ALGORITHMS FOR EFFICIENT VALIDATION OF BLACK-BOX SYSTEMS

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE
WITH A DISTINCTION IN RESEARCH

Robert John Moss
April 2021

I certify that I have read this thesis and that, in my opinion, it is fully adequate in scope and quality as a thesis for the degree of Master of Science.

Mykel J. Kochenderfer, Principal Adviser

I certify that I have read this thesis and that, in my opinion, it is fully adequate in scope and quality as a thesis for the degree of Master of Science.

Dorsa Sadigh

Abstract

Before safety-critical autonomous systems are deployed into the real-world, we must first ensure their validity. One common approach for validation is to stress test these systems in simulation. This thesis proposes several techniques to aid in efficient stress testing of black-box systems, especially when those systems are computationally expensive to evaluate. We first introduce novel variants of the cross-entropy method for stochastic optimization used to find rare failure events. The original cross-entropy method relies on enough objective function calls to accurately estimate the optimal parameters of the proposal distribution and may get stuck in local minima. The variants we introduce attempt to address these concerns and the primary idea is to use every sample to build a surrogate model to offload computation from an expensive system under test. To test our approach, we created a parameterized test objective function with many local minima and a single global minimum, where the test function can be adjusted to control the spread and distinction of the minima.

To find failure events and their likelihoods in computationally expensive open-loop systems, we propose a modification to the black-box stress testing approach called *adaptive stress testing*. This modification generalizes adaptive stress testing to be broadly applied to episodic systems, where a reward is only received at the end of an episode. To test this approach, we analyze an aircraft trajectory predictor from a developmental commercial flight management system. The intention of this work is to find likely failures and report them back to the developers so they can address and potentially resolve shortcomings of the system before deployment. We use a modified Monte Carlo tree search algorithm with progressive widening as our adversarial reinforcement learner with the goal of finding potential problems otherwise not found by traditional requirements-based avionics testing.

When validating a system that relies on a static validation dataset, one could exhaustively evalute the entire dataset, yet that process may be computationally intractable especially when validating minor modification to the system under test. To address this, we reformulate the problem to intelligently select candidate validation data points that we predict to likely cause a failure, using knowledge of the system failures experienced so far. We propose an adaptive black-box validation framework that will learn system weaknesses over time and exploit this knowledge to propose

validation samples that will likely result in a failure. To further reduce computational load, we use a low-dimensional encoded representation of inputs to train the adversarial failure classifier, which will select candidate failures to evaluate.

A motivating principle of this work is a committement to open-source software. We believe everyone benefits when we treat ideas for black-box validation as collaborative and non-competitive; thus resulting in a net safety increase for all. The core software for each of the introduced techniques have been developed as Julia packages and publically released under the MIT license. We introduce the software and their usages at a high level and discuss alternative applications from both a research and industrial perspective.

Acknowledgments

I would especially like to thank my advisor and friend, Mykel Kochenderfer, for his continued guidance over the past decade. Our time together at MIT Lincoln Laboratory has certainly shaped me into the researcher am I today. The positivity in Mykel's leadership is inspirational and his high level of integrity and honesty encouraged me to always do my best. He is a model leader and I thank him for the incredible opportunity both at MIT Lincoln Laboratory at here at Stanford. I'd also like to thank Professor Dorsa Sadigh for being my secondary research adviser and for her advice regarding this thesis.

As with many theses, I am standing on the shoulders of giants. I would like to thank Ritchie Lee from NASA Ames for his original development of the adaptive stress testing approach and for his patience and guidance as he helped shape my own ideas. I want to thank Anthony Corso, Mark Koren, and Alex Koufos for always listening to my ideas, encouraging my excitement in the AI safety field, and always providing constructive feedback. Without their advice, this work would not have been possible. I'd also like to thank members of the Stanford Intelligent Systems Laboratory (SISL) for their encouragement and willingness to listen; particularly Bernard Lange, Shushman Choudhury, Jayesh Gupta, Sydney Katz, and Tomer Arnon. Because this work is built off of other open-source software, I'm forever indebted to the SISL members that developed the POMDPs.jl ecosystem and other open-source tools; this includes Zachary Sunberg, Maxim Egorov, Tim Wheeler, and Xiaobai Ma. I also acknowledge Shreyas Kowshik for his initial implementation of the TRPO and PPO algorithms in Julia. I want to extend a thank you to Edward Balaban at NASA Ames for the opportunity to work on a decision making under uncertainty system in a high-profile NASA mission.

I have also had the pleasure to work with James G. O'Brien as an undergraduate researcher while at Wentworth and I'm grateful for his inspiration and trust he provided me. I owe an enormous debt to everyone I have interacted with while at MIT Lincoln Laboratory over the years—they each played a role in shaping me into a better scientist, engineer, analyst, and friend. This includes Ted Londner, Michael Owen, Wes Olson, Ian Jessen, Adam Panken, Tomas Elder, Cindy McLain, Luis Alvarez, Tan Trinh, Robert Klaus, Justin MacKay, Jared Wikle, Emilie Cowen, Tom Teller, Adam Gjersvik, Sam Wu, Anshu Das, Jack Lepird, Charles Leeper, and Dan Griffith. Particular

appreciation goes to Jeff Bezanson for Julia and showing me its divinity. Also a special thanks to Neal Suchy from the Federal Aviation Administration for his admiration and his faith in my work.

Part of this work had the support from GE's Global Research Center and GE Aviation through the Stanford Center for AI Safety. I want to thank each of these organizations for their fascinating challenges and allowing me to explore research ideas that fit not only my interests but had large industrial impacts. I also want to thank the NASA AOSP System-Wide Safety Project for partially supporting this work and Jerry Lopez, Nicholas Visser, and Joachim Hochwarth for their engineering guidance.

My family and friends have always been there for me, even as we are physically distant. My Mom, Dad, and siblings—Travis, Emily, and Jake—are a big reason I have core values that have helped me succeed. Their love and support is infinite and I could not thank them enough for the life they've provided me. My wife's family have also been incredibly supportive and I am thankful for their immense encouragement. To everyone back in Rockport, MA and beyond, you've seen me grown through every phase in my life, and that bond is irreplaceable; so thank you.

Lastly, but most importantly, I want to thank my wife, Eva Moss, for always being supportive and growing with me during my graduate studies. Eva, you always make me laugh, smile, and think deeply, which has shaped me into a better person because of it. Your logical thinking helps me check my opinions at the door. Your flexibility in leaving our home back in Massachusetts and moving out to California tremendously helped in reducing the stress of graduate school—I love you and I am forever grateful.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the expanding use of artificial intelligent algorithms to solve complex safety-critical problems, there is an increasing need to ensure the validity of such systems. It may be challenging to exhaustively validate a complex system due to the continuous nature of systems deployed in the real-world. Therefore, to validate such systems, we need to rely on simulation. As with all simulations, we model the real-world through a series of approximations, yet despite these approximations, it may still be computationally difficult to find system failures efficiently. The validation problem becomes even more challenging when failures are extremely rare and where a simple random search may severely underestimate the probability of failure (or even estimate it as zero). This work purposes several methods to try and efficiently search for likely failures in safety-critical systems, modeling the system as a black box.

The term *black box* refers to a software system for which we pass inputs and only have access to the provided outputs (i.e., the internals of the black-box system are unknown to us). Framing the problem around black-box systems enables these techniques to be broadly applied to existing systems without the need to gain access to the internal code itself. Techniques that require knowledge of the system internals are termed *white box*, which are not the focus of this work. However, we may use the term *gray box* when we need access to information that's part of the simulation environment in which the system is operating in (e.g., access to transition probabilities in the environment).

The validation problem can be split into *falsification* (i.e., finding failures) and *most-likely failure analysis* (i.e., finding likely failures). We provide several approaches to falsification and build off the *adaptive stress testing* [42] problem formulation for finding likely failures in black-box systems. We consider two types of systems we want to validate: open-loop systems with episodic rewards (i.e., a controller that selects a sequence of actions without feedback from the output) and classification systems (e.g., a hand-written digit classifier). Each type has their own respective challenges to consider. Confronted with computationally expensive systems, we introduce methods to intelligently select when we execute the system to reduce unnecessary evaluations.

1.1 Contributions

The contributions of this thesis are primarily algorithmic with accompanying open-source software tools. Most of this work has been published in conferences [50], open-source journals [49], or is available online [48]. Section 6.1 provides a more detailed description of the contributions of this thesis, but the main contributions can be summarized as follows:

- Two stochastic optimization algorithms called the *cross-entropy surrogate method* and the *cross-entropy mixture method* designed for finding rare failure events using fewer objective function evaluations. We also introduce a novel optimization test function called *sierra* with user-defined control over the spread of local minima and a distinct global minimum.
- A modification to the reinforcement learning-based *adaptive stress testing* problem formulation that is more broadly applicable for episodic-based, open-loop sequential systems (i.e., systems that only receive a reward signal at the end of an episode). We apply this technique to stress test an aircraft trajectory predictor in a developmental commercial flight management system—intended to be complementary to requirements-based avionics testing.
- An adaptive framework for validation of large, static datasets using adversarial weakness recognition to intelligently select candidate inputs that are predicted to be likely failures.
- Open-source software tools to apply general black-box adaptive stress testing (POMDPStressTesting.jl),[1] the cross-entropy method algorithm variants (CrossEntropyVariants.jl),[2] and adversarial weakness recognition (FailureRepresentation.jl).[3]

1.2 Outline

Chapter 2 provides background for the cross-entropy method algorithm and details the development of several novel variants applied to a rare-event simulation. Chapter 3 introduces adaptive stress testing and proposes a reformulation of the problem around sequential systems with episodic rewards, applying the proposed technique to find likely aircraft trajectory prediction failures as part of a developmental commercial flight management system. Chapter 4 introduces a framework for intelligently selecting candidate failure cases from a large, static dataset used to validate a neural network-based classification system. Chapter 5 details the open-source software developed to support this thesis, and chapter 6 provides a detailed summary of the contributions and extensions for future work.

[1] `https://github.com/sisl/POMDPStressTesting.jl`
[2] `https://github.com/mossr/CrossEntropyVariants.jl`
[3] `https://github.com/sisl/FailureRepresentation.jl`

Chapter 2

Cross-Entropy Surrogate Method

The cross-entropy (CE) method is a probabilistic optimization approach that attempts to iteratively fit a distribution to elite samples from an initial proposal distribution [55, 54]. The goal is to estimate a rare-event probability by minimizing the *cross-entropy* between the two distributions [17]. The CE-method has gained popularity in part due to its simplicity in implementation and straightforward derivation. The technique uses *importance sampling* which introduces a proposal distribution over the rare-events to sample from then re-weights the posterior likelihood by the *likelihood ratio* of the true distribution over the proposal distribution.

There are a few key assumptions that make the CE-method work effectively. Through random sampling, the CE-method assumes that there are enough objective function evaluations to accurately represent the objective. This may not be a problem for simple applications, but can be an issue for computationally expensive objective functions. Another assumption is that the initial parameters of the proposal distribution are wide enough to cover the design space of interest. For the case with a multivariate Gaussian distribution, this corresponds to an appropriate mean and wide covariance. In rare-event simulations with many local minima, the CE-method can fail to find the global minimum especially with sparse objective function evaluations.

This work aims to address the key assumptions of the CE-method. We introduce variants of the CE-method that use surrogate modeling to approximate the objective function, thus updating the belief of the underlying objective through estimation. As part of this approach, we introduce evaluation scheduling techniques to reallocate true objective function calls earlier in the optimization when we know the covariance will be large. The evaluation schedules can be based on a distribution (e.g., the Geometric distribution) or can be prescribed manually depending on the problem. We also use a Gaussian mixture model representation of the prior distribution as a method to explore competing local optima. While the use of Gaussian mixture models in the CE-method is not novel, we connect the use of mixture models and surrogate modeling in the CE-method. This connection uses each elite sample as the mean of a component distribution in the mixture, optimized through

a subroutine call to the standard CE-method using the learned surrogate model. To test our approach, we introduce a parameterized test objective function called *sierra*. The sierra function is built from a multivariate Gaussian mixture model with many local minima and a single global minimum. Parameters for the sierra function allow control over both the spread and distinction of the minima. Lastly, we provide an analysis of the weak areas of the CE-method compared to our proposed variants.

2.1 Related Work

The cross-entropy method is popular in the fields of operations research, machine learning, and optimization [31, 32]. The combination of the cross-entropy method, surrogate modeling, and mixture models has been explored in other work [7]. Bardenet and Kégl [7] proposed an adaptive grid approach to accelerate Gaussian-process-based surrogate modeling using mixture models as the prior in the cross-entropy method. Unsurprisingly, they showed that a mixture model performs better than a single Gaussian when the objective function is multimodal. Our work differs in that we augment the "elite" samples both by an approximate surrogate model and by a subroutine call to the CE-method using the learned surrogate model. Other related work use Gaussian processes and a modified cross-entropy method for receding-horizon trajectory optimization [60]. Their cross-entropy method variant also incorporates the notion of exploration in the context of path finding applications. An approach based on *relative entropy*, described in section 2.2.1, proposed a model-based stochastic search that seeks to minimize the relative entropy [2]. They also explore the use of a simple quadratic surrogate model to approximate the objective function. Prior work that relate cross-entropy-based adaptive importance sampling with Gaussian mixture models show that when using mixture models, fewer objective function calls are required relative to a naïve Monte Carlo or standard unimodal cross-entropy-based importance sampling method [37, 63].

2.2 Background

This section provides necessary background on techniques used in this work. We provide introductions to cross-entropy and the cross-entropy method, surrogate modeling using Gaussian processes, and multivariate Gaussian mixture models.

2.2.1 Cross-Entropy

Before understanding the cross-entropy method, we first must understand the notion of *cross-entropy*. Cross-entropy is a metric used to measure the distance between two probability distributions, where the distance may not be symmetric [17]. The distance used to define cross-entropy is called the *Kullback-Leibler (KL) distance* or *KL divergence*. The KL distance is also called the

relative entropy, and we can use this to derive the cross-entropy. Formally, for a random variable $\mathbf{X} = (X_1, \ldots, X_n)$ with a support of $\mathcal{X}$, the KL distance between two continuous probability density functions f and g is defined to be:

$$\begin{aligned}\mathcal{D}_{\mathrm{KL}}(f \mid\mid g) &= \mathbb{E}_f\left[\log \frac{f(\mathbf{X})}{g(\mathbf{X})}\right] \\ &= \int_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) \log f(\mathbf{x}) d\mathbf{x} - \int_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) \log g(\mathbf{x}) d\mathbf{x}\end{aligned}$$

We denote the expectation of some function with respect to a distribution f as $\mathbb{E}_f$. Minimizing the KL distance $\mathcal{D}_{\mathrm{KL}}$ between our true distribution f and our proposal distribution g parameterized by $\boldsymbol{\theta}$, is equivalent to choosing $\boldsymbol{\theta}$ that minimizes the following, called the *cross-entropy*:

$$\begin{aligned}H(f, g) &= H(f) + \mathcal{D}_{\mathrm{KL}}(f \mid\mid g) \\ &= -\mathbb{E}_f[\log g(\mathbf{X})] && \text{(using KL distance)} \\ &= -\int_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) \log g(\mathbf{x}; \boldsymbol{\theta}) d\mathbf{x}\end{aligned}$$

where $H(f)$ denotes the entropy of the distribution f (where we conflate entropy and continuous entropy for convenience). This assumes that f and g share the support $\mathcal{X}$ and are continuous with respect to $\mathbf{x}$. The minimization problem then becomes:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad -\int_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) \log g(\mathbf{x}; \boldsymbol{\theta}) d\mathbf{x} \tag{2.1}$$

Efficiently finding this minimum is the goal of the cross-entropy method algorithm.

2.2.2 Cross-Entropy Method

Using the definition of cross-entropy, intuitively the *cross-entropy method* (CEM or CE-method) aims to minimize the cross-entropy between the unknown true distribution f and a proposal distribution g parameterized by $\boldsymbol{\theta}$. This technique reformulates the minimization problem as a probability estimation problem, and uses adaptive importance sampling to estimate the unknown expectation [17]. The cross-entropy method has been applied in the context of both discrete and continuous optimization problems [54, 36].

The initial goal is to estimate the probability

$$\ell = P_{\boldsymbol{\theta}}(S(\mathbf{x}) \geq \gamma)$$

where S can the thought of as an objective function of $\mathbf{x}$, and $\mathbf{x}$ follows a distribution defined by

$g(\mathbf{x};\boldsymbol{\theta})$. We want to find events where our objective function S is above some threshold γ. We can express this unknown probability as the expectation

$$\ell = \mathbb{E}_{\boldsymbol{\theta}}\Big[\mathbb{1}\{S(\mathbf{x}) \geq \gamma\}\Big] \tag{2.2}$$

where $\mathbb{1}$ denotes the indicator function. A straightforward way to estimate eq. (2.2) can be done through Monte Carlo sampling. But for rare-event simulations where the probability of a target event occurring is relatively small, this estimate becomes inadequate. The challenge of the minimization in eq. (2.1) then becomes choosing the density function for the true distribution $f(\mathbf{x})$. Importance sampling tells us that the optimal importance sampling density can be reduced to

$$f^*(\mathbf{x}) = \frac{\mathbb{1}\{S(\mathbf{x}) \geq \gamma\} g(\mathbf{x};\boldsymbol{\theta})}{\ell}$$

thus resulting in the optimization problem:

$$\begin{aligned}
\boldsymbol{\theta}_g^* &= \arg\min_{\boldsymbol{\theta}_g} - \int_{\mathbf{x}\in\mathcal{X}} f^*(\mathbf{x}) \log g(\mathbf{x};\boldsymbol{\theta}_g) d\mathbf{x} \\
&= \arg\min_{\boldsymbol{\theta}_g} - \int_{\mathbf{x}\in\mathcal{X}} \frac{\mathbb{1}\{S(\mathbf{x}) \geq \gamma\} g(\mathbf{x};\boldsymbol{\theta})}{\ell} \log g(\mathbf{x};\boldsymbol{\theta}_g) d\mathbf{x}
\end{aligned}$$

Note that since we assume f and g belong to the same family of distributions, we get that $f(\mathbf{x}) = g(\mathbf{x};\boldsymbol{\theta}_g)$. Now notice that ℓ is independent of $\boldsymbol{\theta}_g$, thus we can drop ℓ and get the final optimization problem of:

$$\begin{aligned}
\boldsymbol{\theta}_g^* &= \arg\min_{\boldsymbol{\theta}_g} - \int_{\mathbf{x}\in\mathcal{X}} \mathbb{1}\{S(\mathbf{x}) \geq \gamma\} g(\mathbf{x};\boldsymbol{\theta}) \log g(\mathbf{x};\boldsymbol{\theta}_g) d\mathbf{x} \\
&= \arg\min_{\boldsymbol{\theta}_g} -\mathbb{E}_{\boldsymbol{\theta}}[\mathbb{1}\{S(\mathbf{x}) \geq \gamma\} \log g(\mathbf{x};\boldsymbol{\theta}_g)]
\end{aligned} \tag{2.3}$$

The CE-method uses a multi-level algorithm to estimate $\boldsymbol{\theta}_g^*$ iteratively. The parameter $\boldsymbol{\theta}_k$ at iteration k is used to find new parameters $\boldsymbol{\theta}_{k'}$ at the next iteration k'. The threshold γ_k becomes smaller that its initial value, thus artificially making events *less rare* under $\mathbf{X} \sim g(\mathbf{x};\boldsymbol{\theta}_k)$.

