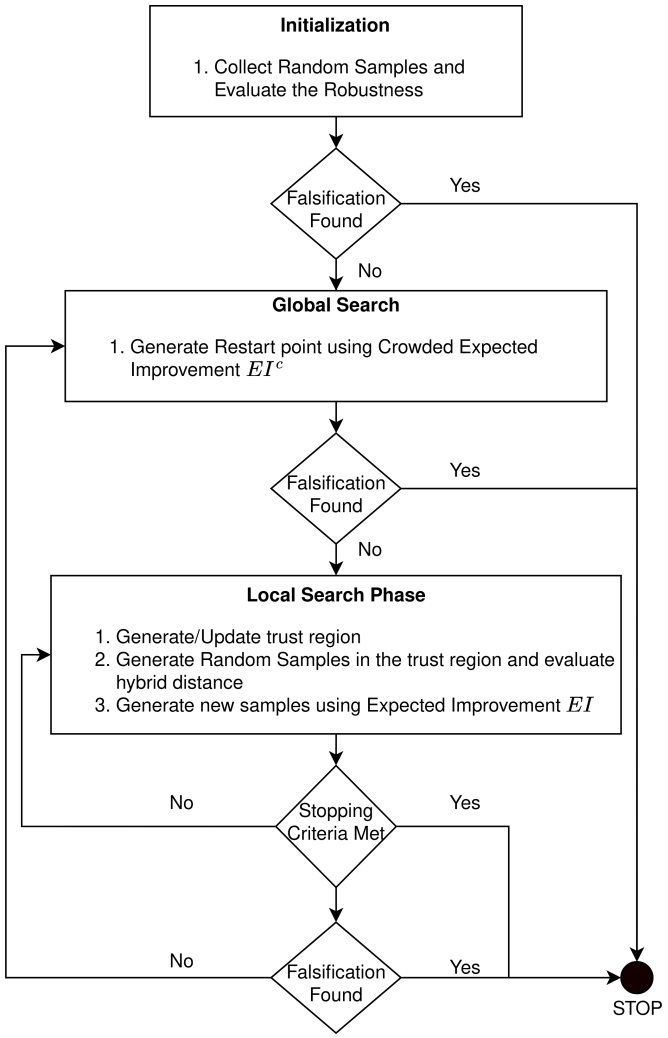
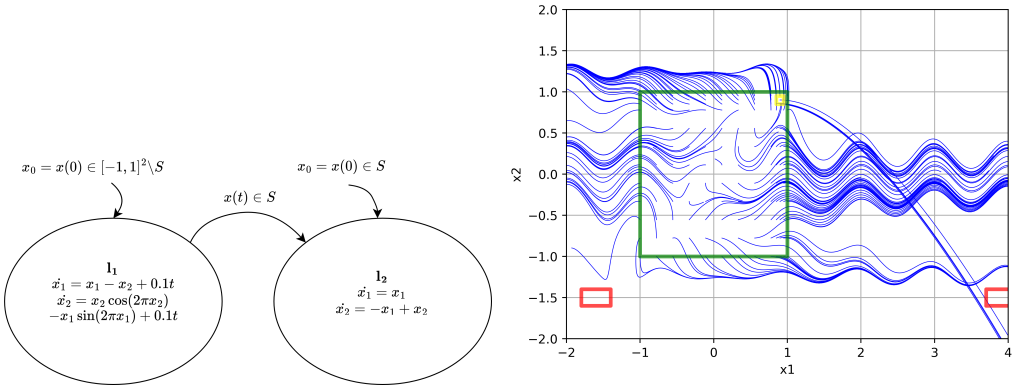


Figure 5.1: Overview of PySOAR-C Algorithm

5.3 PySOAR-C Intuition

Considering that we have the hybrid automata available at our disposal, the question that one may ask is how to exploit the structure to find falsifications. The most intuitive way to do this is to explore the hybrid automata to get an idea of the falsification surface and states that are being accessed. Consider a $2-$ state hybrid automaton $\sum_1$ shown in Figure 5.2a. If the initial state $x_0 = (x_{01}, x_{02}) \in S = [0.85, 0.95]^2$, shown as yellow box in Figure 5.2b, then $\sum_1$ follows the dynamics in location l_2. On the other hand, if the initial system state $x_0 = (x_{01}, x_{02}) \in S^{'} = [1, 1]^2 \backslash S$, then $\sum_1$ follows



(a) Dynamics of a 2-State Hybrid Automaton

(b) Trajectories of HA at time $t \in [0, 2]$ for different initial points x_0

Figure 5.2: Example of 2 state Hybrid automaton from [1]. Figure 5.2a shows the dynamics of a 2-state Hybrid Automaton. Figure 5.2b shows trajectories followed by the hybrid automata from different initial points. Notice that location l_1 is represented by the area inside the yellow box and location l_2 is represented by the green box apart from the yellow box.

the dynamics in location l_1. It is important to note that if the system is operating at a certain location l_1 and enters the set S at any time, the system switches to use the dynamics in l_2. This becomes clearer from the plots of trajectories at times $t \in [0, 2]$ shown in Figure 5.2b. The system is considered to be safe when it never enters the unsafe regions $\mathcal{O}(a) = [-1.8, -1.4] \times [-1.6, -1.4]$ and $\mathcal{O}(b) = [3.7, 4.1] \times [-1.6, -1.4]$. A falsifying behavior in this toy example is when the trajectory enters the $\mathcal{O}(b)$ (Figure 5.2b). Formally, this can be defined in STL by:

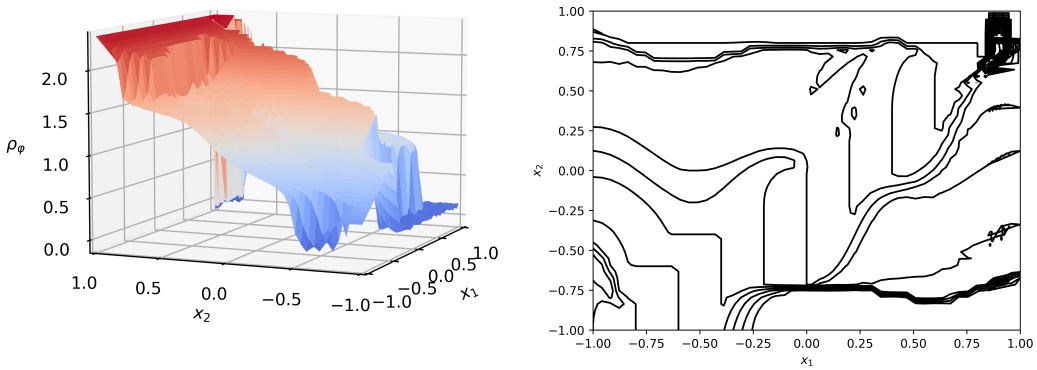
$$\varphi = \square_{[0,2]} \neg (\pi_1 \vee \pi_2) \tag{5.3}$$

$$\text{where} \quad \pi_1 = x_{t1} \in [-1.8, -1.4] \wedge x_{t2} \in [-1.6, -1.4]$$

$$\pi_1 = x_{t1} \in [3.7, 4.1] \wedge x_{t2} \in [-1.6, -1.4]$$

When this system is treated as a black-box, we provide the system input and the black-box simulates for the next 2 seconds to give a timed trajectory in the continuous state space. The corresponding robustness ρ_ϕ can be computed using robustness monitors. Positive values of robustness indicate safe behavior and negative values

indicate that the system has been falsified. The corresponding robustness surface and its contour plots are shown in Figure 5.3.

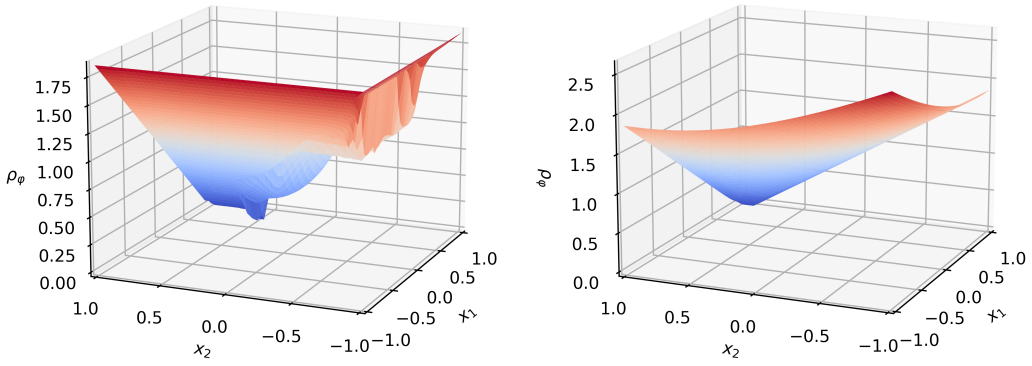


(a) $3-d$ plot of robustness surface (b) Contour plot of robustness surface

Figure 5.3: Robustness Landscape of the 2-State HA.

On closely observing the trajectories taken by the 2−state HA in fig. 5.2b, one can notice that while sampling from the areas in $[-1,1]\times[-1,-0.5]$ leads to trajectories passing closely from the unsafe set and yielding low robustness values, none of them, however, enter the regions $\mathcal{O}(b)$. On the other hand, the falsifications are caused by trajectories originating from $S=[0.85,0.95]^2$. This can also be observed in the robustness landscape shown in fig. 5.3a. While this might appear to be a simple example, the consequence is that if an algorithm treated this system as a black-box, it very unlikely to find falsifications due to the structure of the problem. However, if one could embed requirements that force the algorithm to explore other states finding falsifications seem to be much more possible. In this particular problem, exploring the $\boldsymbol{l_1}$ state will lead to much higher chances of finding falsifying inputs such that the trajectory enters the red areas. Hybrid distance to the explored set S can be visualized from fig. 5.4. Figure 5.4a shows the closest distance that a trajectory can get to the region S, while fig. 5.4b shows the distance of the initial point from the

region S.



(a) Distance of trajectory to S (b) Distance to unexplored state

Figure 5.4: Hybrid Distance

5.4 PySOAR-C

With all the necessary intuition and background to the problem, we are finally ready to present the PySOAR-C algorithm. The key idea of this gray-box falsification algorithm is a 2−phase strategy where the first phase looks at the robustness of the evaluated system and samples global points adaptively, while the local search uses gaussian process based trust region optimization, which aims at exploring the various states of the hybrid automaton. The overview of the algorithm is provided in fig. 5.1.

