Journal of Artificial Intelligence Research x (xxx) x-xx Submitted x/xx; published x/xx

Adaptive Stress Testing: Finding Likely Failure Events with Reinforcement Learning

Ritchie Lee RITCHIE.LEE@NASA.GOV
NASA Ames Research Center, Moffett Field, CA 94035

Ole J. Mengshoel OLE.J.MENGSHOEL@NTNU.NO
Norwegian University of Science and Technology
NO-7491, Trondheim, Norway

Anshu Saksena ANSHU.SAKSENA@JHUAPL.EDU
Ryan W. Gardner RYAN.GARDNER@JHUAPL.EDU
Daniel Genin DANIEL.GENIN@JHUAPL.EDU
Joshua Silbermann JOSHUA.SILBERMANN@JHUAPL.EDU
Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd., Baltimore, MD 20723

Michael Owen MICHAEL.OWEN@LL.MIT.EDU
MIT Lincoln Laboratory, 244 Wood St., Lexington, MA 02421

Mykel J. Kochenderfer MYKEL@STANFORD.EDU
Stanford University, 496 Lomita Mall, Stanford, CA, 94305

Abstract

Finding the most likely path to a set of failure states is important to the analysis of safety-critical systems that operate over a sequence of time steps, such as aircraft collision avoidance systems and autonomous cars. In many applications such as autonomous driving, failures cannot be completely eliminated due to the complex stochastic environment in which the system operates. As a result, safety validation is not only concerned about whether a failure can occur, but also discovering which failures are *most likely* to occur. This article presents adaptive stress testing (AST), a framework for finding the most likely path to a failure event in simulation. We consider a general black box setting for partially observable and continuous-valued systems operating in an environment with stochastic disturbances. We formulate the problem as a Markov decision process and use reinforcement learning to optimize it. The approach is simulation-based and does not require internal knowledge of the system, making it suitable for black-box testing of large systems. We present different formulations depending on whether the state is fully observable or partially observable. In the latter case, we present a modified Monte Carlo tree search algorithm that only requires access to the pseudorandom number generator of the simulator to overcome partial observability. We also present an extension of the framework, called differential adaptive stress testing (DAST), that can find failures that occur in one system but not in another. This type of differential analysis is useful in applications such as regression testing, where we are concerned with finding areas of relative weakness compared to a baseline. We demonstrate the effectiveness of the approach on an aircraft collision avoidance application, where a prototype aircraft collision avoidance system is stress tested to find the most likely scenarios of near mid-air collision.

1. Introduction

Understanding how failures occur is important to the design, evaluation, and certification of safety-critical systems such as aircraft collision avoidance systems (Kochenderfer, Holland, & Chryssan-

thacopoulos, 2012) and autonomous cars (Bouton, Nakhaei, Fujimura, & Kochenderfer, 2018). The knowledge informs decisions that reduce the probability and impact of failures and prevent loss of life and property. We consider one of the key problems in failure analysis, which is finding the most likely sequence of transitions from an initial state to a failure state. That is, we aim not only to find a failure event, but also to maximize the probability of the scenario that causes it. For example, one might be interested in the most likely scenario, where an autonomous vehicle collides with another vehicle given probabilistic models of sensor noise and other vehicles. Maximizing probability is important because in many domains failures can almost always be reached. For example, another car approaching us head-on can swerve into our lane at the last moment, leading to a collision we could not avoid. However, not all failures are equally likely to occur. Incorporating a probabilistic model can significantly improve the relevance of the failure examples uncovered.

The problem is challenging in many ways. Many failure events of interest, such as an autonomous car colliding with a pedestrian, cannot be analyzed by considering the system alone. Because failures occur as a result of sequential interactions between the system and its environment, failure analysis must be performed over the combined system. Systems that operate in large, continuous, and stochastic environments thus present modeling and scalability challenges to analysis. The problem is exacerbated because search occurs over a sequence of time steps, which results in an exponential number of possible futures. Exhaustive consideration of all possible paths is generally intractable. In addition to a large search space, failure states can also be extremely rare and difficult to reach, which is generally the case for mature safety-critical systems.

1.1 Related Work

Existing methods for finding failure events can be broadly separated into two categories. Formal verification constructs a mathematical model of the system and rigorously proves or exhaustively checks whether a safety property holds (D'Silva, Kroening, & Weissenbacher, 2008; Kern & Greenstreet, 1999). The properties are expressed using a formal logic, such as linear temporal logic (LTL) (Pnueli, 1977). Probabilistic model checking (PMC) is a formal verification method that verifies properties over stochastic models with discrete states, such as Markov chains and probabilistic timed automata (Katoen, 2016; von Essen & Giannakopoulou, 2016; Gardner, Genin, McDowell, Rouff, Saksena, & Schmidt, 2016). PMC exhaustively evaluates properties over all states and paths subject to probabilistic constraints. Algorithms can prune infeasible paths to reduce the search space (Katoen, 2016). Automated theorem proving (ATP) uses computer algorithms to automatically generate mathematical proofs (Gallier, 2015). Systems and assumptions are modeled in formal logic, and then ATP is applied to prove whether a property holds (Kouskoulas, Genin, Schmidt, & Jeannin, 2017). If a proof is generated successfully, then that property holds over the entire model. However, if the algorithm fails to generate a proof, then it is uncertain whether the property holds. Hybrid systems theorem proving (HSTP) is a variant of ATP based on differential dynamic logic, which is a real-valued, first-order dynamic logic for hybrid systems (Jeannin, Ghorbal, Kouskoulas, Gardner, Schmidt, Zawadzki, & Platzer, 2015). A hybrid system model can capture both continuous and discrete dynamic behavior. The continuous behavior is described by a differential equation and the discrete behavior is described by a state machine or automaton. Formal verification methods can provide a counterexample when a property does not hold. More importantly, these methods provide completeness guarantees over the entire model, i.e., they can prove the absence of violations. The

major challenge is scalability. Due to their exhaustive nature, they have difficulty scaling to systems with large and complex state spaces.

The second category of methods relies on sampling, which trades completeness in favor of scalability. These methods rely on the availability of a simulator. Simulation models have very few requirements. They only require the ability to draw samples of the next state. As a result, they can contain large sophisticated models and directly embed software systems. Scenarios can be manually crafted by a domain expert or they can sweep over a low-dimensional parametric model (Chludzinski, 2009). An alternative approach is to run simulations using a stochastic model of the system's operating environment (Kochenderfer & Chryssanthacopoulos, 2010; Holland, Kochenderfer, & Olson, 2013). Sequences of states are sampled from the simulator and then the sequences are checked for failures. Because sampling does not optimize for failures, it can take a very large number of simulations to encounter the correct sequence and combination of stochastic values to encounter a failure state. Importance sampling and the cross-entropy method have been used to accelerate the discovery of rare events (Kim & Kochenderfer, 2016; O'Kelly, Sinha, Namkoong, Tedrake, & Duchi, 2018; Zhao, Lam, Peng, Bao, LeBlanc, Nobukawa, & Pan, 2016). However, while this approach improves upon direct Monte Carlo sampling, it does not leverage the sequential structure of the problem.

The problem of finding failure examples directly, known as *falsification*, has been considered in the literature. Falsification does not exhaustively cover the space as in verification, but instead uses search and optimization techniques to find failures. As a result, falsification methods can scale to much larger systems, but generally cannot prove the absence of failures. One approach formulates the problem as a minimization of a robustness measure (Donzé & Maler, 2010). Global optimization algorithms, such as simulated annealing and Nelder-Mead, have been applied, for example in the tool S-TaLiRo (Annapureddy, Liu, Fainekos, & Sankaranarayanan, 2011). Global optimization methods are not aware of the temporal relationship between optimization variables and thus these methods do not leverage the sequential structure of the problem during optimization. An alternative approach formulates the problem as a trajectory planning problem and optimizes using variants of rapidly-exploring random trees (RRTs) (Dreossi, Dang, Donzé, Kapinski, Jin, & Deshmukh, 2015). The RRT approach involves growing a tree from an initial state to a failure state by sampling a random point in the state space and growing the tree towards that target point. In the limit of infinite samples, the tree can reach any point in the reachable search space. However, the procedure requires evaluating the closeness of two states to determine which node in the tree to expand. In its original context, RRT was applied to trajectory planning in physical spaces, which are Euclidean. When the dimensionality of the state space is large and contains variables with mixed types and scales, then it is unclear what distance metric to use. The search also requires directly operating on the state, which cannot be applied to simulators with hidden state. Recent work has also proposed using reinforcement learning (Akazaki, Liu, Yamagata, Duan, & Hao, 2018) and tree search methods (Ernst, Sedwards, Zhang, & Hasuo, 2019), which are similar to the approach in this article. The key difference between existing falsification work and the current work is that we solve a slightly different problem. Existing falsification algorithms aim to find the failure example with the lowest robustness. This article aims to find the most likely failure example given a probabilistic disturbance model. A second difference is that we introduce an abstraction and learning algorithm in this article that can be a applied to (non-Markovian) systems with hidden states and partially unknown disturbance distributions.

1.2 Our Approach

This article presents adaptive stress testing (AST), a method for finding the most likely path to a failure event. We consider simulators that are Markov processes with discrete time and continuous state. AST adaptively guides the sampling of paths to optimize finding failure events and maximizing the path likelihood. It also leverages the sequential structure of the problem for optimization. As a result, it can scale to much larger problems and efficiently search for the most likely failure paths. Our AST method formulates the search problem as a sequential decision process and then applies reinforcement learning algorithms to optimize it. We present formulations for both fully observable systems, where the algorithm has full access to the simulator state, and partially observable systems, where some or all of the simulator states are hidden. In the latter case, we present a modified Monte Carlo tree search (MCTS) algorithm, called Monte Carlo tree search for seed-action simulators (MCTS-SA) that only requires access to the pseudorandom number generator of the simulator to overcome partial observability. AST treats the simulator as a black box, where the system transition behavior is not known. As a result, the approach can be applied to a broad range of systems. We base our algorithm on MCTS (Kocsis & Szepesvári, 2006) because of its ability to scale to large problems and because it can be easily modified to handle non-Markovian systems. Other algorithms can be used as well. For example, deep reinforcement learning has been used for AST to analyze the safety of autonomous cars (Koren, Alsaif, Lee, & Kochenderfer, 2018). Failure scenarios found by AST can then be further analyzed to extract common patterns, e.g., by using an automated categorization algorithm (Lee, Kochenderfer, Mengshoel, & Silbermann, 2018a).

In some applications, it may also be valuable to evaluate failure paths not in absolute terms, but in relation to a baseline system. That is, we are not interested in cases where both systems fail, but rather cases where the test system fails but the baseline system does not. We call this type of analysis *differential stress testing*. Such situations arise, for example, during regression testing where a new version of a system is compared to a previous one to identify areas of comparative weakness. One way to compare the relative behavior of two systems is to evaluate them against a common set of scenarios. For example, scenarios can be generated by running Monte Carlo simulations using the test system, and then replaying them on the baseline system (Holland et al., 2013). However, this approach suffers from the same inefficiency issues as before. The size and complexity of the state space, the rarity of failures, and the exponential explosion of searching over sequences make encountering failure events extremely unlikely. In fact, the issue is even more pronounced in differential stress testing because the failure event needs to occur in the system under test but not the baseline. We present an extension of AST to the differential setting called differential adaptive stress testing (DAST). The approach finds the most likely path to a failure event that occurs in the system under test, but not in the baseline system. DAST follows the same general formulation as AST. However, in the differential setting, we search two simulators in parallel and maximize the difference in the outcomes of the simulators.

1.3 Case Study: ACAS X

We demonstrate the effectiveness of AST and DAST for stress testing the next-generation Airborne Collision Avoidance System (ACAS X) (Kochenderfer et al., 2012). ACAS X has been recently accepted as the next international standard for aircraft collision avoidance. The system replaces the previous system, called Traffic Alert and Collision Avoidance System (TCAS), which has performed very well in the past, but is not optimized for the next-generation airspace (Kuchar & Drumm, 2007).

For example, the number of nuisance alerts is expected to dramatically increase with the rising density of air traffic. This article describes work performed while ACAS X was under development by the Federal Aviation Administration (FAA). As part of the ACAS X validation team, we obtained various prototypes of ACAS X from the FAA and stress tested them in simulated aircraft encounters to find the most likely scenarios of near mid-air collision (NMAC). Our experiments include single-threat (two-aircraft) encounters, multi-threat (three-aircraft) encounters, and differential stress testing against TCAS. Our results were reported to the ACAS X development team to inform development and assess risk. We highlight the main findings from these reports, along with the general methods we introduced.

The main contributions of this article are summarized as follows:

- We present adaptive stress testing (AST), a novel framework that formulates finding the most likely failure scenario as a sequential decision-making problem. The formulation enables reinforcement learning solvers to be applied.
- We present an abstraction for AST that uses pseudorandom seeds to overcome partial observability in the testing simulator.
- We present differential adaptive stress testing (DAST), an extension of AST for finding the most likely scenarios where a failure occurs in the system under test, but not in a baseline system.
- We present a case study of the ACAS X aircraft collision avoidance system, where we characterize its most likely scenarios of near mid-air collisions (NMACs) both in absolute terms and relative to the existing TCAS.

The remaining sections are organized as follows. Section 2 reviews sequential decision processes and MCTS. Section 3 presents an overview of the AST framework, followed by formulations of AST for fully observable and partially observable systems. The section also presents the MCTS-SA algorithm for optimizing partially observable systems. Section 4 presents DAST, an extension of AST to differential stress testing. Finally, Section 5 presents the results of analyzing near mid-air collisions in an aircraft collision avoidance system.

2. Background

In this section, we first present a brief overview of sequential decision processes, which is the mathematical framework underlying AST. We then present an algorithm for solving a sequential decision process, called Monte Carlo tree search.

2.1 Sequential Decision Process

A sequential decision process models situations where an agent makes a sequence of decisions in an environment to maximize a reward function (Kochenderfer, 2015). If the environment is known and its state is fully observable, then the problem can be formulated as a Markov decision process (MDP). An MDP is a 5-tuple $\langle S, A, P, R, \gamma \rangle$, where S is a set of states; A is a set of actions; P is the transition probability function, where $P(s' \mid s, a)$ is the probability of choosing action $a \in A$ in state $s \in S$ and transitioning to next state $s' \in S$; and R is the reward function, where $R(s, a)$ is the reward for taking a in s. We define the transition function T for convenience, where $T(s, a)$ samples the next state s' from the distribution $P(s' \mid s, a)$. The parameter $\gamma \in [0, 1]$ is the discount factor that governs how much to discount the value of future rewards.