In practice, the CE-method algorithm requires the user to specify a number of *elite* samples m_{elite} which are used when fitting the new parameters for iteration k'. Conveniently, if our distribution g belongs to the *natural exponential family* then the optimal parameters can be found analytically [32]. For a multivariate Gaussian distribution parameterized by $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, the optimal parameters

for the next iteration k' correspond to the maximum likelihood estimate (MLE):

$$\boldsymbol{\mu}_{k'} = \frac{1}{m_{\text{elite}}} \sum_{i=1}^{m_{\text{elite}}} \mathbf{x}_i$$
$$\boldsymbol{\Sigma}_{k'} = \frac{1}{m_{\text{elite}}} \sum_{i=1}^{m_{\text{elite}}} (\mathbf{x}_i - \boldsymbol{\mu}_{k'})(\mathbf{x}_i - \boldsymbol{\mu}_{k'})^\top$$

The cross-entropy method algorithm is shown in algorithm 1 [32] For an objective function S and input distribution g, the CE-method algorithm will run for $k_{\max}$ iterations. At each iteration, m inputs are sampled from g and evaluated using the objective function S. The sampled inputs are denoted by $\mathbf{X}$ and the evaluated values are denoted by $\mathbf{Y}$. Next, the top m_{elite} samples are stored in the elite set $\mathbf{e}$, and the distribution g is fit to the elites. This process is repeated for $k_{\max}$ iterations and the resulting parameters $\boldsymbol{\theta}_{k_{\max}}$ are returned. Note that a variety of input distributions for g are supported, but we focus on the multivariate Gaussian distribution and the Gaussian mixture model in this work.

Algorithm 1 Cross-entropy method.

function CROSSENTROPYMETHOD($S, g, m, m_{\text{elite}}, k_{\max}$)
  for $k \in [1, \ldots, k_{\max}]$
    $\mathbf{X} \sim g(\,\cdot\,; \boldsymbol{\theta}_k)$ where $\mathbf{X} \in \mathbb{R}^{|g| \times m}$ ▷ draw m samples from g
    $\mathbf{Y} \leftarrow S(\mathbf{x})$ for $\mathbf{x} \in \mathbf{X}$ ▷ evaluate samples $\mathbf{X}$ using objective S
    $\mathbf{e} \leftarrow$ store top m_{elite} from $\mathbf{Y}$ ▷ select elite samples output from objective
    $\boldsymbol{\theta}_{k'} \leftarrow \text{FIT}(g(\,\cdot\,; \boldsymbol{\theta}_k), \mathbf{e})$ ▷ re-fit distribution g using elite samples
  return $g(\,\cdot\,; \boldsymbol{\theta}_{k_{\max}})$

2.2.3 Mixture Models

A standard Gaussian distribution is *unimodal* and can have trouble generalizing over data that is *multimodal*. A *mixture model* is a weighted mixture of unimodal component distributions used to represent continuous multimodal distributions [31]. Formally, a Gaussian mixture model (GMM) is defined by its parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ and associated weights $\mathbf{w}$ where $\sum_{i=1}^{n} w_i = 1$. We denote that a random variable $\mathbf{X}$ is distributed according to a mixture model as $\mathbf{X} \sim \text{Mixture}(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{w})$. The probability density of the GMM then becomes:

$$P(\mathbf{X} = \mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{w}) = \sum_{i=1}^{n} w_i \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

To fit the parameters of a Gaussian mixture model, it is well known that the *expectation-maximization* (EM) algorithm can be used [18, 5]. The EM algorithm seeks to find the maximum likelihood estimate of a dependent latent variable using observed data. Intuitively, the algorithm

alternates between an expectation step (E-step) and a maximization step (M-step) to guarantee convergence to a local minima. We refer to [18, 5] for further reading on the EM algorithm.

2.2.4 Surrogate Models

In the context of optimization, a surrogate model $\hat{S}$ is used to estimate the true objective function and provide less expensive evaluations. Surrogate models are a popular approach and have been used to evaluate rare-event probabilities in computationally expensive systems [45, 44]. The simplest example of a surrogate model is linear regression. In this work, we focus on the *Gaussian process* surrogate model. A Gaussian process (GP) is a distribution over functions that predicts the underlying objective function S and captures the uncertainty of the prediction using a probability distribution [32]. This means a GP can be sampled to generate random functions, which can then be fit to our given data $\mathbf{X}$. A Gaussian process is parameterized by a mean function $\mathbf{m}(\mathbf{X})$ and kernel function $\mathbf{K}(\mathbf{X}, \mathbf{X})$, which captures the relationship between data points as covariance values. We denote a Gaussian process that produces estimates $\hat{\mathbf{y}}$ as:

$$\hat{\mathbf{y}} \sim \mathcal{N}\left(\mathbf{m}(\mathbf{X}), \mathbf{K}(\mathbf{X}, \mathbf{X})\right)$$

where

$$\mathbf{m}(\mathbf{X}) = \Big[m(\mathbf{x}_1), \ldots, m(\mathbf{x}_n)\Big]$$

$$\mathbf{K}(\mathbf{X}, \mathbf{X}) = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}$$

Note that we use the zero-mean function $m(\mathbf{x}_i) = \mathbf{0}$, which is generally conventional. For the kernel function $k(\mathbf{x}_i, \mathbf{x}_i)$, we use the squared exponential kernel with variance σ^2 and characteristic scale-length ℓ, where larger ℓ values increase the correlation between successive data points, thus smoothing out the generated functions. The squared exponential kernel is defined as:

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{(\mathbf{x} - \mathbf{x}')^\top (\mathbf{x} - \mathbf{x}')}{2\ell^2}\right)$$

We refer to [32] for a detailed overview of Gaussian processes and different kernel functions.

2.3 Algorithms

We can now describe the cross-entropy method variants introduced in this work.[1] This section will first cover the main algorithm introduced, the *cross-entropy surrogate method* (CE-surrogate). Then we introduce a modification to the CE-surrogate method, namely the *cross-entropy mixture method* (CE-mixture). Lastly, we describe various evaluation schedules for redistributing objective function calls over the iterations.

2.3.1 Cross-Entropy Surrogate Method

The main CE-method variant we introduce is the *cross-entropy surrogate method* (CE-surrogate). The CE-surrogate method is a superset of the CE-method, where the differences lie in the evaluation scheduling and modeling of the elite set using a surrogate model. The goal of the CE-surrogate algorithm is to address the shortcomings of the CE-method when the number of objective function calls is sparse and the underlying objective function S has multiple local minima.

The CE-surrogate algorithm is shown in algorithm 2. It takes as input the objective function S, the distribution $\mathbf{M}$ parameterized by $\boldsymbol{\theta}$, the number of samples m, the number of elite samples m_{elite}, and the maximum iterations $k_{\max}$. For each iteration k, the number of samples m are redistributed through a call to EVALUATIONSCHEDULE, where m controls the number of true objective function evaluations of S. Then, the algorithm samples from $\mathbf{M}$ parameterized by the current $\boldsymbol{\theta}_k$ given the adjusted number of samples m. For each sample in $\mathbf{X}$, the objective function S is evaluated and the results are then stored in $\mathbf{Y}$. The top m_{elite} evaluations from $\mathbf{Y}$ are stored in $\mathbf{e}$. Using all of the current function evaluations $\mathbf{Y}$ from sampled inputs $\mathbf{X}$, a modeled elite set $\mathbf{E}$ is created to augment the sparse information provided by a low number of true objective function evaluations. Finally, the distribution $\mathbf{M}$ is fit to the elite set $\mathbf{E}$ and the distribution with the final parameters $\boldsymbol{\theta}_{k_{\max}}$ is returned.

Algorithm 2 Cross-entropy surrogate method.

function CE-SURROGATE(S, $\mathbf{M}$, m, m_{elite}, $k_{\max}$)
  for $k \in [1, \ldots, k_{\max}]$
    $m, m_{\text{elite}} \leftarrow$ EVALUATIONSCHEDULE$(k, k_{\max})$ ▷ number of evaluations from a schedule
    $\mathbf{X} \sim \mathbf{M}(\,\cdot\,; \boldsymbol{\theta}_k)$ where $\mathbf{X} \in \mathbb{R}^{|\mathbf{M}| \times m}$ ▷ draw m samples from $\mathbf{M}$
    $\mathbf{Y} \leftarrow S(\mathbf{x})$ for $\mathbf{x} \in \mathbf{X}$ ▷ evaluate samples $\mathbf{X}$ using true objective S
    $\mathbf{e} \leftarrow$ store top m_{elite} from $\mathbf{Y}$ ▷ select elite samples output from true objective
    $\mathbf{E} \leftarrow$ MODELELITESET$(\mathbf{X}, \mathbf{Y}, \mathbf{M}, \mathbf{e}, m, m_{\text{elite}})$ ▷ find model-elites using a surrogate model
    $\boldsymbol{\theta}_{k'} \leftarrow$ FIT$(\mathbf{M}(\,\cdot\,; \boldsymbol{\theta}_k), \mathbf{E})$ ▷ re-fit distribution $\mathbf{M}$ using model-elite samples
  return $\mathbf{M}(\,\cdot\,; \boldsymbol{\theta}_{k_{\max}})$

[1] Code available at https://github.com/mossr/CrossEntropyVariants.jl

The main difference between the standard CE-method and the CE-surrogate variant lies in the call to algorithm 3, ModelEliteSet. The motivation is to use *all* of the already evaluated objective function values $\mathbf{Y}$ from a set of sampled inputs $\mathbf{X}$. This way, the expensive function evaluations—otherwise discarded—can be used to build a surrogate model of the underlying objective function. First, a surrogate model $\hat{S}$ is constructed from the samples $\mathbf{X}$ and true objective function values $\mathbf{Y}$. We used a Gaussian process with a specified kernel and optimizer, but other surrogate modeling techniques such as regression with basis functions can be used. We chose a Gaussian process because it incorporates probabilistic uncertainty in the predictions, which may more accurately represent our objective function, or at least be sensitive to over-fitting to sparse data. Now we have an approximated objective function $\hat{S}$ that we can inexpensively call. We sample $10m$ values from the distribution $\mathbf{M}$ and evaluate them using the surrogate model. We then store the top $10m_{\text{elite}}$ values from the estimates $\widehat{\mathbf{Y}}_{\text{m}}$. We call these estimated elite values $\mathbf{e}_{\text{model}}$ the *model-elites*. The surrogate model is then passed to SubEliteSet, which returns more estimates for elite values. Finally, the elite set $\mathbf{E}$ is built from the true-elites $\mathbf{e}$, the model-elites $\mathbf{e}_{\text{model}}$, and the subcomponent-elites $\mathbf{e}_{\text{sub}}$. The resulting concatenated elite set $\mathbf{E}$ is returned.

Algorithm 3 Modeling elite set using a surrogate objective.

function ModelEliteSet($\mathbf{X}, \mathbf{Y}, \mathbf{M}, \mathbf{e}, m, m_{\text{elite}}$)
- $\hat{S} \leftarrow$ GaussianProcess($\mathbf{X}, \mathbf{Y}$, kernel, optimizer) ▷ fit a surrogate model to the samples
- $\mathbf{X}_{\text{m}} \sim \mathbf{M}(\,\cdot\,; \boldsymbol{\theta}_k)$ where $\mathbf{X}_{\text{m}} \in \mathbb{R}^{|\mathbf{M}| \times 10m}$ ▷ draw $10m$ samples from $\mathbf{M}$
- $\widehat{\mathbf{Y}}_{\text{m}} \leftarrow \hat{S}(\mathbf{x}_{\text{m}})$ for $\mathbf{x}_{\text{m}} \in \mathbf{X}_{\text{m}}$ ▷ evaluate samples $\mathbf{X}_{\text{m}}$ using surrogate objective $\hat{S}$
- $\mathbf{e}_{\text{model}} \leftarrow$ store top $10m_{\text{elite}}$ from $\widehat{\mathbf{Y}}_{\text{m}}$ ▷ select model-elite samples from surrogate objective
- $\mathbf{e}_{\text{sub}} \leftarrow$ SubEliteSet($\hat{S}, \mathbf{M}, \mathbf{e}$) ▷ generate sub-elite samples using surrogate $\hat{S}$
- $\mathbf{E} \leftarrow \{\mathbf{e}\} \cup \{\mathbf{e}_{\text{model}}\} \cup \{\mathbf{e}_{\text{sub}}\}$ ▷ combine all elite samples into an elite set
- **return E**

To encourage exploration of promising areas of the design space, the algorithm SubEliteSet focuses on the already marked true-elites $\mathbf{e}$. Each elite $e_x \in \mathbf{e}$ is used as the mean of a new multivariate Gaussian distribution with covariance inherited from the distribution $\mathbf{M}$. The collection of subcomponent distributions is stored in $\mathbf{m}$. The idea is to use the information given to us by the true-elites to emphasize areas of the design space that look promising. For each distribution $\mathbf{m}_i \in \mathbf{m}$, we run a subroutine call to the standard CE-method to fit the distribution $\mathbf{m}_i$ using the surrogate model $\hat{S}$. Then the best objective function value is added to the subcomponent-elite set $\mathbf{e}_{\text{sub}}$, and after iterating the full set is returned. Note that we use θ_{CE} to denote the parameters for the CE-method algorithm. In our case, we recommend using a small $k_{\max}$ of around 2 so the subcomponent-elites do not over-fit to the surrogate model but have enough CE-method iterations to tend towards optimal.

Algorithm 4 Subcomponent elite set.

function SUBELITESET($\hat{S}, \mathbf{M}, \mathbf{e}$)
  $\mathbf{e}_{\text{sub}} \leftarrow \emptyset$
  $\mathbf{m} \leftarrow \{e_x \in \mathbf{e} \mid \mathcal{N}(e_x, \mathbf{M}.\Sigma)\}$ ▷ create set of distributions centered at each true-elite sample
  for $\mathbf{m}_i \in \mathbf{m}$
    $\mathbf{m}_i \leftarrow$ CROSSENTROPYMETHOD($\hat{S}, \mathbf{m}_i; \theta_{\text{CE}}$) ▷ run CE-method over each new distribution
    $\mathbf{e}_{\text{sub}} \leftarrow \{\mathbf{e}_{\text{sub}}\} \cup \{\text{BEST}(\mathbf{m}_i)\}$ ▷ append best result into the sub-elite set
  return $\mathbf{e}_{\text{sub}}$

2.3.2 Cross-Entropy Mixture Method

We refer to the variant of our CE-surrogate method that takes an input mixture model $\mathbf{M}$ as the *cross-entropy mixture method* (CE-mixture). The CE-mixture algorithm is identical to the CE-surrogate algorithm, but calls a custom FIT function to fit a mixture model to the elite set $\mathbf{E}$. The input distribution $\mathbf{M}$ is cast to a mixture model using the subcomponent distributions $\mathbf{m}$ as the components of the mixture. We use a default uniform weighting for each mixture component. The mixture model $\mathbf{M}$ is then fit using the expectation-maximization algorithm, and the resulting distribution is returned. The idea is to use the distributions in $\mathbf{m}$ that are centered around each true-elite as the components of the casted mixture model. Therefore, we would expect better performance of the CE-mixture method when the objective function has many competing local minima. Results in section 2.4.3 aim to show this behavior.

Algorithm 5 Fitting mixture models (used by CE-mixture).

function FIT($\mathbf{M}, \mathbf{m}, \mathbf{E}$)
  $\mathbf{M} \leftarrow \text{Mixture}(\mathbf{m})$
  $\widehat{\boldsymbol{\theta}} \leftarrow$ EXPECTATIONMAXIMIZATION($\mathbf{M}, \mathbf{E}$)
  return $\mathbf{M}(\,\cdot\,; \widehat{\boldsymbol{\theta}})$

2.3.3 Evaluation Scheduling

Given the nature of the CE-method, we expect the covariance to shrink over time, thus resulting in a solution with higher confidence. Yet if each iteration is given the same number of objective function evaluations m, there is the potential for elite samples from early iterations dominating the convergence. Therefore, we would like to redistribute the objective function evaluations throughout the iterations to use more truth information early in the process. We call these heuristics *evaluation schedules*. One way to achieve this is to reallocate the evaluations according to a Geometric distribution. Evaluation schedules can also be ad-hoc and manually prescribed based on the current iteration.

We provide the evaluation schedule we use that follows a Geometric distribution with parameter p in algorithm 6. We denote $G \sim \text{Geo}(p)$ to be a random variable that follows a truncated Geometric

distribution with the probability mass function $p_G(k) = p(1-p)^k$ for $k \in \{0, 1, 2, \ldots, k_{\max}\}$. Note the use of the integer rounding function (e.g., $\lfloor x \rceil$), which we later have to compensate for the final iterations. Results in section 2.4.3 compare values of p that control the redistribution of evaluations.

Algorithm 6 Evaluation schedule using a Geometric distribution.

function EVALUATIONSCHEDULE($k, k_{\max}$)
  $G \sim \text{Geo}(p)$
  $N_{\max} \leftarrow k_{\max} \cdot m$
  $m \leftarrow \lfloor N_{\max} \cdot p_G(k) \rceil$
  if $k = k_{\max}$
    $s \leftarrow \sum_{i=1}^{k_{\max}-1} \lfloor N_{\max} \cdot p_G(i) \rceil$
    $m \leftarrow \min(N_{\max} - s, N_{\max} - m)$
  $m_{\text{elite}} \leftarrow \min(m_{\text{elite}}, m)$
  return (m, m_{elite})

2.4 Experiments

In this section, we detail the experiments we ran to compare the CE-method variants and evaluation schedules. We first introduce a test objective function we created to stress the issue of converging to local minima. We then describe the experimental setup for each of our experiments and provide an analysis and results.