The algorithm starts by initializing n_0 samples over the entire search space and evaluating their robustness. The global search phase begins by forming a surrogate over the samples and their evaluations and finding a point that maximizes the crowding-distance based Expected Improvement (EI^c). This is also called a restart point. Using this restart point as a reference, a trust region is generated around the sampled point. Starting from this trust region, the model looks at the number of

points n_{lp} already evaluated inside the trust region. If this is lesser than the number of required samples n_{lr}, the algorithm samples the additional points $n_{lr} - n_{lp}$ points inside the trust region and evaluates their hybrid distances. A surrogate model over the n_{lr} points is generated using gaussian processes and a location that maximizes the Expected Improvement is sampled and robustness and hybrid distances are evaluated and stored. After this, the trust regions are updated using the ratio-comparison test. This process is repeated for n_l iterations unless the stopping criterion is met. Upon exiting the local search, either due to completing all iterations or meeting the stopping criterion, a new restart point is generated and the entire process is repeated again. This is shown in Algorithm 5.

5.4.1 Global Search - Generating a Restart Point

At a certain iteration k during the algorithm, we always have a dataset of robustness values $\mathcal{D}_r : \{X, Y_r\}$ where $X \in \mathbb{R}^d$ and $Y_r \in \mathbb{R}$. This allows us to generate a surrogate $\widehat{Y}(\boldsymbol{x})$, $\widehat{s}^2(\boldsymbol{x})$ (eq. (4.3), eq. (4.4)). We have already discussed how we can use Expected improvement eq. (4.5) to generate new samples. However, it has been shown that using EI to sample new points tend to bias exploitation more than exploration over the input space [61]. In order to alleviate this issue, [13] introduced the crowded expected improvement EI^c. Given an unsampled location $\boldsymbol{x}$ such that $\boldsymbol{x} \notin X$, the EI^c is defined as:

$$EI^c = \begin{cases} \sum_{l=1}^{d} \min_{\boldsymbol{x}_j \in \mathbb{X}^k} \mid x_l - x_{j,l} \mid & EI_k \geqslant \alpha EI_k^* \\ 0 & EI_k < \alpha EI_k^* \end{cases} \tag{5.4}$$

Fundamentally, EI^c uses the $(1-\alpha)-$level set with respect to $EI^* \equiv max_{\boldsymbol{x} \in X} EI((x)$. This means that the EI^c is guaranteed to be at least $(1-\alpha)$ times the EI^*. Note that in normal EI, if the $EI^* = 0$, a random point is sampled. In the EI^c, however,

Algorithm 5 Stochastic Optimization with Adaptive Restart for Coverage (PySOAR-C)

Input: Input space S, Requirement φ_G, coverage Requirement φ_L, initialization budget n_0, Total budget n_{max}
Output: Sample that falsifies the requirement $\boldsymbol{x}$;

STOP = False
$\mathcal{D} \leftarrow \{x_{n_0}, y_{n_0}\}$: Initialize dataset with $n_0 \in S$ points and their evaluation values for requirement φ_G
if Falsified Point Found **then**
  return Falsified Point
end if
$t \leftarrow n_0$
while $t \leqslant n_{max}$ **do**
  Global Search Phase:
  $\{x_{restart}, \rho_{\varphi_G}(\boldsymbol{x_{restart}})\}$: Generate a Restart point by maximising eq. (5.4) for requirement φ_G, and evaluating it.
  $t \leftarrow t + 1$
  if $\rho_{\varphi_G}(\boldsymbol{x_{restart}}) \leqslant 0$ **then**
    Return $\boldsymbol{x_{restart}} \leftarrow$ Falsifying Input found during global search phase.
  end if
  Local Search Phase:
  while Local Search Stopping Criterion not met **do**
    Perform Local Search to maximize coverage by minimizing the robustness for requirement φ_L and update t
    if Falsified Point Found **then**
      return Falsified Point
    end if
    Update Trust Region and Centroid
  end while
end while

when $EI^* = 0$, the problem reduces to $\arg\max_{\boldsymbol{x} \in X} \sum_{l=1}^{d} \min_{\boldsymbol{x}_j \in \mathbb{X}^k} \mid x_l - x_{j,l} \mid$, thus ensuring that previous sampled location is not re-sampled.

5.4.2 Local Search

At a certain iteration k, when the restart point is generated, the trust region is generated such that the restart location is the centroid. The algorithm then enters a local search. The local search is simple in the sense that the number of points and their evaluated hybrid distances are obtained to create a dataset $D_l : \{X, Y_{HD}\}$. If the number of samples in D_l is less than the required number of points n_{lr}, additional

points are randomly sampled in the trust region and are evaluated to obtain hybrid distances. These are then added to D_l, after which a surrogate is generated and the EI is maximized to obtain a new sample location. Using this sampled location, a ratio-comparison test is performed in order to update the trust regions. The test statistic ρ_s is as follows [62, 13]:

$$\rho_s = \frac{f(\tilde{x}^c) - f(\tilde{x}^*)}{f_{\mathcal{GP}}(\tilde{x}^c) - f_{\mathcal{GP}}(\tilde{x}^*)} \tag{5.5}$$

where the numerator is the difference between the true function of the restart point and the sampled point from the local search and the denominator is the prediction of the restart location and the sampled location from the local search using the gaussian process model. If the test statistic $\rho_s > 1$, then a better point has been generated, while $\rho_s < 0$ indicates inadequate improvement over the trust region. The intuition is that if we have better points sampled, we are optimistic about the model and thus increase the trust-region size and update the centroid. On the other hand, if we are not optimistic, we reduce the trust region size. Given the user-defined parameters $\eta_1$andη_2 such that $0 < \eta_1 \leqslant \eta_2 < 1$ the trust region is updated based on the following three cases:

(Case 1.) $\rho_s \leqslant \eta_1$: Keep the current centroid and reduce the trust region size.

(Case 2.) $\eta_1 < \rho_s \leqslant \eta_2$: Update the centroid with the new sample location and maintain the current trust region size.

(Case 3.) $\rho_s < \eta_2$: Update the centroid with the new sample location and expand the current trust region size.

The local search is exited when either the a falsification is found, or the current trust region size reduces. While the former is easier to track, the latter cannot be predicted pre-execution of the algorithm. For that reason, we generate a random

number in $(0, 1)$ and exit if the ratio of of the current trust region and two furthest away points in the subregion is less than the random numbers sampled.

5.5 Preliminary Results

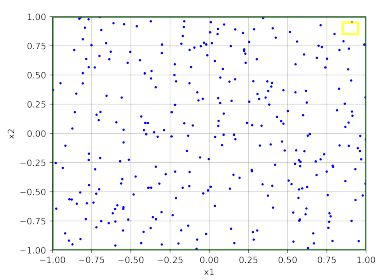
In this section, we compare the performance of PySOAR-C with the PySOAR and the uniform random sampler. The uniform random sampler is run for 50 macro-replications with a budget of 300 each and is considered the oracle. Both PySOAR and PySOAR-C were run with with initialization samples $n_0 = 20$, maximum budget $n_{max} = 300$, required points in local regions $n_{lr} = 15$, local iterations $n_l = 15$, user-defined constants $\eta_1 = 0.25, \eta_2 = 0.75, \delta = 0.75, \gamma = 1.25$. Like the uniform random sampler, each experiment was run for 50 macro-replications. Figure 5.6 and

Table 5.1: Results of PySOAR and PySOAR-C on the 2 state HA. *FR* refers to the Falsification Rate (calculated out of 50), $\overline{S}$ refers to the mean number of simulations to find a falsification, and $\widetilde{S}$ refers to the median number of simulations to find a falsification.

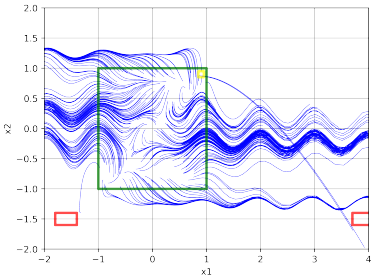
	Uniform Random			**PySOAR-C**			**PySOAR-C**		
Benchmark	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$\widetilde{S}$
2 State HA	17	254.3	300.0	11	258.7	300.0	39	170.5	146.00

fig. 5.5 shows plots of sampled locations and their corresponding trajectories when both falsifications are found and not found. In addition to sampled locations, figures for PySOAR and PySOAR-C show the global points and the corresponding local points beginning from the search. Qualitatively, PySOAR-C is expected to produce samples such that it is biased towards exploring the closest state. This behavior can be observed in fig. 5.5e and fig. 5.6e, where the local samples tend to move towards minimizing the distance between the closest next states. On the other hand, PySOAR is expected to behave such that the global point is at a point further away from the existing samples, which can be observed in fig. 5.5c and fig. 5.6c. Since

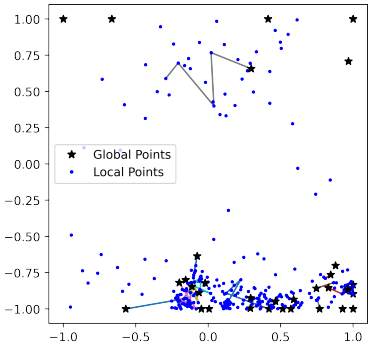
PySOAR just looks at the robustness values with respect to the specification, the search is biased towards the lower areas of the search space S'. Table 5.1 compares the PySOAR-C with uniform random sampler and PySOAR. PySOAR-C has a higher falsification rate along with fewer samples to find falsification, which is what we expected.