In an MDP, the agent chooses an action $a = \pi(s)$ according to its policy π. The system evolves probabilistically to the next state $s' \sim T(s, a)$. The agent then receives reward $r = R(s, a)$ for the transition. The assumption that the transition function depends only on the current state and action is known as the *Markov property*. In cases where the underlying process is Markovian, but the agent only observes part of the state, the problem is a partially observable Markov decision process (POMDP) (Kochenderfer, 2015). At each time step, the agent observes an observation $o \in O$, which depends probabilistically on state s and action a. The actions in a POMDP at time t can only be based on the history of observations up to time t.

A *simulation* iteratively samples the sequential decision process to produce a *path*, which is a sequence of states and actions (and observations in the case of a POMDP). In this article, we assume that the model is *episodic* in that it terminates in a finite number of steps. The *terminal time* t_{end} is the first time the state enters a terminal state or reaches a maximum number of time steps $t_{\max}$. We assume that once the simulation terminates, the agent receives the terminal reward and does not collect any additional rewards thereafter. Because we consider a finite number of steps, we set $\gamma = 1$. Moreover, we define the *return* G to be the sum of rewards collected over a path, i.e., $G = \sum_{t=0}^{t_{\text{end}}} r_t$.

Reinforcement learning algorithms can be used to optimize sequential decision problems through sampling of the transition function T (Sutton & Barto, 1998; Wiering & van Otterlo, 2012). The learners adapt sampling during the search, enabling them to efficiently search large and complex state spaces. Model-free value-based reinforcement learning algorithms are a class of learning algorithms that aim to estimate the *state-action value function* $Q(s, a)$, which is the expected sum of rewards resulting from choosing action a in state s and following an optimal policy $\pi^*(s)$ thereafter. An *optimal action* a^* is an action that maximizes the state-action value function at state s, i.e., $a^* = \arg\max_a Q(s, a)$. An *optimal policy* $\pi^*(s)$ is the function that gives an optimal action a^* for each state s. The objective of reinforcement learning algorithm is to find an optimal policy $\pi^*(s)$.

2.2 Monte Carlo Tree Search

Monte Carlo tree search (MCTS) is a state-of-the-art heuristic search algorithm for optimizing sequential decision processes (Kocsis & Szepesvári, 2006; Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, & Colton, 2012). MCTS incrementally builds a search tree using a combination of directed sampling based on estimates of the state-action value function and undirected sampling based on a fixed distribution, called *rollouts*. During the search, sampled paths from the simulator are used to incrementally estimate $Q(s, a)$ and the optimal policy at nodes in the tree. To account for the uncertainty in the estimates, which may lead to premature convergence, MCTS encourages exploration by adding an exploration term to the state-action value function to encourage choosing paths that have not been explored as often. The exploration term optimally balances the selection of the best action estimated so far with the need for exploration to improve the quality of current value estimates. By doing so, MCTS adaptively focuses the search towards more promising areas of the search space. The effect of the exploration term diminishes as the number of times the state is visited increases.

This article uses a variant of MCTS called Monte Carlo tree search with progressive widening (MCTS-PW), which extends MCTS to large or continuous action spaces (Coulom, 2007; Chaslot, Winands, van den Herik, Uiterwijk, & Bouzy, 2008). When the action spaces are large (or infinite), visited actions are not revisited sufficiently through sampling alone, which hinders the quality of

value estimates. The benefit of the progressive widening in MCTS-PW is that it forces revisits to existing nodes and slowly allows new nodes to be added as the total number of visits increases. Progressive widening, sometimes also called progressive unpruning, stabilizes value estimates and prevents explosion of the branching factor of the tree. MCTS-PW converges asymptotically to the optimal solution as the number of iterations increases (Couëtoux, Hoock, Sokolovska, Teytaud, & Bonnard, 2011).

3. Adaptive Stress Testing

Adaptive stress testing (AST) aims to find the most likely path from a start state to a failure state in a discrete-time simulator (Lee, Kochenderfer, Mengshoel, Brat, & Owen, 2015). The overall approach is to formulate the search as a sequential decision-making problem and then use reinforcement learning to optimize it.

We consider a *simulator* $\mathcal{S}$ that contains a system under test (or simply *system*) $\mathcal{M}$ interacting with an *environment* $\mathcal{E}$. The system with state $\mu \in M$ takes action $a \in A$ based on observation $o \in O$ of the environment state $z \in Z$. The system interacts with the environment over discrete time $t \in [0, \ldots, t_{\text{end}}]$. The simulator state $s \in S$ is the stacked system and environment states $[\mu, z]$. We use subscript to denote the variable at time t and subscript colon range to denote the sequence of a variable over a range of time steps. For example, the simulator state path up to time t is $s_{0:t} = [s_0, \ldots, s_t]$. The state and action of the system depend on the environment observations and are modeled by

$$\mu_{t+1}, a_t = \mathcal{M}(o_{0:t}) \tag{1}$$

The environment state and observation evolve over time depending on the actions of the system and disturbances $x \in X \subseteq \mathbb{R}^n$, where n is the dimensionality of the disturbances. The term disturbances is very broad and encompasses any stochastic variables that can influence the environment. For example, the disturbances x can control the magnitude and direction of the wind, or they can control other actors in the environment such as pedestrians and other vehicles.

$$z_{t+1}, o_{t+1} = \mathcal{E}(a_{0:t}, x_{0:t}) \tag{2}$$

We assume the disturbances are independent across time and distributed with probability density $p(x \mid s)$. The disturbance model can be constructed through expert knowledge or learned from data. We define an *event space* $E \subset S$ where the event of interest occurs. While this article focuses on failure events, an event can be arbitrarily defined. We use the notation $s \in E$ to indicate that an event has occurred. Alternatively, we may use the Boolean variable e to indicate whether an event has occurred.

The goal of AST is to find the *most likely failure path*, which is the path with the highest likelihood subject to the constraint that the final state is an event.

$$\begin{aligned} \max_{x_0, \ldots, x_{t_{\text{end}}}} \quad & \prod_{t=0}^{t_{\text{end}}-1} p(x_t \mid s_t) \\ \text{subject to} \quad & s_{t_{\text{end}}} \in E \end{aligned} \tag{3}$$

AST formulates the search as a reinforcement learning problem by considering a reinforcement learning agent $\mathcal{A}$ that acts as an adversary to the system under test. We let the agent choose dis-

turbances so that the environment is as challenging to the system under test as possible. Figure 1 illustrates the general AST concept. The simulator models the behavior of the system under test and the environment. The simulator is treated as a black box by the agent. At each time step, the agent observes the simulator state, chooses a disturbance, and receives a reward. Through repeated interactions with the simulator, the agent learns to choose disturbances that maximize the reward it receives. By choosing a reward function that rewards failure events and higher likelihood transitions, the agent learns to optimize for the most likely failure path.

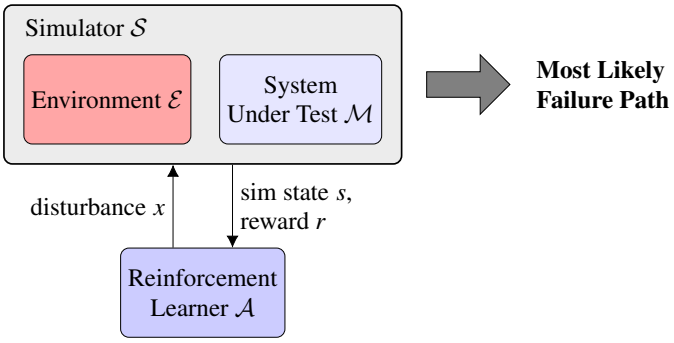

Figure 1: Adaptive stress testing. A reinforcement learning agent chooses disturbances in the environment to be most adversarial to the system under test. The reward function is crafted to search for failure events and maximize transition likelihoods.

3.1 Full Observability

If the simulator state is fully observable and the behavior of $\mathcal{M}$ and $\mathcal{E}$ depend only on the input at the current time, then the transition equations become:

$$\mu_{t+1}, a_t = \mathcal{M}(\mu_t, o_t) \tag{4}$$

$$z_{t+1}, o_{t+1} = \mathcal{E}(z_t, a_t, x_t) \tag{5}$$

We formulate the AST problem as an MDP as follows. The state of the MDP is the state s of the simulator. The agent observes the state s and chooses disturbance x. The transition to the next state is given by the transition behavior of the simulator $\mathcal{S}$, which consists of the combined behavior of $\mathcal{M}$ and $\mathcal{E}$ according to Equations 4 and 5. The reward function R is crafted to optimize Equation 3, which searches for the most likely failure path. We describe the reward function in the following subsection. We set $\gamma = 1$ to properly account for the transition likelihood in the reward function. Figure 2 illustrates the AST framework for the fully observable case.

Reward Function. The reward function is designed to find failure events as the primary objective and maximize the path likelihood as a secondary objective. Let $R_E \in \mathbb{R}_{\geq 0}$ be the event reward, $d \in \mathbb{R}_{\geq 0}$ be the miss distance, and $E \subset S$ be the event states. Then the reward function is given by:

$$R(s, x) = \begin{cases} R_E & \text{if } s \text{ is terminal and } s \in E \\ -d & \text{if } s \text{ is terminal and } s \notin E \\ \log(p(x \mid s)) & \text{otherwise} \end{cases} \tag{6}$$

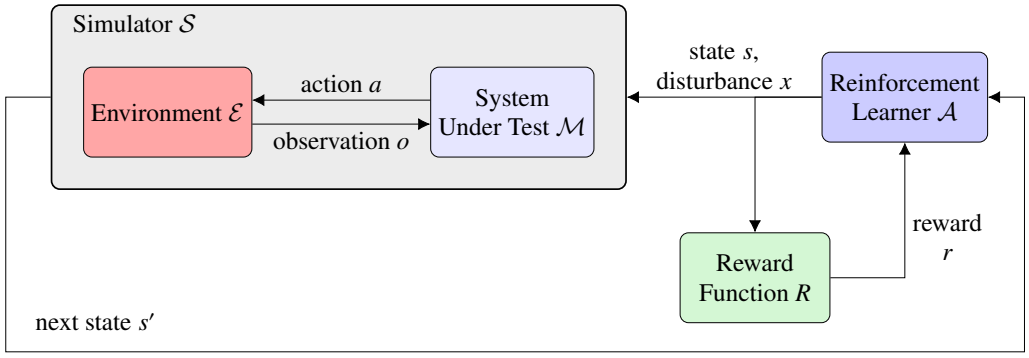

Figure 2: Adaptive stress testing of a fully observable system. The problem is modeled as an MDP where the agent observes the simulator state and chooses a disturbance at each time step. Traditional reinforcement learning algorithms can be applied to solve the MDP.

The first term of Equation 6 gives a non-negative constant reward R_E if the path terminates and a failure event occurs. If the path terminates and an event does not occur, the second term of Equation 6 penalizes the learner by assigning the negative of the miss distance to the learner. The miss distance d is some measure defined by the user that depends on s and indicates how close the simulation came to a failure. If such a measure is not available, then $-d$ can be set to a large negative constant. However, providing an appropriate miss distance can greatly accelerate the search by giving the learner the ability to distinguish the desirability of two paths that do not contain failure events. The third term of Equation 6 maximizes the overall path likelihood by awarding the log likelihood of each transition. Recall that reinforcement learning maximizes the expected sum of rewards. By choosing a reward of the log likelihood at each step, the reinforcement learning algorithm then maximizes the sum of the log likelihoods, which is equivalent to maximizing the product of the likelihoods.

Maximizing the reward function in Equation 6 also maximizes the AST objective in Equation 3. The first two terms in the reward function apply at the terminal time t_{end} and incentivizes the agent to satisfy the constraint $s_{t_{\text{end}}} \in E$ in Equation 3. The third term of the reward function maximizes the path likelihood. We show the relation as follows:

$$
\begin{aligned}
&\max_{x_0,\ldots,x_{t_{\text{end}}}} G \\
&= \max_{x_0,\ldots,x_{t_{\text{end}}}} \sum_{t=0}^{t_{\text{end}}} R(s_t, x_t) \\
&= \max_{x_0,\ldots,x_{t_{\text{end}}}} \left[\sum_{t=0}^{t_{\text{end}}-1} \log(p(x_t \mid s_t)) + R_E \cdot \mathbb{1}\{s_{t_{\text{end}}} \in E\} - d \cdot \mathbb{1}\{s_{t_{\text{end}}} \notin E\} \right] \\
&= \max_{x_0,\ldots,x_{t_{\text{end}}}} \left[\prod_{t=0}^{t_{\text{end}}-1} p(x_t \mid s_t) + R_E \cdot \mathbb{1}\{s_{t_{\text{end}}} \in E\} - d \cdot \mathbb{1}\{s_{t_{\text{end}}} \notin E\} \right]
\end{aligned}
$$

where we have used the convexity of the logarithm function to replace the sum with a product in the last line. The indicator function $\mathbb{1}\{b\}$ returns 1 if b is true and 0 otherwise. If the difference between R_E and $-d$ is sufficiently large, the learner will be incentivized to first satisfy $s_{t_{\text{end}}} \in E$ to replace the miss distance penalty with the event reward, and then maximize the path likelihood. We

do not distinguish between varying degrees of a failure. Once we have found a failure event, all optimization effort is spent towards maximizing the path likelihood.

The MDP can be optimized using standard reinforcement learning algorithms (Sutton & Barto, 1998; Wiering & van Otterlo, 2012), which only require sampling of the transitions. Existing reinforcement learning algorithms such as MCTS (Kocsis & Szepesvári, 2006; Chaslot et al., 2008) and Q-Learning (Watkins & Dayan, 1992) can be applied to optimize the decision process and find the optimal path.

3.2 Partial Observability

Many simulators simply do not allow access to all or any state information. For example, the collision avoidance system we analyze in Section 5 was provided as a software binary and maintained hidden state over function calls. We introduce an abstraction that relaxes the need for the simulator to expose its underlying state and disturbance. First, instead of explicitly representing and passing the state into and out of the simulator as in the fully observable case, we now assume that the simulator maintains state internally and the state is updated in-place. In other words, we have previously assumed that the simulator is stateless, but now we assume that the simulator is stateful and that the state is hidden from the learner. Second, rather than passing disturbance values x as input, we pass a pseudorandom seed $\bar{x}$ as a proxy. A *pseudorandom seed*, or just *seed*, is a vector of integers used to initialize a pseudorandom number generator. We assume that all random processes in the simulator are derived from the pseudorandom number generator and seed, so that the result of sampling from these processes is deterministic given the seed. The simulator uses $\bar{x} \sim \mathbb{U}_{\text{seed}}$ to seed an internal random process that samples $x \sim p(x \mid s)$. Setting the seed makes the sampling process deterministic and thus a particular seed $\bar{x}$ is deterministically tied to a particular sample of disturbance x.