2.4.1 Test Objective Function Generation

To stress the cross-entropy method and its variants, we created a test objective function called *sierra* that is generated from a mixture model comprised of 49 multivariate Gaussian distributions. We chose this construction so that we can use the negative peeks of the component distributions as local minima and can force a global minimum centered at our desired $\widetilde{\boldsymbol{\mu}}$. The construction of the sierra test function can be controlled by parameters that define the spread of the local minima. We first start with the center defined by a mean vector $\widetilde{\boldsymbol{\mu}}$ and we use a common covariance $\widetilde{\boldsymbol{\Sigma}}$:

$$\widetilde{\boldsymbol{\mu}} = [\mu_1, \mu_2], \quad \widetilde{\boldsymbol{\Sigma}} = \begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix}$$

Next, we use the parameter δ to control the clustered distance between symmetric points:

$$\mathbf{G} = \big\{[+\delta, +\delta], [+\delta, -\delta], [-\delta, +\delta], [-\delta, -\delta]\big\}$$

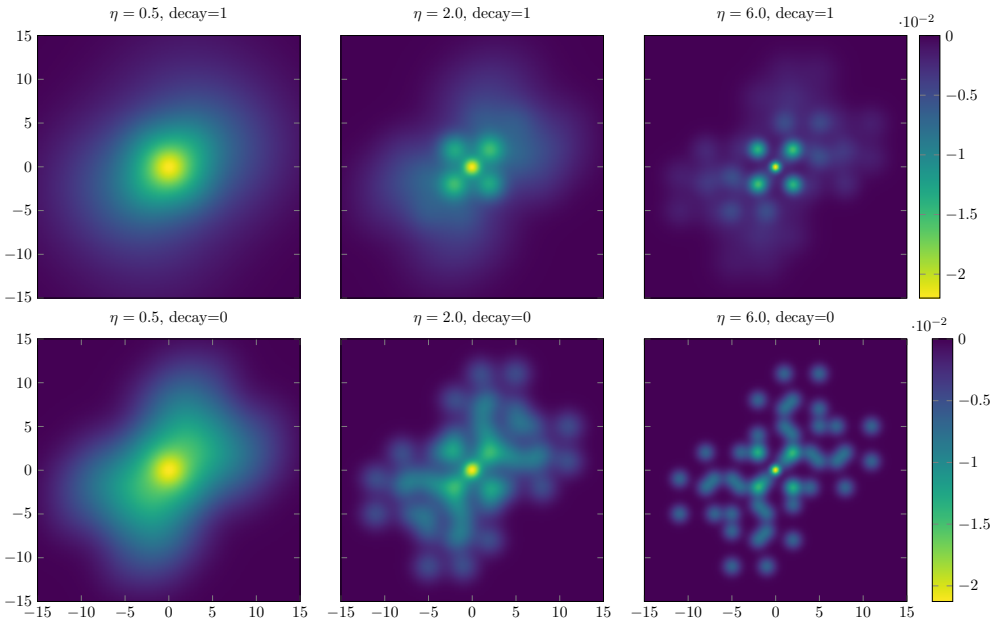


Figure 2.1: Example test objective functions generated using the sierra function.

We chose points $\mathbf{P}$ to fan out the clustered minima relative to the center defined by $\widetilde{\boldsymbol{\mu}}$:

$$\mathbf{P} = \big\{[0,0],[1,1],[2,0],[3,1],[0,2],[1,3]\big\}$$

The vector $\mathbf{s}$ is used to control the $\pm$ distance to create an 's' shape comprised of minima, using the standard deviation σ: $\mathbf{s} = [+\sigma, -\sigma]$. We set the following default parameters: standard deviation $\sigma = 3$, spread rate $\eta = 6$, and cluster distance $\delta = 2$. We can also control if the local minima clusters "decay", thus making those local minima less distinct (where decay $\in \{0, 1\}$). The parameters that define the sierra function are collected into $\boldsymbol{\theta} = \langle \widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\Sigma}}, \mathbf{G}, \mathbf{P}, \mathbf{s} \rangle$. Using these parameters, we can define the mixture model used by the sierra function as:

$$\mathbf{M}_{\mathcal{S}} \sim \text{Mixture}\left(\left\{\boldsymbol{\theta} \;\middle|\; \mathcal{N}\left(\mathbf{g} + s\mathbf{p}_i + \widetilde{\boldsymbol{\mu}},\ \widetilde{\boldsymbol{\Sigma}} \cdot i^{\text{decay}}/\eta\right)\right\}\right) \quad \text{for} \quad (\mathbf{g}, \mathbf{p}_i, s) \in (\mathbf{G}, \mathbf{P}, \mathbf{s})$$

We add a final component to be our global minimum centered at $\widetilde{\boldsymbol{\mu}}$ and with a covariance scaled by $\sigma\eta$. Namely, the global minimum is $\mathbf{x}^* = \mathbb{E}\left[\mathcal{N}(\widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\Sigma}}/(\sigma\eta))\right] = \widetilde{\boldsymbol{\mu}}$. We can now use this constant mixture model with 49 components and define the sierra objective function $\mathcal{S}(\mathbf{x})$ to be the negative probability density of the mixture at input $\mathbf{x}$ with uniform weights:

$$\mathcal{S}(\mathbf{x}) = -P(\mathbf{M}_{\mathcal{S}} = \mathbf{x}) = -\frac{1}{|\mathbf{M}_{\mathcal{S}}|}\sum_{j=1}^{n}\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

An example of six different objective functions generated using the sierra function are shown in fig. 2.1, sweeping over the spread rate η, with and without decay.

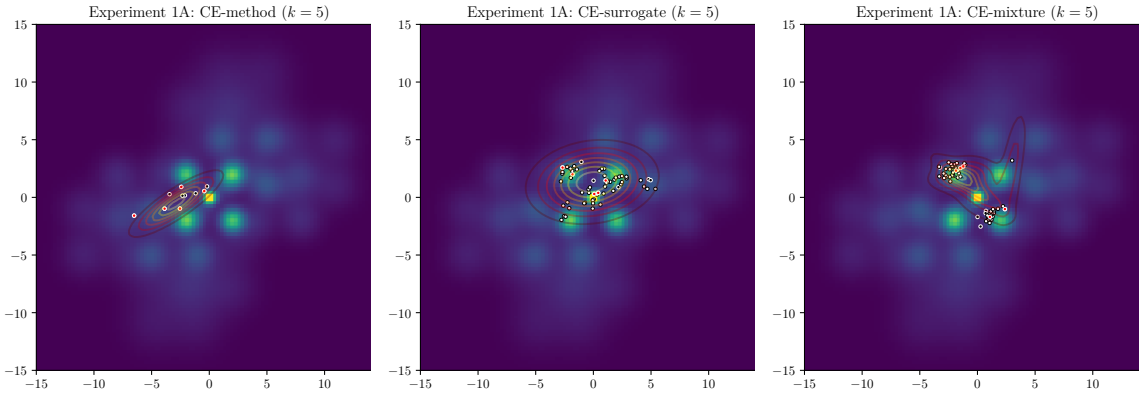
2.4.2 Experimental Setup

Experiments were run to stress a variety of behaviors of each CE-method variant. The experiments are split into two categories: algorithmic and scheduling. The algorithmic category aims to compare features of each CE-method variant while holding common parameters constant (for a better comparison). While the scheduling category experiments with evaluation scheduling heuristics.

Because the algorithms are stochastic, we run each experiment with 50 different random number generator seed values. To evaluate the performance of the algorithms in their respective experiments, we define three metrics. First, we define the average "optimal" value $\bar{b}_v$ to be the average of the best so-far objective function value (termed "optimal" in the context of each algorithm). Again, we emphasize that we average over the 50 seed values to gather meaningful statistics. Another metric we monitor is the average distance to the true global optimal $\bar{b}_d = \|\mathbf{b}_\mathbf{x} - \mathbf{x}^*\|$, where $\mathbf{b}_\mathbf{x}$ denotes the $\mathbf{x}$-value associated with the "optimal". We make the distinction between these metrics to show both "closeness" in *value* to the global minimum and "closeness" in the *design space* to the global minimum. Our final metric looks at the average runtime of each algorithm, noting that our goal is to off-load computationally expensive objective function calls to the surrogate model.

For all of the experiments, we use a common setting of the following parameters for the sierra test function (shown in the top-right plot in fig. 2.1):

$$(\widetilde{\boldsymbol{\mu}} = [0, 0],\ \sigma = 3,\ \delta = 2,\ \eta = 6,\ \text{decay} = 1)$$



(a) The cross-entropy method. (b) The cross-entropy surrogate method. (c) The cross-entropy mixture method.

Figure 2.2: Iteration $k = 5$ illustrated for each algorithm. The covariance is shown by the contours.

Algorithmic Experiments

We run three separate algorithmic experiments, each to test a specific feature. For our first algorithmic experiment (1A), we want to test each algorithm when the user-defined mean is centered at the global minimum and the covariance is arbitrarily wide enough to cover the design space. Figure 2.2 illustrates experiment (1A) for each algorithm. Let $\mathbf{M}$ be a distribution parameterized by $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and for experiment (1A) we set the following:

$$\boldsymbol{\mu}^{(1A)} = [0, 0] \qquad \boldsymbol{\Sigma}^{(1A)} = \begin{bmatrix} 200 & 0 \\ 0 & 200 \end{bmatrix}$$

For our second algorithmic experiment (1B), we test a mean that is far off-centered with a wider covariance:

$$\boldsymbol{\mu}^{(1B)} = [-50, -50] \qquad \boldsymbol{\Sigma}^{(1B)} = \begin{bmatrix} 2000 & 0 \\ 0 & 2000 \end{bmatrix}$$

This experiment is used to test the "exploration" of the CE-method variants introduced in this work. In experiments (1A) and (1B), we set the following common parameters across each CE-method variant:

$$(k_{\max} = 10,\ m = 10,\ m_{\text{elite}} = 5)^{(1A,1B)}$$

This results in $m \cdot k_{\max} = 100$ objective function evaluations, which we define to be *relatively* low.

For our third algorithmic experiment (1C), we want to test how each variant responds to an extremely low number of function evaluations. This sparse experiment sets the common CE-method parameters to:

$$(k_{\max} = 10,\ m = 5,\ m_{\text{elite}} = 3)^{(1C)}$$

This results in $m \cdot k_{\max} = 50$ objective function evaluations, which we defined to be *extremely* low. We use the same mean and covariance defined for experiment (1A):

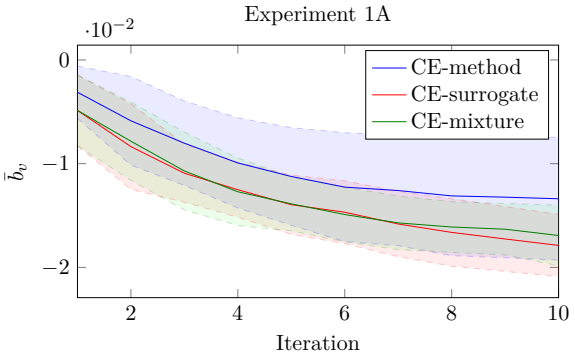
$$\boldsymbol{\mu}^{(1C)} = [0, 0] \qquad \boldsymbol{\Sigma}^{(1C)} = \begin{bmatrix} 200 & 0 \\ 0 & 200 \end{bmatrix}$$

Scheduling Experiments

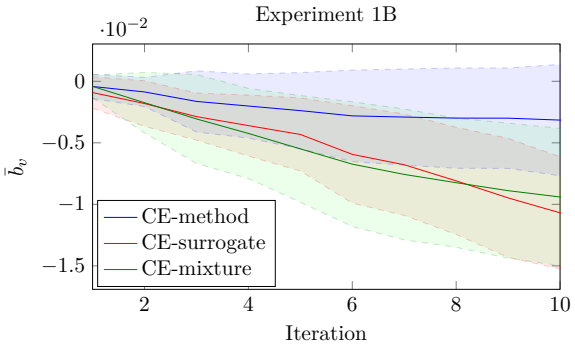
In our final experiment (2), we test the evaluation scheduling heuristics which are based on the Geometric distribution. We sweep over the parameter p that determines the Geometric distribution which controls the redistribution of objective function evaluations. In this experiment, we compare the CE-surrogate methods using the same setup as experiment (1B), namely the far off-centered mean. We chose this setup to analyze exploration schemes when given very little information about the true objective function.

2.4.3 Results and Analysis

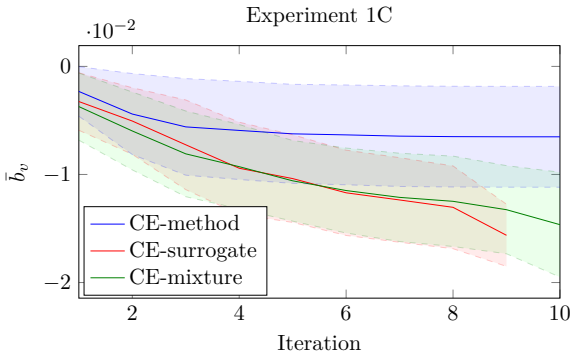
Figure 2.3a shows the average value of the current optimal $\bar{b}_v$ for the three algorithms for experiment (1A). One standard deviation is plotted in the shaded region. Notice that the standard CE-method converges to a local minima before $k_{\max}$ is reached. Both CE-surrogate method and CE-mixture stay below the standard CE-method curve, highlighting the mitigation of convergence to local minima. Minor differences can be seen between CE-surrogate and CE-mixture, differing slightly towards the tail in favor of CE-surrogate. The average runtime of the algorithms along with the performance metrics are shown together for each experiment in table 2.1.



(a) Average optimal value for experiment (1A) when the initial mean is centered at the global minimum and the covariance sufficiently covers the design space.



(b) Average optimal value for experiment (1B) when the initial mean is far from the global minimum with a wide covariance.



(c) Average optimal value for experiment (1C) when we restrict the number of objective function calls.

Figure 2.3: Cross-entropy method variant experiment results.

An apparent benefit of the standard CE-method is in its simplicity and speed. As shown in table 2.1, the CE-method is the fastest approach by about 2-3 orders of magnitude compared to CE-surrogate and CE-mixture. The CE-mixture method is notably the slowest approach. Although the runtime is also based on the objective function being tested, recall that we are using the same number of true objective function calls in each algorithm, and the metrics we are concerned with in optimization are to minimize $\bar{b}_v$ and $\bar{b}_d$. We can see that the CE-surrogate method consistently out

performs the other methods. Surprisingly, a uniform evaluation schedule performs the best even in the sparse scenario where the initial mean is far away from the global optimal.

When the initial mean of the input distribution is placed far away from the global optimal, the CE-method tends to converge prematurely as shown in fig. 2.3b. This scenario is illustrated in fig. 2.4. We can see that both CE-surrogate and CE-mixture perform well in this case.

Given the same centered mean as before, when we restrict the number of objective function calls even further to just 50 we see interesting behavior. Notice that the results of experiment (1C) shown in fig. 2.3c follow a curve closer to the far away mean from experiment (1B) than from the same setup as experiment (1A). Also notice that the CE-surrogate results cap out at iteration 9 due to the evaluation schedule front-loading the objective function calls, thus leaving none for the final iteration (while still maintaining the same total number of evaluations of 50).

Table 2.1: Experimental Results.

Exper.	**Algorithm**	**Runtime**	$\bar{b}_v$	$\bar{b}_d$
1A	CE-method	**0.029** s	−0.0134	23.48
	CE-surrogate	1.47 s	**−0.0179**	**12.23**
	CE-mixture	9.17 s	−0.0169	16.87
1B	CE-method	**0.046** s	−0.0032	138.87
	CE-surrogate	11.82 s	**−0.0156**	**18.24**
	CE-mixture	28.10 s	−0.0146	33.30
1C	CE-method	**0.052** s	−0.0065	43.14
	CE-surrogate	0.474 s	**−0.0156**	**17.23**
	CE-mixture	2.57 s	−0.0146	22.17
2	CE-surrogate, Uniform	—	**−0.0193**	**8.53**
	CE-surrogate, Geo(0.1)	—	−0.0115	25.35
	CE-surrogate, Geo(0.2)	—	−0.0099	27.59
	CE-surrogate, Geo(0.3)	—	−0.0089	30.88
			$-0.0220 \approx \mathbf{x}^*$	

2.5 Discussion

We presented variants of the popular cross-entropy method for optimization of objective functions with multiple local minima. Using a Gaussian process-based surrogate model, we can use the same number of true objective function evaluations and achieve better performance than the standard CE-method on average. We also explored the use of a Gaussian mixture model to help find global minimum in multimodal objective functions. We introduce a parameterized test objective function with a controllable global minimum and spread of local minima. Using this test function, we showed that the CE-surrogate algorithm achieves the best performance relative to the standard CE-method, each using the same number of true objective function evaluations.

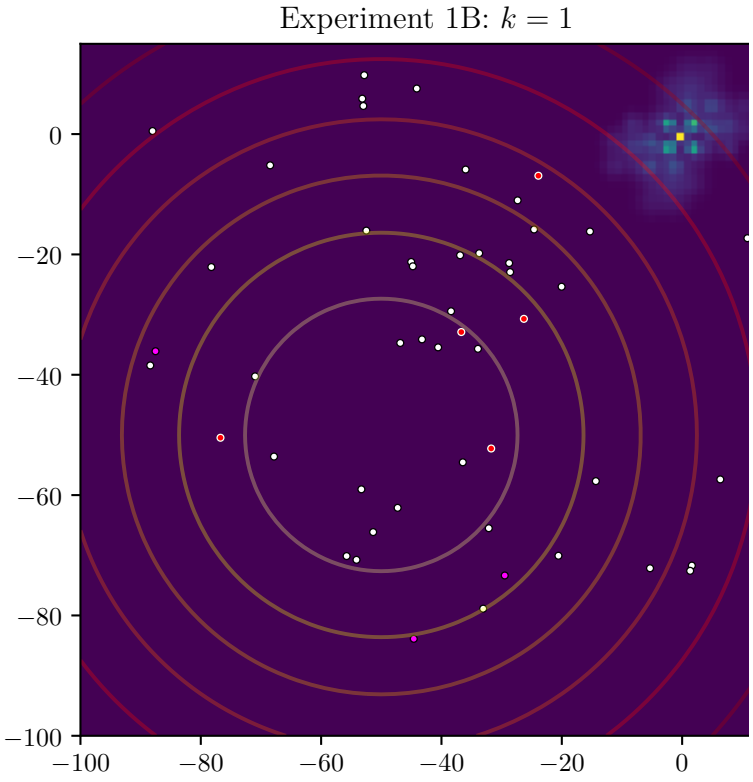


Figure 2.4: First iteration of the scenario in experiment (1B) where the initial distribution is far away form the global optimal. The red dots indicate the true-elites, the magenta dots with black outlines indicate the "non-elites" (i.e., all others) evaluated from the true objective function, and the white dots with black outlines indicate the samples evaluated using the surrogate model. Notice the trend of the white dots (those from the surrogate) that can help guide the proposal distribution towards the area of interest (in the top right).

Chapter 3

Episodic Adaptive Stress Testing

We now consider black-box validation of an open-loop sequential system, namely components of an aircraft flight management system (FMS). A primary function of an FMS is to provide guidance in the form of navigational waypoints between origin and destination airports. A trajectory predictor is the subsystem that provides trajectories to the guidance subsystem that commands the autopilot. Failures within the trajectory prediction system can occur if the output waypoints are unreachable given the physical limitations of the aircraft, or if there are problems with the implementation or design of the software. The goal of this work is to find likely failure cases before system deployment so that the engineers can resolve or address potential problems in the system.

