Two-Layered Falsification of Hybrid Systems Guided by Monte Carlo Tree Search

Anonymous Author(s)

ABSTRACT

Not many real-world hybrid systems are amenable to formal verification, due to their complexity and black box components. *Optimization-based falsification*—a methodology of search-based testing that employs stochastic optimization—is thus attracting attention as an alternative quality assurance method. Inspired by the recent work that advocates *coverage* and *exploration* in falsification, we introduce a two-layered optimization framework that uses *Monte Carlo tree search (MCTS)*, a popular machine learning technique with solid mathematical and empirical foundations (e.g. in computer Go). MCTS is used in the upper layer of our framework; it guides the lower layer of local hill-climbing optimization, thus balancing exploration and exploitation in a disciplined manner. We demonstrate the proposed framework through experiments with benchmarks from the automotive domain.

CCS CONCEPTS

• Computer systems organization → Embedded and cyber-physical systems; **• Mathematics of computing → Stochastic control and optimization**; **• Theory of computation → Random search heuristics**;

KEYWORDS

cyber-physical system, hybrid system, testing, falsification, stochastic optimization, temporal logic

ACM Reference Format:
Anonymous Author(s). 2018. Two-Layered Falsification of Hybrid Systems Guided by Monte Carlo Tree Search. In *Proceedings of ACM International Conference on Embedded Software (EMSOFT'18).* ACM, New York, NY, USA, 10 pages.

1 INTRODUCTION

Hybrid Systems. Quality assurance of *cyber-physical systems* (CPS) is a problem of great interest. Errors in CPS, such as cars and aircrafts, can lead to economic and social damage, including loss of human lives. Unique challenges in quality assurance are posed by the nature of CPS: in the form of *hybrid systems* they comprise the discrete dynamics of computers and the continuous dynamics of physical components. Continuous dynamics combined with other features, such as complexity (a modern car can contain 10^8 lines of code) and black-box components (such as parts coming from external suppliers), make it very hard to apply formal verification to CPS.

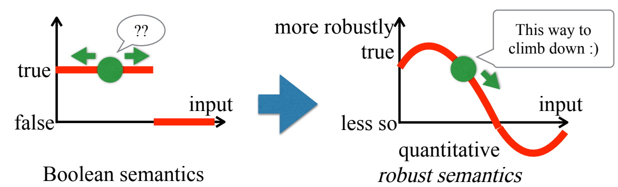

Figure 1: From Boolean to robust semantics

An increasing number of researchers and practitioners are therefore turning to *optimization-based falsification* as a quality assurance measure for CPS. The problem is formalized as follows.

The falsification problem

- **Given:** a *model* $\mathcal{M}$ (that takes an input signal $\mathbf{u}$ and yields an output signal $\mathcal{M}(\mathbf{u})$), and a *specification* φ (a temporal formula)
- **Answer:** *error input*, that is, an input signal $\mathbf{u}$ such that the corresponding output $\mathcal{M}(\mathbf{u})$ violates φ

$$\xrightarrow{\mathbf{u}} \boxed{\mathcal{M}} \xrightarrow[\not\models \varphi\,?]{\mathcal{M}(\mathbf{u})}$$

In the optimization-based falsification approach, the above falsification problem is turned into an optimization problem. This is possible thanks to *robust semantics* of temporal formulas [18]. Instead of the Boolean satisfaction relation $\mathbf{v} \models \varphi$, robust semantics assigns a quantity $[\![\mathbf{v}, \varphi]\!] \in \mathbb{R} \cup \{\infty, -\infty\}$ that tells us, not only whether φ is true or not (by the sign), but also *how robustly* the formula is true or false. This allows one to employ hill-climbing optimization (see Fig. 1): we iteratively generate input signals, in the direction of decreasing robustness, hoping that eventually we hit negative robustness.

Optimization-based falsification is a subclass of *search-based testing*: it adaptively chooses test cases (input signals $\mathbf{u}$) based on previous observations. One can use stochastic algorithms for optimization, such as simulated annealing (SA), Global Nelder-Mead (GNM) and covariance matrix adaptation evolution strategy (CMA-ES [5]), which turn out to be much more scalable than model checking algorithms that rely on exhaustive search. Note also that the system model $\mathcal{M}$ can be black box: observing the correspondence between input $\mathbf{u}$ and output $\mathcal{M}(\mathbf{u})$ is enough. Observing an error $\mathcal{M}(\mathbf{u}')$ for some input $\mathbf{u}'$ is sufficient evidence for a system designer to know that the system needs improvement. Besides these practical advantages, optimization-based falsification is an interesting topic from a scientific point of view, combining formal and structural reasoning with stochastic optimization.