3.2.1 Seed-Action Simulator

A *seed-action simulator* $\bar{\mathcal{S}}$ is a stateful simulator that uses a pseudorandom seed input to update its state in-place. The state s is not exposed externally, making the simulator appear non-Markovian to external processes, such as the reinforcement learner. The simulator uses $\bar{x}$ to draw a sample of the disturbance x and transition to the next state s'. The next state replaces the current state in-place. The simulator returns the transition likelihood $\rho = p(x \mid s)$; a Boolean indicating whether an event occurred $e = (s \in E)$; and the miss distance d. While the state cannot be observed or set, the simulator transitions are deterministic given the pseudorandom seed $\bar{x}$. This property allows a previously visited state to be revisited by replaying the sequence of pseudorandom seeds $\bar{x}_{0:t-1} = [\bar{x}_0, \ldots, \bar{x}_{t-1}]$ that leads to it starting from the initial state. The seed-action simulator $\bar{\mathcal{S}}$ exposes the following simulation control functions:

- Initialize($\bar{\mathcal{S}}$) resets the simulator $\bar{\mathcal{S}}$ to a deterministic initial state s_0. The simulation state is modified in-place.
- Step($\bar{\mathcal{S}}, \bar{x}$) advances the state of the simulator by pseudorandom sampling. First, the pseudorandom seed $\bar{x}$ is used to set the state of the simulator's pseudorandom process. Second, a sample $x \sim p(x \mid s)$ is drawn. Third, the simulator evaluates the values of the transition likelihood ρ, whether the event occurred e, and the miss distance d of the current state. Then, the simulator transitions to the next state s' using the simulator transition functions in Equations 1 and 2 replacing s with s'. The simulator returns (ρ, e, d).

- IsTERMINAL($\bar{\mathcal{S}}$) returns true if the current state of the simulator is terminal and false otherwise. The simulator terminates if an event occurs, i.e., $s \in E$, or if the simulation has reached a maximum number of time steps $t_{\max}$.

Figure 3 illustrates the AST framework under the pseudorandom seed abstraction, where the simulator has been replaced by a seed-action simulator $\bar{\mathcal{S}}$, which maintains a hidden state. The simulator takes a seed input $\bar{x}$ and transitions state internally. The simulator outputs the transition likelihood ρ, a Boolean indicating whether an event occurred e, and the miss distance d. The reward function translates the simulator outputs into a reward r. The optimization algorithm learns from the reward to optimize over its seed inputs.

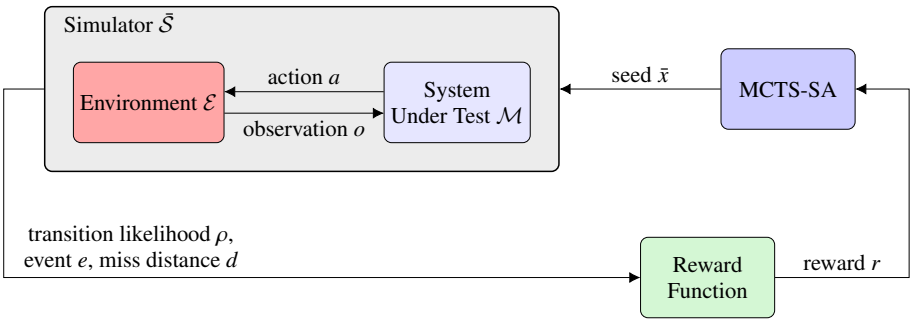

Figure 3: Adaptive stress testing of a partially observable system. We use a state-action simulator model where the simulator retains state, but does not expose it. The learner sets the pseudorandom seed that controls the pseudorandom number generator of the simulator making transitions deterministic. The simulator outputs the transition likelihood of the current transition, whether an event occurred, and a miss distance.

The seed-action abstraction also provides significant practical benefits. Large software simulators are often written in a distributed and modular fashion where each component maintains its own state. The simulator may consist of many of these components. Since the simulator state is the concatenation of the states of all the individual components, explicitly assembling and handling the state can break modularity and be a major implementation inconvenience. This abstraction, which uses in-place state update and pseudorandom seeds as a proxy to the actions, alleviates the need to explicitly form the state and enables the simulator to maintain modularity of the components. For implementation, all that is required is the ability to set the global pseudorandom seed of the simulator, which is generally easy to do in software simulators.

3.2.2 REWARD FUNCTION

The reward function for AST of a state-action simulator is given in Equation 7. The reward is expressed as a function of state-action simulator outputs and a Boolean variable τ which indicates whether the simulator has terminated, i.e., τ = IsTERMINAL($\bar{\mathcal{S}}$). The three components mirror those in Equation 6.

$$R(\rho, e, d, \tau) = \begin{cases} R_E & \text{if } \tau \wedge e \\ -d & \text{if } \tau \wedge \neg e \\ \log \rho & \text{otherwise} \end{cases} \tag{7}$$

3.2.3 MCTS-SA Algorithm

We present Monte Carlo tree search for seed-action simulators (MCTS-SA). We base the algorithm on the progressive widening variant of MCTS because the actions of the reinforcement learner are now pseudorandom seeds, which are vast (Coulom, 2007; Chaslot et al., 2008). The transition behavior of the simulator is deterministic given a pseudorandom seed input. Consequently, only a single next state is possible and there is no need to limit the number of next states. Parameters k and α are used for the progressive widening of actions. The action space is the space of all pseudorandom seeds. Since the seeds are discrete and do not have any semantic relationship, there is no need to distinguish between them. We choose the rollout policy and the action expansion function of MCTS to uniformly sample over all seeds $\bar{x} \sim \mathbb{U}_{\text{seed}}$. Sampling seeds uniformly generates $x \sim p(x \mid s)$ in the simulator. Because we are sampling from a continuous distribution, the samples of x will be unique. The hidden state s of the simulator is not available to the reinforcement learner. However, since the simulator is deterministic given the seed input, we can revisit a previous state by replaying the sequence of seeds that leads to it starting from the initial state. As a result, we use the sequence of seeds $[\bar{x}_0, \ldots, \bar{x}_{t-1}]$ as the state $\bar{s}_t$ in the algorithm. The path with the highest path return may, and likely will, be from a path that is encountered during a rollout. Since rollouts are not individually recorded in the tree, information about the best path can be lost. To ensure that the algorithm returns the best path seen over the entire search, we explicitly track the highest return seen G^* and the corresponding seed sequence $\bar{s}^*$. The MCTS-SA algorithm is shown in Algorithm 1. The algorithm takes as input a seed-action simulator $\bar{\mathcal{S}}$ and returns the most likely failure path represented by its seed sequence $\bar{s}^*$. The algorithm consists of a main loop that repeatedly performs forward simulations of the system while building the search tree and updating the state-action value estimates. The search tree $\mathcal{T}$ is initially empty.

Each simulation runs from initial state to terminal state. The path is determined by the sequence of pseudorandom seeds chosen by the algorithm, which falls into three stages for each simulation:

- *Search.* In the search stage, which is implemented by Simulate (line 13), the algorithm starts at the root of the tree and recursively selects a child to follow. At each visited state node, the progressive widening criteria (line 19) determines whether to choose amongst existing seeds or to expand the number of children by sampling a new seed. The criterion limits the number of seeds at a state $\bar{s}$ to be no more than polynomial in the total number of visits to that state (Chaslot et al., 2008). Specifically, a new seed $\bar{x}$ is sampled from a discrete uniform distribution over all seeds $\mathbb{U}_{\text{seed}}$ if $|\bar{X}(\bar{s})| < kN(\bar{s})^\alpha$, where k and α are parameters, $\bar{X}(\bar{s})$ is the set of previously applied seeds from state $\bar{s}$, $|\bar{X}(\bar{s})|$ is the cardinality of $\bar{X}(\bar{s})$, and $N(\bar{s})$ is the total number of visits to state $\bar{s}$. Otherwise, the existing action that maximizes

$$Q(\bar{s}, \bar{x}) + c\sqrt{\frac{\log N(\bar{s})}{N(\bar{s}, \bar{x})}} \tag{8}$$

  is chosen (line 23), where c is a parameter that controls the amount of exploration in the search, and $N(\bar{s}, \bar{x})$ is the total number of visits to seed $\bar{x}$ in state $\bar{s}$. Equation 8 is the upper confidence tree (UCT) equation (Kocsis & Szepesvári, 2006). The second term in the equation is an *exploration bonus* that encourages selecting seeds that have not been tried as frequently. The seed is used to advance the simulator to the next state and the reward is evaluated. The search stage continues in this manner until the system transitions to a state that is not in the tree.

Algorithm 1 MCTS for seed-action simulators

▷ Inputs: Seed-action simulator $\bar{\mathcal{S}}$
▷ Returns: Seed sequence $\bar{s}^*$ that induces path with highest return G^*
function MCTS-SA($\bar{\mathcal{S}}$)
    global $\bar{s}_{\text{end}} \leftarrow \emptyset$
    $(\bar{s}^*, G^*) \leftarrow (\emptyset, -\infty)$
    loop
        $\bar{s} \leftarrow \emptyset$
        INITIALIZE($\bar{\mathcal{S}}$)
        $G \leftarrow$ SIMULATE($\bar{\mathcal{S}}, \bar{s}$)
        if $G > G^*$
            $(\bar{s}^*, G^*) \leftarrow (\bar{s}_{\text{end}}, G)$
    return $\bar{s}^*$
function SIMULATE($\bar{\mathcal{S}}, \bar{s}$)
    if $\bar{s} \notin \mathcal{T}$
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\bar{s}\}$ ▷ Expansion, add new node
        $(N(\bar{s}), \bar{X}(\bar{s})) \leftarrow (0, \emptyset)$
        return ROLLOUT($\bar{\mathcal{S}}, \bar{s}$)
    $N(\bar{s}) \leftarrow N(\bar{s}) + 1$
    if $|\bar{X}(\bar{s})| < kN(\bar{s})^\alpha$ ▷ Progressive widening of seeds
        $\bar{x} \sim \mathbb{U}_{\text{seed}}$ ▷ New seed
        $(N(\bar{s}, \bar{x}), Q(\bar{s}, \bar{x})) \leftarrow (0, 0)$
        $\bar{X}(\bar{s}) \leftarrow \bar{X}(\bar{s}) \cup \{\bar{x}\}$
    $\bar{x} \leftarrow \arg\max_x Q(\bar{s}, x) + c\sqrt{\frac{\log N(\bar{s})}{N(\bar{s}, x)}}$ ▷ UCT selection criterion
    $\tau \leftarrow$ ISTERMINAL($\bar{\mathcal{S}}$)
    $(\rho, e, d) \leftarrow$ STEP($\bar{\mathcal{S}}, \bar{x}$) ▷ Deterministic transition
    $r \leftarrow$ REWARD(ρ, e, d, τ)
    if τ
        $\bar{s}_{\text{end}} \leftarrow \bar{s}$
        return r
    $\bar{s}' \leftarrow [\bar{s}, \bar{x}]$
    $q \leftarrow r +$ SIMULATE($\bar{\mathcal{S}}, \bar{s}'$)
    $N(\bar{s}, \bar{x}) \leftarrow N(\bar{s}, \bar{x}) + 1$
    $Q(\bar{s}, \bar{x}) \leftarrow Q(\bar{s}, \bar{x}) + \frac{q - Q(\bar{s}, \bar{x})}{N(\bar{s}, \bar{x})}$ ▷ Update estimate of Q
    return q
function ROLLOUT($\bar{\mathcal{S}}, \bar{s}$)
    $\bar{x} \sim \mathbb{U}_{\text{seed}}$ ▷ Sample seeds uniformly
    $\tau \leftarrow$ ISTERMINAL($\bar{\mathcal{S}}$)
    $(\rho, e, d) \leftarrow$ STEP($\bar{\mathcal{S}}, \bar{x}$)
    $r \leftarrow$ REWARD(ρ, e, d, τ)
    if τ
        $\bar{s}_{\text{end}} \leftarrow \bar{s}$
        return r
    $\bar{s}' \leftarrow [\bar{s}, \bar{x}]$
    return $r +$ ROLLOUT($\bar{\mathcal{S}}, \bar{s}'$)

- *Expansion.* Once we have reached a state that is not in the tree $\mathcal{T}$, we create a new node for the state and add it (line 14). The set of previously applied seeds from this state $\bar{X}(\bar{s})$ is initially empty and the number of visits to this state $N(\bar{s})$ is initialized to zero.
- *Rollout.* Starting from the state created in the expansion stage, we perform a *rollout* that repeatedly samples state transitions until the desired termination is reached (line 17). In the ROLLOUT function (line 35), state transitions are drawn from the simulator with seeds chosen according to a rollout policy, which we set to sampling from $\mathbb{U}_{\text{seed}}$.

At each step in the simulation, the reward function is evaluated and the reward is used to update estimates of the state-action values $Q(\bar{s}, \bar{x})$ (line 33). The values are used to direct the search. At the end of each simulation, the best return G^* and best path $\bar{s}^*$ are updated (lines 10–11). Simulations are run until the stopping criterion is met. The criterion is a fixed number of iterations for all our experiments except for the performance study (Section 5.7) where we used a fixed computational budget. The algorithm returns the path with the highest return represented as a sequence of pseudo-random seeds. The sequence of seeds can be used to replay the simulator to reproduce the failure event.

3.2.4 COMPUTATIONAL COMPLEXITY

Each iteration of the MCTS main loop simulates a path from initial state to terminal state. As a result, the number of calls to the simulator is linear in the number of loop iterations. The computation time thus varies as $O(N_{\text{loop}} \cdot (T_{\text{INITIALIZE}} + N_{\text{steps}} \cdot T_{\text{STEP}}))$, where N_{loop} is the number of loop iterations, $T_{\text{INITIALIZE}}$ is the computation time of INITIALIZE, N_{steps} is the average number of steps in the simulation, and T_{STEP} is the computation time of the STEP function.

4. Differential Adaptive Stress Testing

The previous section presented AST, which can be used to find the most likely failure path in a system. In some applications, it may also be valuable to identify failure scenarios not in absolute terms, but compared to another system—that is, areas where the system is *relatively* weak compared to a baseline system. In other words, we are not interested in the cases where both systems perform poorly, but rather where the system under test performs poorly but the baseline system performs well. This analysis may arise, for example, when comparing two candidate solutions to determine which one may be more desirable for release. Another use case is for regression testing where a new version of a system is compared to a previous one to see whether any new issues have been introduced.

One way to compare the behavior of two systems is to evaluate them against a common set of testing scenarios. For example, testing inputs can be randomly drawn using Monte Carlo from a stochastic model of the system's operating environment. Then, the inputs can be applied to both systems and the scenarios where failure occurs in the system under test but not in the baseline system are kept. While this method can generate failures, the undirected nature of this approach can be very inefficient due to the size and complexity of the state space, and the rarity of failure events. Moreover, the method does not find the most likely path to a failure, which is very valuable in the analysis of failure events. Another, perhaps slightly better, approach is to use a stress testing method, such as AST, to identify failure scenarios in the system under test, and then replay the scenario on the baseline system. If the scenario does not fail on the baseline system, then accept

the scenario. This method improves upon the first method in that at least the failure scenarios on the system under test are being optimized. However, the optimization process does not take into account the behavior of the baseline system.