Traditionally, large-scale Monte Carlo testing is used to generate these failure cases [11]. However, Monte Carlo testing can be inadequate for large input spaces with rare failure events [28]. We investigate the use of adaptive stress testing (AST), an advanced black-box stress testing approach that has been successfully applied to find failures in safety-critical systems [40, 34, 43, 39]. Adaptive stress testing is a method that uses reinforcement learning to adversarially search for rare failure events in sequential decision making systems [38]. This work applies the AST approach to trajectory prediction systems to efficiently find failure events and their likelihoods.

The trajectory prediction system is treated as a black-box simulator and AST controls the selection of waypoints and other environmental input parameters. Monte Carlo tree search (MCTS) with progressive widening is used to explore the possible trajectories and a notion of "miss distance" to a failure event is used to help guide the search. Transition probabilities between states are also used to guide the search towards the most likely failures. In traditional AST formulations, a sequential decision making problem is assumed, but our open-looped trajectory predictor does not provide a sequence of decisions, but rather generates trajectories based on complete flight plans. We regard each waypoint in the flight plan as a (hidden) step in the sequence. Although we can forcibly fit the problem to a strict sequential framework by calling the system with partial flight plans, this is prohibitively expensive and unnecessary. Instead, we collect the intermediate states

and actions and only evaluate the system at the end of a simulated rollout, then back-propagate the reward to unevaluated parts of the tree. We can do this due to the structure of our proposed AST reward function. Therefore, this work extends the AST approach to be applied more generally to sequential decision making problems with episodic reward (i.e., rewards accumulated at the end of an episode with intermediate rewards of zero).

We analyze a trajectory predictor from a developmental commercial FMS which takes as input winds aloft, origin and destination airports, and a collection the lateral waypoints. The trajectory predictor outputs the discrete-time controls that determine translational motion which would be input to the FMS. Within the FMS, the trajectories are passed to the guidance subsystem which determines how to command the autopilot. Although we focus on arc length discrepancies as the primary failure mode (described in section 3.5), this work can be easily extended to other failure events (see appendix A). The performance of AST is compared to two baselines: direct Monte Carlo simulation (i.e., random search) and the cross-entropy method. Current failure assessment is performed exhaustively over a navigational database of predefined aircraft routes defined by lateral waypoints. This testing method is used during development while requirement-based testing is used for final system certification following RTCA DO-178C [53]. The intention of testing during development is to find failures otherwise not covered by requirements-based tests. As a comparison of developmental testing approaches, we sample routes from the navigational database as another baseline to relate the simulation-based approach to the standard navigational database approach. Experiments were run to generate likely failure events that are provided to the developers of the trajectory predictor to analyze and resolve potential shortcomings of the system.

3.1 Prior Work

AST has been successfully applied to safety-critical systems such as aircraft collision avoidance systems [40, 43] and autonomous vehicles [34]. Lee et al. [41] applied AST to trajectory planning systems, but as will be discussed further in section 3.3.1, that problem had access to the full FMS, thus fitting the model of traditional AST. They looked at runtime behavior of the FMS on a simulated aircraft, where the disturbances were added as sensor noise. In our work, we do not rely on simulating aircraft dynamics and use input waypoints as the disturbances.

Other work has been proposed to efficiently search for failures in black-box cyber-physical systems [14]. Norden, O'Kelly, and Sinha [51] propose an importance sampling approach to find rare failure events to assess autonomous vehicle safety. Their work is similar to AST but relies on an accurate importance distribution. Other approaches formulate the problem of falsification as an optimization problem and solve it using Bayesian optimization [3], simulated annealing [1, 4], or rapidly-exploring random trees [61]. Each of these approaches are formulated as a classical optimization problem and use different techniques to search the input space for failures—although

there is no likelihood estimation for a given falsifying input. A different approach altogether uses an existing set of falsifying inputs to bootstrap the search for neighboring failures [20]. That work relies on existing counterexamples to base the neighboring search upon. Our work addresses falsification of sequential systems with episodic reward and includes most likely failure analysis.

The remainder of this chapter is organized as follows. Section 3.2 provides necessary background of AST and commercial aircraft FMS. Section 3.3 describes how this work extends existing AST approaches and modifications made to the MCTS algorithm. Section 3.4 details the implementation of this work using the Julia programming language and provides a description of the interface for AST to interact with black-box systems. Section 3.5 describes the simulation environment constructed for the FMS application and the failure events we are searching for. Section 3.6 describes the experiments and discusses the analysis of the results, and section 3.7 provides a discussion on the conclusions from this work.

3.2 Background

This section describes background of the adaptive stress testing problem formulation and flight management systems, particularly the trajectory prediction subsystem.

3.2.1 Adaptive Stress Testing

Adaptive stress testing (AST) is a black-box approach to find rare failure events in cyber-physical systems [40, 42]. The AST problem is formulated as a Markov decision process (MDP) and can be solved using reinforcement learning algorithms to guide the search towards likely failure events. AST can also be formulated more generally as other sequential decision making processes, such as a partially observable Markov decision process (POMDP) [40, 34, 41].

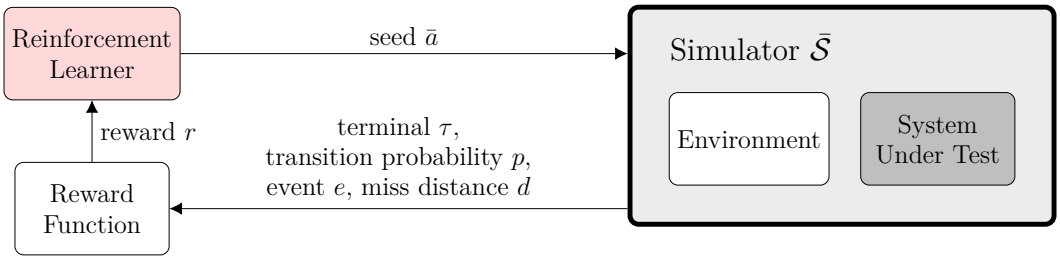


Figure 3.1: Adaptive stress testing formulation.

Figure 3.1 illustrates how the AST concept is formulated. The system under test (SUT) is treated as a black box while the simulator $\bar{\mathcal{S}}$ is treated as a gray box that passes the transition probability p, event indicator e, miss distance d, and termination state indicator τ to the reward function. The AST problem is solved using an adversarial reinforcement learner that selects a random number generator seed $\bar{a}$ to indirectly control the SUT through the simulator. In other

types of simulators, AST could directly control input disturbances rather than seeds. In prior work, MCTS and deep reinforcement learning have been used to solve the MDP [40, 34]. The output of the AST process is a set of state trajectories deterministically controlled by a set of seeds. These seeds are used to deterministically playback the simulation starting from an initial state.

A necessary clarification is differentiating what is black-box with respect to the AST problem. Given that the simulator must provide $\langle p, e, d, \tau \rangle$ to the AST reward function, the output needs access to the transition probabilities p and terminal indication τ from the environment, but can determine the event indication e and miss distance d from the output of the SUT. Thus, depending on the problem, the environment may be required to be white-box but the SUT is strictly black-box.

The standard AST reward function is designed to guide the search towards failure events and to maximize the likelihoods of those events. It is also affected by the notion of miss distance d: a measure of "closeness" to a particular event. The miss distance helps guide the search towards failures to search efficiently. The standard reward function is given by:

$$R(p, e, d, \tau) = \begin{cases} R_E & \text{if } \tau \wedge e \\ -d & \text{if } \tau \wedge \neg e \\ \log(p) & \text{otherwise} \end{cases} \tag{3.1}$$

The non-negative constant reward for finding an event is given by R_E and is generally set to 0. The boolean e indicates when an event has been found and the boolean τ indicates that the simulation is in a terminal state. If the simulation terminates without finding an event, then the negative miss distance $-d$ is awarded to guide the search towards failure. Otherwise, the log-likelihood of each state transition is used, denoted by $\log(p)$. This term is used to maximize the likelihood of the overall trajectory and is designed to guide the search towards likely failures. Recall that the goal of reinforcement learning is to maximize the expected sum of rewards [59]. Using log-likelihood means we can maximize the summations, which is equivalent to maximizing the product of the likelihoods.

3.2.2 Flight Management Systems

Aircraft flight management systems (FMS) have been a critical part in reducing workload of pilots in commercial aircraft by contributing to in-flight automation [52]. Major components of FMS include flight planning, navigation, guidance, performance optimization, and trajectory prediction [46]. This work focuses on the subsystem of the FMS that generates the aircraft trajectory given a flight plan. This subsystem, called the trajectory predictor, provides deterministic trajectories based on an operator-defined input flight plan and estimates of environmental conditions. Inputs include winds aloft, origin and destination airports, aircraft weight, cost index (a balance between

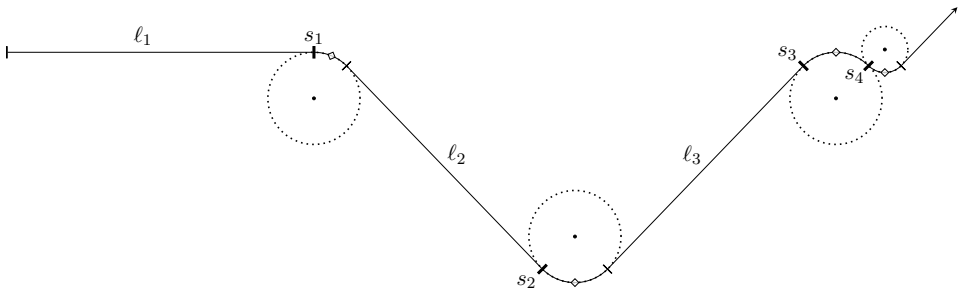


Figure 3.2: Lateral packets output by the trajectory predictor. Lateral packets consist of latitude and longitude points that describe straight line segments ℓ_i and turning arc segments starting at s_i. Straight segments are optional which can result in multiple turn segments sequenced together, as seen at s_3 and s_4.

cost of fuel and time of arrival), and a set of navigational waypoints that define the lateral route. For our purposes, we focus on the winds aloft, origin and destination airports, and the placement of the lateral waypoints. The trajectory predictor outputs the discrete-time controls that determine translational motion (i.e., both vertical and horizontal motion) which become input to the guidance subsystem of the FMS. The trajectories may be processed before being passed to the guidance in the cases where a change in the lateral or vertical trajectory needs to be anticipated and controlled towards. Therefore, the trajectory predictor is not a strictly sequential decision making problem because the full sequence of lateral paths are deterministically constructed based solely on the inputs. The outputs of the trajectory predictor are illustrated in fig. 3.2.

3.3 Approach

Several modifications were made to adapt AST for open-loop sequential systems with episodic reward. Modifications to the standard reward function were made to guide the search towards severe failures as well as likely failures. Modifications to the Monte Carlo tree search algorithm are also described; adapting the search algorithm for efficient SUT evaluations and to use progressive widening with a single deterministic next state.

3.3.1 Adaptive Stress Testing for Episodic Reward Problems

Traditionally, AST is used to steer sequential decision making systems towards likely failures by controlling the seed used to sample environmental variables within the simulator at each time step (as seen in fig. 3.1). The issue with applying this formulation directly to the trajectory predictor is that this system does not rely on sequential feedback to generate the trajectory and solely relies on its set of inputs. Therefore, we propose a modification to the AST formulation to abstract the sequential nature of the problem to the simulation environment. In other words, we collect the state

transitions during the search and evaluate the system at the end of the rollout. This distinction can be seen in fig. 3.3. The transition probability p is output from the environment and the event indication e, miss distance d, and terminal state indication τ are output from the black-box SUT, i.e., the trajectory predictor. The 4-tuple $\langle p, e, d, \tau \rangle$ is passed as input to the reward function and the transition probability p and miss distance d are used to guide the search.

The standard AST reward function described in eq. (3.1) was modified to collect all rewards at the termination state and to incorporate a severity measurement when a failure event occurs. Both the log-likelihood and miss distance d are used throughout the search. A multiplicative bonus R_E is applied when a failure event occurs and we set $R_E = 100$ for our experiments. The modified reward function becomes:

$$R(p, e, d, \tau) = \begin{cases} (\log(p) - d)R_E & \text{if } \tau \wedge e \\ \log(p) - d & \text{if } \tau \wedge \neg e \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

The transition probability p is given by the probability of sampling a set of waypoints from a multivariate Gaussian distribution $\mathbf{w} \overset{\bar{a}}{\sim} \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ composed of waypoint direction and distance (relative to the previous waypoint) and wind direction and magnitude with mean vector $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, deterministically controlled by the seed $\bar{a}$. If an event is found, then the reward is the negative miss distance combined with the log-likelihood and adjusted by the multiplicative bonus R_E. This modification is used to incorporate severity of an event gauged by the miss distance when an event is found. If the simulation terminates without finding an event, no multiplicative bonus is applied. This reward function may not be suitable for certain AST problem formulations, but in section 3.5 we discuss how these modifications are applicable for the failure event and miss distance we investigate.

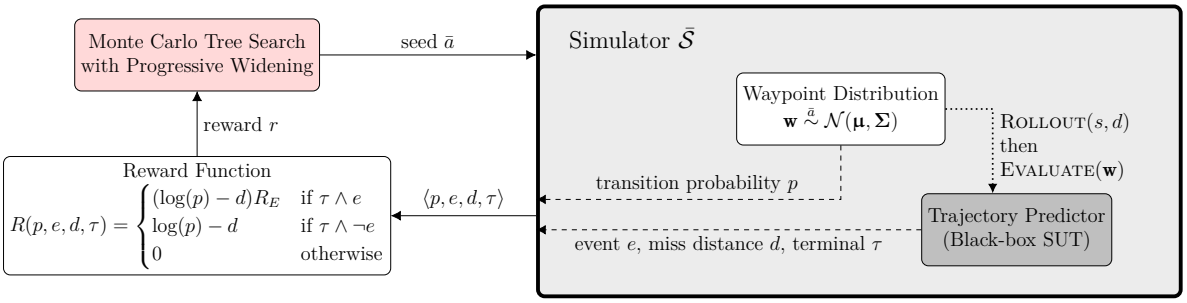


Figure 3.3: Modified adaptive stress testing formulation for the trajectory predictor with episodic reward. The simulation environment samples waypoints from a distribution and passes those waypoints as input to the SUT at the end of the rollout. The modified reward function is guided by both the severity and likelihood of the failure event. Information on the dashed lines is only provided to the reward function when the SUT is evaluated.

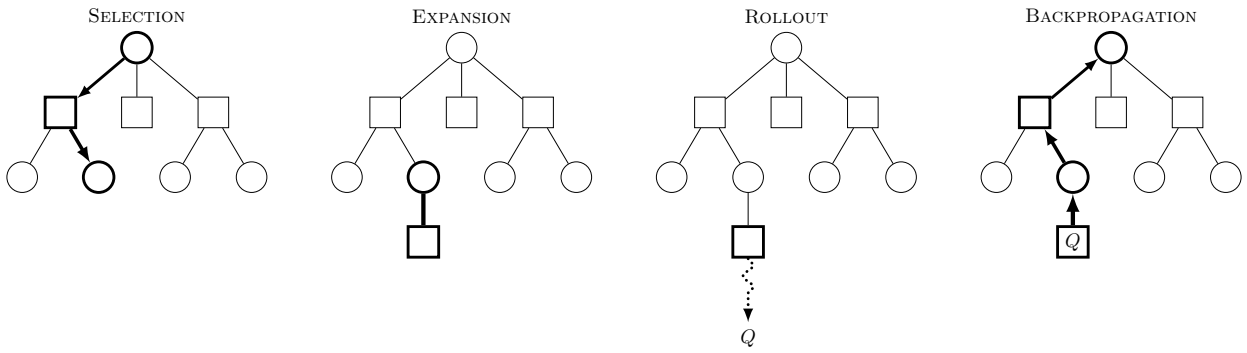


Figure 3.4: The four steps of the Monte Carlo tree search algorithm.

3.3.2 Modified Monte Carlo Tree Search

Monte Carlo tree search (MCTS) is an anytime algorithm that uses rollouts of a random policy to estimate the value of each state-action node in the tree [16, 31]. MCTS has found success in recent years in the reinforcement learning field, notably playing games such as Go [58]. There are four main stages in each simulation: *selection*, *expansion*, *rollout* (or *simulation*), and *backpropagation*. Figure 3.4 illustrates these four steps. The algorithm is "anytime" because a policy can be constructed after any single iteration, but the state-action value estimates become increasingly accurate as more simulations are performed and the tree depth is expanded. The tree $\mathcal{T}$ is iteratively expanded and the policy improves over time as the algorithm balances exploration with exploitation of the state and action spaces.

A commonly used extension of MCTS for large or continuous spaces is progressive widening (PW) [13, 15, 26, 8]. We apply PW on the action space of seeds because there are an infinite number of seeds. There is no need to apply PW to the state transitions, since a seed uniquely

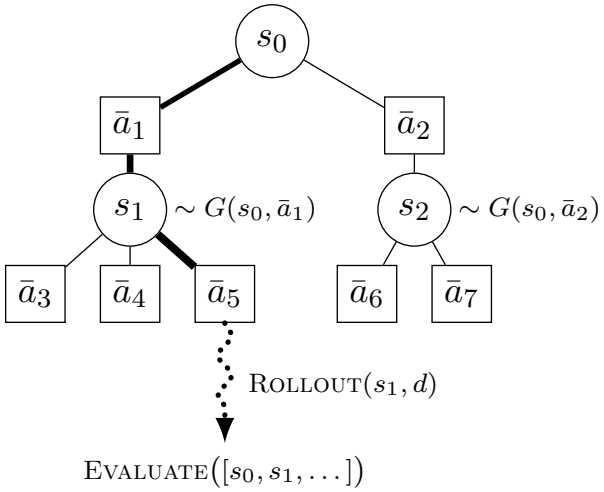


Figure 3.5: MCTS-PW with deterministic next states and SUT evaluated at the end of the rollout.

determines the next state. This notion can be seen in fig. 3.5 where each action $\bar{a}_i$ leads to a single deterministic state s_i as its child node. The states are deterministically sampled from the generative model G given the current state and action. We define the state s as a collection of all preceding actions deterministically leading to that point in the simulation. Note that because the state s is a sequence of seeds that uniquely determines the state of the simulator $\bar{S}$, we may overload the notation when calling EVALUATE and ISTERMINAL for convenience. In our formulation, the generative model samples state transitions and does not evaluate the underlying system (unlike standard MCTS). This difference is in algorithm 9, ROLLOUT, where new state transitions are generated during the rollout and the system is only evaluated at the end. By default, actions are uniformly selected from a random policy. However, to encourage exploration of promising actions, the current best action $\bar{a}^*$ is used mid-rollout. The best action is updated at the end of the rollout based on updated Q-values.