The approach of optimization-based falsification was initiated in [18] and has been actively pursued ever since [1, 3, 4, 11, 13–15, 17, 30, 37, 38]. See [28] for a recent survey. There are now mature tools, such as Breach [13] and S-Taliro [4], which work with industry-standard Simulink models.

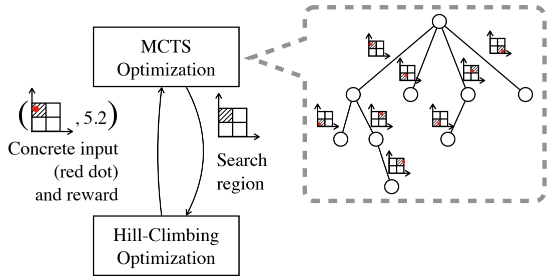

Figure 2: Our two-layered optimization framework

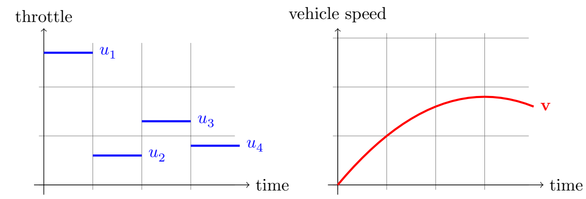

Figure 3: A piecewise constant input signal (one-dimensional, throttle, left) for a simple automotive powertrain model, and the corresponding output signal (one-dimensional, vehicle speed, right).

The Exploration-Exploitation Trade-off in Falsification. In optimization-based falsification, the important role of *coverage* is advocated by many authors [1, 11, 15, 30] (see also §5). One reason is that in highly nonconvex optimization problems for falsification, eager hill climbing can easily be trapped in local minima and thus fail to find an error input (i.e. a global minimum) that exists elsewhere. Another reason is that coverage gives a certain degree of confidence for absence of error input, in case search for error input is unsuccessful.

This puts us in the *exploration-exploitation trade-off*,[1] a typical dilemma in stochastic optimization and machine learning (specifically in reinforcement/active learning). While exploitation guides us to pursue the direction that seems promising based on the previous observations, we have to occasionally explore in order to avoid getting stuck local minima. Many common stochastic hill-climbing algorithms, such as SA, GNM and CMA-ES, contain implicit exploration mechanisms. At the same time, explicit methods for exploration in falsification have been pursued e.g. in [1, 11, 15, 30] (see §5).

Contribution: Two-Layered Optimization for Falsification Guided by Monte Carlo Tree Search. Our main contribution is, in the context of hybrid system falsification, to balance exploration and exploitation in a systematic and mathematically disciplined way using *Monte Carlo tree search (MCTS)* [7, 29]. We integrate hill-climbing optimization in MCTS, and obtain a two-layered optimization framework.

MCTS uses a *search tree* whose nodes are usually organized according to causal relationships, and interleaves *search* (walking down the already expanded tree, in a promising direction) with *playout* (expanding a new node and estimating its reward). Typical applications allowing such a structured search space are *decision problems* as games. In particular, MCTS is attracting a lot of attention thanks to its success in computer Go [36]. One main cause for the success of MCTS is search strategies[2] that nicely balance exploration and exploitation. For example, the most common search strategy called UCT (upper confidence tree [29]) is derived from a solid theoretical background, namely the upper confidence bounds (UCB) strategy for the multi-armed bandit problem. While MCTS is a relatively new methodology, it has established its position in the rapidly growing community of machine learning. See [7] for a survey.

Our framework uses robustness values as rewards in MCTS, and employs hill-climbing optimization for playout in MCTS. This way we integrate hill-climbing in Monte Carlo tree search in a systematic way. In our two-layered framework (Fig. 2), the upper optimization layer picks (by MCTS) a region in the input space, from which a concrete input value should be sampled. The lower layer then picks (by hill-climbing) an optimal concrete input value within the prescribed region. We also compute the robustness of the specification under the chosen input. This value is fed back to the upper layer as a reward, which is then used by the tree search strategy to balance exploration and exploitation.