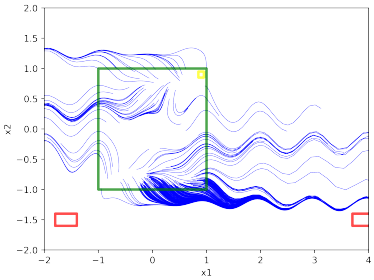
(a) Uniform Random Sampler



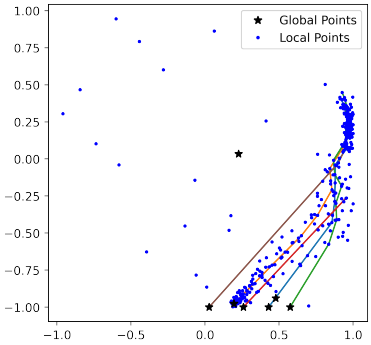
(b) Uniform Random Sampler



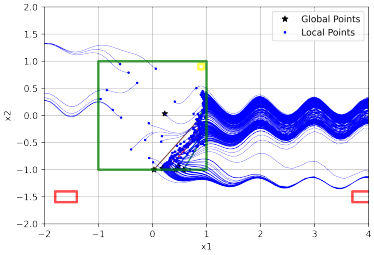
(c) PySOAR



(d) PySOAR

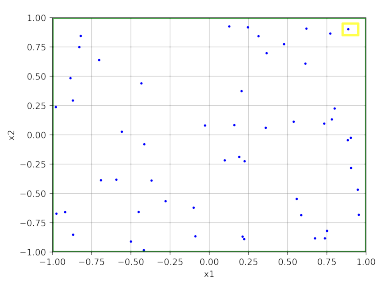


(e) PySOAR

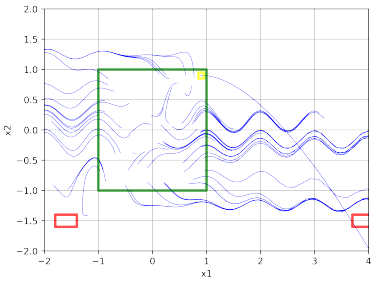


(f) PySOAR

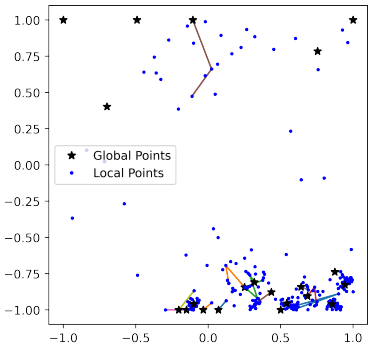
Figure 5.5: Random replications in which no falsifications were Found. Global points and their corresponding local searches are shown for PySOAR and PySOAR-C.



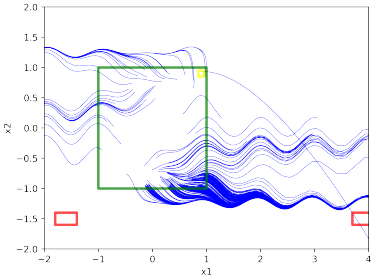
(a) Uniform Random Sampler



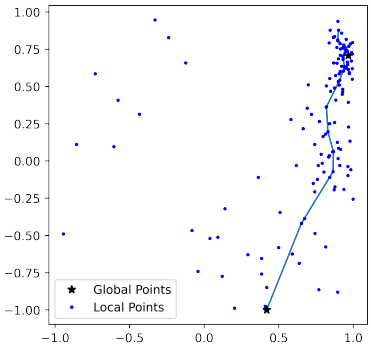
(b) Uniform Random Sampler



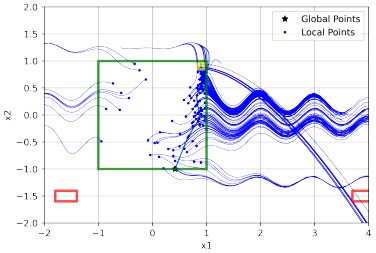
(c) PySOAR



(d) PySOAR



(e) PySOAR



(f) PySOAR

Figure 5.6: Random replications in which falsifications were Found. Global points and their corresponding local searches are shown for PySOAR and PySOAR-C.

Chapter 6

LARGE SCALE EXTENDED MINIMUM BAYESIAN OPTIMIZATION (LS-EMIBO)

6.1 Introduction

In this chapter, the Large Scale Extended Minimum Bayesian Optimization (LS-EMIBO) is introduced which helps in gray-box testing, where the information from the structure of the requirement is exploited in order to find falsifying inputs to the System-under-Test (SUT). Specifically, the algorithm deals with conjunctive requirements where a requirement is the conjunction of multiple sub-requirements. The quantitative definition of robustness says that violating a conjunctive requirement is equivalent to violating at least one of the sub-requirement. Formally, a conjunctive requirement $\varphi(\boldsymbol{x})$ is:

$$\varphi(\boldsymbol{x}) = \varphi_1(\boldsymbol{x}) \wedge \varphi_2(\boldsymbol{x}) \wedge \ldots \wedge \varphi_h(\boldsymbol{x}). \tag{6.1}$$

If $\rho_{\varphi_h}(\boldsymbol{x})$ refers to robustness corresponding to h^{th} sub-requirement $\varphi_h(\boldsymbol{x})$ and $\rho_{\varphi}(\boldsymbol{x})$ corresponds to the robustness in eq. (6.1), then we have the following:

$$\boldsymbol{x} \vDash \varphi \quad \text{if } \min\left(\rho_{\varphi_1}(\boldsymbol{x}), \rho_{\varphi_2}(\boldsymbol{x}), ..., \rho_{\varphi_h}(\boldsymbol{x})\right) > 0 \tag{6.2}$$

$$\boldsymbol{x} \nvDash \varphi \quad \text{if } \min\left(\rho_{\varphi_1}(\boldsymbol{x}), \rho_{\varphi_2}(\boldsymbol{x}), ..., \rho_{\varphi_h}(\boldsymbol{x})\right) \leqslant 0 \tag{6.3}$$

Thus, with this in mind, finding a falsifying input boils down to solving the following problem:

$$\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x} \subset \mathbb{R}^d} \left(\rho_{\varphi}(\boldsymbol{x})\right) \tag{6.4}$$

$$\text{with} \quad \rho_{\varphi}(\boldsymbol{x}) = \min\left(\rho_{\varphi_1}(\boldsymbol{x}), \rho_{\varphi_2}(\boldsymbol{x}), ..., \rho_{\varphi_h}(\boldsymbol{x})\right)$$

The state-of-the-art minBO algorithm was proposed in [41] which aimed at solving the problem described in eq. (6.4). Specifically, they solved the problem by employing a surrogate model for all the sub-requirements by using Gaussian Process (described in eq. (4.3) and eq. (4.4)) and calculating the Expected Improvement (described in eq. (4.5)) for every component of the requirement. The requirement which produced the best EI over all the components was chosen in order to sample the next location, and the process was repeated. One of the key advantages of the minBO algorithm was it avoided the masking effect that occurs while considering the entire requirement. However, there was no way to model the dependencies that certain components might have between each other. The LS-EMIBO algorithm aims at modeling the dependencies between the various components of the requirement.

The overview of the algorithm is shown in fig. 6.1. The algorithm starts by sampling n_0 location in the input space S and computing the corresponding robustness for all the components of the requirement φ. This can formatted to form a dataset $\{\boldsymbol{x}_n; \rho_{\varphi_1}(\boldsymbol{x}), \rho_{\varphi_2}(\boldsymbol{x}), \ldots, \rho_{\varphi_h}(\boldsymbol{x})\}$. Next, a classifier is trained to predict the component having the minimum value for a given input $\boldsymbol{x}$. Subsequently, the probability distribution of component φ_h having the minimum robustness can be computed. Surrogate models for Top-k components are then generated and the probability distribution is used to compute the emi-EI. The subsequent testing location which maximizes the emi-EI is generated and added to the dataset. If the generated test is a falsifying one or the budget for evaluation is exhausted, the algorithm terminates. If not, the algorithm is repeated again. The detailed discussion of the algorithms is shown in algorithm 6.

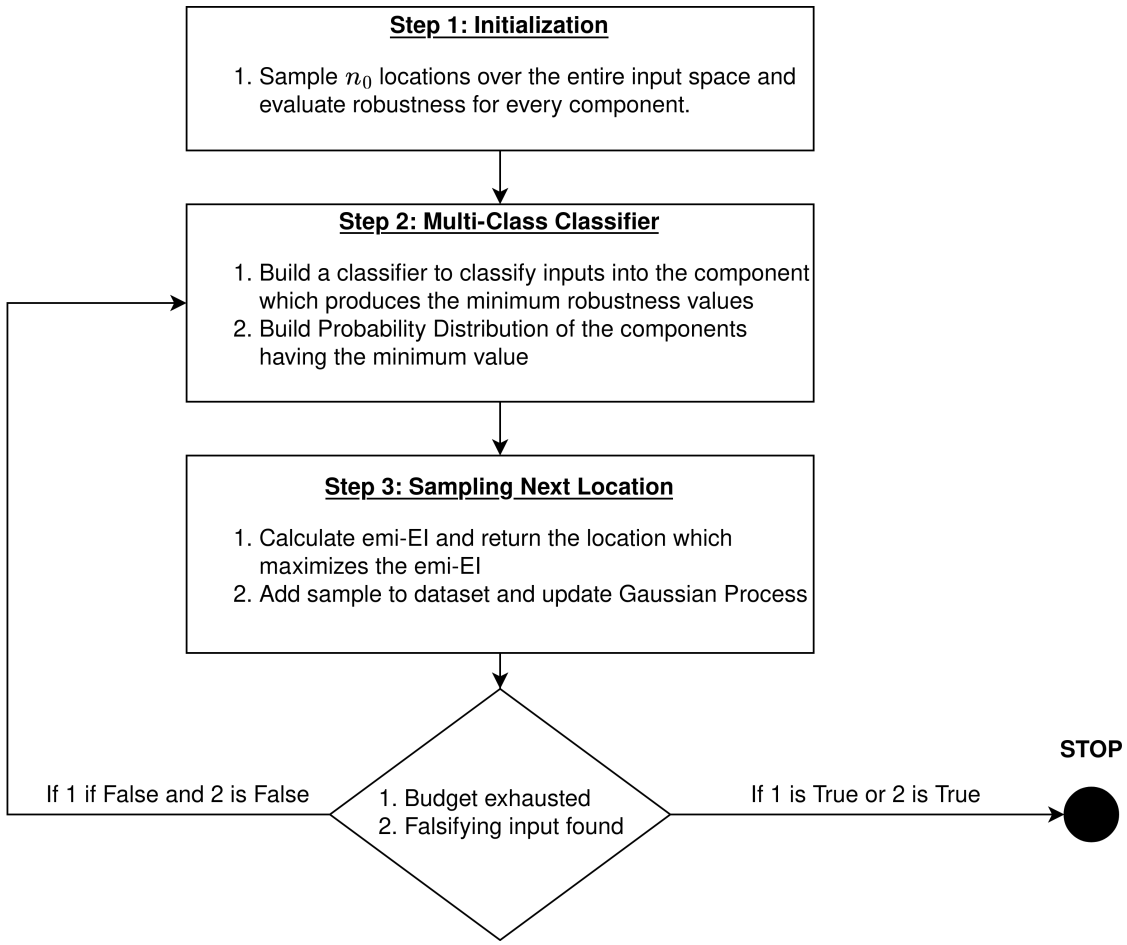


Figure 6.1: LS-EMIBO Algorithm Overview

6.2 Extended Minimum Expected Improvement Function

Consider the eq. (6.4) where we have a set of samples and the robustness for h components of the requirements. With this, we can build a dataset of inputs and outputs such that the inputs are the sampled locations, and outputs are a one-hot encoding where the index of the component having the minimum robustness is 1, while others are 0.