4.1 Approach

We present a stress testing method, called differential adaptive stress testing (DAST), that extends the AST framework to the differential analysis setting while retaining its desirable properties, including scalability, efficiency, and support for black-box systems. DAST finds the most likely path to a failure event that occurs in the system under test, but not in the baseline system (Lee, Mengshoel, Saksena, Gardner, Genin, Brush, & Kochenderfer, 2018b). The key idea behind DAST is to drive two simulators in parallel and maximize the difference in their outcomes. To achieve this, we craft a new reward function that accepts the outputs of two simulators and encourages failures in one simulator but not the other. For optimization, we regard the two parallel simulators as a larger combined simulator, then we follow the AST approach to formulate stress testing as a sequential decision-making problem and optimize it using reinforcement learning. One of the core advantages of the seed-action formulation introduced previously is that it uses control of the pseudorandom seed to abstract the internal state and transition behavior from the optimization procedure. By composing two seed-action simulators into a combined simulator that is also a seed-action simulator, we can apply the MCTS-SA algorithm described in Algorithm 1 for optimization without any modification.

Figure 4 illustrates the DAST framework. We create two instances of the simulator $\bar{\mathcal{S}}^{(1)}$ and $\bar{\mathcal{S}}^{(2)}$. The instances are identical except that $\bar{\mathcal{S}}^{(1)}$ contains the system under test, while $\bar{\mathcal{S}}^{(2)}$ contains the baseline system. In particular, they contain identical models of the environment with which the test systems interface. The simulators are driven by the same pseudorandom seed input, which leads to the same sequence of disturbances being drawn in the simulator when the behaviors of the test and baseline systems match. When the behavior of the two systems diverge, the seed automatically allows different disturbances to be drawn from each simulator following their diverging states. We define a combined simulator that contains the two parallel simulators $\bar{\mathcal{S}}^{(1)}$ and $\bar{\mathcal{S}}^{(2)}$. They are both driven by the same input seed $\bar{x}$. Each simulator produces its own set of outputs, which include the transition likelihood ρ, an indicator of whether an event occurred e, and the miss distance d. These variables are combined in a reward function, where a single reward is provided to the reinforcement learner. Finally, the MCTS-SA algorithm chooses seeds to maximize the reward it receives. The superscripts on the variables ρ, e, d and τ indicate the associated simulator.

4.2 Reward Function

The reward function combines the output from the two individual simulators to produce a single reward for the MCTS-SA learner. The primary objective of the reward function is to maximize the difference in outcomes of the simulators driving the first simulator to a failure event, while keeping the second simulator away from one. The secondary objective is to maximize the path likelihoods of the two simulators to produce the most likely paths. The DAST reward function is given by Equation 9.

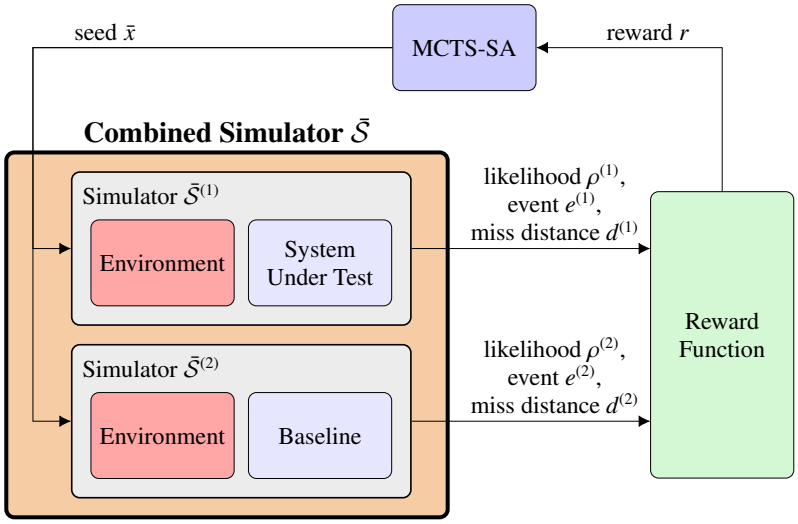

Figure 4: Differential adaptive stress testing framework. Two parallel simulators, one running the system under test and one running the baseline, are searched simultaneously. The search drives one simulator to failure while keeping the second simulator away from one.

$$\begin{aligned} R(\rho^{(1)}, e^{(1)}, d^{(1)}, \tau^{(1)}, \rho^{(2)}, e^{(2)}, d^{(2)}, \tau^{(2)}) = \quad & R_E \cdot \mathbb{1}\{\tau^{(1)} \wedge e^{(1)}\} \\ & - d^{(1)} \cdot \mathbb{1}\{\tau^{(1)} \wedge \neg e^{(1)}\} \\ & - R_E \cdot \mathbb{1}\{\tau^{(2)} \wedge e^{(2)}\} \\ & + d^{(2)} \cdot \mathbb{1}\{\tau^{(2)} \wedge \neg e^{(2)}\} \\ & + (\log \rho^{(1)} + \log \rho^{(2)}) \cdot \mathbb{1}\{\neg \tau\} \end{aligned} \tag{9}$$

The DAST reward function extends the AST reward function to the differential setting and has a similar structure:

- The first term in Equation 9 gives a non-negative reward R_E to the learner if the first simulator $\bar{\mathcal{S}}^{(1)}$ terminates in an event.
- If $\bar{\mathcal{S}}^{(1)}$ terminates and an event did not occur, then the second term penalizes the agent by giving the negative miss distance $-d^{(1)}$.
- The third term gives $-R_E$ if $\bar{\mathcal{S}}^{(2)}$ terminates in an event.
- The fourth term gives $d^{(2)}$ if $\bar{\mathcal{S}}^{(2)}$ terminates and an event did not occur.
- To maximize the likelihoods of the paths, the fifth term gives the sum of the log transition likelihoods of both simulators.

The third and fourth terms are the negations of the first and second terms, respectively, applied to the second simulator $\bar{\mathcal{S}}^{(2)}$. The terminal state of the simulators are treated as *absorbing*. That is, once a simulator enters a terminal state and collects the terminal reward, it stays there for all subsequent transitions and collects zero reward for these transitions. The Boolean variables $\tau^{(1)}$ and $\tau^{(2)}$ indicate whether $\bar{\mathcal{S}}^{(1)}$ and $\bar{\mathcal{S}}^{(2)}$ have terminated, respectively. The combined simulator terminates when both simulators have terminated, i.e., $\tau = \textsc{IsTerminal}(\bar{\mathcal{S}}) = \tau^{(1)} \wedge \tau^{(2)}$, where τ (without superscript) indicates whether the combined simulator has terminated.

Due to the seed-action abstraction, we can optimize the reward function using the MCTS-SA algorithm described in Algorithm 1 as used in AST. Because the algorithm is based on scalar rewards and pseudorandom seeds, the internal details of the simulator are abstracted from the reinforcement learner. As a result, no modifications to the algorithm are necessary.

5. Aircraft Collision Avoidance Application

Aircraft collision avoidance systems are mandated on all large transport and cargo aircraft in the United States and other countries around the world to help prevent mid-air collisions. Their operation has played a crucial role in the exceptional level of safety in the national airspace (Kuchar & Drumm, 2007). The Traffic Alert and Collision Avoidance System (TCAS) is currently deployed in the United States and many countries around the world. TCAS has been very successful at protecting aircraft from mid-air collisions. However, studies have revealed fundamental limitations in TCAS that prevent it from operating effectively in the next-generation airspace where the number of aircraft is expected to increase significantly (Kuchar & Drumm, 2007). To address the growing needs of the national airspace, the Federal Aviation Administration (FAA) has decided to create a new aircraft collision avoidance system. The next-generation Airborne Collision Avoidance System (ACAS X) was created promising a number of improvements over TCAS including a reduction in collision risk while simultaneously reducing the number of unnecessary alerts (Kochenderfer et al., 2012). This research work was performed on prototypes of ACAS X while the system was still being developed and tested. On September 20$^{\text{th}}$, 2018, the RTCA[1] accepted ACAS X to replace TCAS as the next standard for airborne collision avoidance and the system is expected to be widely deployed in the near future.

ACAS X has been shown to be much more operationally suitable than TCAS (Federal Aviation Administration, 2018). Table 1 is a comparison of ACAS X and TCAS on several key operational metrics evaluated on a number of simulated aircraft encounter datasets. The data is excerpted from the results of many studies performed at MIT Lincoln Laboratory (MIT-LL) and presented at RTCA (Federal Aviation Administration, 2018). ACAS X shows significant improvements in overall safety and alert rates compared to TCAS. The primary metric of safety is the probability of a near mid-air collision (NMAC), P_{NMAC}. An NMAC is defined as two aircraft coming closer than 500 feet horizontally and 100 feet vertically. On large encounter datasets, ACAS X is shown to reduce the probability of NMAC by 17% to 54% compared to TCAS. Alert metrics show significant improvements over TCAS as well, including the aggregate alert rate, which is the number of times the collision avoidance system alerts (counted by aircraft); and the 500 feet corrective alert rate, which is the number of alerts issued in encounters where the aircraft are flying level and vertically separated by exactly 500 feet. ACAS X also significantly improves metrics on generally undesirable scenarios such as the altitude crossing rate, which is the number of encounters where two aircraft cross in altitude; and the reversal rate, which is the number of times the collision avoidance system first advises the pilot to maneuver in one direction, then later advises the pilot to maneuver in the opposite direction.

Studies have shown that the risk of NMAC is extremely small and moreover ACAS X reduces the risk even further over TCAS overall. However, the risk of NMAC cannot be completely eliminated due to factors such as surveillance noise, pilot response delay, and the need for an acceptable alert rate (Kochenderfer et al., 2012). Because NMACs are such important safety events, it is im-

1. RTCA was formerly known as the Radio Technical Commission for Aeronautics, but is now known simply as RTCA.

Table 1: A comparison of ACAS X and TCAS operational metrics on various encounter datasets. The data is excerpted from (Federal Aviation Administration, 2018). ACAS X significantly improves overall safety, alert rates, and other operational metrics compared to TCAS.

Metric	Dataset	Number of encounters	TCAS v7.1	ACAS Xa 0.10.3	Improvement over TCAS
Safety (P_{NMAC})	LLCEM	5,956,128	$2.179 \cdot 10^{-4}$	$1.744 \cdot 10^{-4}$	19.57%
Safety (P_{NMAC})	SAVAL	75,173,906	$4.361 \cdot 10^{-4}$	$3.627 \cdot 10^{-4}$	16.82%
Safety (P_{NMAC})	SA01	100,000	$4.106 \cdot 10^{-2}$	$1.873 \cdot 10^{-2}$	54.37%
Alert Rate (by aircraft)	TRAMS	293,101	252,656	121,267	52.00%
500' Corrective Alert Rate	TRAMS	175,184	14,912	9,919	33.48%
Altitude Crossing Rate	TRAMS	293,101	3,196	1,582	50.5%
Reversal Rate	TRAMS	293,101	1,029	556	45.97%

portant to study and understand the rare circumstances under which they can still occur even if they are extremely unlikely. Understanding the nature of the residual NMAC risk has been important for certification and informing the iterative development of the system. This article uses AST and DAST to find and analyze the rare corner cases where an NMAC can still occur.

5.1 ACAS X Operation

There are several versions of ACAS X under development. This article considers a development (and not final) version of ACAS Xa, which uses active surveillance and is designed to be a direct replacement to TCAS. Despite the internal logics of ACAS X and TCAS being derived completely differently, the input and output interfaces of these two systems are identical. As a result, the following description of aircraft collision avoidance systems applies to both ACAS X and TCAS.

Airborne collision avoidance systems monitor the airspace around an aircraft and issue alerts to the pilot if a conflict with another aircraft is detected. These alerts, called resolution advisories (RAs), instruct the pilot to maneuver the aircraft to a certain target vertical velocity and maintain it. The advisories are typically issued when the aircraft are within approximately 20–40 seconds to a potential collision. Table 2 lists the possible primary RAs. We use $\dot{z}_{\text{own}}$ to denote the current vertical velocity of own aircraft.

Table 2: Primary ACAS X advisories

Abbreviation	Description	Rate to Maintain (ft/min)
COC	clear of conflict	N/A
DND	do not descend	0
DNC	do not climb	0
DNDx	do not descend at greater than x ft/min	$\max(-x, \dot{z}_{\text{own}})$
DNCx	do not climb at greater than x ft/min	$\min(x, \dot{z}_{\text{own}})$
MAINTAIN	maintain current rate	$\dot{z}_{\text{own}}$
DS1500	descend at 1,500 ft/min	−1500
CL1500	climb at 1,500 ft/min	+1500
DS2500	descend at 2,500 ft/min	−2500
CL2500	climb at 2,500 ft/min	+2500

The COC advisory stands for "clear of conflict" and is equivalent to no advisory. The pilot is free to choose how to control the aircraft. The DND and DNC advisories stand for "do not descend" and "do not climb", respectively. They restrict the pilot from flying in a certain direction. The DNDx and DNCx advisories extend DND and DNC, respectively, with a maximum vertical rate x. They restrict the pilot from descending or climbing at a vertical rate greater than x feet per minute. The MAINTAIN advisory is preventative and instructs the pilot to maintain the current vertical rate of the aircraft. The advisories DS1500 and CL1500 instruct the pilot to descend or climb at 1,500 feet per minute. The pilot is expected to maneuver the aircraft at $\frac{1}{4}g$ acceleration until the target vertical rate is reached then maintain that vertical rate. The DS2500 and CL2500 advisories instruct the pilot to descend or climb at an increased rate of 2,500 feet per minute. These advisories are strengthened advisories and expect a stronger response from the pilot. For these strengthened RAs, the pilot is expected to maneuver at $\frac{1}{3}g$ acceleration until the target vertical rate is reached then maintain that vertical rate. Strengthened RAs must follow a weaker RA of the same vertical direction. They cannot be issued directly. For example, a CL1500 advisory must precede a CL2500 advisory. Advisories issued by collision avoidance systems on different aircraft are not completely independent. When an RA is issued, a coordination message is broadcasted to other nearby equipped aircraft to prevent other collision avoidance systems from accidentally issuing an RA in the same vertical direction.

5.2 Experimental Setup

We construct a seed-action simulator modeling an aircraft mid-air encounter. We focus on encounters where all aircraft are equipped with identical collision avoidance systems. The overall architecture of the simulator for two aircraft is shown in Figure 5. Simulation models capture the key aspects of the encounter, including the initial state, sensors, collision avoidance system, pilot response, and aircraft dynamics.