In our two-layered framework, hill-climbing optimization—whose potential in falsification of hybrid systems has been established, see e.g. [28]—is supervised by MCTS, with MCTS dictating which region to sample from. By expanding new children, MCTS can tell hill-climbing optimization to try an input region that has not yet explored, or to exploit and dig deep in a direction that seems promising. Such combination of MCTS and application-specific lower-layer optimization seems to be a useful approach that can apply to problems other than hybrid system falsification. See §5 for further discussion.

Our use of MCTS depends on incrementally synthesizing K input segments one after another. Those input segments are for the time intervals $[0, \frac{T}{K}), [\frac{T}{K}, \frac{2T}{K}), \dots, [\frac{(K-1)T}{K}, T]$, where T is the time horizon. The Monte Carlo search tree will then be of depth K. See Fig. 3. In this paper we restrict input signals to piecewise-constant ones (this is a common assumption in falsification); an edge in the MCTS search tree from depth $i-1$ to i (see Fig. 2) determines the input value u_i for the interval $[\frac{(i-1)T}{K}, \frac{iT}{K})$.

We have implemented our two-layered falsification framework in MATLAB, building on Breach [13].[3] Our experiments with benchmarks from [12, 24, 27] demonstrate the possible performance improvements, especially in the ability of finding rare counterexamples.

Organization. In §2 we formulate the falsification problem. In §3 we present our main contribution, namely a two-layered optimization framework for falsification that combines MCTS and hill-climbing. Our experimental results are in §4. In §5 we discuss related work, locating the current work in the context of falsification and also of other applications of MCTS and related machine learning methods. In §6 we conclude with some directions of future research.

Notations. The set of (positive, nonnegative) real numbers is denoted by $\mathbb{R}$ (and $\mathbb{R}_+, \mathbb{R}_{\geq 0}$, respectively). Closed and open intervals are denoted such as $[0, 2]$ and $(2, 3)$; $[0, 2) = \{x \in \mathbb{R} \mid 0 \leq x < 2\}$ is a half-closed half-open interval. For a set X, $|X|$ denotes its cardinality.

[1] Also called the *variance-bias compromise* in the literature.
[2] Often called *tree policies* in the MCTS literature.
[3] Code obtained at https://github.com/decyphir/breach.

2 PROBLEM: HYBRID SYSTEM FALSIFICATION

We formulate the problem of hybrid system falsification. We also introduce robust semantics of temporal logics [14, 18] that allows us to reduce falsification to an optimization problem.

Definition 2.1 (time-bounded signal)**.** Let $T \in \mathbb{R}_+$ be a positive real. An *m-dimensional signal* with a time horizon T is a function $\mathbf{w} \colon [0, T] \to \mathbb{R}^m$.

Let $\mathbf{w} \colon [0, T] \to \mathbb{R}^m$ and $\mathbf{w}' \colon [0, T'] \to \mathbb{R}^m$ be m-dimensional signals. Their *concatenation* $\mathbf{w} \cdot \mathbf{w}' \colon [0, T + T'] \to \mathbb{R}^m$ is an m-dimensional signal defined by $(\mathbf{w} \cdot \mathbf{w}')(t) := \mathbf{w}(t)$ if $t \in [0, T]$, and $\mathbf{w}'(t - T)$ if $t \in (T, T + T']$.

Let $T_1, T_2 \in (0, T]$ such that $T_1 < T_2$. The *restriction* $\mathbf{w}|_{[T_1,T_2]} \colon [0, T_2 - T_1] \to \mathbb{R}^m$ of $\mathbf{w} \colon [0, T] \to \mathbb{R}^m$ to the interval $[T_1, T_2]$ is defined by $(\mathbf{w}|_{[T_1,T_2]})(t) := \mathbf{w}(T_1 + t)$.

Definition 2.2 (system model $\mathcal{M}$)**.** A *system model*, with m-dimensional input and n-dimensional output, is a function $\mathcal{M}$ that takes an input signal $\mathbf{u} \colon [0, T] \to \mathbb{R}^m$ and returns a signal $\mathcal{M}(\mathbf{u}) \colon [0, T] \to \mathbb{R}^n$. Here the common time horizon $T \in \mathbb{R}_+$ is arbitrary.

Some recent works including [25] use sequences of time-stamped values as basic objects in their problem formulation, in place of continuous-time signals (as we do in the above). This difference is mostly presentational and not essential.

As a specification language we use *signal temporal logic* (STL) [32]. We do so for simplicity of presentation; we can also use more expressive logics such as the one in [2].

In what follows **Var** is the set of variables. Variables stand for physical quantities, control modes, etc. $\equiv$ denotes syntactic equality.