$$\widehat{I}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \rho_{\varphi_i}(\boldsymbol{x}) = \min_j \rho_{\varphi_j}(\boldsymbol{x}) \\ 0 & \text{otherwise} \end{cases} \quad \forall \ i \in [1, 2, \ldots, h] \tag{6.5}$$

A classifier can then be trained in order to learn the unknown function that maps inputs to components with minimum robustness. Such a surrogate allows us to compute the component prior as:

$$p_i = \frac{1}{V(X)} \int_X \widehat{I}_i(\boldsymbol{x}) dx \ \ , i = 1, 2, \ldots, h. \tag{6.6}$$

From this distribution is the algorithm samples $k \leqslant h$ models for which surrogates are generated (using eq. (4.1), and eq. (4.2)). Assuming that $\widehat{Y}_i$, $\widehat{s}_i^2$ are the predictors and model variance associated with the surrogate model of i^{th} component such that $i \in [1, k]$, the extended minimum EI is then computed as follows.

$$\text{emi-}EI = \sum_{i=1}^{k} EI_i(x) p(i = i^*_{t+1} \mid x) \tag{6.7}$$

where i^*_{t+1} is the index of model with lowest robustness at location x, formally:

$$i^*_{t+1} \mid x \in \arg\min_i \rho_{\varphi_i}(\boldsymbol{x}). \tag{6.8}$$

Denoting the best value obtained from the evaluated samples by y^*_t, the $p(i = i^*_{t+1} \mid x)$ can estimated using:

$$p\left(i = i^*_{t+1} | x\right) = \frac{P\left(\widehat{Y}_i\left(x\right) < y^*_t\right)}{\sum_{j=1}^{k} P\left(\widehat{Y}_j\left(x\right) < y^*_t\right)} \tag{6.9}$$

While the closed form for calculation of the probability $P(.)$ in eq. (6.9) is not readily available, the desired probability can be estimated using monte-carlo approximation. Finally, the $EI(.)$ in eq. (6.7) is computed using:

$$EI_i\left(\boldsymbol{x}\right) = (y^*_t - \widehat{Y}_i(\boldsymbol{x}))\Phi\left(\frac{(y^*_t - \widehat{Y}_i(\boldsymbol{x}))}{\widehat{s}_i(\boldsymbol{x})}\right) + \widehat{s}_i(\boldsymbol{x})\phi\left(\frac{(y^*_t - \widehat{Y}_i(\boldsymbol{x}))}{\widehat{s}_i(\boldsymbol{x})}\right) \tag{6.10}$$

With all the components of the eq. (6.7), the new testing locations can be sampled by solving the following optimization problem.

$$x_{t+1} \in \arg\max_x \text{emi-}EI(x) \tag{6.11}$$

To show the various aspects of the LS-EMIBO algorithm, let's solve a simple numerical example that imitates the problem we try to solve in eq. (6.4). Specifically, consider the following optimization problem and the parameters in table 6.1:

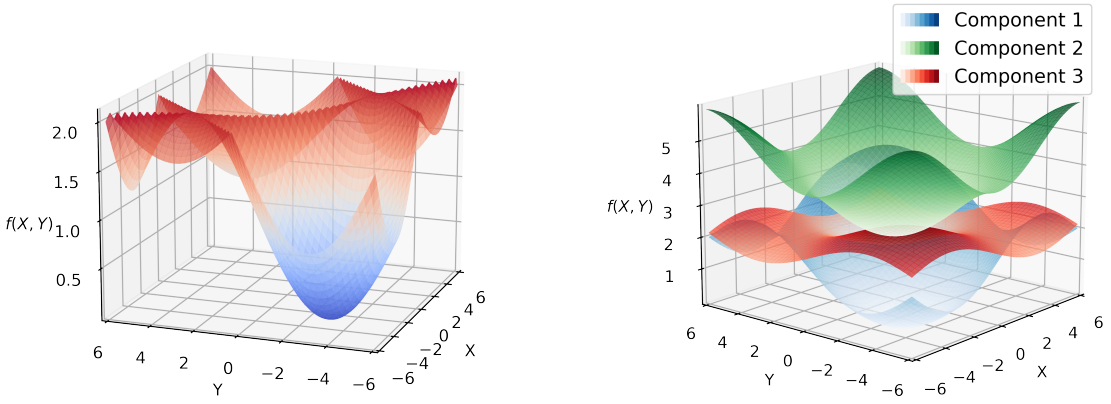
$$x \in \arg\min_{x \in \mathbb{X}} (\min(h_1(x), h_2(x), h_3(x))) \tag{6.12}$$

$$\text{where} \quad h_j(x) = A_j \left(\sin\left(a_j x_1 + b_j\right) + \sin\left(a_j x_2 + b_j\right)\right) + B_j \quad \forall\, j \in [1, 2, 3]$$

The landscape for the various components $h_1(x), h_2(x), h_3(x)$ is shown in fig. 6.2b and

Table 6.1: Parameters Corresponding to A_j, B_j, a_j, b_j in eq. (6.12)

Component	A	B	a	b
h_1	1.00	2.00	0.50	0.00
h_2	1.00	4.00	0.50	−1.50
h_3	0.50	2.00	0.50	−3.00



(a) Optimization landscape of $\min(h_1(x), h_2(x), h_3(x))$.

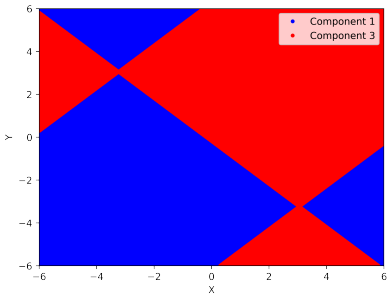
(b) Optimization landscape for Component 1 ($h_1(x)$), Component 2 ($h_2(x)$) and Component 3 ($h_3(x)$).

Figure 6.2: Optimization Surface of eq. (6.12) and Table 6.1.

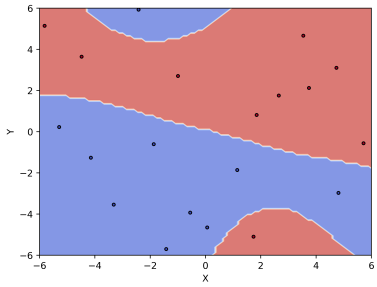
the $\min(h_1(x), h_2(x), h_3(x))$ is shown in fig. 6.2a. It can be observed that component

2 is never the minimizer of out of the three components. Out of components 1 and 3, component 3 achieves the minimum function values.

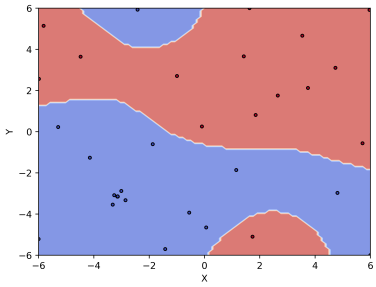
Starting with $n_0 = 20$ initial samples, $n_{BO} = 130$ samples, and selecting Top-$k = 2$ components every time, the problem is framed such that the algorithm reaches the minimizer of the function, but never falsifies. This allows us to look at the quality of samples generated by the algorithm. While the algorithm is able to find the minimizer, it is critical to compare how the classifier performs over various iterations and evaluate the quality of samples generated by the algorithm. Figure 6.3a shows



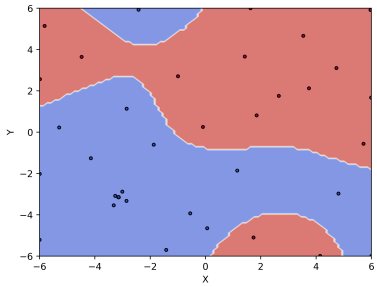
(a) Empirical surface



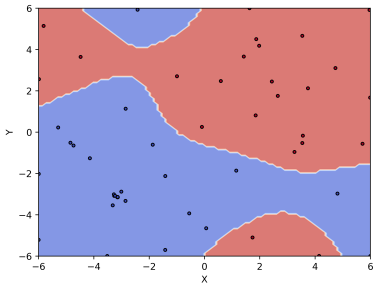
(b) Iteration 20



(c) Iteration 50



(d) Iteration 80



(e) Iteration 150

Figure 6.3: The classifier decision surface evolving over multiple iterations. Figure 6.3a denotes the empirical classification surface.

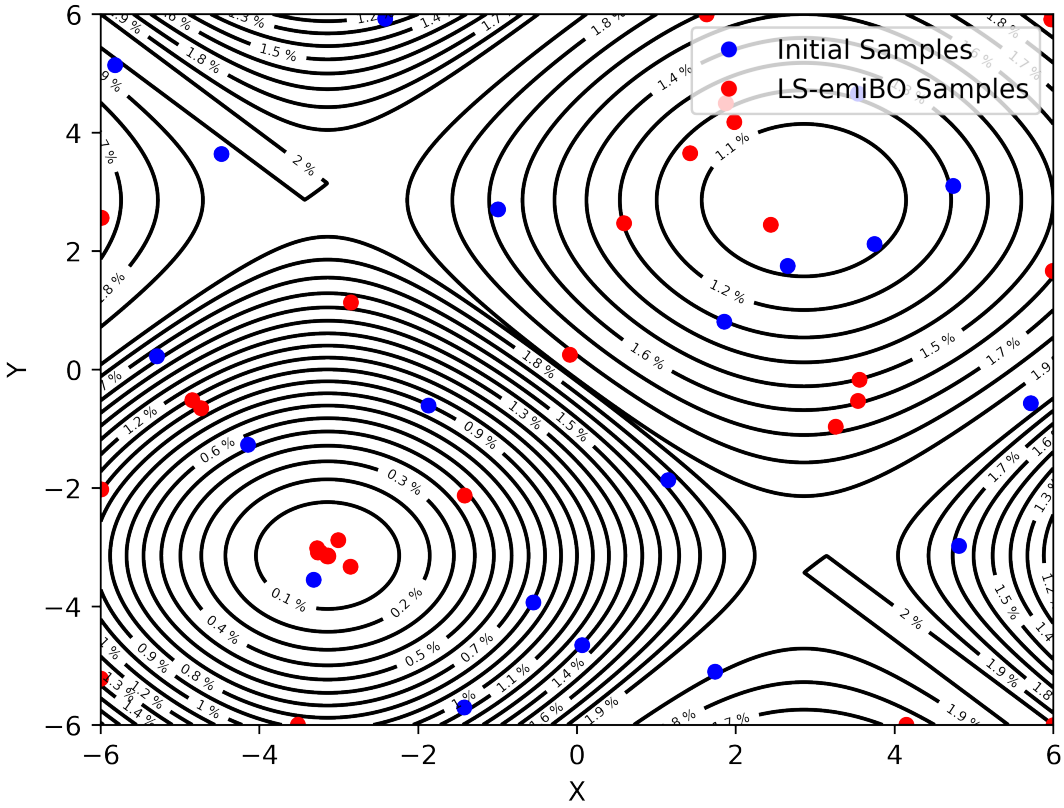


Figure 6.4: Samples generated using LS-EMIBO.

the empirical decision surface over 10^8 points in the S'. fig. 6.3b - fig. 6.3e shows how the classifier evolves over various iterations of the algorithm. Figure 6.4 shows the samples generated (represented by blue dots) which tend to minimize the function.