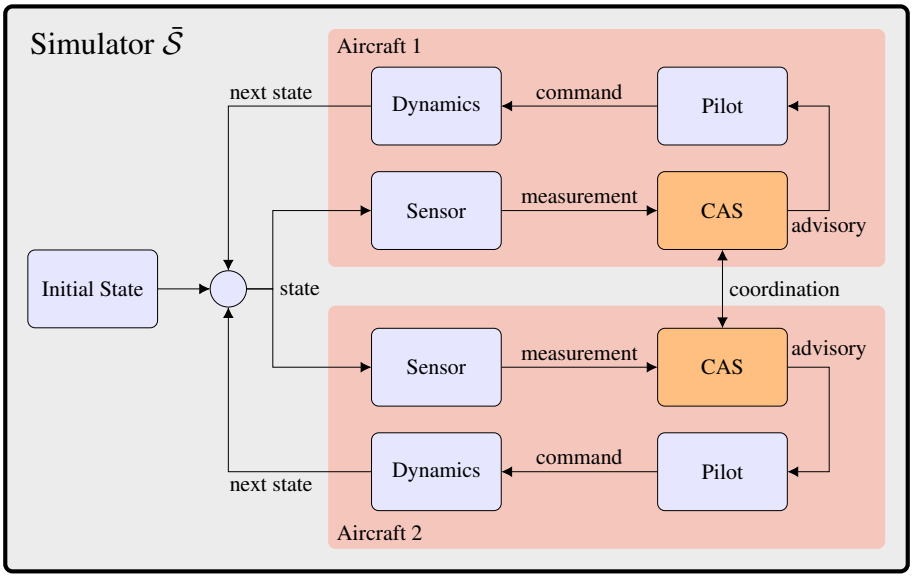

Figure 5: System diagram for pairwise encounters. Two aircraft simulation loops are used. Simulation models sensors, collision avoidance system, pilot response, aircraft dynamics, and interactions.

5.2.1 SIMULATION MODEL

Initial State. The initial state of the encounter includes initial positions, velocities, and headings of the aircraft. The initial state is drawn from a distribution that gives realistic initial configurations of aircraft that are likely to lead to NMAC. Once the initial state is sampled, it is fixed for the duration of the search. In our experiments, pairwise (two-aircraft) encounters are initialized using the Lincoln Laboratory Correlated Aircraft Encounter Model (LLCEM) (Kochenderfer, Espindle, Kuchar, & Griffith, 2008; Kochenderfer, Edwards, Espindle, Kuchar, & Griffith, 2010). LLCEM is a statistical model learned from a large body of radar data of the entire national airspace. We follow the encounter generation procedure described in the paper (Kochenderfer et al., 2008). Multi-threat (three-aircraft) encounters use the *Star model*, which initializes aircraft on a circle heading towards the origin spaced apart at equal angles. Initial airspeed, altitude, and vertical rate are sampled from a uniform distribution over a prespecified range. The horizontal distance from the origin is set such that without intervention, the aircraft intersect at approximately 40 seconds into the encounter.

Sensor Model. The sensor model captures how the collision avoidance system perceives the world. We assume active, beacon-based radar capability with no noise. For own aircraft, the sensor measures the vertical rate, barometric altitude, heading, and height above ground. For each intruding aircraft, the sensor measures slant range (relative distance to intruder), bearing (relative horizontal angle to intruder), and relative altitude.

Collision Avoidance System. The collision avoidance system is the system under test. We use a prototype of ACAS X in the form of a binary library obtained from the FAA. The binary has a minimal interface that allows initializing and stepping the state forward in time. The system maintains internal state, but does not expose it. The primary output of the ACAS X system is the RA. ACAS X has a coordination mechanism to ensure that issued RAs from different aircraft are compatible with one another, i.e., that two aircraft are not instructed to maneuver in the same vertical direction. The messages are communicated to all nearby aircraft through coordination messages. Our differential studies compare ACAS X to TCAS. Our implementation of TCAS is also a binary library obtained from the FAA. Both binaries have identical input and output interfaces making them interchangeable in the simulator.

Pilot Model. The pilot model consists of a model for the pilot's intent and a model for how the pilot responds to an RA. The pilot's intent is how the pilot would fly the aircraft if there are no RAs. To model intended commands, we use the pilot command model in LLCEM, which gives a realistic stochastic model of aircraft commands in the airspace (Kochenderfer et al., 2008). The pilot response model defines how pilots respond to an issued RA. Pilots are assumed to respond deterministically to an RA with perfect compliance after a fixed delay (International Civil Aviation Organization, 2007). Pilots respond to initial RAs with a five-second delay and subsequent RAs (i.e., strengthenings and reversals) with a three-second delay. During the initial delay period, the pilot continues to fly the intended trajectory. During response delays from subsequent RAs, the pilot continues responding to the previous RA. Multiple RAs issued successively are queued so that both their order and timing are preserved. In the case where a subsequent RA is issued within 2 seconds or less of an initial RA, the timing of the subsequent RA is used and the initial RA is skipped. The pilot command includes commanded airspeed acceleration, commanded vertical rate, and commanded turn rate.

Aircraft Dynamics Model. The aircraft dynamics model determines how the state of the aircraft propagates with time. The aircraft state includes the airspeed, position north, position east, altitude, roll angle, pitch angle, and heading angle. The aircraft state is propagated forward at 1 Hz using forward Euler integration.

In our experiments, the commands of the pilots when not responding to an advisory are stochastic and are being optimized by the algorithm. When no RA is active, the pilot commands follow the stochastic dynamic model of LLCEM. When an RA is active, the vertical component of the pilot's command follows the pilot response model, while the other components follow LLCEM. Other simulation components are deterministic in our experiments. However, in the future, we may consider stochastic models for sensors, aircraft dynamics, and pilot response.

5.2.2 Reward Function

We use the reward function defined in Equation 7 for optimization. The event space E is defined to be an NMAC, which occurs when two aircraft are closer than 100 feet vertically and 500 feet horizontally. We define the miss distance d to be the distance of closest approach, which is the Euclidean distance of the aircraft at their closest point in the encounter. The distance of closest approach is a good metric because it is monotonically decreasing as trajectories get closer and reach a minimum at an NMAC.

5.3 Stress Testing Single-Threat Encounters

We apply AST to analyze NMACs in encounters involving two aircraft. We searched 100 encounters initialized using samples from LLCEM. The configuration is shown in Table 3. Of the 100 encounters searched, 18 encounters contained an NMAC, yielding an empirical find rate of 18%. When the optimization algorithm completes, it returns the path with the highest reward regardless of whether the path contains an NMAC. When the returned path does not contain an NMAC, it is uncertain whether an NMAC exists and the algorithm was unable to find it, or whether no NMAC is reachable given the initial state of the encounter. We manually cluster the NMAC encounters and present our findings.

Table 3: Single-threat configuration

Simulation	
number of aircraft	2
initialization	LLCEM
sensors	active, beacon-based, noiseless
collision avoidance system	ACAS Xa Run 13 libcas 0.8.5
pilot response model	deterministic 5s–3s
MCTS	
maximum steps	50
iterations	2000
exploration constant	100.0
k	0.5
α	0.85

Crossing Time. We observed a number of NMACs resulting from well-timed vertical maneuvers. In particular, several encounters included aircraft crossing in altitude during the pilot response delay to an initial RA. Figure 6 shows one such encounter that eventually ends in an NMAC at 36 seconds into the encounter. The probability density of the encounter evaluated under LLCEM is $5.3 \cdot 10^{-18}$. This measure can be used as an unnormalized measure of likelihood of occurrence. In this encounter, the aircraft cross in altitude during pilot 1's response delay. The crossing leads to aircraft 1 starting the climb from below aircraft 2. The subsequent reversal later in the encounter is unable to resolve the conflict due to the pilot response delay.

NMAC encounter plots contain horizontal tracks (top-down view) and vertical profile (altitude versus time). Aircraft numbers are indicated at the start and end of each trajectory. RA codes are labeled at the time of occurrence. Marker colors indicate the RA issued: blue for COC, orange for CL1500, red for CL2500, cyan for DS1500, purple for DS2500, and gray for DNC/DND/MAINTAIN/MULTITHREAT. We prepend the aircraft number followed by a slash to each RA code for readability since the aircraft may cross multiple times during the encounter. Symbols inside the markers indicate the state of the pilot response: no symbol for not responding, dash for responding to previous RA, and asterisk for responding to current RA.

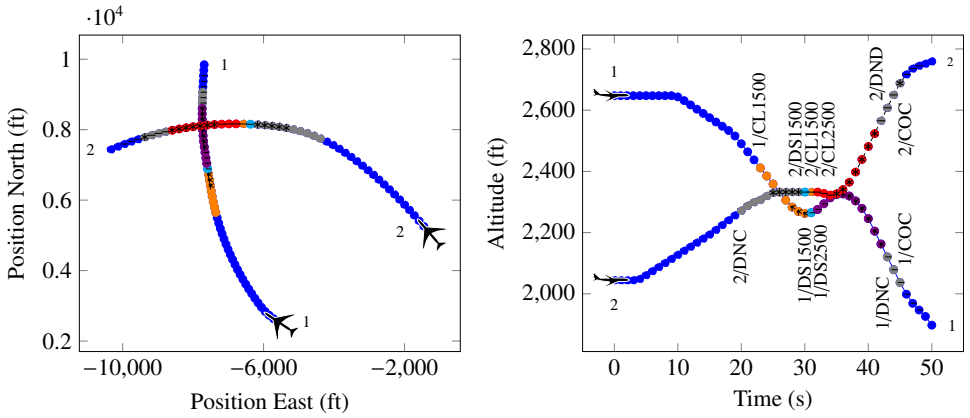

Figure 6: NMAC encounter where the aircraft cross in altitude during pilot delay. The aircraft cross in altitude after aircraft 1's RA has been issued, but before they start to respond. The aircraft starts to climb from below the intruder and the encounter ends in an NMAC at 36 seconds.

Maneuvering Against RA. Our analysis revealed a number of NMAC encounters where the pilot initially maneuvers against the issued RA before complying. That is, after the pilot receives the RA, they maneuver the aircraft in the opposite direction of what is instructed by the RA for the duration of the pilot response delay before subsequently complying and reversing direction. Pilots do not normally maneuver in this manner and so this scenario represents a very operationally rare case. Even so, ACAS X does seem to be able to resolve the majority of these initially disobeying cases. In most cases, the maneuvering must be very aggressive against the RA to result in an NMAC.

High Turn and Vertical Rates. Turns at high rates quickly shorten the time to closest approach. ACAS X does not have full state information about its intruder and must estimate it by tracking relative distance, relative angle, and the intruder altitude. Figure 7 shows an example of an encounter that has similar crossing behavior as Figure 6 but exacerbated by the high turn rate of aircraft 2

(approximately 1.5 times the standard turn rate). In this scenario, the aircraft become almost head-on at the time of closest approach and a reversal is not attempted. An NMAC with a probability density of $6.5 \cdot 10^{-17}$ occurs at 48 seconds into the encounter. Aircraft 1 is also coincidently descending at a high vertical rate. The combination makes this encounter operationally very unlikely.

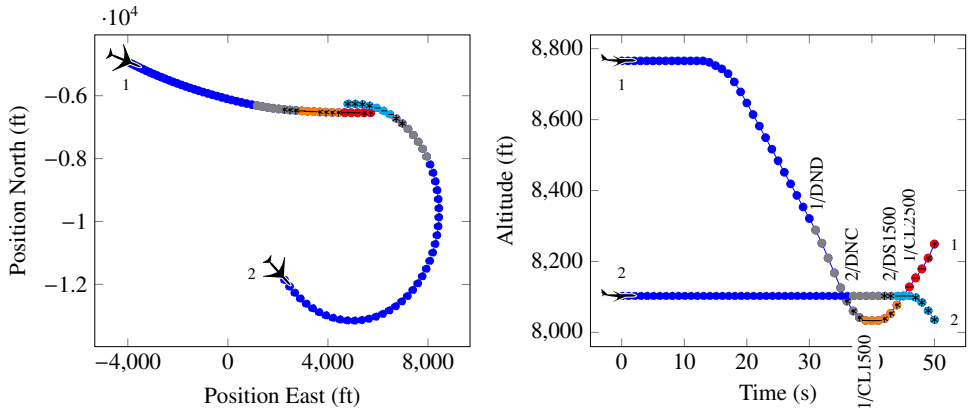

Figure 7: NMAC encounter where one aircraft is turning at high rate while the other is descending at high vertical rate. The encounter ends in an NMAC, but the combination is operationally very rare.

Sudden Aggressive Maneuvering. Sudden maneuvers can lead to NMACs if they are sufficiently aggressive. In particular, we observed some NMAC encounters where two aircraft are approaching one another separated in altitude and flying level, then one aircraft suddenly accelerates vertically towards the other aircraft as they are about to pass. Under these circumstances, given the pilot response delays and dynamic limits of the aircraft, there is insufficient time and distance remaining for the collision avoidance system to resolve the conflict. Pilots do not normally fly so aggressively in operation, so this case is extremely unlikely. In fact, they are even more rare than our model predicts. ACAS X issues traffic alerts (TAs) to alert pilots to nearby traffic, so that pilots are made aware of intruding aircraft well before the initial RA. These advance warnings increase the pilot's situational awareness and reduce blunders like these. Our simulator does not model the effect of such TAs, however. As a designer of ACAS X, one course of action would be to tune ACAS X to intervene preemptively in such scenarios. While this reduces the risk of possible sudden behavior, it also increases the alert rate of the system. Ultimately, the ACAS X designer must assess the probabilities of various scenarios and find the delicate balance between risk of collision and issuing too many advisories. Since these scenarios are extremely rare, we must trade off accordingly.

Combined Factors. In our experiments, it is rare for an NMAC to be attributable to a single cause. More commonly, a combination of factors contribute to the NMAC. Figure 6 shows an example of an encounter where multiple factors contribute to the NMAC. In Figure 6, crossing time played a crucial role in the NMAC. However, there are a number of other factors that are important as well. The horizontal behavior where they are turning into each other is significant as it reduces the time to closest approach. The two vertical maneuvers of aircraft 1 before receiving an RA are also important. Similar observations can be made for many of the other NMAC encounters found.

5.4 Stress Testing Multi-Threat Encounters

We applied AST to analyze NMACs in three-aircraft encounters. We searched 100 encounters initialized using samples from the Star model. The configuration is shown in Table 4. We found 25 NMACs out of 100 encounters searched, yielding a find rate of 25%.