Definition 2.3 (syntax)**.** In STL, *atomic propositions* and *formulas* are defined as follows, respectively: $\alpha \ ::\equiv \ \big(f(x_1, \dots, x_n) > 0\big)$, and $\varphi \ ::\equiv \ \alpha \mid \bot \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi\, \mathcal{U}_I\, \varphi$. Here f is an n-ary function $f : \mathbb{R}^n \to \mathbb{R}$, $x_1, \dots, x_n \in$ **Var**, and I is a closed non-singular interval in $\mathbb{R}_{\geq 0}$, i.e. $I = [a, b]$ or $[a, \infty)$ where $a, b \in \mathbb{R}$ and $a < b$.

We omit subscripts I for temporal operators if $I = [0, \infty)$. Other common connectives like $\vee, \to, \top, \Box_I$ (always) and $\Diamond_I$ (eventually), are introduced as abbreviations: $\Diamond_I \varphi \equiv \top\, \mathcal{U}_I\, \varphi$ and $\Box_I \varphi \equiv \neg \Diamond_I \neg \varphi$. Atomic formulas like $f(\vec{x}) \leq c$, where $c \in \mathbb{R}$ is a constant, are also accommodated by using negation and the function $f'(\vec{x}) := f(\vec{x}) - c$.

Definition 2.4 (robust semantics [14])**.** For an n-dimensional signal $\mathbf{w} \colon \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ and $t \in \mathbb{R}_{\geq 0}$, $\mathbf{w}^t$ denotes the *t-shift* of $\mathbf{w}$, that is, $\mathbf{w}^t(t') := \mathbf{w}(t + t')$.

Let $\mathbf{w} \colon \mathbb{R}_{\geq 0} \to \mathbb{R}^{|\mathbf{Var}|}$ be a signal, and φ be an STL formula. We define the *robustness* $[\![\mathbf{w}, \varphi]\!] \in \mathbb{R} \cup \{\infty, -\infty\}$ as follows, by induction. Here $\bigsqcap$ and $\bigsqcup$ denote infimums and supremums of real numbers, respectively. Their binary version $\sqcap$ and $\sqcup$ denote minimum and maximum.

$$
\begin{aligned}
[\![\mathbf{w}, f(x_1, \cdots, x_n) > 0]\!] &:= f\big(\mathbf{w}(0)(x_1), \cdots, \mathbf{w}(0)(x_n)\big) \\
[\![\mathbf{w}, \bot]\!] &:= -\infty \qquad [\![\mathbf{w}, \neg\varphi]\!] := -[\![\mathbf{w}, \varphi]\!] \\
[\![\mathbf{w}, \varphi_1 \wedge \varphi_2]\!] &:= [\![\mathbf{w}, \varphi_1]\!] \sqcap [\![\mathbf{w}, \varphi_2]\!] \\
[\![\mathbf{w}, \varphi_1\, \mathcal{U}_I\, \varphi_2]\!] &:= \textstyle\bigsqcup_{t \in I} \big([\![\mathbf{w}^t, \varphi_2]\!] \sqcap \bigsqcap_{t' \in [0,t)} [\![\mathbf{w}^{t'}, \varphi_1]\!] \big)
\end{aligned}
$$

Here are some intuitions and consequences of the definition. The robustness $[\![\mathbf{w}, f(\vec{x}) > c]\!]$ stands for the vertical margin $f(\vec{x}) - c$ for the signal $\mathbf{w}$ at time 0. A negative robustness value indicates how far the formula is from being true. The robustness for the eventually modality is computed by $[\![\mathbf{w}, \Diamond_{[a,b]}(x > 0)]\!] = \bigsqcup_{t \in [a,b]} \mathbf{w}(t)(x)$.

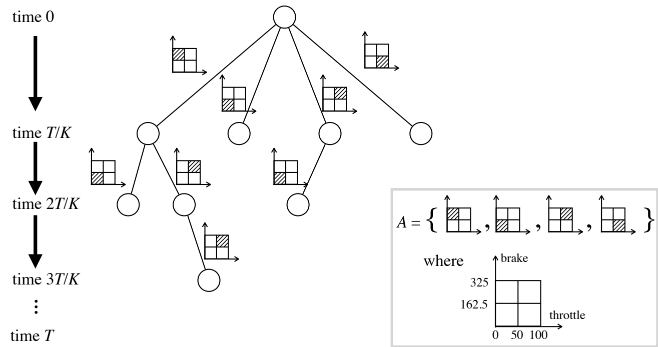

Figure 4: Our MCTS search tree, for a system model $\mathcal{M}$ with two input values (throttle and brake) whose ranges are $[0, 100]$ and $[0, 325]$, respectively. We partition each interval to two (i.e. $L_1 = L_2 = 2$); thus the branching degree $|A|$ is 2×2.