Algorithm 6 Large Scale Extended Minimum Bayesian Optimization

Input: Input space S, function $f(\boldsymbol{x})$, initialization budget n_0, Bayesian optimization budget, n_{BO}, Top$-k$ Models k, Requirements φ_i, Monte-Carlo Estimation R
Output: Sample that produces falsifies the conjunctive requirement $\boldsymbol{x}$;

$\mathcal{D} \leftarrow \varnothing$: Initialize empty dataset of sampled and evaluated the robustness of all components
$t \leftarrow 0$
while $t \leqslant n_0$ **do**
  $\boldsymbol{x} \leftarrow$ Sample $\boldsymbol{x} \in S$
  for i = 1, . . . , h **do**
    $\rho_{\varphi_i}(\boldsymbol{x}) \leftarrow$ Evaluate Robustness of i^{th} component.
    if $\rho_{\varphi_i}(\boldsymbol{x}) \leqslant 0$ **then**
      Return $\boldsymbol{x} \leftarrow$ Falsifying Input found during Initialization Phase
    end if
  end for
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{\boldsymbol{x}; \rho_{\varphi_i}(\boldsymbol{x})\}, i = 1, \ldots h$: Add input and robustness of all components to dataset $\mathcal{D}$
  $t \leftarrow t + 1$
end while
$t \leftarrow 0$
while $t \leqslant n_{BO}$ **do**
  $\mathcal{D}^{'} \leftarrow$ Build dataset of inputs and one-hot encoding output vector using eq. (6.5).
  Train Classifier with dataset $\mathcal{D}^{'}$ to component prior (eq. (6.6)) and sample top-k models.
  $\widehat{Y}_i(\boldsymbol{x}), \widehat{s}_i^2(\boldsymbol{x}),\ i \in 1, \ldots, k \leftarrow$ Estimate GP for the top-k sampled components.
  $num \leftarrow \varnothing$, $den \leftarrow 0$
  for $i = 1, \ldots, k$ **do**
    for $r = 1, \ldots, R$ **do**
      $\boldsymbol{x}_r \in S$: Sample random samples and evaluate $\widehat{Y}_i(\boldsymbol{x}), \widehat{s}_i^2(\boldsymbol{x})$
      $num \leftarrow num \cup P\left(\widehat{Y}_i(x) < y_t^*\right)$, $den \leftarrow den + P\left(\widehat{Y}_i(x) < y_t^*\right)$
    end for
    Compute $p(i = i_{t+1}^* \mid x)$ using num and den.
  end for
  $x_{t+1} \leftarrow \arg\max_x$ emi-$EI(x)$
  for i = 1, . . . , k **do**
    $\rho_{\varphi_i}(\boldsymbol{x_{t+1}}) \leftarrow$ Evaluate Robustness of i^{th} component.
    if $\rho_{\varphi_i}(\boldsymbol{x_{t+1}}) \leqslant 0$ **then**
      Return $\boldsymbol{x_{t+1}} \leftarrow$ Falsifying Input found.
    else
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{\boldsymbol{x_{t+1}}; \rho_{\varphi_i}(\boldsymbol{x})\}, i = 1, \ldots k$: Add input and robustness of all components to dataset $\mathcal{D}$
      $t \leftarrow t + 1$
    end if
  end for
end while

Chapter 7

EXPERIMENTS ON ARCH BENCHMARKS

PART-X and LS-EMIBO have been implemented as Python libraries that be used as a part of the PSY-TALIRO tool or as standalone packages in order to integrate them with other System-in-the-loop testing frameworks. In this chapter, we discuss the experiments that were conducted with the proposed algorithms and their results. Due to the similarity in PART-X and LS-EMIBO, we compare both of them in the chapter on the set of experiments that have conjunctive experiments. This chapter is structured such that we first discuss the experimental setup and discussion of results for both PART-X and LS-EMIBO followed by their comparison.

7.1 Experimental Setup

In order the convey the efficacy and efficiency of the PART-X on engineering applications, an add-on PART-X module is developed which integrates with the SBTG Python package PSY-TALIRO [20] (as discussed in section 2.2.1). This section will first discuss the experimental setup followed by the results of PART-X applied to find falsifying ($0-$level) sets for various multiple against various cyber-physical systems.

7.1.1 Cyber-Physical Systems Models

In order to test the algorithms on complex cyber-physical systems, with various requirements, benchmarks from the ARCH-COMP 2021 [14] are selected. A brief overview of the systems is as follows:

1. **Automatic Transmission (AT)**: The AT model [63] is a Simulink model of an automatic transmission controller with hybrid dynamics. The model is

deterministic in nature. At any time t, the AT model takes in two inputs: throttle and brake, and outputs the speed of the vehicle, rotations-per-minute (RPM), and the gear of the system.

2. **Chasing Cars (CC)**: The CC model is a Simulink model of automatic chasing car [64]. The model consists of five cars, one of which is driven by the inputs: throttle and brake, while the other 4 are driven by the algorithm in [64]. At any time t, the model takes in these inputs and outputs a 5-dimensional vector denoting the locations of the cars.

3. **Neural-Network Controller (NN)**: The NN model is a Simulink-based neural network controller for a system that levitates a magnet over an electromagnet at a certain reference position [14]. At any time t, the model takes the reference position (ref) as input and returns the current position of the magnet.

4. **Steam Condenser (SC)**: The SC model is a simulink-based model that consists of a neural network controller for energy and cooling water mass balance [65]. At a certain time t, the input is the Steam Flow Rate and the system outputs Circulating Water Outlet Temperature, Steam Flow, Cooling water Flow, and Condenser Pressure. However, the requirements for this system are based only on the condenser pressure.

5. **Fuel Control of an Automatic Powertrain (AFC)**: The AFC model is a Simulink model that models the automotive air-fuel control model [66]. The system takes as input the throttle and the engine speed, and the system outputs the mode. The system is tested against requirements involving the throttle and the mode of the system.

Black-box models for all these models are generated by calling MATLAB engine from

Table 7.1: These tables summarize the inputs and their corresponding ranges along with the simulation time horizon. The input type refers to the kind of the model must receive in order to model them. Time-varying input is a time trajectory (signal) that is passed to the model in order to get a timed trajectory of their outputs over which requirements are evaluated. Time invariant inputs are static inputs in the form of initial conditions, using which the model simulates itself and return a timed trajectory of the states. NN is used for experiments with non-conjunctive requirements while NNx is used for experiments with conjunctive requirements.

Model	Inputs	Input Ranges	Sim. Time Horizon (s)	Input Type	Instance 1	Instance 2
AT	Throttle Brake	$[0, 100]$ $[0, 325]$	50	Time Varying Input	7 3	7 3
CC	Throttle Brake	$[0, 1]$ $[0, 1]$	100	Time Varying Input	10 10	10 10
NN	Reference	$[1, 3]$	40	Time Varying Input	13	3
NNx	Reference	$[1.95, 2.05]$	3	Time Inariant	3	3
SC	Steam Flow Rate	$[3.99, 4.01]$	35	Time Varying Input	18	20
AFC	Throttle Engine	$[0, 61.2]$ $[900, 1100]$	50	Time Varying Input	- -	10 1

python which is then attached to the PSY-TaLiRo tool for finding falsifications against various requirements. As per the definition of black-box models, they take in a timed trajectory of input signals (in the case of time-varying inputs) and/or the initial conditions (in the case of time-invariant inputs) and returns the timed trajectory of outputs.

7.1.2 Input Parametrization

For every model we describe above, the objective while performing a falsification search can be: (1) finding the initial conditions that falsify a system, (2) finding timed trajectories (signals) that lead to falsifications, or (3) both of the above. In the first case, the objective reduces to finding a combination of initial input values that lead to violating a requirement against a specification. It is important to note that the dimensionality of the problem in Part-X is equal to the number of inputs that form the initial conditions. In the second case, the objective is to generate a signal that is passed as an input to the black-box model, and output signals are generated.

These input signals are often discretized at a small time step leading to trajectories of length in the order of 10^3 even for simpler benchmarks. While we can theoretically use Part-X to find falsifying inputs at all the discretized time steps, it is practically very hard. A way to deal with the situation is to provide the control points of the signal at different time steps and use interpolation methods to estimate the signal at the required discretization. We stick to piecewise continuous and piecewise constant (as used in [14]) methods which are described as follows:

1. Arbitrary piece-wise continuous input signals: These input signals are generated using the Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) [67] methods, such that a set of control points at different time steps produces the interpolated signal.

2. Constrained input signals: These input signals are generated using Piecewise Constant Interpolation methods such that the signal values take the same value as the control point until the next control point.

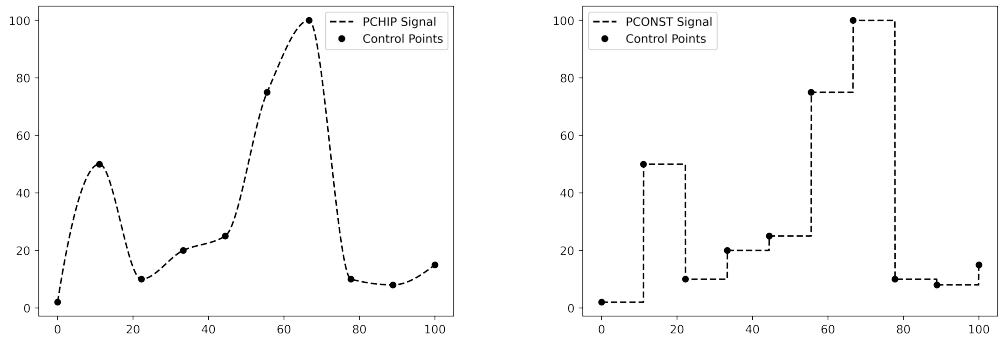
An example of the control points over 100 seconds is described in table 7.2 and resulting signals are shown in fig. 7.1.