These modifications were made to MCTS-PW specifically for episodic reward problems. The evaluation of our SUT is expensive so we limit the evaluations to the end of the rollout (rather than at node creation and throughout the rollout). This way, we can reduce the number of external system executions but still provide the search with information during tree expansion, namely using back-propagated values of the transition probabilities and the miss distance from previously finished rollouts. Choosing to evaluate at the end of the rollout provides the SUT with an expanded set of waypoints which it evaluates once the rollout has reached its maximum depth. Generally for AST problem formulations, the discount factor γ is set to 1. Algorithm 7 is the entry point of MCTS-PW and algorithms 8 to 11 detail the sub-routines.

Algorithm 7 Top-level Monte Carlo tree search algorithm.

function MONTECARLOTREESEARCH(s, d)
  loop
    SIMULATE(s, d)
  return $\arg\max_{\bar{a} \in A(s)} Q(s, \bar{a})$

3.4 Implementation

The AST implementation was written in the Julia programming language [10]. Implementation of the simulation environment around the SUT was also written in Julia, but this section will focus on the algorithms required for AST and MCTS-PW. Modifications to MCTS were implemented and merged into the existing MCTS.jl[1] Julia package.

[1] `https://github.com/JuliaPOMDP/MCTS.jl`

Algorithm 8 Monte Carlo tree search simulation.

function SIMULATE(s, d)
  if $d = 0$
    return 0
  if $s \notin \mathcal{T}$
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{s\}$
    $N(s) \leftarrow N_0(s)$
    return ROLLOUT(s, d)
  $N(s) \leftarrow N(s) + 1$
  $\bar{a} \leftarrow$ SELECTACTION(s) ▷ selection
  $(s', r) \leftarrow$ DETERMINISTICSTEP($s, \bar{a}$) ▷ expansion
  $q \leftarrow r + \gamma$SIMULATE($s', d - 1$) ▷ simulation/rollout
  $N(s, \bar{a}) \leftarrow N(s, \bar{a}) + 1$
  $Q(s, \bar{a}) \leftarrow Q(s, \bar{a}) + \frac{q - Q(s,\bar{a})}{N(s,\bar{a})}$ ▷ backpropagation
  return q

Algorithm 9 Modified rollout with end-of-depth evaluation.

function ROLLOUT(s, d)
  if $d = 0$
    $(p, e, d) \leftarrow$ EVALUATE(s)
    $\tau \leftarrow$ ISTERMINAL(s)
    $\bar{a}^* \leftarrow$ UPDATEBESTACTION(s, Q)
    return $R(p, e, d, \tau)$
  else if $d = \lfloor d_{\max}/2 \rfloor$
    $\bar{a} \leftarrow \bar{a}^*$ ▷ exploit the best action
  else
    $\bar{a} \leftarrow$ SAMPLEACTION(s, Q)
  $(s', r) \sim G(s, \bar{a})$
  return $r + \gamma$ ROLLOUT($s', d - 1$)

Algorithm 10 Action selection with progressive widening.

function SELECTACTION(s)
  if $|A(s)| \leq kN(s)^\alpha$
    $\bar{a} \leftarrow$ SAMPLEACTION(s, Q)
    $(N(s, \bar{a}), Q(s, \bar{a})) \leftarrow (N_0(s, \bar{a}), Q_0(s, \bar{a}))$
    $A(s) \leftarrow A(s) \cup \{\bar{a}\}$
  return $\arg\max_{\bar{a} \in A(s)} Q(s, \bar{a}) + c\sqrt{\frac{\log N(s)}{N(s,\bar{a})}}$

Algorithm 11 Single deterministic next state.

function DETERMINISTICSTEP($s, \bar{a}$)
  if $N(s, \bar{a}, \cdot) = \emptyset$
    $(s', r) \sim G(s, \bar{a})$
    SETCACHE($s, \bar{a}, s', r$)
    $N(s, \bar{a}, s') \leftarrow N_0(s, \bar{a}, s')$
  else
    $(s', r) \leftarrow$ GETCACHE($s, \bar{a}$)
    $N(s, \bar{a}, s') \leftarrow N(s, \bar{a}, s') + 1$
  return (s', r)

3.4.1 Interface

To apply AST to a general black-box system, a user has to provide the interface defined in table 3.1. The simulation object $\bar{\mathcal{S}}$ is the user-defined data structure that holds parameters for their simulation. All of the following functions take the simulation object $\bar{\mathcal{S}}$ as input and can modify the object in place. The INITIALIZE function resets the simulation and the SUT to an initial state. The EVALUATE function executes the SUT and returns the transition probability p, a boolean indicating an event occurred e, and the miss distance d. Three subroutines determine these output values: TRANSITION, MISSDISTANCE, and ISEVENT (where all three subroutines are used by the EVALUATE function, but may also be called individually). Finally, the ISTERMINAL function returns a boolean τ to indicate if the simulation is in a terminal state.

Table 3.1: Adaptive Stress Testing Interface

Function	**Input $\mapsto$ Output**
INITIALIZE	$\bar{\mathcal{S}} \mapsto \emptyset$
EVALUATE	$\bar{\mathcal{S}} \mapsto \langle p, e, d \rangle$
TRANSITION	$\bar{\mathcal{S}} \mapsto p \in \mathbb{R}$
MISSDISTANCE	$\bar{\mathcal{S}} \mapsto d \in \mathbb{R}$
ISEVENT	$\bar{\mathcal{S}} \mapsto e \in \mathbb{B}$
ISTERMINAL	$\bar{\mathcal{S}} \mapsto \tau \in \mathbb{B}$

As an example, the functions in the above interface can either be implemented directly in Julia or can call out to C++, Python, MATLAB® or run an executable on the command line. Typically, implementing the MISSDISTANCE and ISEVENT functions rely solely on the output of the SUT, thus keeping in accordance with the black-box formulation.

3.4.2 Stress Testing Julia Framework

We have implemented the AST interface written in Julia as part of a new package called POMDPStressTesting.jl[2] (see chapter 5). This package is inspired by work originally done in the AdaptiveStressTesting.jl[3] package, but POMDPStressTesting.jl adheres to the `MDP` interface defined by the POMDPs.jl[4] package [22]. Thus, POMDPStressTesting.jl fits into the POMDPs.jl ecosystem, which is why it can use the MCTS.jl package as an off-the-shelf solver. This design choice allows other Julia packages within the POMDPs.jl ecosystem to be used; this includes solvers, simulation tools, policies, and visualizations. The intention of the POMDPStressTesting.jl package is to provide the user with a virtual black-box interface they must define, and provide the necessary AST algorithms to run the search. Future modifications will focus on inclusion of benchmark falsification problems from the literature [23]. Section 5.1 provides more details about this new package.

[2] `https://github.com/sisl/POMDPStressTesting.jl`
[3] `https://github.com/sisl/AdaptiveStressTesting.jl`
[4] `https://github.com/JuliaPOMDP/POMDPs.jl`

3.5 Application

The primary goal of this work is applying the modified AST formulation to a trajectory predictor in a developmental commercial FMS. The following sections will describe the input and output specification of the trajectory predictor and detail the investigated failure event and the associated miss distance. We will also describe the simulation environment constructed to run the SUT.

3.5.1 Trajectory Predictor

The inputs of the trajectory predictor controlled by AST are the origin and destination airports, a set of intermediate waypoints, and the wind direction and magnitude at each waypoint. The output of the trajectory predictor is a detailed flight path which provides predicted vertical data, predicted lateral data (i.e., lateral packets, illustrated in fig. 3.2), and other flight path information currently unused in this application. The following failure event and miss distance is calculated by parsing the SUT output after each evaluation.

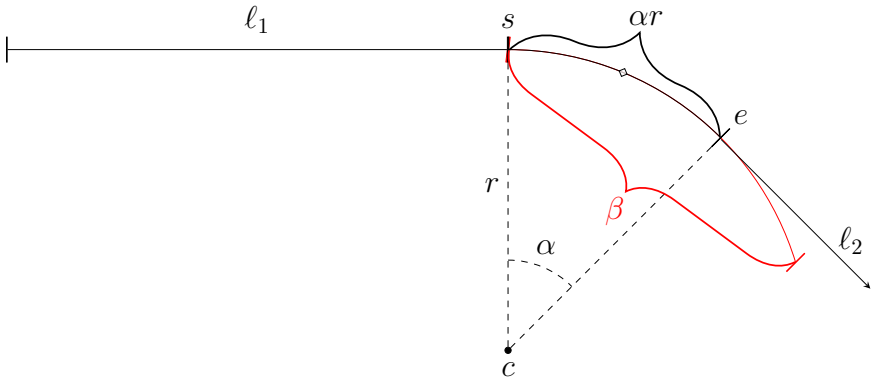


Figure 3.6: Arc length β and calculated arc length αr, showing a failure in red.

Arc Length Failure

Arc length is defined as the distance traveled across the arc from the starting point s and ending point e, shown in fig. 3.6. Failures can arise when the arc length β does not agree with the arc length computed using the angular extent α and arc radius r. Angular extent is computed as

$$\alpha = \text{sign}(r) \cdot |z_s - z_e| + 2\pi \tag{3.3}$$

where z_s is the azimuth from the center waypoint c to the starting waypoint s, and z_e is the azimuth from the center waypoint c to the ending waypoint e. The sign of r determines the turn direction, where negative values represent left turns. A failure occurs when the calculated difference $|\beta - \alpha r|$ is above the threshold $h = 10$ ft.

The arc length miss distance is how close the arc length difference comes to the threshold h. We transform this difference by scaling the log-ratio of the threshold h and the maximum miss distance from that trajectory. This way, non-positive values indicate an event. Namely, we define miss distance to be:

$$d = \rho \log\left(\frac{h}{\max|\beta - \alpha r|}\right) \tag{3.4}$$

We use a scale of $\rho = 100$ to match the expected range of the log-likelihood in our problem so that the log-likelihood does not dominate the miss distance in the reward function.

3.5.2 Simulation Environment

A simulator was constructed to sample waypoint trajectories and evaluate the SUT. Starting from an origin airport, waypoints were sampled from a multivariate Gaussian distribution of independent normals that encodes waypoint direction and distance (relative to the previous waypoint) and wind direction and magnitude, deterministically controlled by the seed $\bar{a}$:

$$\mathbf{w} \overset{\bar{a}}{\sim} \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{3.5}$$

Mean and variance values for the waypoint direction and distance were set by a domain expert and the values for winds aloft were learned from observational weather data.

$$\boldsymbol{\mu} = [180\,^\circ,\ 50\,\text{nmi},\ -88.5^\circ,\ 66.8\,\text{kts}]$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} 45^\circ & 0 & 0 & 0 \\ 0 & 30\,\text{nmi} & 0 & 0 \\ 0 & 0 & 39.5^\circ & 0 \\ 0 & 0 & 0 & 24.4\,\text{kts} \end{bmatrix}$$

The simulator implements the interface defined in section 3.4.1 for the trajectory predictor. The failure event and miss distance calculations described in section 3.5 were also implemented within the simulator.

3.5.3 Navigational Database

In addition to the simulator, we have access to a navigational database of aircraft routing procedures. The routes are encoded as collections of waypoints describing departure, arrival, and en-route airways. Exhaustively searching all combinations of the waypoints in the navigational database is the current approach to finding failures during development. We employ the navigational database as a baseline by sampling the same allotted number of SUT evaluations to assess the miss distance distribution and search for failures.

3.6 Experiments

Experiments were run to test the AST approach using MCTS-PW against direct Monte Carlo (MC) simulations as a naïve baseline and the cross-entropy method as an importance sampling baseline. We also perform Monte Carlo sampling over the routes in the navigational database as another baseline. Algorithm 12 describes the direct Monte Carlo simulation approach for n episodes, starting at an initial state s_0, with a rollout depth d. Note this rollout function does not exploit the best action as described in section 3.3.2.

Algorithm 12 Direct Monte Carlo simulation.

```
function DirectMonteCarlo(s_0, n, d)
    for 1 → n
        Initialize(S̄)
        Rollout(s_0, d)                    ▷ without expoiting the best action
```

The cross-entropy method (CEM) is a probabilistic optimization algorithm that iteratively fits a proposal distribution to elite samples (see section 2.2.2 in the previous chapter) [54, 55]. The method uses importance sampling, which introduces a proposal distribution over rare events to sample from then re-weights the posterior likelihood by the *likelihood ratio* of the true distribution ($\mathbf{w}$ in our case) over the proposal distribution. The idea is to artificially make failure events less rare under the newly fit proposal distribution. We set the proposal distribution to be the same as the true distribution $\mathbf{w}$, with the exception that the waypoint distance was reduced to $\mu = 1\,\text{nmi}$ and $\sigma = 3\,\text{nmi}$ to encourage smaller distances between the waypoints.

For the experiments, the San Francisco International Airport (KSFO) was used as an origin airport and the Los Angeles International Airport (KLAX) was used as a destination airport. Comparisons were run to assess the effectiveness of AST against CEM and direct MC in finding high-likely severe failure events. Metrics include the number of failure events found N_E, iteration of first failure i_{FF}, and statistics about the miss distance. We also report the mean log-likelihood of failures found by each algorithm relative to the mean log-likelihood of failures found by the direct MC approach. For a given algorithm, this is computed as:

$$\text{rel-log}(p) = \frac{\log(p_{\text{alg}})}{\log(p_{\text{MC}})} \tag{3.6}$$

Values larger than one indicate a higher relative likelihood.

All experiments were run with the MCTS hyperparameters listed in table 3.2. Sensitivity analysis of the various hyperparameter values has been omitted from this chapter for brevity. When controlling the parameters for progressive widening, to encourage widening let $k \to \infty$ and $\alpha \to 1$. To discourage widening, let $k \to 1$ and $\alpha \to 0$.

Table 3.2: Algorithm Hyperparameters

Hyperparameter	**Value**
episodes *	5000
maximum tree depth $d_{\max}$ (i.e., number of waypoints) *	12
rollout depth d †	12
exploration constant c	10
progressive widening k	10
progressive widening α	0.3

* Used by all algorithms.
† Used by MCTS and direct Monte Carlo.

3.6.1 Results and Analysis

We first look at the performance of each approach over all episodes. An initial seed is set across each experiment and we run each algorithm for 5000 episodes. For each of these approaches, the number of episodes also corresponds to the number of SUT evaluation calls.

The first two plots in fig. 3.7 show the running mean and minimum miss distance over each episode. Notice that the direct Monte Carlo approach applied to the navigational database baseline converges quickly to a minimum miss distance that remains above the rest, and a running mean that only outperforms the CEM approach. Evident from fig. 3.7 is the initial behavior that MCTS and direct Monte Carlo share. Recall that MCTS balances exploration and exploitation, and initially acts similar to direct Monte Carlo. This similarity is based on the choice of exploration hyperparameters and the miss distance in the reward function. This behavior suggests that the miss distance for this problem is a noisy measurement of the actual distance to a failure event. At about episode 500, the MCTS approach starts to exploit found failures which can be seen as the descent of the running mean passing the origin (i.e., the event horizon).

One goal of the AST approach using MCTS is to exploit known failures to maximize their likelihood. The bottom plot in fig. 3.7 shows the cumulative number of failure events which highlights this behavior. Notice that each approach finds failures relatively early in the search, suggesting that failure events may be common given the choice of simulation environment.

We are also interested in the distribution of the miss distances collected from each approach. Recall that the miss distance d is a transformation of the arc length discrepancy relative to a threshold, detailed in eq. (3.4). Thus, the value for d is unitless and non-positive values indicate an event. The top plot of fig. 3.8 shows the miss distance distributions, indicating the event horizon at the origin. The miss distance distribution from the navigational database is used as a proxy for miss distance distributions we would expect in the real-world. The CEM approach converges to a local minima and stays there, which is evident in the concentration of the CEM miss distance distribution. MCTS and the direct MC approach share similar distributions to the left of the event horizon (indicating non-failure events), further suggesting that the miss distance is a noisy

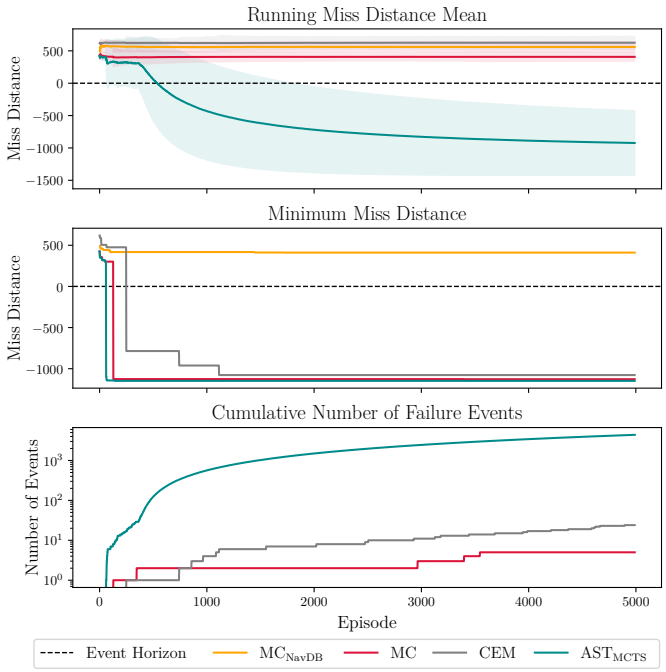


Figure 3.7: Running miss distance mean, minimum miss distance, and log-scaled cumulative number of failure events across episodes. One standard deviation is reported in the shaded regions.

measurement of the true distance to a failure. The spike to the right of the origin is the distribution of failure events found by our AST approach using MCTS. The bottom plot in fig. 3.8 shows the distribution of log-likelihoods filtered for the failure events, suggesting that AST finds failures with higher likelihood than the CEM approach.

The collected aggregate results are shown in table 3.3. AST finds failures with relative likelihood about an order of magnitude greater than that of direct Monte Carlo. The CEM approach finds a small number of failures with very low relative likelihood. This is because CEM is using importance sampling and after re-weighting the samples using the true distribution, we would expect to get these extremely small likelihood values. AST has the lowest mean miss distance $\bar{X}_d$, noting the large standard deviation which is a result of large differences between miss distances from failure and non-failure events. Each approach finds their first failure early in the experiment, with AST

Table 3.3: Experimental Results

Algorithm*	N_E	i_{FF}	$\bar{X}_d$	min(d)	rel-log(p)†
MC_{NavDB}	0	—	560.21 ± 75.09	410.98	—
MC	5	128	407.44 ± 64.85	−1127.7	1.0
CEM	24	249	625.65 ± 97.80	−1077.3	4.5×10^{-161}
AST_{MCTS}	**4394**	**61**	$\mathbf{-923.49 \pm 497.4}$	**−1147.9**	**13.1**

* Hyperparameters listed in table 3.2.
† Mean log-likelihood relative to direct Monte Carlo.