The original semantics of STL is Boolean, given by a binary relation $\models$ between signals and formulas. The robust semantics refines the Boolean one as follows: $[\![\mathbf{w}, \varphi]\!] > 0$ implies $\mathbf{w} \models \varphi$, and $[\![\mathbf{w}, \varphi]\!] < 0$ implies $\mathbf{w} \not\models \varphi$, see [18, Prop. 16]. Optimization-based falsification via robust semantics hinges on this refinement. Although the definitions so far are for time-unbounded signals only, we note that the robust semantics $[\![\mathbf{w}, \varphi]\!]$, as well as the Boolean satisfaction $\mathbf{w} \models \varphi$, can be easily adapted to time-bounded signals (Def. 2.1).

Finally, here is a formalization of the falsification problem. It refines the description in §1. In particular, its use of real-valued robust semantics enables use of hill-climbing optimization. See Fig. 1.

Definition 2.5 (falsifying input)**.** Let $\mathcal{M}$ be a system model, and φ be an STL formula. A signal $\mathbf{u} \colon [0, T] \to \mathbb{R}^m$ is a *falsifying input* if $[\![\mathcal{M}(\mathbf{u}), \varphi]\!] < 0$ (that implies $\mathcal{M}(\mathbf{u}) \not\models \varphi$).

3 TWO-LAYERED OPTIMIZATION FRAMEWORK WITH MCTS

In this section we present our main contribution, namely a two-layered optimization framework for hybrid system falsification. It combines: *Monte Carlo tree search (MCTS)* [7, 29] (the upper layer) for high-level planning; and hill-climbing optimization (such as SA, GNM and CMA-ES, the lower layer) for local input search (see Fig. 2 for a schematic overview). The upper layer steers the lower layer by the UCT strategy [29], an established method in machine learning for balancing exploration and exploitation.

We present two algorithms: the *basic* two-layered algorithm (Alg. 1), and the one enhanced with *progressive widening* (Alg. 3). The auxiliary functions used therein are presented in Alg. 2. Our algorithms work on an *MCTS search tree*; its example is shown in Fig. 4.

3.1 The Basic Two-Layered Algorithm (Alg. 1)

We start with Alg. 1, using the example in Fig. 4.

Time Staging. We search for a falsifying input signal, focusing on piecewise-constant signals (Fig. 3, left). The interval $[0, T]$ is divided into K intervals of the same size (here K is a tunable parameter). The