Table 7.2: A signal can be defined by describing the control points at various time steps. This is an example of one such signal, and the generated signal for both PCHIP and PCONST is shown in fig. 7.1

Time (s)	$0.\overline{00}$	$11.\overline{11}$	$22.\overline{22}$	$33.\overline{33}$	$44.\overline{44}$	$55.\overline{55}$	$66.\overline{66}$	$77.\overline{77}$	$88.\overline{88}$	$99.\overline{99}$
Control Points	2	50	10	20	25	75	100	10	8	15

7.1.3 Specifications

The specifications are picked up from the ARCH-COMP 2021 [14]. They are stated here for reader's convenience.



(a) PCHIP Signal (b) PConst Signal

Figure 7.1: Signals generated using controls points described in table 7.2

Table 7.3: Requirement formulas for the benchmarks

Key	STL formula	Remarks/Constraints
AT1	$\square_{[0,20]} v < 120$	
AT2	$\square_{[0,10]} \omega < 4750$	
AT51	$\square_{[0,30]}((\neg g1 \wedge \circ\, g1) \rightarrow \circ\, \square_{[0,2.5]} g1)$	where $\circ\, \phi \equiv \Diamond_{[0.001,0.1]}\ \phi$
AT52	$\square_{[0,30]}((\neg g2 \wedge \circ\, g2) \rightarrow \circ\, \square_{[0,2.5]} g2)$	
AT53	$\square_{[0,30]}((\neg g3 \wedge \circ\, g3) \rightarrow \circ\, \square_{[0,2.5]} g3)$	
AT54	$\square_{[0,30]}((\neg g4 \wedge \circ\, g4) \rightarrow \circ\, \square_{[0,2.5]} g4)$	
AT6a	$(\square_{[0,30]} \omega < 3000) \rightarrow (\square_{[0,4]} v < 35)$	
AT6b	$(\square_{[0,30]} \omega < 3000) \rightarrow (\square_{[0,8]} v < 50)$	
AT6c	$(\square_{[0,30]} \omega < 3000) \rightarrow (\square_{[0,20]} v < 65)$	
AT6abc	AT6a $\wedge$ AT6b $\wedge$ AT6c	cojunctive requirement
AFC27	$\square_{[11,50]}((\mathit{rise} \vee \mathit{fall}) \rightarrow (\square_{[1,5]} \lvert\mu\rvert < \beta))$	$0 \leqslant \theta < 61.2$ (normal mode)
AFC29	$\square_{[11,50]} \lvert\mu\rvert < \gamma$	$0 \leqslant \theta < 61.2$ (normal mode)
AFC33	$\square_{[11,50]} \lvert\mu\rvert < \gamma$	$61.2 \leqslant \theta \leqslant 81.2$ (power mode)
	where $\beta = 0.008$, $\gamma = 0.007$ $\mathit{rise} = (\theta < 8.8) \wedge (\Diamond_{[0,0.05]}(\theta > 40.0))$ $\mathit{fall} = (\theta > 40.0) \wedge (\Diamond_{[0,0.05]}(\theta < 8.8))$	
NN	$\square_{[1,37]}\big(\lvert Pos - Ref\rvert > \alpha + \beta\lvert Ref\rvert \rightarrow \Diamond_{[0,2]}\square_{[0,1]}\neg(\alpha + \beta\lvert Ref\rvert \leqslant \lvert Pos - Ref\rvert)\big)$ where $\alpha = 0.005$ and $\beta = 0.03$	
NNx	$\Diamond_{[0,1]}(Pos > 3.2) \wedge \Diamond_{[1,1.5]}(\square_{[0,0.5]}(1.75 < Pos < 2.25)) \wedge \square_{[2,3]}(1.825 < Pos < 2.175)$	conjunctive requiremet $1.95 \leqslant Ref \leqslant 2.05$
CC1	$\square_{[0,100]} y_5 - y_4 \leqslant 40$	
CC2	$\square_{[0,70]}\Diamond_{[0,30]} y_5 - y_4 \geqslant 15$	
CC3	$\square_{[0,80]}((\square_{[0,20]} y_2 - y_1 \leqslant 20) \vee (\Diamond_{[0,20]} y_5 - y_4 \geqslant 40))$	
CC4	$\square_{[0,65]}\Diamond_{[0,30]}\square_{[0,20]} y_5 - y_4 \geqslant 8$	
CC5	$\square_{[0,72]}\Diamond_{[0,8]}((\square_{[0,5]} y_2 - y_1 \geqslant 9) \rightarrow (\square_{[5,20]} y_5 - y_4 \geqslant 9))$	
CCx	$\bigwedge_{i=1..4} \square_{[0,50]}(y_{i+1} - y_i > 7.5)$	conjunctive requirement
F16	$\square_{[0,15]}\, \mathit{altitude} > 0$	
SC	$\square_{[30,35]}(87 \leqslant \mathit{pressure} \wedge \mathit{pressure} \leqslant 87.5)$	

7.2 Results

There are mainly two sets of experiments depending upon the input parameterization. Instance 1 experiment refers to models whose inputs are parameterized by PCHIP interpolation, while Instance 2 refers to those whose inputs are parameterized by PCONST interpolation. Technically, since our task is to come up with control points, the dimensionality of the problem is equal to the number of control points that we are predicting. The summary of experiments and their control points is shown in table 7.1.

PART-X was run with an initialization budget $n_0 = 20$, per-subregion budget of unclassified subregions $n_{\text{BO}} = 10$, classified subregions budget $n_c = 100$, maximum budget $T = 2000$, number of Monte Carlo iterations $R = 20$, number of evaluations per iterations $M = 500$, number of cuts $B = 2$, and classification percentile $\delta_{\text{C}} = 0.05$. Also, we used $\delta_v = 0.001$ to identify dimensions that should not be branched. PART-X was run for 10 microreplication with these settings. Results shown in table 7.4 show the falsification rate out of 10, the mean and median number of simulations to find the first falsifying point, and the simulation time ratio for both instances. Along with this, the falsification volume is also reported. Results for PART-X are discussed in table 7.4. It can be seen that we get probabilistic estimates of falsifications at 95% confidence bounds. In some cases, like Steam Condenser for Instance 2, it can observed that even though no falsificiations are found, estimates for falsification volume still show up. In cases where do not get any probabilistic estimates (like AFC33 for Instance 2), the probabilistic estimates show up as we got to we increase our confidence bounds.

Next, the LS-EMIBO was run with an initialization budget $n_0 = 20$ along with bayesian optimization budget $n_{BO} = 1880$ and top-$k = 2$ in case of conjunctive requirements. When LS-EMIBO is run on the non-conjunctive requirement, it re-

Table 7.4: PART-X results for piecewise continuous input signals (instance 1) and constrained input signals (instance 2). *FR*: falsification rate, $\overline{S}$: mean number of simulations, $\widetilde{S}$: median (rounded down) number of simulations, *LCB*: Lower Confidence Bound at 95% confidence, *UCB*: Upper Confidence Bound at 95% confidence, *R*: $(\frac{SimulationTime}{TotalTime}) * 100$ (%).

Tool	**PSY-TaLiRo**						**PSY-TaLiRo**					
Approach	PART-X						PART-X					
Instance	1						1					
Property	*FR*	$\overline{S}$	$\widetilde{S}$	*LCB*	*UCB*	*R*	*FR*	$\overline{S}$	$\widetilde{S}$	*LCB*	*UCB*	*R*
AT1	10	34.9	28.5	0.00E+00	7.03E-04	70.71	10	30.5	25.5	0.00E+00	5.58E-04	85.71
AT2	10	6.7	5.5	9.45E-02	1.80E-01	52.78	10	6.5	5.0	1.16E-01	2.77E-01	50.64
AT51	0	2000.0	2000.0	0.00E+00	0.00E+00	93.88	10	13.3	11.5	2.22E-01	5.86E-01	64.31
AT52	10	5.6	2.0	1.81E-01	9.02E-01	62.53	10	66.5	53.5	0.00E+00	0.00E+00	93.46
AT53	10	15.7	15.5	2.45E-02	4.26E-01	59.72	10	2.2	2.0	8.38E-01	1.00E+00	57.05
AT54	3	1658.8	2000.0	0.00E+00	3.60E-05	90.97	10	85.0	65.0	0.00E+00	7.68E-02	76.17
AT6a	10	134.3	51.5	1.18E-01	2.47E-01	58.18	10	153.7	72.0	5.75E-02	1.94E-01	53.09
AT6b	10	212.2	150.0	9.45E-02	2.88E-01	57.83	10	307.9	111.5	3.40E-02	1.97E-01	56.44
AT6c	10	200.5	138.0	9.94E-02	2.86E-01	58.08	10	334.4	249.5	4.34E-02	1.98E-01	59.32
AT6abc	10	126.1	50.0	1.02E-01	2.67E-01	68.69	10	106.9	67.5	5.72E-02	2.06E-01	69.38
CC1	10	19.0	16.5	2.71E-01	8.31E-01	69.15	10	17.6	21.0	2.83E-01	8.86E-01	68.65
CC2	10	23.9	12.0	4.82E-01	1.00E+00	68.61	10	17.8	12.0	2.27E-01	1.00E+00	66.34
CC3	10	23.1	24.0	1.28E-01	4.58E-01	69.87	10	13.5	12.0	1.18E-01	1.00E+00	69.53
CC4	0	2000.0	2000.0	0.00E+00	0.00E+00	95.33	0	2000.0	2000.0	0.00E+00	0.00E+00	94.51
CC5	10	45.8	29.0	3.83E-02	7.10E-01	79.43	10	29.9	22.5	2.09E-01	5.90E-01	73.84
CCx	9	813.7	703.0	0.00E+00	0.00E+00	95.96	10	607.1	156.0	0.00E+00	0.00E+00	96.17
NN	10	15.2	16.0	4.84E-01	8.80E-01	83.54	10	145.8	89.5	0.00E+00	1.36E-01	87.31
NNx	-	-	-	-	-	-	10	190.7	40.0	0.00E+00	1.20E-02	66.43
SC	0	2000.0	2000.0	0.00E+00	0.00E+00	78.87	0	2000.0	2000.0	0.00E+00	2.70E-05	45.48
F16	0	2000.0	2000.0	0.00E+00	0.00E+00	39.22	-	-	-	-	-	-
AFC27	-	-	-	-	-	-	10	34.3	27.0	5.90E-01	7.27E-01	89.44
AFC29	-	-	-	-	-	-	10	12.1	11.0	2.31E-01	5.36E-01	87.88
AFC33	-	-	-	-	-	-	0	2000.0	2000.0	0.00E+00	0.00E+00	96.14

duces to the vanilla Bayesian optimization (BO). LS-EMIBO was run for 10 macro-replications. The results for instance 1 are reported in table 7.5 and instance 2 are reported in table 7.6. These are also compared with their respective counter part, the uniform random, which was also run for 10 microreplication with a budget of 2000 each. We can see that LS-EMIBO works efficiently when compared to their respective counterparts. One of the most significant results is the CCx experiment in both instances, where the LS-EMIBO algorithm finds a falsifying input in about a quarter of the simulations performed by the uniform random.