Table 4: Multi-threat configuration

Simulation	
number of aircraft	3
initialization	Star model
sensors	active, beacon-based, noiseless
collision avoidance system	ACAS Xa Run 13 libcas 0.8.5
pilot response model	deterministic 5s–3s
MCTS	
maximum steps	50
iterations	1000
exploration constant	100.0
k	0.5
α	0.85

Limited Maneuverable Space. In general, multi-threat encounters are more challenging to resolve than pairwise encounters because there is less open space for the aircraft to maneuver. Figure 8 shows an example of an NMAC encounter where aircraft 1 (the aircraft in the middle altitude between 10 and 36 seconds) needs to simultaneously avoid an aircraft below and a vertically closing aircraft from above. An NMAC with a probability density of $1.0 \cdot 10^{-16}$ occurs at 39 seconds into the encounter. Aircraft 2's downward maneuver greatly reduces the maneuverable airspace of aircraft 1. These encounters are undoubtedly extremely challenging for a collision avoidance system and it is unclear whether any satisfactory resolution exists. Nevertheless, we gain insight by observing how the collision avoidance system behaves under such extremely rare circumstances.

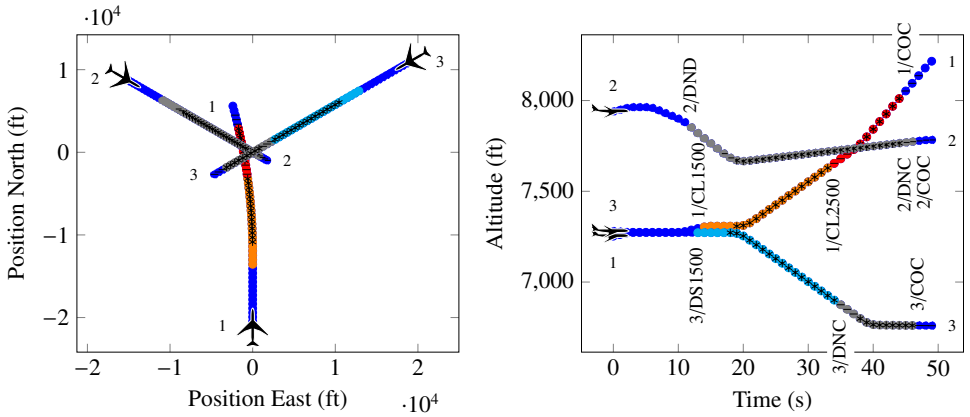

Figure 8: NMAC encounter where the aircraft have limited maneuverable airspace. Aircraft 1 must avoid aircraft 2 during maneuver away from aircraft 3. The encounter ends in NMAC at 39 seconds.

Pairwise Coordination in Multi-Threat. Our algorithm discovered a number of NMAC encounters where all aircraft are issued a MULTITHREAT (MTE) RA and asked to follow an identical climb rate. Complying with the RA results in the aircraft closing horizontally without gaining vertical separation. Figure 9 shows an example of such an encounter where an NMAC occurs with probability density $5.8 \cdot 10^{-7}$ at 38 seconds into the encounter. In discussing these results with the ACAS X development team, we learned that this behavior is a known issue that can arise when performing multi-aircraft coordination using a pairwise coordination mechanism. The pairwise coordination messages in essence determine which aircraft will climb and which will descend in an encounter. Since coordination messaging occurs pairwise, under rare circumstances it is possible for each aircraft to receive conflicting coordination messages from the other aircraft in the scenario. In nominal encounters, the aircraft that receives conflicting coordination messages from two aircraft remains level and lets the other aircraft climb or descend around it. However, in these encounters, all three aircraft receive conflicting coordination messages. Although very rare, this is an important case that is being addressed by both TCAS and ACAS X development teams.

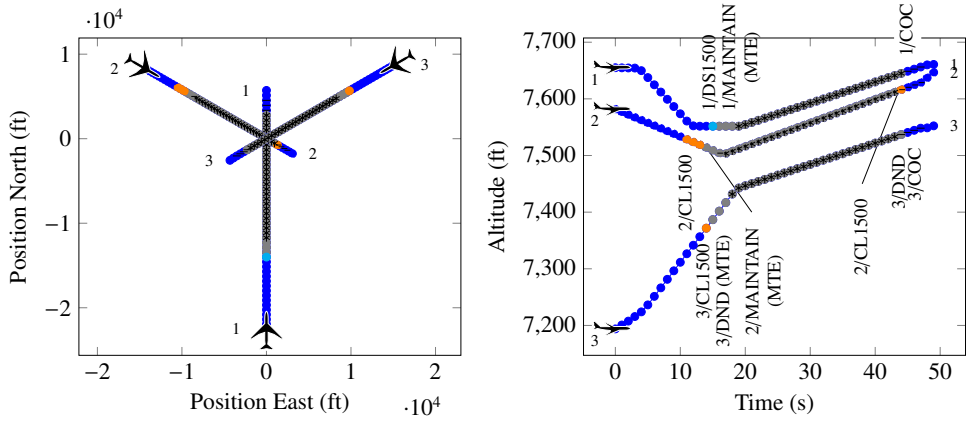

Figure 9: NMAC encounter where all aircraft receive pairwise conflicting coordination messages and do not maneuver. The encounter ends in an NMAC at 38 seconds. This is a very rare but important case that the ACAS X team is addressing.

Pairwise Phenomena. Phenomena that appear in pairwise encounters also appear in multi-threat encounters. The presence of the third aircraft typically exacerbates the encounter. In our multi-threat analysis, we noted similar phenomena related to crossing time, maneuvering against RA, and sudden aggressive maneuvering as discussed previously. We did not observe any cases related to high turn rates in the multi-threat setting due to our use of the Star model.

5.5 Stress Testing ACAS X Relative to TCAS Baseline

We apply DAST to perform differential stress testing of ACAS X relative to a TCAS baseline. We seek to find NMAC encounters that occur in ACAS X but not in TCAS. The search is extremely difficult. Not only are both ACAS X and TCAS systems extremely safe, which means that NMACs are extremely rare, but also ACAS X is a much safer system than TCAS overall, which makes cases where ACAS X has NMAC but TCAS does not extremely rare. We seek to find those extremely rare corner cases.

We searched 2700 pairwise encounters initialized with samples from LLCEM. The configuration is shown in Table 5. The top 10 highest reward paths were returned for each encounter initialization producing a total of 27,000 paths. Of these paths, a total of 28 contained NMACs, which originated from 10 encounter initializations. We analyzed the scenarios and confirmed that all were operationally rare scenarios. We present examples of NMAC found by the algorithm and discuss their properties.

Table 5: Differential stress testing configuration

Simulation	
number of simulators	2
number of aircraft per simulator	2
initialization	LLCEM
sensors	active, beacon-based, noiseless
collision avoidance system (test)	ACAS Xa Run 15 libcas 0.10.3
collision avoidance system (baseline)	TCAS II v7.1
pilot response model	deterministic 5s–3s
MCTS	
maximum steps	50
iterations	3000
exploration constant	100.0
k	0.5
α	0.85

ACAS X Issues RA, But TCAS Does Not. We found some NMAC cases where ACAS X issued an RA but TCAS did not. An example is shown in Figure 10 where an NMAC occurs at 40 seconds in the ACAS X simulation but no NMAC occurs in the TCAS simulation. The encounter occurs with a probability density of $1.7 \cdot 10^{-19}$. No RA was issued in the TCAS simulation. In the example, both aircraft are traveling at a high absolute vertical rate exceeding 70 feet per second toward each other. Both aircraft receive an RA to level-off but the aircraft cannot respond in time due to the high vertical rates and the pilot response delay. The aircraft proceed to cross in altitude. After crossing, ACAS X increases the strength of the advisory in the same direction as the previous RA. Since the aircraft have crossed in altitude, responding to the RA results in a loss of vertical separation and an NMAC.

High vertical rates are known to make conflict resolution more difficult, especially for vertical-only collision avoidance systems like TCAS and this version of ACAS X. Aircraft with high vertical rates take longer to reverse direction vertically and more vertical distance is traveled during the pilot's response delay. As a result, advisories take longer to take full effect. Moreover, in cases

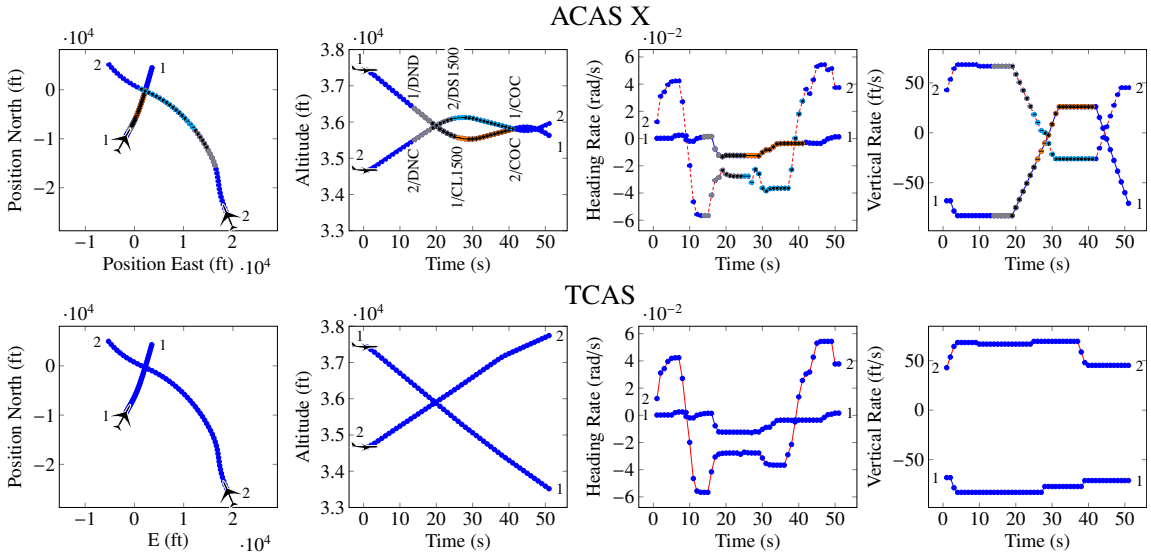

Figure 10: An encounter where ACAS X issues an RA but TCAS does not. The aircraft are initially traveling at high vertical rates and aircraft 2 also performs some horizontal maneuvering. An NMAC occurs in the ACAS X simulation at 40 seconds.

where both aircraft are maneuvering in the same vertical direction, an aircraft may lose the ability to "outrun" the other aircraft. For example, even if a maximal climb advisory is issued, it may not be sufficient for the aircraft to stay above a second aircraft climbing at an even higher rate. Another interesting feature of this encounter is the horizontal behavior. Aircraft 2 is initially turning away from the other aircraft before turning towards it at 8 seconds into the encounter. The initial RA is issued shortly after that maneuver. Large rapid changes in turn rate around the time of an RA can make it difficult for a collision avoidance system to accurately estimate the time to horizontal intersection.

Overall, ACAS X is much safer and more operationally suitable than TCAS. However, there are some trade-offs between the two systems as highlighted by our methods. ACAS X's late altering characteristic, which reduces the number of unnecessary alerts, can sometimes hurt encounters with higher vertical rates, such as seen in this example. In deciding the trade-off, a designer must weigh the relative likelihood of these encounters versus the effect on other more frequently observed trajectories.

Simultaneous Horizontal and Vertical Maneuvering. Many of the NMAC encounters found involve an aircraft turning while simultaneously climbing or descending very rapidly. Figure 11 shows an example where an NMAC occurs at 45 seconds in the ACAS X simulation but no NMAC occurs in the TCAS simulation. The encounter occurs with probability density $2.7 \cdot 10^{-4}$. In this example, aircraft 1 is flying generally straight and level while aircraft 2 is simultaneously turning and climbing at a vertical rate exceeding 80 feet per second. Aircraft 2 receives an RA to level-off but is unable to maneuver in time before losing vertical separation resulting in an NMAC. In this example, TCAS is able to resolve the conflict by issuing RAs to both aircraft earlier in the encounter than ACAS X.

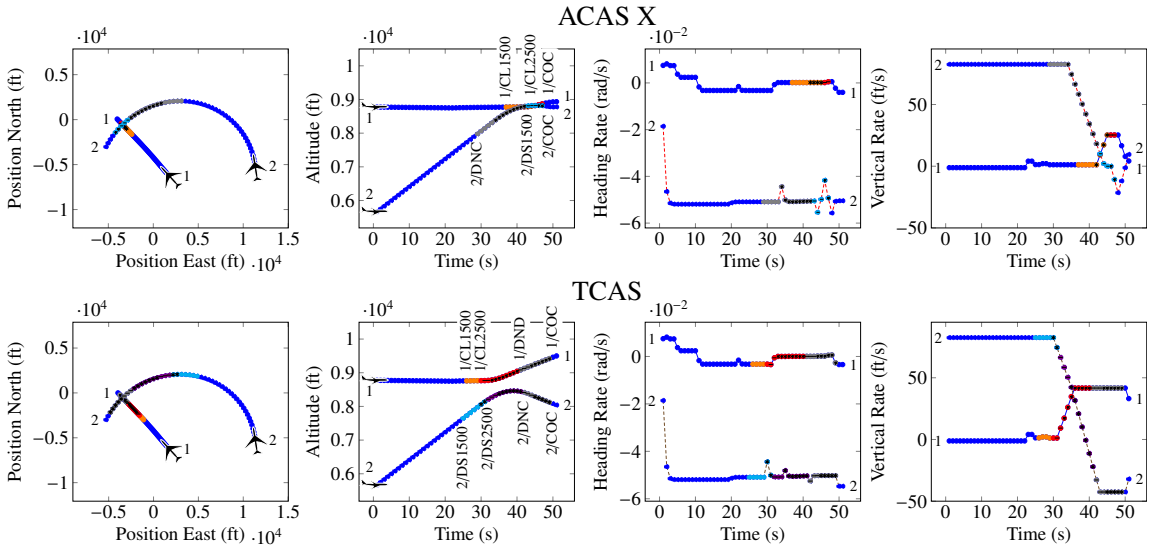

Figure 11: An NMAC encounter where one aircraft turns while simultaneously climbing very rapidly. An NMAC occurs in the ACAS X simulation at 45 seconds whereas no NMAC occurs in the TCAS simulation.

Horizontally, aircraft 2's turn quickly shortens the time to closest approach between the two aircraft. Vertically, aircraft 2 receives a level-off advisory but the high climb rate and pilot response delay limit how quickly the aircraft can be brought to compliance. Scenarios that involve turning and simultaneously climbing or descending at high rate are operationally very rare.

Horizontal Maneuvering. In some very rare cases, NMACs can also result from horizontal maneuvering alone. An example is shown in Figure 12 where an NMAC occurs at 40 seconds in the ACAS X simulation but no NMAC occurs in the TCAS simulation. The encounter occurs with probability density $4.3 \cdot 10^{-11}$. The aircraft are initially headed away from each other but they are also turning towards each other. At 25 seconds into the encounter, the aircraft turn more tightly towards each other, rapidly reducing the time to closest approach. Crossing advisories are issued to the aircraft 9 seconds prior to NMAC. However, there is not enough time remaining to cross safely and an NMAC occurs. In this example, TCAS is able to resolve the conflict by issuing RAs to both aircraft earlier in the encounter. However, it is unclear how the aircraft got to their initial positions in the encounter and whether this initial position is generally reachable.