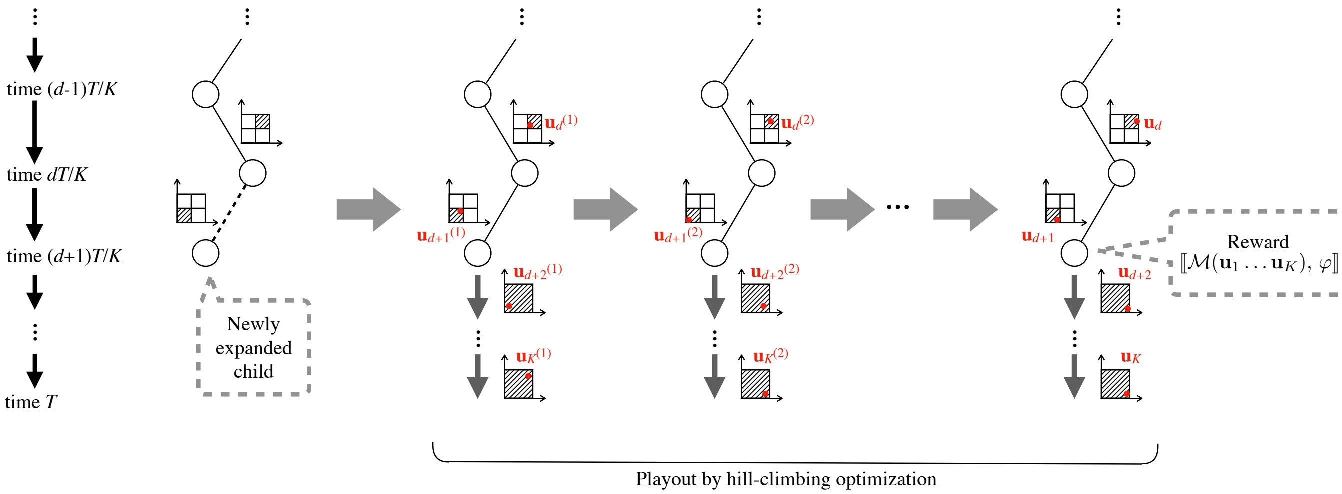

Figure 5: Playout by hill-climbing optimization

time points $0, \frac{T}{K}, \frac{2T}{K}, \dots, \frac{(K-1)T}{K}$—at which those intervals start—are called *control points*. Our goal is therefore to find a sequence $\mathbf{u}_1, \dots, \mathbf{u}_K$, where each $\mathbf{u}_i = (u_{i1}, \dots, u_{iM})$ is an M-dimensional real vector (M is the number of input for the model $\mathcal{M}$), so that the corresponding piecewise-constant signal is a falsifying one (Def. 2.5).

We assume intervals $I_i = [u_i^{\min}, u_i^{\max}]$ ($i \in \{1, \dots, M\}$) for the ranges of input $u_1, \dots, u_M$ of the model $\mathcal{M}$.

The Search Tree. A search tree in MCTS has a branching degree $|A|$, where the set A is called an *action set* in the MCTS literature. In Go, for example, an action set A consists of possible moves.

We use, as the action set A, a *partitioning* of the input space $I_1 \times \cdots \times I_M$. We partition the input space into $L_1 \times \cdots \times L_M$ hypercubes of the equal size, according to predetermined parameters $L_1, \dots, L_M$, where M is the number of input signals of the system and L_i indicates how finely the i-th input should be partitioned. In Fig. 4 we present an example where $M = 2$ and $L_1 = L_2 = 2$. There we have four actions in the set A, corresponding to the four square regions.

An edge in our search tree represents a choice of an input region—from which we choose the input value $\mathbf{u}_i$ from—for a single control point $\frac{(i-1)T}{K}$. The depth of the tree is K (the number of control points). We follow the usual convention and specify a node of a $|A|$-branching tree by a word $w = a_1 a_2 \dots a_j$ over the alphabet A (here $j \le K$). That is: the root is ε (the empty word), its child in the direction $a_1 \in A$ is a_1, its children are $a_1 a_1, a_1 a_2, \dots$, and so on.

In general, a node in an MCTS search tree is decorated by two values: *reward* R and *visit count* N. In our case, R stores the current estimate of the smallest (i.e. the best) robustness value. Both values are updated explicitly during back-propagation (see below).

Monte Carlo Tree Search Sampling. Much like usual MCTS, Alg. 1 iteratively expands the search tree $\mathcal{T}$. Initially the tree $\mathcal{T}$ is root-only (Line 3), and in each iteration—called *MCTS sampling*—the invocation of MCTSSample on Line 10 adds one new node to $\mathcal{T}$. (In the MCTS literature, *expanding* a child means adding the child to $\mathcal{T}$.) We repeat MCTS sampling until a counterexample is found, or the MCTS budget is used up after the max number of iterations (Line 9).

The exploration-exploitation trade-off in MCTS is in the choice of the node to add. In each MCTS sampling, we start from the root (Line 10), walk down in the tree $\mathcal{T}$ choosing already expanded nodes (Lines 19–20), until we expand a child (Lines 23–24). Growing a wider tree means exploration, while a deeper tree means exploitation.

We use the UCT strategy [29], the most commonly used strategy in MCTS, to resolve the dilemma. UCT is based on the UCB strategy for the multi-armed bandit problems; Line 2 of Alg. 2 follows UCB, where the exploitation score $1 - \frac{R(wa)}{\max_{w' \in \mathcal{T}} R(w')}$ and the exploration score $\sqrt{\frac{2 \ln N(w)}{N(wa)}}$ are superposed using a scaler c. Recall that our rewards $R(wa)$ for w's children are given by robustness estimates from previous simulations, and that falsification favors smaller R. Note also that values of R can be greater than 1 in general. In the exploitation score $1 - \frac{R(wa)}{\max_{w' \in \mathcal{T}} R(w')}$, therefore, we normalize rewards to the interval $[0, 1]$ and reverse their order.[4] The exploration score $\sqrt{\frac{2 \ln N(w)}{N(wa)}}$ is taken from UCB: the visit count $N(w)$ tells how many times the node w has been visited, that is, how many offsprings the node w currently has in $\mathcal{T}$. The scaler, for the trade-off, is a tunable parameter, as usual in MCTS.