While comparing PART-X and LS-EMIBO, one might notice that the LS-EMIBO tends to find falsifications quickly in comparison to PART-X. However, not much can be said in PART-X, we do not have a criterion on which node of the PART-X tree should be explored first.

Table 7.5: LS-EMIBO results for piecewise continuous input signals (instance 1). *FR*: falsification rate, $\overline{S}$: mean number of simulations, $\widetilde{S}$: median (rounded down) number of simulations, *LCB*: Lower Confidence Bound at 95% confidence, *UCB*: Upper Confidence Bound at 95% confidence, *R*: $(\frac{SimulationTime}{TotalTime}) * 100(\%)$.

Tool: Approach:	**Uniform Random**				**PSY-TaLiRo** LSemiBO			
Benchmark	*FR*	$\overline{S}$	$\widetilde{S}$	*R*	*FR*	$\overline{S}$	$\widetilde{S}$	*R*
AT1	0	2000.0	2000.0	99.96	10	106.3	105.5	72.73
AT2	10	7.6	5.0	99.95	10	15.5	12.5	57.48
AT51	1	1892.3	2000.0	99.96	1	1802.2	2000.0	91.99
AT52	10	4.1	2.0	99.95	10	3.2	2.5	61.69
AT53	10	18.6	15.0	99.95	10	28.0	21.0	60.25
AT54	4	1648.5	2000.0	99.96	7	1100.7	973.0	90.63
AT6a	10	74.4	41.5	99.96	10	76.6	89.0	58.08
AT6b	10	251.3	189.0	99.96	7	898.3	373.0	58.24
AT6c	10	185.2	86.0	99.96	9	495.4	167.5	53.27
AT6abc	10	58.8	33.5	99.97	10	34.2	29.5	37.02
NN	10	38.6	27.5	99.98	10	36.4	35.5	84.56
CC1	10	10.4	9.5	99.98	10	13.1	8.5	71.55
CC2	10	15.4	15.0	99.97	10	16.4	11.0	65.45
CC3	10	77.9	54.5	99.98	10	21.5	15.0	71.75
CC4	0	2000.0	2000.0	99.98	1	1925.3	2000.0	93.58
CC5	10	28.5	14.5	99.98	10	47.3	39.0	84.76
CCx	9	801.0	472.5	99.98	10	210.6	70.0	20.82
F16	0	2000.0	2000.0	99.90	-	-	-	-
SC	0	2000.0	2000.0	99.88	-	-	-	-

Table 7.6: LS-EMIBO results for piecewise constrained input signals (instance 1). *FR*: falsification rate, $\overline{S}$: mean number of simulations, $\widetilde{S}$: median (rounded down) number of simulations, *LCB*: Lower Confidence Bound at 95% confidence, *UCB*: Upper Confidence Bound at 95% confidence, *R*: $(\frac{SimulationTime}{TotalTime}) * 100(\%)$.

Tool: Approach:	**Uniform Random**				**S-TaLiRo** LSemiBO			
Benchmark	*FR*	$\overline{S}$	$\widetilde{S}$	*R*	*FR*	$\overline{S}$	$\widetilde{S}$	*R*
AT1	0	2000.0	2000.0	99.95	10	105.4	105.0	86.77
AT2	10	18.8	13.5	99.95	10	11.6	11.0	50.26
AT51	10	20.5	16.5	99.96	10	13.7	8.5	69.32
AT52	10	74.1	65.0	99.96	10	79.1	95.0	92.88
AT53	10	1.5	1.0	99.95	10	2.7	2.0	57.06
AT54	10	47.9	42.0	99.96	10	37.7	32.0	79.63
AT6a	10	156.6	138.0	99.96	9	429.5	252.5	61.66
AT6b	10	472.2	588.0	99.96	9	722.0	523.5	49.41
AT6c	10	326.8	176.0	99.96	8	689.2	248.5	54.73
AT6abc	10	149.0	125.5	99.97	10	240.5	74.0	7.76
AFC27	10	2000.0	2000.0	99.99	10	113.2	109.5	97.75
AFC29	10	25.1	19.0	99.99	10	19.6	19.0	100.00
AFC33	10	2000.0	2000.0	99.99	0	2000.0	2000.0	35.01
NN	10	277.2	158.5	99.99	10	155.5	100.0	88.79
NNx	10	712.7	488.0	99.95	10	46.8	48.0	37.25
CC1	10	16.4	9.5	99.97	10	10.8	8.0	73.94
CC2	10	12.4	13.0	99.97	10	9.6	7.0	68.30
CC3	10	19.6	21.0	99.98	10	11.7	8.0	70.54
CC4	0	2000.0	2000.0	99.98	3	1882.4	2000.0	96.94
CC5	10	37.4	22.0	99.98	10	28.3	27.0	75.33
CCx	7	1029.8	735.0	99.98	10	240.5	74.0	35.38
SC	0	2000.0	2000.0	99.99	-	-	-	-

Chapter 8

CONCLUSION

With this chapter, we come to the conclusion of the thesis. The thesis started by stating the three research questions:

RQ1. *In black-box systems, when no information is available, can we find falsifying behaviors, while also providing probabilistic estimates of a point leading to falsification as well as regions that can lead to finding inputs that produce a falsifying behavior?*

Response: Through the work in chapter 4, it was shown that the PART-X algorithm is indeed able to find structure in the problem and provide point and region-wide probability estimates. These can be of use to practitioners when trying to find falsifications in a complex system, and further investigate the probably falsifying regions with other SBTG methods.

RQ2. *In gray-box systems, when we have some knowledge about the dynamics of the system-under-test, can we come up method that can utilize this information in order to find inputs that lead to falsifying behaviors?*

Response: In chapter 5, we show that global restart points with aim of minimizing robustness coupled with local search to move towards exploring new states can lead to utilizing the information from the dynamics to find falsifying behaviors. This is shown through an example of the 2-state hybrid automaton in [1] where PYSOAR-C is compared with the uniform random sampler and PYSOAR. Extensive experimentation will be shown in the future idea where we explore this idea and prove it efficacy through experimentation on complex

cyber-physical systems.

RQ3. *If we have some knowledge on the structure of the requirement, can we exploit the structure in order to efficiently come up with faster falsification?*

Response: In chapter 6, finding falsifications in conjunctive requirements is formulated such that we consider the requirements individually and utilize the $emi - EI$ to get the next best sample locations. Results on this are shown through extensive experiments on numerical experiments as well as on complex cyber-physical systems from [14].

This thesis has explored various algorithms that can be applied to different dynamic systems depending upon the type of information available. While these are just initial results, we plan to test these algorithms extensively on more complex cyber-physical systems.

REFERENCES

[1] Adel Dokhanchi, Aditya Zutshi, Rahul T. Sriniva, Sriram Sankaranarayanan, and Georgios Fainekos. Requirements driven falsification with coverage metrics. In *2015 International Conference on Embedded Software (EMSOFT)*, pages 31–40, 2015.

[2] Robert N Charette. How software is eating the car. *IEEE Spectrum, June*, 2021.

[3] Ye Yuan, Xiuchuan Tang, Wei Zhou, Wei Pan, Xiuting Li, Hai-Tao Zhang, Han Ding, and Jorge Goncalves. Data driven discovery of cyber physical systems. *Nature communications*, 10(1):1–9, 2019.

[4] Sarah M. Loos, André Platzer, and Ligia Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In Michael Butler and Wolfram Schulte, editors, *FM 2011: Formal Methods*, pages 42–56, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[5] Houbing Song, Danda B Rawat, Sabina Jeschke, and Christian Brecher. *Cyber-physical systems: foundations, principles and applications*. Morgan Kaufmann, 2016.

[6] Jyotirmoy V. Deshmukh and Sriram Sankaranarayanan. *Formal Techniques for Verification and Testing of Cyber-Physical Systems*, pages 69–105. Springer International Publishing, Cham, 2019.

[7] Zhihao Jiang, Miroslav Pajic, Rajeev Alur, and Rahul Mangharam. Closed-loop verification of medical devices with model abstraction and refinement. *Int. J. Softw. Tools Technol. Transf.*, 16(2):191–213, apr 2014.

[8] Joseph B Lyons, Nhut T Ho, William E Fergueson, Garrett G Sadler, Samantha D Cals, Casey E Richardson, and Mark A Wilkins. Trust of an automatic ground collision avoidance technology: A fighter pilot perspective. *Military Psychology*, 28(4):271–277, 2016.

[9] Greg Zacharias. *Autonomous horizons: the way forward*. Air University Press; Curtis E. LeMay Center for Doctrine Development and Education, Maxwell Air Force Base, Alabama, 2019.

[10] Frank J. Furrer. *Cyber-Physical Systems*, pages 9–76. Springer Fachmedien Wiesbaden, Wiesbaden, 2022.

[11] James Kapinski, Jyotirmoy V Deshmukh, Xiaoqing Jin, Hisahiro Ito, and Ken Butts. Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. *IEEE Control Systems Magazine*, 36(6):45–64, 2016.

[12] Logan Mathesen, Shakiba Yaghoubi, Giulia Pedrielli, and Georgios Fainekos. Falsification of cyber-physical systems with robustness uncertainty quantification through stochastic optimization with adaptive restart. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 991–997, 2019.

[13] Logan Mathesen, Giulia Pedrielli, Szu Hui Ng, and Zelda B. Zabinsky. Stochastic optimization with adaptive restart: A framework for integrated local and global learning. *J. of Global Optimization*, 79(1):87–110, jan 2021.