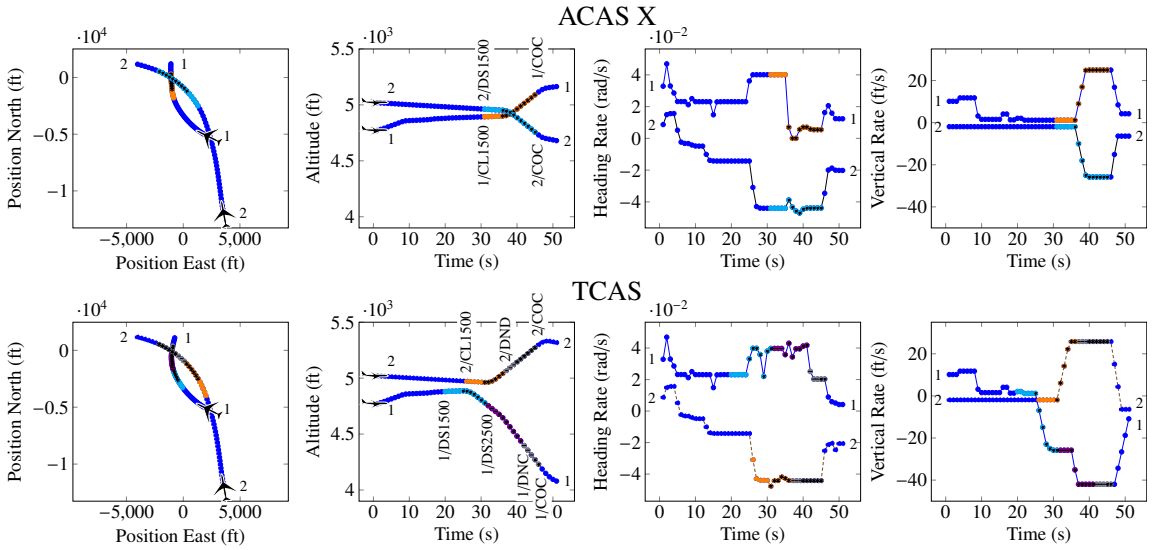

Figure 12: An NMAC encounter where the aircraft maneuver horizontally only. The aircraft start in an unlikely initial configuration. An NMAC occurs at 40 seconds in the ACAS X simulation but not in the TCAS simulation.

5.6 Stress Testing TCAS Relative to ACAS X Baseline

For comparison, we perform DAST experiments where we consider TCAS as the system under test and ACAS X as the baseline, which is the inverse of the experiment in Section 5.5. We searched 2700 pairwise encounters with samples from LLCEM. We use the same configuration as in Table 5 with the exception that the test and baseline systems are reversed. The top 10 highest reward paths are returned for each encounter initialization producing a total of 27,000 paths as before. Of these paths, a total of 39 contained NMACs, which represents an increase of 39.3%. It is reassuring that this result, where DAST found 39.3% more NMACs with TCAS compared to ACAS X, is qualitatively similar to the results from other studies which use direct Monte Carlo sampling and also show a safety benefit of ACAS X compared to TCAS (Holland et al., 2013). The NMACs also originate from a broader set of encounter initializations as well (22 versus 10 previously). We present examples of NMAC found by the algorithm and discuss their properties.

Crossing RA Followed by a Reversal. An example is shown in Figure 13 where an NMAC occurs at 41 seconds in the TCAS simulation but no NMAC occurs in the ACAS X simulation. The encounter occurs with a probability density of $6.6 \cdot 10^{-12}$. In the encounter, the aircraft converge vertically while they approach in a perpendicular configuration horizontally. TCAS issues crossing initial advisories to the aircraft, but then reverses the RAs shortly before the aircraft cross in altitude. The reversal leads to the aircraft reversing vertical direction after they have crossed in altitude, reducing their vertical separation, and eventually resulting in an NMAC. In contrast, ACAS X issues preventative advisories to the aircraft early in the encounter and keeps the aircraft vertically separated throughout the encounter.

The crossing TCAS advisory combined with a reversal shortly afterwards suggests that the encounter may be operating near a decision boundary and TCAS may be having trouble deciding whether the aircraft should cross in altitude. In this case, it appears the initial crossing RA may have successfully resolved the conflict had it been maintained, and it is the reversal that complicated the resolution. The lateness of the TCAS initial RA likely played a major role in the NMAC, leaving a very difficult decision for the direction selection of the RA. ACAS X gives a desirable outcome in this case deciding early in the encounter that the aircraft should not cross in altitude.

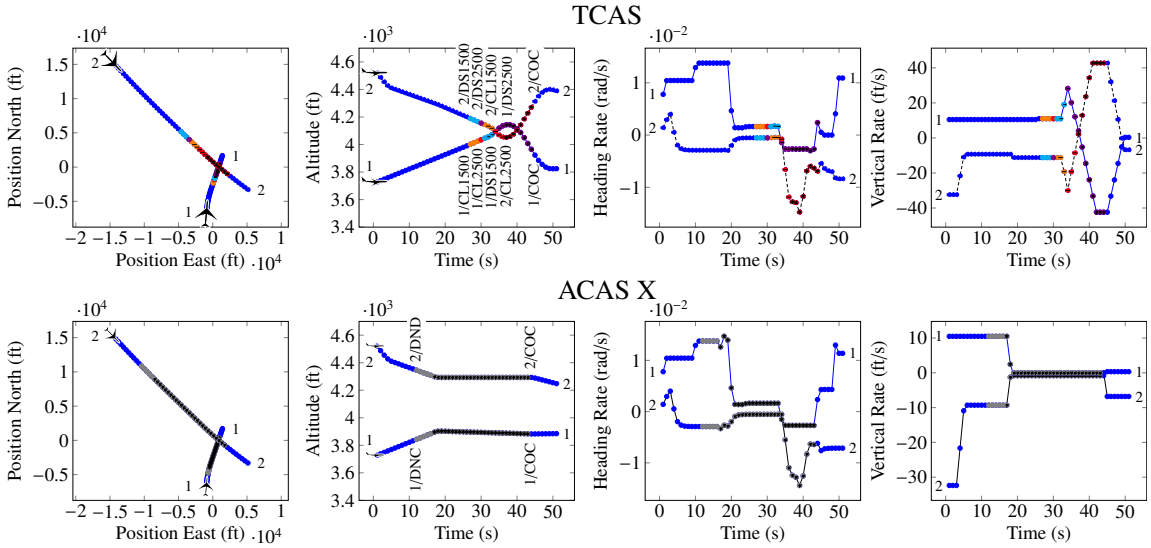

Figure 13: An NMAC encounter where a crossing RA is subsequently reversed prior to altitude crossing. An NMAC occurs at 41 seconds in the TCAS simulation but not in the ACAS X simulation.

Double Horizontal Crossing. An example is shown in Figure 14 where an NMAC occurs at 50 seconds in the TCAS simulation, but no NMAC occurs in the ACAS X simulation. The encounter occurs with a probability density of $2.2 \cdot 10^{-14}$. In the encounter, the aircraft turn as they converge both horizontally and vertically, eventually resulting in a horizontal double crossing. TCAS issues preventative DND2000 and DNC1000 advisories to the aircraft, respectively, predicting that they will cross safely, then further clears the advisories after the first horizontal crossing. However, after the first crossing, the aircraft continue to turn toward each other into a second horizontal crossing. TCAS issues a second sequence of advisories, but it is too late and an NMAC occurs. Sequences of RAs separated by COC are called *split advisories* and are undesirable because they may cause confusion or doubt in the pilots. Another contributor to the difficulty of the encounter is aircraft 2 increasing its turn rate mid-encounter, which significantly reduces time to collision. The ACAS X system handles the encounter much more desirably, choosing a DND advisory to maintain vertical separation as the aircraft cross horizontally.

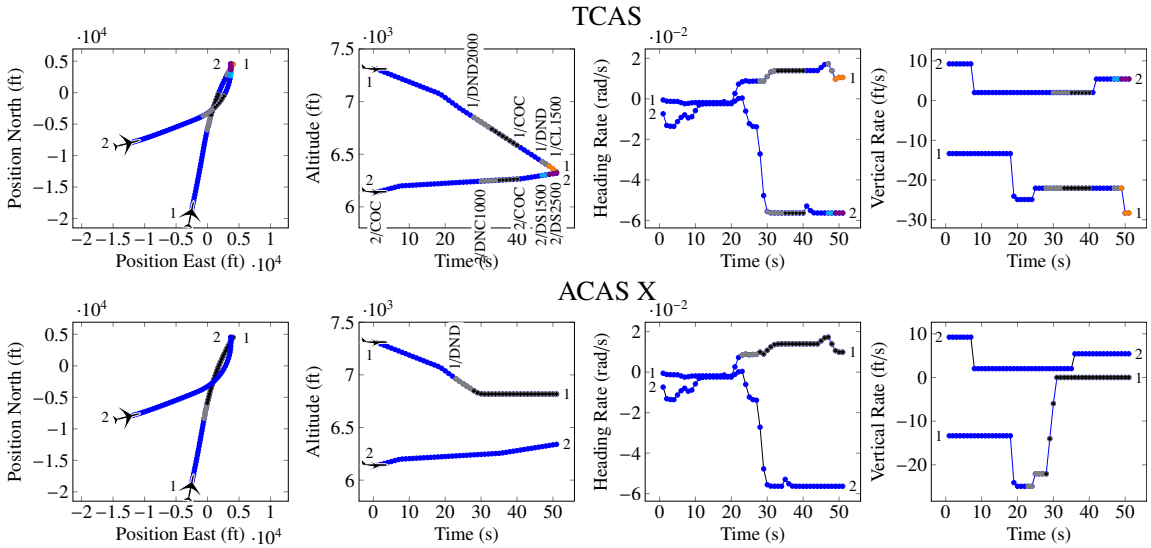

Figure 14: An NMAC encounter with a double horizontal crossing and a split advisory. An NMAC occurs at 50 seconds in the TCAS simulation but not in the ACAS X simulation.

Triple Altitude Crossing. An example is shown in Figure 15 where an NMAC occurs at 40 seconds in the TCAS simulation but no NMAC occurs in the ACAS X simulation. The encounter occurs with probability density $1.5 \cdot 10^{-9}$, which is more likely than the first two examples presented in this section according to our model. The aircraft approach almost straight, head-on, and co-altitude. TCAS issues initial RAs just as the aircraft cross in altitude. Responding to the RAs actually results in bringing the aircraft closer together vertically. TCAS subsequently reverses the RAs, but the aircraft cross in altitude for a second time due to the pilot response delay. The aircraft, following the reversal RAs, cross in altitude for a third time, where an NMAC occurs. In contrast, ACAS X is able to resolve the conflict by issuing crossing RAs slightly earlier than TCAS and maintaining the RAs for the duration of the encounter.

In the TCAS simulation, interactions between the RA and the pilot response delay lead to oscillations in the aircraft altitudes. In particular, the response delay causes RAs and their responses to be separated by altitude crossings, resulting in the responses reducing vertical separation rather than increasing it. The oscillations result in multiple altitude crossings and ultimately an NMAC. Another interesting feature of the encounter is the change in vertical rate by aircraft 2 at 17 seconds, which immediately precedes the initial RAs and the oscillations. Due to the proximity of the maneuver, the effects of state estimation may be playing a significant role in the initial RA. Due to hardware limitations, the collision avoidance system does not directly measure the vertical rate of the intruder and must estimate it by tracking altitude over time. Furthermore, the altitude of the intruder is not known exactly, but quantized to 25 feet increments. To deal with this uncertainty, tracking and filtering techniques are used, which introduce a small amount of tracking error and lag. TCAS uses an alpha-beta filter while ACAS X uses a more sophisticated modified Kalman filter (Asmar, Kochenderfer, & Chryssanthacopoulos, 2013). The modified Kalman filter has been shown to give better tracking error and lag performances. The improved state tracking may be a key contributor to the earlier initial RA and better sense selection in ACAS X in this encounter.

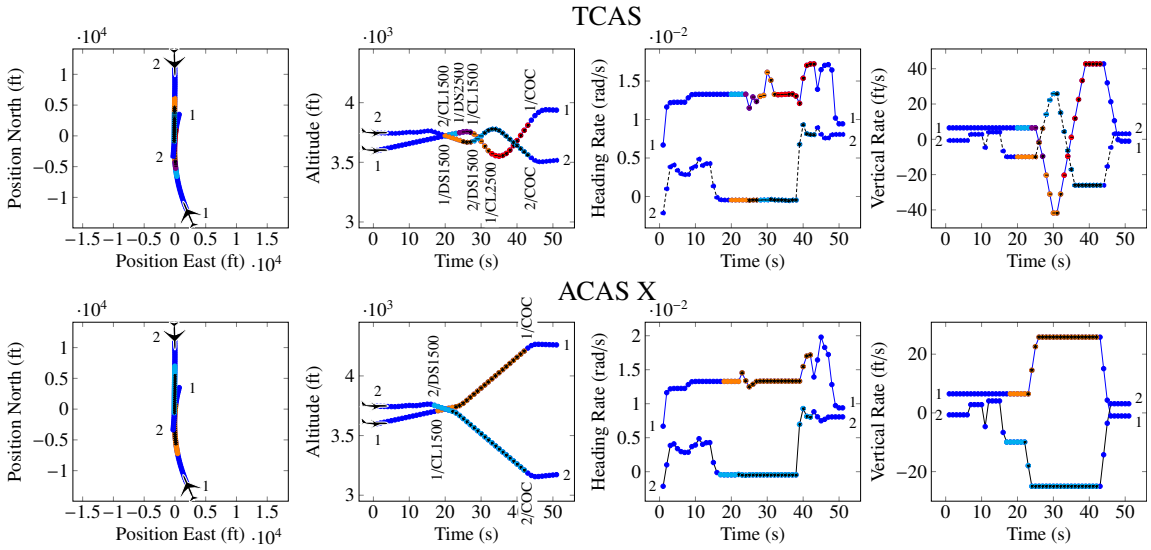

Figure 15: An NMAC encounter where the aircraft cross in altitude three times. An NMAC occurs at 40 seconds in the TCAS simulation but not in the ACAS X simulation.

Vertical Chase. An example is shown in Figure 16 where an NMAC occurs at 43 seconds in the TCAS simulation but no NMAC occurs in the ACAS X simulation. The encounter occurs with probability density $8.4 \cdot 10^{-9}$, which is more likely than the first two examples presented in this section according to our model. In the encounter, the aircraft approach perpendicularly in the horizontal direction and are engaged in a vertical chase. The aircraft are both descending but at different vertical rates. The aircraft cross in altitude and begin to diverge vertically when TCAS issues a crossing RA. Responding to the initial RA, the aircraft cross in altitude for a second time. TCAS then reverses the advisory causing the aircraft to cross in altitude a third time, where an NMAC occurs. In contrast, ACAS X resolves the conflict by issuing a preventative DND advisory early followed by a CL1500 to maintain vertical separation of the aircraft.