Playout and Back-Propagation. In MCTS, the reward of a newly expanded node $a_1 a_2 \dots a_d a$ (see e.g. Line 24) is computed by an operation called *playout*. The result is then *back-propagated*, in a suitable manner, to the ancestors: $a_1 \dots a_d, a_1 \dots a_{d-1}, \dots$ and finally ε.

In our MCTS algorithms for falsification we use hill-climbing optimization (such as SA, GNM and CMA-ES) for playout. See Line 25, where input values $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_K$ are sampled by stochastic hill-climbing optimization, so that the resulting robustness value of the specification φ becomes smaller. The regions to sample those values from are dictated by the MCTS tree: $\mathbf{u}_1 \in \textsc{Reg}(a_1), \dots, \mathbf{u}_d \in \textsc{Reg}(a_d)$ follow the actions $a_1, \dots, a_d$ determined so far (here Reg is from Alg. 2); $\mathbf{u}_{d+1} \in \textsc{Reg}(a)$ follows the newly chosen action a (Line 23); and the remaining values $\mathbf{u}_{d+2}, \dots, \mathbf{u}_K$ can be chosen from the whole input range $I_1 \times \cdots \times I_M$.

[4] We can assume nonnegative values of R , otherwise we already have a falsifying input.

Algorithm 1 *Basic* Two-Layered Algorithm

Require: a system model $\mathcal{M}$, an STL formula φ, intervals $I_i = [u_i^{\min}, u_i^{\max}]$ $(i \in \{1, \dots, M\})$ for the ranges of input $u_1, \dots, u_M$ of $\mathcal{M}$, time horizon $T \in \mathbb{R}_+$, and the following tunable parameters: the number K of control points, the number L_i of partitions of the input range $[u_i^{\min}, u_i^{\max}]$ for each $i \in \{1, \dots, M\}$, the scaler c in Line 2 of Alg. 2, and an MCTS budget (the maximum number of MCTS sampling, Line 9)

1: **function** MCTSPreprocess
2: $A \leftarrow \{1, \dots, L_1\} \times \cdots \times \{1, \dots, L_M\}$ ▹ the set of actions
3: $\mathcal{T} \leftarrow \{\varepsilon\}$ ▹ the MCTS search tree, initially root-only
4: $N \leftarrow (\varepsilon \mapsto 0)$ ▹ visit count N initialized, defined only for ε
5: $R \leftarrow (\varepsilon \mapsto \infty)$ ▹ reward function R initialized
6: $\vec{\mathbf{u}} \leftarrow$ null ▹ place holder for a falsifying input
7: $R_{\min} \leftarrow \infty$ ▹ place holder for a minimum reward
8: $\vec{a}_{\min} \leftarrow$ null ▹ the most promising action sequence
9: **while** $R(\varepsilon) \geq 0$ and within the MCTS budget **do**
10: MCTSSample(ε)
11: **if** $\vec{\mathbf{u}} \neq$ null **then**
▹ a falsifying input is found already in preprocessing
12: **return** $\vec{\mathbf{u}}$
13: **else** ▹ return the most promising action sequence
14: **return** $\vec{a}_{\min}$

15: **function** MCTSSample(w) ▹ let $w = a_1 \dots a_d$ with $a_i \in A$
16: $N(w) \leftarrow N(w) + 1$
17: **if** $|w| < K$ **then**
18: **if** $wa' \in \mathcal{T}$ for all $a' \in A$ **then**
▹ if all children have been expanded
19: $a \leftarrow$ UCBSample(w) ▹ pick a child wa by UCB
20: MCTSSample(wa) ▹ recursive call
21: $R(w) \leftarrow \min_{a' \in A} R(wa')$ ▹ back-propagation
22: **else**
23: randomly sample $a \in A$ from $\{a \mid wa \notin \mathcal{T}\}$
▹ expand a random unexpanded child wa
24: $\mathcal{T} \leftarrow \mathcal{T} \cup \{wa\}$
25: $\mathbf{u}_1, \dots, \mathbf{u}_K$ ▹ playout by hill-climbing
$$\leftarrow \operatorname*{arg\,min^{HillClimb}}_{\substack{\mathbf{u}_1 \in \textsc{Reg}(a_1), \dots, \mathbf{u}_d \in \textsc{Reg}(a_d), \\ \mathbf{u}_{d+1} \in \textsc{Reg}(a), \\ \mathbf{u}_{d+2}, \dots, \mathbf{u}_K \in I_1 \times \cdots \times I_M}} [\![\mathcal{M}(\mathbf{u}_1 \dots \mathbf{u}_K), \varphi]\!]$$
26: $N(wa) \leftarrow 0;$ $R(wa) \leftarrow [\![\mathcal{M}(\mathbf{u}_1 \dots \mathbf{u}_K), \varphi]\!]$
27: **if** $R(wa) < 0$ **then**
28: $\vec{\mathbf{u}} \leftarrow \mathbf{u}_1 \dots \mathbf{u}_K$
▹ a falsifying input is found and stored in $\vec{\mathbf{u}}$
29: **if** $R(wa) < R_{\min}$ **then**
30: $R_{\min} \leftarrow R(wa);$ $\vec{a}_{\min} \leftarrow a_1 \dots a_d a$
31: $R(w) \leftarrow \min_{a' \in A} R(wa')$ ▹ back-propagation