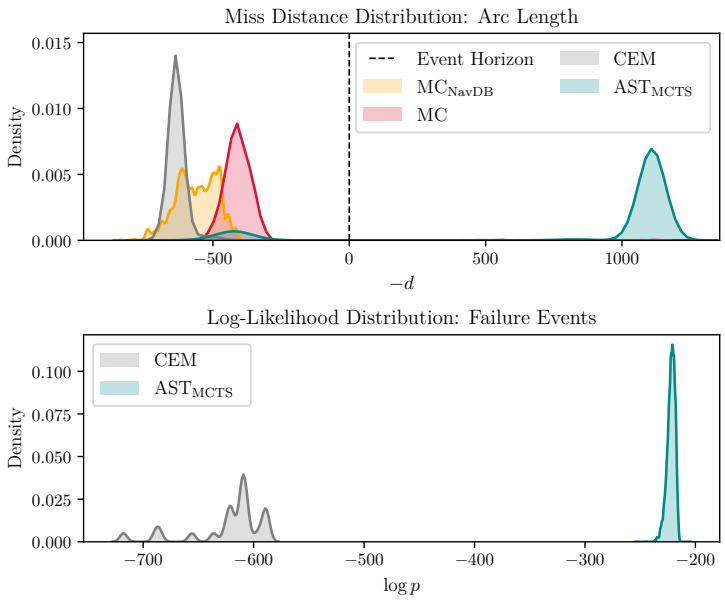


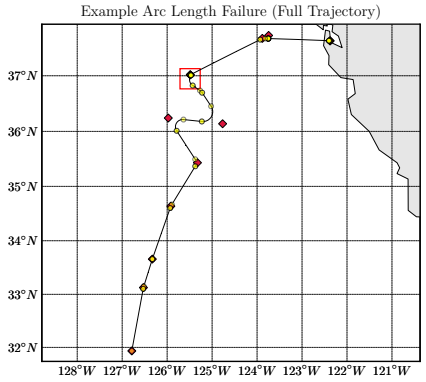
Figure 3.8: Distribution of negative miss distances for all episodes (where values to the right of the origin are events) and distribution of log-likelihoods filtered by failure events.

finding failures the earliest. The effect of exploiting the best action midway through the rollout accelerates finding these failures. Once found, AST will optimize the failures to maximize their likelihood. We see that AST finds failures in about 88% of episodes (i.e., system executions), where as standard MC and CEM find failures in about 0.1% and 0.48% of episodes, respectively.

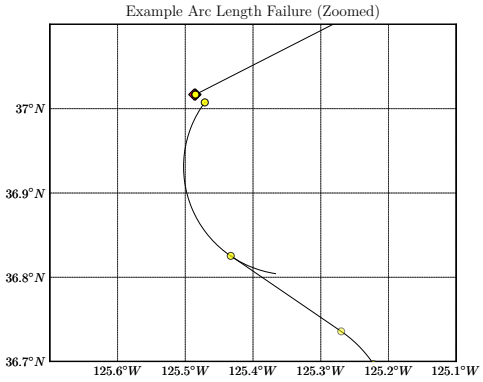
3.6.2 Example Failure

Many of the failures are a result of two duplicate waypoints being generated in sequence. Based on the defined environmental distributions, this is possible, but unlikely. We observed that duplicate waypoints do not always cause failures and that certain assemblages of neighboring waypoints affect whether a failure is caused by the duplicates. Despite failures arising from these duplicate waypoints, certain failures found only by AST have waypoints close in range to each other—not necessarily identical—which can also result in arc length failures. Figure 3.9a shows an example failure trajectory and fig. 3.9b zooms in on the specific arc length failure. Refer to the figure captions for further descriptions. A set of failure trajectories was generated and provided to the engineers of the trajectory predictor. These failures are encoded in input files than can be deterministically played back through the system.

Due to the nature of exploiting known failures, certain failure cases may only have minor differences between them. To assess the impact of the trajectory predictor failures on broader flight operations, each failure case would have to be evaluated on the full FMS in simulation. This would include modeling aircraft dynamics, guidance systems, and control feedback. Full assessment of the



(a) Full trajectory of an example arc length failure originating from KSFO. Red diamonds indicate the input waypoints selected by MCTS and the yellow circles indicate the output lateral packets from the SUT. The red box shows where the failure occurs, shown in more detail in fig. 3.9b.



(b) Example arc length failure, zoomed in from fig. 3.9a. Notice two almost identical red waypoint diamonds, which are separated by about 0.08 nmi or about 486 ft (zoom in further for more detail). The arc length failure is shown as the extending arc after the center yellow waypoint, which extends about 3 nmi past its intended end waypoint. This extension is the negative miss distance.

Figure 3.9: Example failure found by adaptive stress testing.

trajectory predictor failures would help inform the system developers in their decision to mitigate issues before deployment. Further extensions of this work include searching for other failure events and stress testing other components of the FMS.

3.7 Discussion

Adaptive stress testing was extended for sequential systems with episodic reward to find likely failures in FMS trajectory predictors. To improve search performance, we used Monte Carlo tree search with progressive widening and modified the rollout with end-of-depth evaluations. We replace the current with the best action midway through the rollout to encourage further exploration of promising actions, resulting in exploiting failures to maximize their likelihood. A simulation environment was constructed to evaluate the trajectory predictor, and a navigational database was sampled to compare to existing methods of finding failures during development. Performance of AST using MCTS-PW was compared against direct Monte Carlo simulations and the cross-entropy method. Results suggest that the AST approach finds more failures with both higher severity and higher relative likelihood. The failure cases are provided to the system engineers to address unwanted behaviors before system deployment. In addition to requirements-based tests, we show that AST can be used for confidence testing during development.

Chapter 4

Adversarial Weakness Recognition

Black-box systems are not always sequential decision making systems, and to validate non-sequential systems we have to reformulate the problem. When validating a non-sequential black-box system, exhaustively evaluating over the entire validation dataset may be computationally intractable. Then the challenge becomes how to intelligently select candidate inputs that are likely to lead to failures. The motivation to find such candidate failure inputs is to reduce the need to exhaustively evaluate an entire validation dataset, especially for black-box systems that may be computationally expensive to call. Each input sample in the dataset may also be high-dimensional, therefore we also want to learn a low-dimensional representation of the dataset and use that representation to learn features that caused failures. We propose an adversarial failure classifier that inputs the low-dimensional representation and determines if a sampled input will likely result in a failure. Our proposed framework combines the components of a dataset encoder, an adversarial failure classifier, and a candidate failure selector to propose dataset inputs that will likely result in failure, all to reduce the computational cost of evaluating the system under test and to focus only on evaluating predicted failures. The framework is open sourced and available under the MIT license.[1]

4.1 Related Work

Current approaches to validate black-box systems focus their search over input disturbances to find failures [14]. Adaptive stress testing—the black-box reinforcement learning approach described in the previous chapter—has been used to find the most likely failures in aircraft collision avoidance systems [40, 42], aircraft flight management systems [50], and autonomous vehicles [34, 33]. An underlying assumption of the AST problem formulation is that the system under test can be modeled as a sequential decision making process with explicitly defined states. This assumption limits the application of AST for validation over a dataset and ultimately creates *more* data to validate due to applying input disturbances. Other approaches use model-based clustering to

[1] `https://github.com/sisl/FailureRepresentation.jl`

efficiently sample large datasets, but rely on tuned initialization parameters for good performance [64]. Bayesian methods have been used to efficiently sample data from large hierarchical datasets using techniques to fit clusters over randomly partitioned subsets of the data [27]. To be effective, these techniques assume hierarchical structure in the data. Compression-based approaches have also been applied to reduce data size without loss of useful information [25], but are generally domain specific. The most straightforward approach is to evaluate the system exhaustively over the entire validation dataset, which may be expensive and is the main motivation of our framework.

4.2 Dataset and Features

We use the MNIST handwritten digit dataset [19] as our collection of inputs we want to selectively sample. We chose MNIST because it is a well-known machine learning dataset with many benchmarks, and is small enough to quickly iterate our framework without worrying about computational concerns. The MNIST training data contains 60,000 gray-scale images of handwritten digits, each represented as 28×28 pixels. The test dataset contains 10,000 gray-scale images with the same 28×28 pixel dimensions. For feature extraction, we left that up to the autoencoder described in Section 4.3.1.

4.2.1 Black-Box System Under Test

To test our framework, we trained an MNIST classifier to be our black-box system under test $\mathcal{S}$. We are using the Julia machine learning package `Flux.jl` [29] for building the neural network model and training. The black-box classifier $\mathcal{S}$ consists of two dense layers and a ReLU activation, mapping the input size of $28 \times 28 = 784$ to 32 activations, and then an output layer of size 10 (for each digit class). We use the logit cross-entropy loss, which is equivalent to the cross-entropy loss after applying the softmax function to the predicted output $\widehat{\mathbf{y}}$:

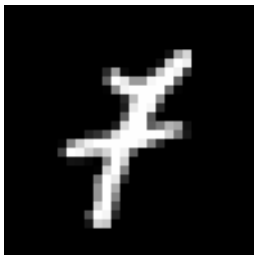


Figure 4.1: Example failure (i.e., misclassification) which classified this image as a 1 instead of a 7.

$$\mathcal{L}_{\mathcal{S}}(\text{softmax}(\widehat{\mathbf{y}}), \mathbf{y}) = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(\hat{y}_i)$$

We trained the system over 20 epochs, with a mini-batch size of 1024, and using the Adam optimizer [30] with a learning rate of $\alpha = 3e^{-4}$. This classifier achieves around 93.2% accuracy, so there is room to find weaknesses to exploit failures, where we define a failure as a misclassification. Figure 4.1 shows an example failure where the system misclassified a particular digit.

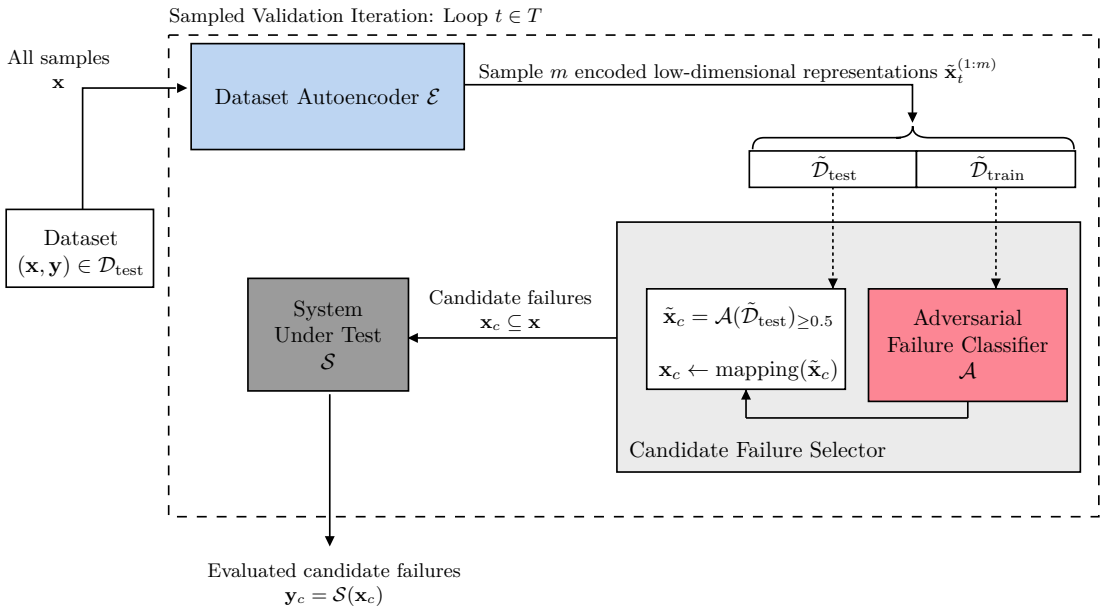
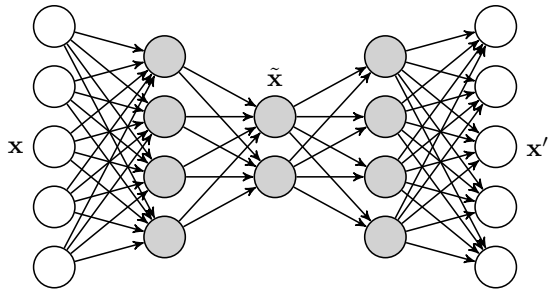


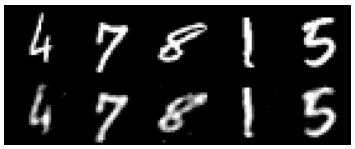
Figure 4.2: Validation framework: a dataset autoencoder $\mathcal{E}$ is trained on the entire validation dataset $\mathcal{D}_{\text{test}}$ consisting of input samples $\mathbf{x}$. Then m samples of encoded low-dimensional representations of the inputs $\tilde{\mathbf{x}}$ are selected for this iteration t, denoted $\tilde{\mathbf{x}}_t^{(1:m)}$ for all m samples. The low-dimensional representations are then split into a training and test dataset. The training dataset $\tilde{\mathcal{D}}_{\text{train}}$ is used to train an adversarial failure classifier $\mathcal{A}$ on the encoded representations. Then the test dataset $\tilde{\mathcal{D}}_{\text{test}}$ is used to select candidate failures $\tilde{\mathbf{x}}_c$ as predicted by the adversary. Finally, the candidate failures from the adversary $\tilde{\mathbf{x}}_c$ are mapped back to the original inputs $\mathbf{x}_c \subseteq \mathbf{x}$ and evaluated by the system under test $\mathcal{S}$.

4.3 Method

Our proposed framework, illustrated in fig. 4.2, consists of two major components: a dataset autoencoder, and an adversarial failure selector. These components are iteratively called within a *sampled validation loop*. The dataset autoencoder is used to sample m low-dimensional representations of the encoded input samples $\tilde{\mathbf{x}}$. We encode inputs into a lower-dimensional space for two reasons: 1) to reduce the potential high-dimensionality of the inputs $\mathbf{x}$ and 2) to learn features in this low-dimensional space that likely caused failures. We then split the m low-dimensional samples into a training set $\tilde{\mathcal{D}}_{\text{train}}$ and test set $\tilde{\mathcal{D}}_{\text{test}}$. The training set $\tilde{\mathcal{D}}_{\text{train}}$ is passed to an adversarial agent that learns characteristics of the low-dimensional feature representation that led to failures. We use a failure classifier as our adversary, and then predict which inputs led to failures over the test data $\tilde{\mathcal{D}}_{\text{test}}$, then map the predicted failures from the low-dimensional space back to the original representation, and then run the candidate inputs expected to result in a failure through the system under test.



(a) Dataset autoencoder architecture, with inputs $\mathbf{x} \in \mathbb{R}^{28\times28}$, encoded through a dense LeakyReLU layer of size $\frac{28\times28}{2}$ to a low-dimensional representation $\tilde{\mathbf{x}} \in \mathbb{R}^{64}$, then decoded through a LeakyReLU layer of size $\frac{28\times28}{2}$, outputting $\mathbf{x}' \in \mathbb{R}^{28\times28}$.



(b) Sampled output from the MNIST autoencoder: true input is on top and the recovered input on bottom.

Figure 4.3: Dataset autoencoder architecture and sample decoded output.

4.3.1 Dataset Autoencoder

To get a low-dimensional representation of the inputs $\mathbf{x}$, we used an autoencoder network [35]. We trained the autoencoder $\mathcal{E}$ on the MNIST test dataset $\mathcal{D}_{\text{test}}$ and sample from the low-dimensional representation $\tilde{\mathbf{x}}$ as inputs into our adversarial failure classifier. We use the MNIST test set because this is our input validation set—thus, we want our autoencoder to only have information about the validation set, without the need to have access to the training set used by the system under test. The autoencoder network maps the 28×28 input image $\mathbf{x}$ into a low-dimensional latent space of size 64 using a LeakyReLU activation. Then the decoder will take the 64-dimensional representation $\tilde{\mathbf{x}}$, again using a LeakyReLU activation layer, and attempt to recover the original input $\mathbf{x}'$. We pre-trained the autoencoder over 20 epochs, with a mini-batch size of 1000, and tuned the network parameters using the Adam optimizer with a learning rate of $\alpha = 1e^{-3}$. Training is unsupervised and we use the mean squared error loss function:

$$\mathcal{L}_{\mathcal{E}}(\mathbf{x}', \mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} \left(x_i' - x_i\right)^2$$

Figure 4.3a illustrates the autoencoder network architecture and Figure 4.3b shows samples of the true inputs and their output after encoding/decoding.

4.3.2 Adversarial Failure Classifier

To learn the low-dimensional features that are likely to cause failures, we train an adversary $\mathcal{A}$ in the validation loop to classify failures. The supervised adversary is trained on the partition $\tilde{\mathcal{D}}_{\text{train}}$ of the low-dimensional samples $\tilde{\mathbf{x}}$ and outputs a prediction that a given input would lead to a system failure. We train the adversary on the low-dimensional representation for computational efficiency, rather than using the potentially high-dimensional input. To get the target labels $\mathbf{y}$, we run the

system $\mathcal{S}$ using the *true* inputs associated to the *encoded* inputs which are part of the training data $\tilde{\mathcal{D}}_{\text{train}}$. This gives us targets we can now train our adversary on using the binary cross-entropy loss:

$$\mathcal{L}_{\mathcal{A}}(\widehat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(\hat{y}_i) - (1 - y_i)\log(1 - \hat{y}_i)$$

The adversarial network architecture consists of 3 dense layers which map the low-dimensional representation of $\tilde{\mathbf{x}} \in \mathbb{R}^{64}$ through a ReLU layer of size 128, another ReLU layer of size 64, and finally an output sigmoid layer to map the predictions to a probability. For each sampled validation iteration t (shown in Figure 4.2), we retrain the adversary $\mathcal{A}$ for 20 epochs using the Adam optimizer with learning rate $\alpha = 3e^{-5}$. Notice that our learning rate is very small, this is so we do not overfit to early iterations in t and can generalize across different samples of the low-dimensional space. The adversary will use the test data partition $\tilde{\mathcal{D}}_{\text{test}}$ to select the encoded input that it predicted would lead to a failure. We use the threshold of $\mathcal{A}(\tilde{x}) \geq 0.5$ to indicate the input $\tilde{x} \in \tilde{\mathcal{D}}_{\text{test}} \subset (\tilde{\mathbf{x}}, \mathbf{y})$ led to a failure. All encoded inputs in the test dataset that led to a failure are considered candidate failure scenarios, and we denote them as $\tilde{\mathbf{x}}_c$. We use a mapping from the encoded inputs $\tilde{\mathbf{x}}_c$ to the original inputs $\mathbf{x}_c \subseteq \mathbf{x}$, and finally pass the failure candidates to the true system $\mathcal{S}$ for evaluation.