The TCAS encounter shows similar oscillations as in the previous example in Figure 15. However, there are a couple of key differences. First, the crossing initial advisories are issued after the aircraft have already crossed in altitude and are beginning to diverge vertically. Second, this encounter has a change in heading rate immediately preceding the initial RAs rather than a change in vertical rate. As a result, it is likely that both vertical state tracking and the estimation of time to closest horizontal approach are playing a role in the initial RAs.

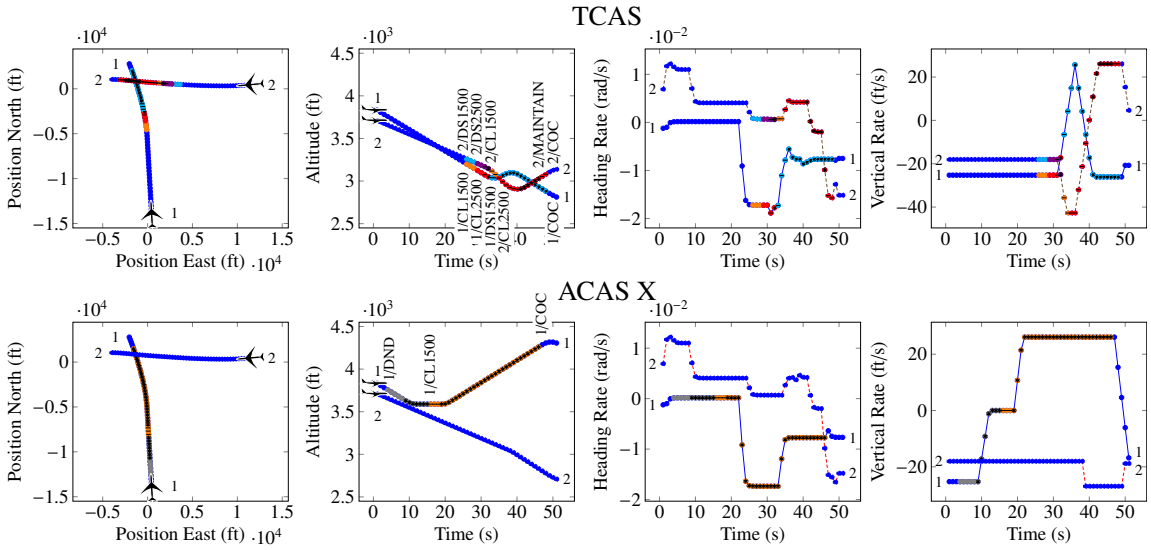

Figure 16: An NMAC encounter where the aircraft are in a vertical chase configuration. TCAS issues crossing RAs after the aircraft have crossed in altitude and are beginning to diverge vertically. An NMAC occurs at 43 seconds in the TCAS simulation but not in the ACAS X simulation.

5.7 Performance Comparison with Direct Monte Carlo Simulation

We compare the performance of MCTS against direct Monte Carlo sampling given a fixed computational budget. The algorithms are given a fixed amount of wall clock time and the best path found at the end of that time is returned. We compare the wall clock time of the algorithms rather than number of samples to account for the additional computations performed in the MCTS algorithm. We use the same configuration as the single-threat encounter experiments as shown in Table 3 except that we limit the search based on computation time instead of a fixed number of iterations. The experiments were performed on a laptop with an Intel i7 4700HQ quad-core processor and 32 GB of memory.

Figure 17 shows the performance of the two algorithms as computation time varies. Figure 17a compares the return of the best encounters found by the algorithms. Each data point shows the mean and standard error of the mean of 100 pairwise encounters. Figure 17b shows the NMAC find rate of NMACs out of the 100 encounters searched. In both cases, MCTS clearly outperforms the baseline Monte Carlo search. The effectiveness of MCTS in finding NMACs is particularly important and we see that MCTS greatly outperforms the baseline in this regard. As the computational budget increases, MCTS is able to find increasingly many NMACs, whereas at the computational budgets considered, Monte Carlo is unable to find any NMACs.

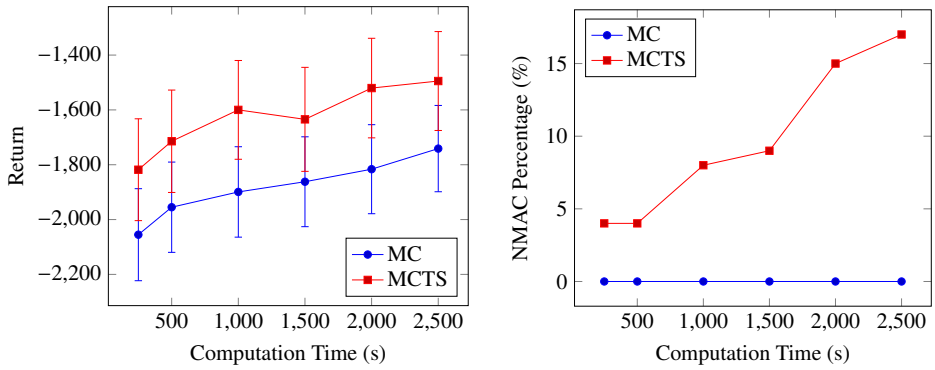

Figure 17: Performance of MCTS and Monte Carlo with computation time. MCTS is able to find an increasing number of NMACs while Monte Carlo is unable to find any due to the vast search space and rare failure events.

6. Conclusion

This article presented adaptive stress testing (AST), a reinforcement learning-based stress testing approach for finding the most likely path to a failure event. We described AST formulations for the case where the state of the simulator is fully observable and also for the case where the state is hidden. For the latter case, we presented the MCTS-SA algorithm that uses the pseudorandom seed of the simulator to overcome partial observability. We also presented differential adaptive stress testing (DAST), an extension of AST for stress testing relative to a baseline. We applied AST and DAST to stress test a prototype of the next-generation ACAS X in an aircraft encounter simulator and found a number of categories of near mid-air collisions, which we reported to the ACAS X team. Our differential studies of ACAS X with TCAS give us additional confidence that ACAS X will offer a significant added safety benefit compared to TCAS. Our results contributed to the certification case of ACAS X, which led to the acceptance of ACAS X by the RTCA. Our implementation of AST is available as an open source Julia package at `https://github.com/sisl/AdaptiveStressTesting.jl`.

Acknowledgments

We thank Neal Suchy at the Federal Aviation Administration; Wes Olson, Robert Klaus, and Cindy McLain at MIT Lincoln Laboratory; Rachel Szczesiul and Jeff Brush at Johns Hopkins Applied Physics Laboratory; and others on the ACAS X team. We thank Guillaume Brat at NASA Ames Research Center and Corina Pasareanu at Carnegie Mellon University Silicon Valley for their valuable feedback. We thank Anthony Corso, Robert Moss, and Mark Koren for their useful feedback on the article. This work was supported by the Safe and Autonomous Systems Operations (SASO) Project and System-Wide Safety (SWS) Project under NASA Aeronautics Research Mission Directorate (ARMD) Airspace Operations and Safety Program (AOSP); and also supported by the Federal Aviation Administration (FAA) Traffic-Alert & Collision Avoidance System (TCAS) Program Office (PO) AJM-233, Volpe National Transportation Systems Center Contract No. DTRT5715D30011.

References

Akazaki, T., Liu, S., Yamagata, Y., Duan, Y., & Hao, J. (2018). Falsification of cyber-physical systems using deep reinforcement learning. In *International Symposium on Formal Methods*, pp. 456–465. Springer.

Annapureddy, Y., Liu, C., Fainekos, G., & Sankaranarayanan, S. (2011). 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*, pp. 254–257. Springer.

Asmar, D. M., Kochenderfer, M. J., & Chryssanthacopoulos, J. P. (2013). Vertical state estimation for aircraft collision avoidance with quantized measurements. *AIAA Journal on Guidance, Control, and Dynamics*, *36*(6), 1797–1802.

Bouton, M., Nakhaei, A., Fujimura, K., & Kochenderfer, M. J. (2018). Scalable decision making with sensor occlusions for autonomous driving. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(1), 1–43.

Chaslot, G. M., Winands, M. H., van den Herik, H. J., Uiterwijk, J. W., & Bouzy, B. (2008). Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, *4*(03), 343–357.

Chludzinski, B. J. (2009). Evaluation of TCAS II version 7.1 using the FAA fast-time encounter generator model. Project report ATC-346, Massachusetts Institute of Technology, Lincoln Laboratory.

Couëtoux, A., Hoock, J.-B., Sokolovska, N., Teytaud, O., & Bonnard, N. (2011). Continuous upper confidence trees. In *Learning and Intelligent Optimization (LION)*, pp. 433–445.

Coulom, R. (2007). Computing âĂIJELO ratingsâĂİ of move patterns in the game of Go. *ICGA journal*, *30*(4), 198–208.

Donzé, A., & Maler, O. (2010). Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pp. 92–106. Springer.

Dreossi, T., Dang, T., Donzé, A., Kapinski, J., Jin, X., & Deshmukh, J. V. (2015). Efficient guiding strategies for testing of temporal properties of hybrid systems. In *NASA Formal Methods Symposium*, pp. 127–142. Springer.

D'Silva, V., Kroening, D., & Weissenbacher, G. (2008). A survey of automated techniques for formal software verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *27*(7), 1165–1178.

Ernst, G., Sedwards, S., Zhang, Z., & Hasuo, I. (2019). Fast falsification of hybrid systems using probabilistically adaptive input. In *International Conference on Quantitative Evaluation of Systems*, pp. 165–181. Springer.

Federal Aviation Administration (2018). Next-generation airborne collision avoidance system (ACAS X) requirements matrix. RTCA (Unpublished).

Gallier, J. H. (2015). *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Courier Dover Publications.

Gardner, R. W., Genin, D., McDowell, R., Rouff, C., Saksena, A., & Schmidt, A. (2016). Probabilistic model checking of the next-generation airborne collision avoidance system. In *Digital Avionics Systems Conference (DASC)*. AIAA/IEEE.

Holland, J. E., Kochenderfer, M. J., & Olson, W. A. (2013). Optimizing the next generation collision avoidance system for safe, suitable, and acceptable operational performance. *Air Traffic Control Quarterly*, *21*(3), 275–297.

International Civil Aviation Organization (2007). Surveillance, radar and collision avoidance. In *International Standards and Recommended Practices* (4th edition)., Vol. IV, annex 10.

Jeannin, J.-B., Ghorbal, K., Kouskoulas, Y., Gardner, R., Schmidt, A., Zawadzki, E., & Platzer, A. (2015). A formally verified hybrid system for the next-generation airborne collision avoidance system. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*.

Katoen, J.-P. (2016). The probabilistic model checking landscape. In *ACM/IEEE Symposium on Logic in Computer Science*, pp. 31–45. ACM.

Kern, C., & Greenstreet, M. R. (1999). Formal verification in hardware design: A survey. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, *4*(2), 123–193.

Kim, Y., & Kochenderfer, M. J. (2016). Improving aircraft collision risk estimation using the cross-entropy method. *Journal of Air Transportation*, *24*(2), 55–62.

Kochenderfer, M. J. (2015). *Decision Making under Uncertainty: Theory and Application*. MIT Press.

Kochenderfer, M. J., & Chryssanthacopoulos, J. P. (2010). A decision-theoretic approach to developing robust collision avoidance logic. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1837–1842.

Kochenderfer, M. J., Edwards, M. W. M., Espindle, L. P., Kuchar, J. K., & Griffith, J. D. (2010). Airspace encounter models for estimating collision risk. *AIAA Journal on Guidance, Control, and Dynamics*, *33*(2), 487–499.

Kochenderfer, M. J., Espindle, L. P., Kuchar, J. K., & Griffith, J. D. (2008). Correlated encounter model for cooperative aircraft in the national airspace system. Project report ATC-344, Massachusetts Institute of Technology, Lincoln Laboratory.

Kochenderfer, M. J., Holland, J. E., & Chryssanthacopoulos, J. P. (2012). Next-generation airborne collision avoidance system. *Lincoln Laboratory Journal*, *19*(1), 17–33.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, pp. 282–293.

Koren, M., Alsaif, S., Lee, R., & Kochenderfer, M. J. (2018). Adaptive stress testing for autonomous vehicles. In *Intelligent Vehicles Symposium (IV)*. IEEE.

Kouskoulas, Y., Genin, D., Schmidt, A., & Jeannin, J. (2017). Formally verified safe vertical maneuvers for non-deterministic, accelerating aircraft dynamics. In *International Conference on Interactive Theorem Proving*, pp. 336–353.

Kuchar, J. K., & Drumm, A. C. (2007). The traffic alert and collision avoidance system. *Lincoln Laboratory Journal*, *16*(2), 277–296.

Lee, R., Kochenderfer, M. J., Mengshoel, O. J., Brat, G. P., & Owen, M. P. (2015). Adaptive stress testing of airborne collision avoidance systems. In *Digital Avionics Systems Conference (DASC)*. AIAA/IEEE.

Lee, R., Kochenderfer, M. J., Mengshoel, O. J., & Silbermann, J. (2018a). Interpretable categorization of heterogeneous time series data. In *International Conference on Data Mining (SDM)*. SIAM.

Lee, R., Mengshoel, O. J., Saksena, A., Gardner, R., Genin, D., Brush, J., & Kochenderfer, M. J. (2018b). Differential adaptive stress testing of airborne collision avoidance systems. In *SciTech, Modeling and Simulation Technologies Conference (MST)*. AIAA.

O'Kelly, M., Sinha, A., Namkoong, H., Tedrake, R., & Duchi, J. C. (2018). Scalable end-to-end autonomous vehicle testing via rare-event simulation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 9827–9838.

Pnueli, A. (1977). The temporal logic of programs. In *Foundations of Computer Science, 1977*, pp. 46–57. IEEE.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

von Essen, C., & Giannakopoulou, D. (2016). Probabilistic verification and synthesis of the next generation airborne collision avoidance system. *International Journal on Software Tools for Technology Transfer*, *18*(2), 227–243.

Watkins, C. J. C. H., & Dayan, P. (1992). Technical note: Q-learning. *Machine Learning*, *8*, 279–292.

Wiering, M., & van Otterlo, M. (Eds.). (2012). *Reinforcement Learning: State of the Art*. Springer, New York.

Zhao, D., Lam, H., Peng, H., Bao, S., LeBlanc, D. J., Nobukawa, K., & Pan, C. S. (2016). Accelerated evaluation of automated vehicles safety in lane-change scenarios based on importance sampling techniques. *IEEE Transactions on Intelligent Transportation Systems*, *18*(3), 595–607.