[14] Gidon Ernst, Paolo Arcaini, Ismail Bennani, Aniruddh Chandratre, Alexandre Donz\'e, Georgios Fainekos, Goran Frehse, Khouloud Gaaloul, Jun Inoue, Tanmay Khandait, Logan Mathesen, Claudio Menghi, Giulia Pedrielli, Marc Pouzet, Masaki Waga, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. Archcomp 2021 category report: Falsification with validation of results. In Goran Frehse and Matthias Althoff, editors, *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)*, volume 80 of *EPiC Series in Computing*, pages 133–152. EasyChair, 2021.

[15] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification*, pages 167–170. Springer, 2010.

[16] Georgios E Fainekos and George J Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[17] Houssam Abbas, Georgios Fainekos, Sriram Sankaranarayanan, Franjo Ivančić, and Aarti Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 12(2s), may 2013.

[18] Dejan Ničković and Tomoya Yamaguchi. Rtamt: Online robustness monitors from stl. In *International Symposium on Automated Technology for Verification and Analysis*, pages 564–571. Springer, 2020.

[19] Joseph Cralley, Ourania Spantidi, Bardh Hoxha, and Georgios Fainekos. Tltk: A toolbox for parallel robustness computation of temporal logic specifications. In *International Conference on Runtime Verification*, pages 404–416. Springer, 2020.

[20] Quinn Thibeault, Jacob Anderson, Aniruddh Chandratre, Giulia Pedrielli, and Georgios Fainekos. Psy-taliro: A python toolbox for search-based test generation for cyber-physical systems. In Alberto Lluch Lafuente and Anastasia Mavridou, editors, *Formal Methods for Industrial Critical Systems*, pages 223–231, Cham, 2021. Springer International Publishing.

[21] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram 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*, pages 254–257. Springer, 2011.

[22] Robert L Grossman, Anil Nerode, Anders P Ravn, and Hans Rischel. *Hybrid systems*, volume 736. Springer, 1993.

[23] Yashwanth Singh Rahul Annapureddy and Georgios E. Fainekos. Ant colonies for temporal logic falsification of hybrid systems. In *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, pages 91–96, 2010.

[24] Truong Nghiem, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancić, Aarti Gupta, and George J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '10, page 211–220, New York, NY, USA, 2010. Association for Computing Machinery.

[25] Jyotirmoy Deshmukh, Xiaoqing Jin, James Kapinski, and Oded Maler. Stochastic local search for falsification of hybrid systems. In Bernd Finkbeiner, Geguang Pu, and Lijun Zhang, editors, *Automated Technology for Verification and Analysis*, pages 500–517, Cham, 2015. Springer International Publishing.

[26] Fabio Schoen. Two-phase methods for global optimization. In *Handbook of global optimization*, pages 151–177. Springer, 2002.

[27] Jan Kuřátko and Stefan Ratschan. Combined global and local search for the falsification of hybrid systems. In Axel Legay and Marius Bozga, editors, *Formal Modeling and Analysis of Timed Systems*, pages 146–160, Cham, 2014. Springer International Publishing.

[28] Arvind Adimoolam, Thao Dang, Alexandre Donzé, James Kapinski, and Xiaoqing Jin. Classification and coverage-based falsification for embedded control systems. In *International Conference on Computer Aided Verification*, pages 483–503. Springer, 2017.

[29] Houssam Abbas, Andrew Winn, Georgios Fainekos, and A Agung Julius. Functional gradient descent method for metric temporal logic specifications. In *2014 American Control Conference*, pages 2312–2317. IEEE, 2014.

[30] Shakiba Yaghoubi and Georgios Fainekos. Hybrid approximate gradient and stochastic descent for falsification of nonlinear systems. In *2017 American Control Conference (ACC)*, pages 529–534, 2017.

[31] Shakiba Yaghoubi and Georgios Fainekos. Local descent for temporal logic falsification of cyber-physical systems. In *International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems*, pages 11–26. Springer, 2017.

[32] Shakiba Yaghoubi and Georgios Fainekos. Falsification of temporal logic requirements using gradient based local search in space and time. *IFAC-PapersOnLine*, 51(16):103–108, 2018.

[33] Ezio Bartocci, Luca Bortolussi, Laura Nenzi, and Guido Sanguinetti. System design of stochastic models using robustness of temporal properties. *Theoretical Computer Science*, 587:3–25, 2015. Interactions between Computer Science and Biology.

[34] Gang Chen, Zachary Sabato, and Zhaodan Kong. Active requirement mining of bounded-time temporal properties of cyber-physical systems. *arXiv preprint arXiv:1603.00814*, 2016.

[35] Takumi Akazaki. Falsification of conditional safety properties for cyber-physical systems with gaussian process regression. In Yliès Falcone and César Sánchez, editors, *Runtime Verification*, pages 439–446, Cham, 2016. Springer International Publishing.

[36] Simone Silvetti, Alberto Policriti, and Luca Bortolussi. An active learning approach to the falsification of black box cyber-physical systems. In Nadia Polikarpova and Steve Schneider, editors, *Integrated Formal Methods*, pages 3–17, Cham, 2017. Springer International Publishing.

[37] Jyotirmoy Deshmukh, Marko Horvat, Xiaoqing Jin, Rupak Majumdar, and Vinayak S. Prabhu. Testing cyber-physical systems through bayesian optimization. *ACM Trans. Embed. Comput. Syst.*, 16(5s), sep 2017.

[38] Shromona Ghosh, Felix Berkenkamp, Gireeja Ranade, Shaz Qadeer, and Ashish Kapoor. Verifying controllers against adversarial examples with bayesian optimization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7306–7313. IEEE, 2018.

[39] Masaki Waga. Falsification of cyber-physical systems with robustness-guided black-box checking. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, HSCC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[40] Chuchu Fan, Xin Qin, and Jyotirmoy Deshmukh. Parameter searching and partition with probabilistic coverage guarantees. *arXiv preprint arXiv:2004.00279*, 2020.

[41] Logan Mathesen, Giulia Pedrielli, and Georgios Fainekos. Efficient optimization-based falsification of cyber-physical systems with multiple conjunctive requirements. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 732–737, 2021.

[42] Zhenya Zhang and Paolo Arcaini. Gaussian process-based confidence estimation for hybrid system falsification. In *International Symposium on Formal Methods*, pages 330–348. Springer, 2021.

[43] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *Ieee access*, 2:56–77, 2014.

[44] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. Hybrid systems: from verification to falsification by combining motion planning and discrete search. *Formal Methods in System Design*, 34(2):157–182, 2009.

[45] Tarik Nahhal and Thao Dang. Test coverage for continuous and hybrid systems. In *International Conference on Computer Aided Verification*, pages 449–462. Springer, 2007.

[46] Tommaso Dreossi, Thao Dang, Alexandre Donzé, James Kapinski, Xiaoqing Jin, and Jyotirmoy V Deshmukh. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *NASA Formal Methods Symposium*, pages 127–142. Springer, 2015.

[47] Zhenya Zhang, Gidon Ernst, Sean Sedwards, Paolo Arcaini, and Ichiro Hasuo. Two-layered falsification of hybrid systems guided by monte carlo tree search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2894–2905, 2018.

[48] Gidon Ernst, Sean Sedwards, Zhenya Zhang, and Ichiro Hasuo. Fast falsification of hybrid systems using probabilistically adaptive input. In *International Conference on Quantitative Evaluation of Systems*, pages 165–181. Springer, 2019.

[49] Zhenya Zhang, Deyun Lyu, Paolo Arcaini, Lei Ma, Ichiro Hasuo, and Jianjun Zhao. Effective hybrid system falsification using monte carlo tree search guided by qb-robustness. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification*, pages 595–618, Cham, 2021. Springer International Publishing.

[50] Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. Sample-efficient neural architecture search by learning actions for monte carlo tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[51] Takumi Akazaki, Shuang Liu, Yoriyuki Yamagata, Yihai Duan, and Jianye Hao. Falsification of cyber-physical systems using deep reinforcement learning. In Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik de Vink, editors, *Formal Methods*, pages 456–465, Cham, 2018. Springer International Publishing.

[52] Shaohua Zhang, Shuang Liu, Jun Sun, Yuqi Chen, Wenzhi Huang, Jinyi Liu, Jian Liu, and Jianye Hao. Figcps: Effective failure-inducing input generation for cyber-physical systems with deep reinforcement learning. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 555–567, 2021.

[53] Edmund M Clarke, Orna Grumberg, and David E Long. Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

[54] Aditya Zutshi, Jyotirmoy V Deshmukh, Sriram Sankaranarayanan, and James Kapinski. Multiple shooting, cegar-based falsification for hybrid systems. In *Proceedings of the 14th International Conference on Embedded Software*, pages 1–10, 2014.

[55] Claudio Menghi, Shiva Nejati, Lionel Briand, and Yago Isasi Parache. Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 372–384, New York, NY, USA, 2020. Association for Computing Machinery.

[56] Lennart Ljung. System identification toolbox. *The Matlab user's guide*, 1988.

[57] Giulia Pedrielli, Tanmay Kandhait, Surdeep Chotaliya, Quinn Thibeault, Hao Huang, Mauricio Castillo-Effen, and Georgios Fainekos. Part-x: A family of stochastic algorithms for search-based test generation with probabilistic guarantees. *arXiv preprint arXiv:2110.10729*, 2021.

[58] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[59] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[60] Thomas J Santner, Brian J Williams, William I Notz, and Brain J Williams. *The design and analysis of computer experiments*, volume 1. Springer, 2003.

[61] Pritam Ranjan, Ronald Haynes, and Richard Karsten. A computationally stable approach to gaussian process interpolation of deterministic computer simulation data. *Technometrics*, 53(4):366–378, 2011.

[62] Isaac E Lagaris and Ioannis G Tsoulos. Stopping rules for box-constrained stochastic global optimization. *Applied Mathematics and Computation*, 197(2):622–632, 2008.

[63] Bardh Hoxha, Houssam Abbas, and Georgios Fainekos. Benchmarks for temporal logic requirements for automotive systems. In Goran Frehse and Matthias Althoff, editors, *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, volume 34 of *EPiC Series in Computing*, pages 25–30. EasyChair, 2015.

[64] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 160–173. Springer, 2000.

[65] Shakiba Yaghoubi and Georgios Fainekos. Gray-box adversarial testing for control systems with machine learning components. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 179–184, 2019.

[66] Xiaoqing Jin, Jyotirmoy V Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. Powertrain control verification benchmark. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 253–262, 2014.

[67] Frederick N Fritsch and Ralph E Carlson. Monotone piecewise cubic interpolation. *SIAM Journal on Numerical Analysis*, 17(2):238–246, 1980.