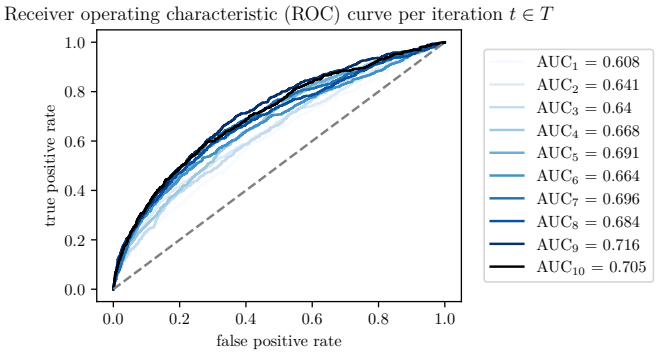
4.4 Experiments and Results

To evaluate our approach, we ran $T = 10$ sampled validation iterations, sampling $m = 500$ random encodings and partitioning these samples in half into $\tilde{\mathcal{D}}_{\text{train}}$ and $\tilde{\mathcal{D}}_{\text{test}}$ for the adversary. Because the failure rate for our system under test is about 0.0677, we augment the training dataset by duplicating known failures 10 times after running each training set through the system to get the true outputs $\mathbf{y}_{\text{train}}$. We use *precision* and *recall* to evaluate the performance of the adversary. During each iteration t, we save off the current adversary $\mathcal{A}_t$ and evaluate the *area under the ROC curve* (AUC) as shown in Figure 4.4a. The ROC curve highlights incremental improvement of the adversary after each iteration. Note that we retrain the new adversary $\mathcal{A}_t$ starting from the network weights learning by the previous adversary $\mathcal{A}_{t-1}$. To balance these metrics, we swept over the prediction threshold to determine which threshold to use (see Figure 4.4b). From the tuning sweep, we chose a threshold of $\hat{y} \geq 0.5$ to indicate a positive failure prediction by the adversary.

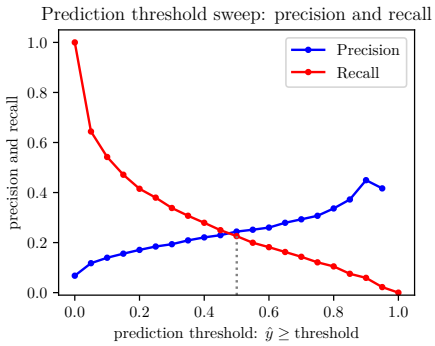
Table 4.1: Evaluation Metrics

Failure Selector	**Precision**[*]	**Recall**[*]	**Sampled Precision**[†]	**Sampled Recall**[†]
Adversary $\mathcal{A}$	0.2441	0.2260	0.2374 ± 0.11	0.3244 ± 0.17
Random	0.0647	0.4712	0.0618 ± 0.04	0.0910 ± 0.07

[*] Run over $\mathcal{D}_{\text{test}}$ only calculated for the "failure" class.
[†] Calculated from $T = 10$ iterations of the *sampled validation loop*.



(a) ROC curve and AUC for each adversary $\mathcal{A}_t$.



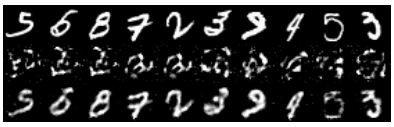
(b) Tuning prediction threshold.

Figure 4.4: ROC curve and evaluation metrics.

During each iteration t, the adversary selects k candidate inputs predicted to be likely failures. For comparison, we employ a random selection of k candidates and evaluate the precision and recall metrics of the random scheme. This allows us to compare our adversarial learning approach to a baseline and test that our approach is better than guessing randomly. Table 4.1 quantifies the evaluation metrics for the adversary and random candidate selector.

4.5 Analysis and Discussion

Shown by the confusion matrix in Figure 4.6a, the adversarial failure classifier achieves about 0.24 in precision and 0.23 in recall. Compared to random, the precision rate is about 3 times better using the adversary. Random recall—as expected—is around 0.5. These results show that our approach learned the low-dimensional representation of a failure over the validation set, and based on this information it selected candidate failures to be evaluated. To see which element of the low-dimensional feature vector contributed the most to a likely failure as predicted by the adversary, we encoded a one-hot vector over $\mathbb{R}^{28\times28}$ (i.e., the input space) and plotted the output likelihood of failure for each of the pixels, shown in Figure 4.6b.



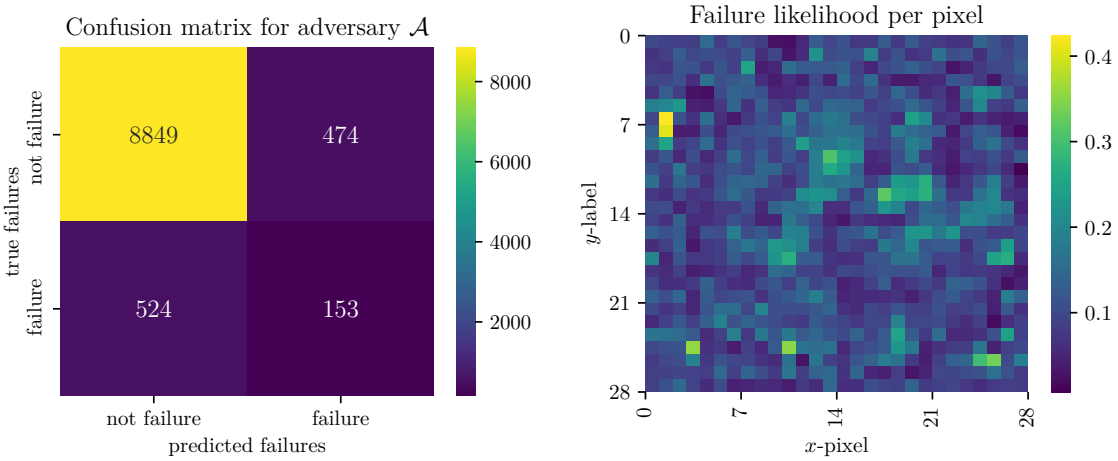
(a) Adversarial true positives.

(b) Adversarial false negatives.

Figure 4.5: MNIST failure predictions from the adversary.

Figure 4.5 compares the failures classified by the adversary. For the true positives in Figure 4.5a, the top row of digits are the 10 digits with the highest predicted failure likelihood that were true failures, the middle row shows the feature that had the strongest influence on the failure

classification (decoding a one-hot vector representation of the maximum of a softmax over low-dimensional inputs), and the bottom row shows the reconstructed digits after passing through the autoencoder. Similarly for the false negatives in Figure 4.5b, the top row are the 10 digits with the lowest predicted failure likelihood that were true failures, the middle row shows the strongest influential feature, and the bottom row shows the output of the autoencoder. Notice that certain features in the middle row are shared among the other digits, indicating features that play a larger role in classifying failures.



(a) Confusion matrix.

(b) Failure likelihood prediction per pixel.

Figure 4.6: Failure classifier performance and predictions.

4.6 Discussion

To avoid exhaustively searching an entire validation set for failures, we show that an iterative framework that uses an adversarial failure classifier trained on low-dimensional representations of the inputs can select failures about 3 times more likely than randomly choosing candidates to evaluate. To extend this framework, we could investigate different adversarial architectures, particularly around deep reinforcement learning. We could also explore how to take the true failures found, automatically improve the system under test, and then in a continual learning approach we could rerun this validation loop to use prior knowledge of previous system failures to find new system failures.

Chapter 5

Open Source Tools for Validation

This chapter discusses open-source software tools built and used in the previous chapters. These tools were designed for general applications of this work and were written in the scientific computing language Julia [10]. Open-source software is important for safety validation as it encourages reproducibility and enables others to apply these methods to different applications. More importantly, we view the field of validation as collaborative and non-competitive because ultimately safer systems benefit us all. We first discuss the adaptive stress testing package built for chapter 3, then provide quick usage examples of the optimization package built for chapter 2 and the weakness recognition framework built for chapter 4. All software packages described in this section are publicly available under the MIT license.

5.1 POMDPStressTesting.jl Summary

POMDPStressTesting.jl[1] is a package that uses reinforcement learning and stochastic optimization to find likely failures in black-box systems using adaptive stress testing [39]. The POMDPStressTesting.jl package is written in Julia [10] and is part of the wider POMDPs.jl ecosystem [22], which provides access to simulation tools, policies, visualizations, and—most importantly—solvers. We provide different solver variants including online planning algorithms such as Monte Carlo tree search [16] and deep reinforcement learning algorithms such as trust region policy optimization (TRPO) [57] and proximal policy optimization (PPO) [56]. Stochastic optimization solvers such as the cross-entropy method [54] are also available and random search is provided as a baseline. Additional solvers can easily be added by adhering to the POMDPs.jl interface.

[1] `https://github.com/sisl/POMDPStressTesting.jl`

5.1.1 Interface and Related Software

The AST formulation treats the falsification problem (i.e., finding failures) as a Markov decision process (MDP) with a reward function that uses a measure of distance to a failure event to guide the search towards failure. The reward function also uses the state transition probabilities to guide towards *likely* failures. Reinforcement learning aims to maximize the discounted sum of expected rewards, therefore maximizing the sum of log-likelihoods is equivalent to maximizing the likelihood of a trajectory. A gray-box simulation environment steps the simulation and outputs the state transition probabilities, and the black-box system under test is evaluated in the simulator and outputs an event indication and the real-valued distance metric (i.e., how close we are to failure). To apply AST to a general black-box system, a user has to implement the following Julia interface:

```
# GrayBox simulator and environment
abstract type GrayBox.Simulation end
function GrayBox.environment(sim::Simulation)::GrayBox.Environment end
function GrayBox.transition!(sim::Simulation)::Real end

# BlackBox.interface(input::InputType)::OutputType
function BlackBox.initialize!(sim::Simulation)::Nothing end
function BlackBox.evaluate!(sim::Simulation)::Tuple{Real, Real, Bool} end
function BlackBox.distance(sim::Simulation)::Real end
function BlackBox.isevent(sim::Simulation)::Bool end
function BlackBox.isterminal(sim::Simulation)::Bool end
```

Our package builds off work originally done in the AdaptiveStressTesting.jl package [39], but POMDPStressTesting.jl adheres to the interface defined by POMDPs.jl and provides different action modes and solver types. Related falsification tools (i.e., tools that do not include most-likely failure analysis) are S-TaLiRo [6], Breach [21], and FalStar [65]. These packages use a combination of optimization, path planning, and reinforcement learning techniques to solve the falsification problem. The tool most closely related to POMDPStressTesting.jl is the AST Toolbox in Python [34], which wraps around the gym reinforcement learning environment [12]. The author has contributed to the AST Toolbox and found the need to create a similar package in pure Julia for better performance and to interface with the POMDPs.jl ecosystem.

5.1.2 Statement of Need

Validating autonomous systems is a crucial requirement before their deployment into real-world environments. Searching for likely failures using automated tools enable engineers to address potential problems during development. Because many autonomous systems are in environments with rare failure events, it is especially important to incorporate likelihood of failure within the search to help inform the potential problem mitigation. This tool provides a simple interface for general black-box systems to fit into the adaptive stress testing problem formulation and gain access to

solvers. Due to varying simulation environment complexities, random seeds can be used as the AST action when the user does not have direct access to the environmental probability distributions or when the environment is complex. Alternatively, directly sampling from the distributions allows for finer control over the search. The interface is designed to easily extend to other autonomous system applications and explicitly separating the simulation environment from the system under test allows for wider validation of complex black-box systems.

5.1.3 Research and Industrial Usage

POMDPStressTesting.jl has been used to find likely failures in aircraft trajectory prediction systems (chapter 3) [50], which are flight-critical subsystems used to aid in-flight automation. This software was used to stress test a development commercial FMS so that the system engineers could mitigate potential issues before system deployment. In addition to traditional requirements-based testing for avionics certification [53], this work is being used to find potential problems during development. There has also been research on the use of POMDPStressTesting.jl for assessing the risk of autonomous vehicles and determining failure scenarios of autonomous lunar rovers.

5.2 CrossEntropyVariants.jl Summary

The CrossEntropyVariants.jl[2] package provides a general implementation of the algorithms introduced in chapter 2. Specifically, we provide the following algorithms that take an objective function S and initial proposal distribution $\mathbf{M}$ as input, and output an optimized distribution $\mathbf{M}'$:

```
M′ = cross_entropy_method(S, M) # standard CEM
M′ = ce_surrogate(S, M) # surrogate-based CEM
M′ = ce_mixture(S, M) # surrogate-based CEM using mixture models
```

To illustrate why we found Julia to be our language of choice, below we present a slightly cleaned-up version of our actual implementation of algorithm 2 (i.e., the CE-Surrogate algorithm), noting the direct translation of the pseudocode into executable Julia code:

```
function ce_surrogate(S, M; m, m_elite, k_max)
    for k in 1:k_max
        mₑ, m_elite = evaluation_schedule(k, k_max) # number of evaluations from a schedule
        X = rand(M, mₑ) # draw mₑ samples from M
        Y = map(S, eachcol(X)) # evaluate samples X using true objective S
        e = X[:, sortperm(Y)[1:m_elite]] # select elite samples output from true objective
        E = model_elite_set!(X, Y, M, e, m, m_elite) # find model-elites using a surrogate
        M = fit(M, E) # re-fit distribution M using model-elite samples
    end
    return M
end
```

[2] https://github.com/mossr/CrossEntropyVariants.jl

As an example, to run an simple optimization problem, consider the following code:

```
using CrossEntropyVariants
using Distributions

S = sierra # objective function
μ = [0, 0] # initial mean
Σ = [200 0; 0 200] # initial covariance
M = MvNormal(μ, Σ) # proposal distribution

(M, best_x, best_v) = ce_surrogate(S, M)
```

Here we select the `sierra` test function described in section 2.4.1 and optimize a multivariate Gaussian proposal distribution $\mathbf{M}$ using a call to `ce_surrogate`.

The performance of the surrogate models may be dependent on the underlying objective function. The default surrogate model is a Gaussian process with the squared exponential kernel function, yet we provide other surrogate models including linear regression, polynomial regression, and radial basis regression. As an example, to use radial basis regression instead of a Gaussian process, you can specify a `basis` keyword input:

```
S = paraboloid # objective function
M = MvNormal([0, 0], [200 0; 0 200]) # proposal distribution

(M, best_x, best_v) = ce_surrogate(S, M; basis=:squared) # squared radial basis surrogate
```

Our package relies an other open-source Julia software including Distributions.jl [9], GaussianMixtures.jl,[3] GaussianProcesses.jl [24], and Optim.jl [47]. Their own open-source contributions enabled the quick prototyping of our software and we appreciate their effort and commitment to open-source software. We were inspired by Kochenderfer and Wheeler [32] and borrowed the `cross_entropy_method` algorithm from their work.

5.3 FailureRepresentation.jl Summary

The FailureRepresentation.jl[4] package captures the work from chapter 4 primarily for reproducibility but can also be extended to other black-box systems aside from the MNIST neural network classifier. This requires the user to define a `predict(`$\mathcal{M}$`, x)` function given their system $\mathcal{M}$ and input x, replacing the `SystemUnderTest` module within the framework. We also provide experiment code and pre-trained adversarial models produced from chapter 4.

A sample usage would load the user-defined system under test and the adversarial autoencoder, then call the `sampled_validation_iteration` function illustrated in fig. 4.2.

[3] `https://github.com/davidavdav/GaussianMixtures.jl`
[4] `https://github.com/sisl/FailureRepresentation.jl`

Chapter 6

Conclusions

The applications of complex systems often built using artificial intelligence algorithms for use in safety-critical domains is very promising and this work attempts to address how we can efficiently validate these systems before they're deployed. There are many challenges presented when dealing with complex, black-box systems that require clever algorithms to solve the validation problem. We present several approaches to validation of black-box systems, which were applied to open-loop sequential systems, non-sequential classification systems, and general objective functions. Here we reiterate the contributions of this thesis and discuss open questions in the field of black-box validation.

6.1 Summary of Contributions

The main contributions of this thesis are algorithmic along with open-source tools to reproduce this work and apply these methods to other applications. We first introduced surrogate model-based extensions to the cross-entropy method algorithm used for stochastic optimization, particularly with an interest in applying the algorithms to computationally expensive objective functions with rare failure events. A byproduct of that work was an configurable test function with many local minima and a single global minimum; providing an infinite number of objectives to test, with control over their difficulty. We then propose a reformulation of the adaptive stress testing framework to validate episodic-based systems (i.e., systems that only receive a reward signal at the end of an episode) and apply this technique to stress test a developmental commercial flight management system. Next, we consider how to intelligently select data points in a large static dataset to validate black-box classification systems and develop a framework to apply this technique during a system's development cycle. Finally, we describe the open-source software developed for this thesis—all publicly available under the MIT license—both to reproduce our work and to be applied to other black-box validation applications.

Cross-entropy method variants for expensive objective functions. In chapter 2, we introduced variants of the popular cross-entropy method algorithm (CE-method) used for rare-event optimization. Using a Gaussian process as our surrogate model, section 2.4.3 shows that we can achieve better optimization performance than the standard CE-method while using the same number of true objective function evaluations. We achieve this by connecting ideas from the surrogate modeling community and introduced novel sample-efficient algorithms. These algorithms—i.e., CE-surrogate (algorithm 2) and CE-mixture (algorithm 5)—use *every* sample to fit a surrogate model, then for each true elite sample will create a sub-component distribution centered around that sample, optimized through the standard CE-method using the surrogate objective. We tested our method in a variety of experiments and introduced a novel test function called *sierra* (described in section 2.4.1) to control the spread and distinction of the local and global minima, and provide an analysis of the weak areas of the CE-method compared to our proposed variants.

Episodic adaptive stress testing applied to aircraft trajectory predictions. In chapter 3, we extend the adaptive stress testing framework for sequential systems with episodic reward to find likely failures in an aircraft trajectory prediction system. This work reported likely failure cases back to the developers and engineers to mitigate potential shortcomings of the system before deployment. Our extension improves search performance by collecting the state transitions during the search, and evaluating the system at the end of a simulated rollout; enabled by our proposed episodic AST reward function (eq. (3.2)). We modified the Monte Carlo tree search algorithm to fit this formulation by employing progressive widening strictly on the action space (algorithm 10 and algorithm 11), collecting state transitions during the Simulate stage (algorithm 8), feeding the best action midway through the rollout to encourage exploration of promising actions (algorithm 9), and only evaluating the system (i.e., collect true reward signal) at the end of the rollout. These modification of MCTS help exploit failures to maximize their likelihood. Performance of our approach was compared to direct (random) Monte Carlo search and the cross-entropy method, and section 3.6.1 shows that our approach finds more failures with both higher severity and higher relative likelihood. This work is complementary to requirements-based avionics testing (e.g., DO-178C [53]) and is currently being used for confidence testing during system development.

Adversarial weakness recognition of black-box classification systems. In chapter 4, we reframe the black-box validation problem around classification systems that are typically validated across a large, static dataset. We proposed a framework that combines the components of a dataset encoder (to reduce dimensionality, thus computation), an adversarial failure classifier (trained on past failure experiences), and a candidate failure selector to propose inputs from the dataset that will likely result in failure; all to reduce the computational cost of evaluating the system under test and to focus only on evaluating predicted failures. Section 4.4 shows that our framework selects

failures about 3 times more often than random, specifically when applied to a neural network classifier trained on the MNIST dataset. The main contribution outlined in fig. 4.2 is a general framework to efficiently apply this technique to other black-box classification systems.

Open source tools for validation. In chapter 5, we present the open-source software developed to support this thesis. We hold a principle that open-source software, and more specifically reproducibility, is vital in the field of validation. We view this field as collaborative and non-competitive; everyone wins when critical software is safer. Our software contributions enable the adaptive stress testing techniques described in chapter 3 to be broadly and quickly applied to other black-box systems (with or without the episodic restriction). We have also released the optimization algorithms detailed in chapter 2 to be applied generally to other objective functions, along with a configurable test function. Finally, we present the software used to define the framework from chapter 4, both to reproduce the work and to allow users to extend to other black-box classification systems.

6.2 Future Work

Black-box validation presents interesting challenges and there are many ways the methods proposed in this thesis could be improved. Finding the most likely failure is a difficult problem itself—aside from strictly falsification. Improving the AST techniques to be more data efficient in its search could drastically improve the failure likelihood estimate *and* speed-up falsification. This problem is not necessarily specific to AST, but rather many data-efficient reinforcement learning and black-box optimization algorithm improvements would directly transfer to the AST search. Data efficiency extensions involving the rollout stage of MCTS could be explored to produce a (biased) trajectory towards failures, rather than a random (unbiased) trajectory. The next steps in the validation process need to assess the risk of the found failures, and researching methods for risk assessment would close the loop on the full safety validation problem. Yet, in order to effectively assess risk we must collect or estimate a distribution over failures. The work in this thesis could be used as a means of collecting risk metrics during the search. Extensions in this space would enable risk assessment and would allow this work to answer more concrete questions like *how safe is the system?* and *what areas of weakness were found and exploited?*, among other interesting questions.

Appendix A

Alternative FMS Failure Events

This section details alternative failure events investigated and their associated miss distance calculations when stress testing the trajectory predictions in a flight management system (FMS) performed in chapter 3. Ultimately, analysis of these failure events showed their inadequacy and arc length failures were selected as the primary event.

A.1 Tangency Kinks

Tangency kinks arise when a small angle threshold is exceeded between the inbound straight segment and the tangent of the turn segment at a given waypoint. The failure event occurs when this angle difference θ is above the threshold $\tau_k = 0.001$ rad.

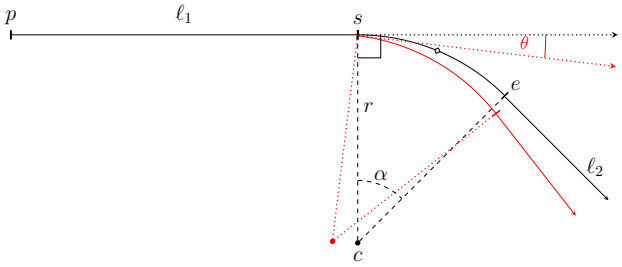


Figure A.1: Tangency kink failure event and miss distance.

Miss distance for the tangency kinks is calculated by first determining the azimuth angle z between the start point of the arc s and the center point c. The azimuth is calculated using a WGS84 Earth flattening leveraging the Geodesic.jl[1] package which implements Vincenty's formula [62]. The tangency angle θ is then calculated for each $\ell \in L$ where L is the set of lateral packets that each have a straight segment and a turn arc with radius r. The sign of the radius r indicates the direction of turn, where left is negative and right is positive. The course-in angle of the straight segment ℓ is denoted $\angle\ell$, which we then get a distance metric of:

$$d = \tau_k - \max_{(\ell,r)\in L} \left| z - \angle\ell + \text{sign}(r)\frac{\pi}{2} \right|$$

[1] `https://github.com/anowacki/Geodesics.jl`

A.2 Disconnections

Disconnected lateral packets occur when two sequential lateral packets are not connected end-to-start, thus leaving a distance δ between them. A failure occured if this geodesic distance $\delta = \|e_i - p_{i+1}\|$ is above the threshold $\tau_d = 10$ ft.

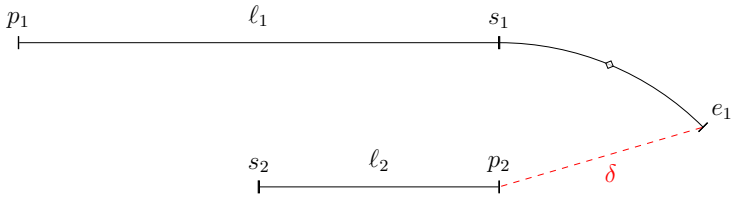


Figure A.2: Disconnected failure event and miss distance.

Miss distance is calculated as the threshold τ_d minus the maximum distance between the end points e_i (or s_i if no arc is provided) and the initial point p_{i+1} of the next lateral packet:

$$d = \tau_d - \max_{\substack{e_i \in L_i \\ p_{i+1} \in L_{i+1}}} \|e_i - p_{i+1}\|$$

A.3 Course Directions

In-bound and out-bound course directions may deviate from one another and can be classified as a failure event. Closely related to the *disconnection* failure event, the course direction failures can arise when two sequential lateral packets are disconnected. This failure specifically looks for angle differences between the course-out θ_{out} and course-in θ_{in} directions of the sequential lateral packets. If this angle difference $\omega = |\theta_{\text{out}} - \theta_{\text{in}}|$ is above the threshold $\tau_c = 1°$ then it is classified as a failure.

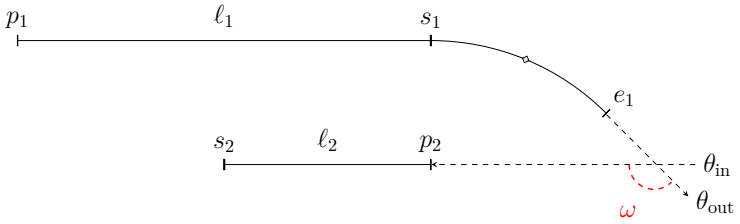


Figure A.3: Course direction failure event and miss distance.

Miss distance is calculated as how close to the threshold τ_c is the maximum wrapped angle difference between the course-out angle and course-in angle denoted by ω, namely

$$d = \tau_c - \max_{\substack{\theta_{\text{out}} \in L_i \\ \theta_{\text{in}} \in L_{i+1}}} |\theta_{\text{out}} - \theta_{\text{in}}| \,.$$

Bibliography

[1] Houssam Abbas et al. "Probabilistic Temporal Logic Falsification of Cyber-Physical Systems". In: *ACM Transactions on Embedded Computing Systems (TECS)* 12.2s (2013), pp. 1–30. DOI: 10.1145/2465787.2465797.

[2] Abbas Abdolmaleki et al. "Model-Based Relative Entropy Stochastic Search". In: *Advances in Neural Information Processing Systems (NIPS)*. 2015, pp. 3537–3545.

[3] Yasasa Abeysirigoonawardena, Florian Shkurti, and Gregory Dudek. "Generating Adversarial Driving Scenarios in High-Fidelity Simulators". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8271–8277. DOI: 10.1109/ICRA.2019.8793740.

[4] Arend Aerts et al. "Temporal Logic Falsification of Cyber-Physical Systems: An Input-Signal-Space Optimization Approach". In: *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. 2018, pp. 214–223. DOI: 10.1109/ICSTW.2018.00052.

[5] Murray Aitkin and Granville Tunnicliffe Wilson. "Mixture Models, Outliers, and the EM Algorithm". In: *Technometrics* 22.3 (1980), pp. 325–331. DOI: 10.2307/1268316.

[6] Yashwanth Annapureddy et al. "S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer. 2011, pp. 254–257. DOI: 10.1007/978-3-642-19835-9_21.

[7] Rémi Bardenet and Balázs Kégl. "Surrogating the Surrogate: Accelerating Gaussian-Process-Based Global Optimization with a Mixture Cross-Entropy Algorithm". In: *International Conference on Machine Learning (ICML)*. 2010, pp. 55–62. DOI: 10.5555/3104322.3104331.

[8] Dimitris Bertsimas et al. *A Comparison of Monte Carlo Tree Search and Mathematical Optimization for Large Scale Dynamic Resource Allocation*. 2014. arXiv: 1405.5498.

[9] Mathieu Besançon et al. "Distributions.jl: Definition and Modeling of Probability Distributions in the JuliaStats Ecosystem". In: (July 2019). arXiv: 1907.08611.

[10] Jeff Bezanson et al. "Julia: A Fresh Approach to Numerical Computing". In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671.

[11] G. A. Bird. "Monte-Carlo Simulation in an Engineering Context". In: *Progress in Astronautics and Aeronautics* 74 (Jan. 1981), pp. 239–255. DOI: 10.2514/5.9781600865480.0239.0255.

[12] Greg Brockman et al. "OpenAI gym". In: (2016). arXiv: 1606.01540.

[13] G. M. J. B. Chaslot et al. "Progressive Strategies for Monte-Carlo Tree Search". In: *Joint Conference on Information Sciences (JCIS)*. 2007, pp. 655–661. DOI: 10.1142/S1793005708001094.

[14] Anthony Corso et al. *A Survey of Algorithms for Black-Box Safety Validation*. 2020. arXiv: 2005.02979.

[15] Adrien Couëtoux et al. "Continuous Upper Confidence Trees". In: *International Conference on Learning and Intelligent Optimization*. Springer. 2011, pp. 433–445. DOI: 10.1007/978-3-642-25566-3_32.

[16] Rémi Coulom. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search". In: *International conference on computers and games*. Springer. 2006, pp. 72–83. DOI: 10.1007/978-3-540-75538-8_7.

[17] Pieter-Tjerk De Boer et al. "A Tutorial on the Cross-Entropy Method". In: *Annals of Operations Research* 134.1 (2005), pp. 19–67. DOI: 10.1007/s10479-005-5724-z.

[18] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22. DOI: 10.1111/j.2517-6161.1977.tb01600.x.

[19] Li Deng. "The MNIST database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142. DOI: 10.1109/MSP.2012.2211477.

[20] Ram Das Diwakaran, Sriram Sankaranarayanan, and Ashutosh Trivedi. "Analyzing Neighborhoods of Falsifying Traces in Cyber-Physical Systems". In: *International Conference on Cyber-Physical Systems (ICCPS)*. 2017, pp. 109–119. DOI: 10.1145/3055004.3055029.

[21] Alexandre Donzé. "Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems". In: *International Conference on Computer Aided Verification (CAV)*. Springer. 2010, pp. 167–170. DOI: 10.1007/978-3-642-14295-6_17.

[22] Maxim Egorov et al. "POMDPs.jl: A Framework for Sequential Decision Making under Uncertainty". In: *Journal of Machine Learning Research* 18.26 (2017), pp. 1–5. DOI: 10.5555/3122009.3122035.

[23] Gidon Ernst et al. "ARCH-COMP 2019 Category Report: Falsification". In: *EPiC Series in Computing* 61 (2019), pp. 129–140. DOI: 10.29007/68dk.

[24] Jamie Fairbrother et al. "GaussianProcesses.jl: A Nonparametric Bayes Package for the Julia Language". In: (2018). arXiv: 1812.09064.

[25] Carlotta Ferrari, Giorgia Foca, and Alessandro Ulrici. "Handling large datasets of hyperspectral images: Reducing data size without loss of useful information". In: *Analytica chimica acta* 802 (2013), pp. 29–39. DOI: 10.1016/j.aca.2013.10.009.

[26] J. Daniel Griffith et al. "Automated Dynamic Resource Allocation for Wildfire Suppression". In: *Lincoln Laboratory Journal* 22.2 (2017).

[27] Zaijing Huang and Andrew Gelman. "Sampling for Bayesian computation with large datasets". In: *Available at SSRN 1010107* (2005). DOI: 10.2139/ssrn.1010107.

[28] D. E. Huntington and C. S. Lyrintzis. "Improvements to and limitations of Latin hypercube sampling". In: *Probabilistic Engineering Mechanics* 13.4 (1998), pp. 245–253. DOI: 10.1016/S0266-8920(97)00013-1.

[29] Mike Innes. "Flux: Elegant Machine Learning with Julia". In: *Journal of Open Source Software* (2018). DOI: 10.21105/joss.00602.

[30] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980.

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

[32] Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. MIT Press, 2019.

[33] Mark Koren and Mykel J Kochenderfer. "Efficient Autonomy Validation in Simulation with Adaptive Stress Testing". In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. 2019, pp. 4178–4183. DOI: 10.1109/ITSC.2019.8917403.

[34] Mark Koren et al. "Adaptive Stress Testing for Autonomous Vehicles". In: *IEEE Intelligent Vehicles Symposium (IV)*. June 2018, pp. 1–7. DOI: 10.1109/IVS.2018.8500400.

[35] Mark A Kramer. "Nonlinear Principal Component Analysis Using Autoassociative Neural Networks". In: *AIChE Journal* 37.2 (1991), pp. 233–243. DOI: 10.1002/aic.690370209.

[36] Dirk P. Kroese, Sergey Porotsky, and Reuven Y. Rubinstein. "The Cross-Entropy Method for Continuous Multi-Extremal Optimization". In: *Methodology and Computing in Applied Probability* 8.3 (2006), pp. 383–407. DOI: 10.1007/s11009-006-9753-0.

[37] Nolan Kurtz and Junho Song. "Cross-entropy-based adaptive importance sampling using Gaussian mixture". In: *Structural Safety* 42 (2013), pp. 35–44. DOI: 10.1016/j.strusafe.2013.01.006.

[38] Ritchie Lee. "AdaStress: Adaptive Stress Testing and Interpretable Categorization for Safety-Critical Systems". PhD thesis. Carnegie Mellon University, 2019.

[39] Ritchie Lee, Ole J. Mengshoel, and Mykel J. Kochenderfer. "Adaptive Stress Testing of Safety-Critical Systems". In: *Safe, Autonomous and Intelligent Vehicles*. Springer, 2019, pp. 77–95. DOI: 10.1007/978-3-319-97301-2_5.

[40] Ritchie Lee et al. "Adaptive Stress Testing of Airborne Collision Avoidance Systems". In: (Sept. 2015). ISSN: 2155-7209. DOI: 10.1109/DASC.2015.7311450.

[41] Ritchie Lee et al. "Adaptive Stress Testing of Trajectory Planning Systems". In: *AIAA Scitech Forum*. 2019, p. 1454. DOI: 10.2514/6.2019-1454.

[42] Ritchie Lee et al. "Adaptive Stress Testing: Finding Likely Failure Events with Reinforcement Learning". In: *Journal of Artificial Intelligence Research* 69 (2020), pp. 1165–1201. DOI: 10.1613/jair.1.12190.

[43] Ritchie Lee et al. "Differential Adaptive Stress Testing of Airborne Collision Avoidance Systems". In: *AIAA Modeling and Simulation Technologies Conference*. 2018. DOI: 10.2514/6.2018-1923.

[44] Jing Li, Jinglai Li, and Dongbin Xiu. "An efficient surrogate-based method for computing rare failure probability". In: *Journal of Computational Physics* 230.24 (2011), pp. 8683–8697. DOI: 10.1016/j.jcp.2011.08.008.

[45] Jing Li and Dongbin Xiu. "Evaluation of failure probability via surrogate models". In: *Journal of Computational Physics* 229.23 (2010), pp. 8966–8980. DOI: 10.1016/j.jcp.2010.08.022.

[46] S. Liden. "The Evolution of Flight Management Systems". In: *Digital Avionics Systems Conference (DASC)*. Oct. 1994, pp. 157–169. DOI: 10.1109/DASC.1994.369487.

[47] Patrick Kofod Mogensen and Asbjørn Nilsen Riseth. "Optim: A mathematical optimization package for Julia". In: *Journal of Open Source Software* 3.24 (2018), p. 615. DOI: 10.21105/joss.00615.

[48] Robert J. Moss. *Cross-Entropy Method Variants for Optimization*. 2020. arXiv: 2009.09043.

[49] Robert J. Moss. "POMDPStressTesting.jl: Adaptive Stress Testing for Black-Box Systems". In: *Journal of Open Source Software* 6.60 (2021), p. 2749. DOI: 10.21105/joss.02749.

[50] Robert J. Moss et al. "Adaptive Stress Testing of Trajectory Predictions in Flight Management Systems". In: *Digital Avionics Systems Conference (DASC)* (2020). DOI: 10.1109/DASC50938.2020.9256730.

[51] Justin Norden, Matthew O'Kelly, and Aman Sinha. *Efficient Black-box Assessment of Autonomous Vehicle Safety*. 2019. arXiv: 1912.03618.

[52] Alan H. Roscoe. "Flight Deck Automation and Pilot Workload". In: *Vigilance and Performance in Automatized Systems*. Springer, 1989, pp. 111–122. DOI: 10.1007/978-94-009-0981-6_11.

[53] RTCA. *Software Considerations in Airborne Systems and Equipment Certification*. DO-178C. Dec. 2011.

[54] Reuven Rubinstein. "The Cross-Entropy Method for Combinatorial and Continuous Optimization". In: *Methodology and Computing in Applied Probability* 1.2 (1999), pp. 127–190. DOI: 10.1023/A:1010091220143.

[55] Reuven Y. Rubinstein and Dirk P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Springer, 2004. DOI: 10.1007/978-1-4757-4321-0.

[56] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347.

[57] John Schulman et al. "Trust Region Policy Optimization". In: *International Conference on Machine Learning (ICML)*. 2015, pp. 1889–1897. DOI: 10.5555/3045118.3045319.

[58] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961.

[59] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[60] Y. T. Tan, A. Kunapareddy, and M. Kobilarov. "Gaussian Process Adaptive Sampling Using the Cross-Entropy Method for Environmental Sensing and Monitoring". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 6220–6227. DOI: 10.1109/ICRA.2018.8460821.

[61] Cumhur Erkan Tuncali and Georgios Fainekos. "Rapidly-exploring Random Trees-based Test Generation for Autonomous Vehicles". In: (2019). arXiv: 1903.10629.

[62] Thaddeus Vincenty. "Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations". In: *Survey Review* 23.176 (1975), pp. 88–93. DOI: 10.1179/sre.1975.23.176.88.

[63] Ziqi Wang and Junho Song. "Cross-entropy-based adaptive importance sampling using von Mises-Fisher mixture for high dimensional reliability analysis". In: *Structural Safety* 59 (2016), pp. 42–52. DOI: 10.1016/j.strusafe.2015.11.002.

[64] Ron Wehrens et al. "Model-Based Clustering for Image Segmentation and Large Datasets via Sampling". In: *Journal of Classification* 21.2 (2004), pp. 231–253. DOI: 10.1007/s00357-004-0018-8.

[65] Zhenya Zhang et al. "Two-Layered Falsification of Hybrid Systems Guided by Monte Carlo Tree Search". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), pp. 2894–2905. DOI: 10.1109/TCAD.2018.2858463.