32: **function** Main
33: $\vec{x} \leftarrow$ MCTSPreprocess
34: **if** $\vec{x} = \vec{\mathbf{u}}$, an input signal **then** ▹ Line 11
35: **return** $\vec{\mathbf{u}}$
36: **else** ▹ $\vec{x} = a_1 a_2 \dots a_{K'} \in A^*$ with some $K' \leq K$, Line 13
37: **return** $\operatorname*{arg\,min^{HillClimb}}_{\substack{\mathbf{u}_1 \in \textsc{Reg}(a_1), \dots, \mathbf{u}_{K'} \in \textsc{Reg}(a_{K'}), \\ \mathbf{u}_{K'+1}, \dots, \mathbf{u}_K \in I_1 \times \cdots \times I_M}} [\![\mathcal{M}(\mathbf{u}_1 \dots \mathbf{u}_K), \varphi]\!]$

Algorithm 2 Auxiliary Functions for Algs. 1 & 3

1: **function** UCBSample(w)
2: **return** $\displaystyle \operatorname*{arg\,max}_{a \in A} \left(\left(1 - \frac{R(wa)}{\max_{w' \in \mathcal{T}} R(w')}\right) + c\sqrt{\frac{2 \ln N(w)}{N(wa)}} \right)$
3: **function** Reg(a) ▹ The input region for an action a
▹ $a \in A$ is of the form $(k_1, \dots, k_M)$, see Line 2 of Alg. 1
4: **return** $\prod_{i=1}^{M} \Big[u_i^{\min} + \frac{k_i - 1}{L_i}(u_i^{\max} - u_i^{\min}),$ $u_i^{\min} + \frac{k_i}{L_i}(u_i^{\max} - u_i^{\min}) \Big)$

Algorithm 3 Two-Layered Algorithm with *Progressive Widening*

Require: The same data as required in Alg. 1, and additionally, constants C, α (used in Line 4)

The algorithm is the same as Alg. 1, except that the function MCTSSample is replaced by the following one.

1: **function** MCTSSample(w) ▹ let $w = a_1 \dots a_d$ with $a_i \in A$
2: $N(w) \leftarrow N(w) + 1$
3: **if** $|w| < K$ **then**
4: **if** $\left(\begin{array}{l} \big|\{a' \in A \mid wa' \in \mathcal{T}\}\big| \geq C \cdot N(w)^{\alpha} \\ \text{or } wa' \in \mathcal{T} \text{ for all } a' \in A \end{array} \right)$ **then**
▹ progressive widening: all or enough children expanded
5: $a \leftarrow$ UCBSample(w) ▹ pick a child wa by UCB
6: MCTSSample(wa) ▹ recursive call
7: $R(w) \leftarrow \min_{a' \in A} R(wa')$ ▹ back-propagation
8: **else**
9: $S \leftarrow$ (a maximal convex subset of $\bigcup_{wa' \notin \mathcal{T}} \textsc{Reg}(a')$)
10: $\mathbf{u}_1, \dots, \mathbf{u}_K$ ▹ playout by hill-climbing
$$\leftarrow \operatorname*{arg\,min^{HillClimb}}_{\substack{\mathbf{u}_1 \in \textsc{Reg}(a_1), \dots, \mathbf{u}_d \in \textsc{Reg}(a_d), \\ \mathbf{u}_{d+1} \in S, \\ \mathbf{u}_{d+2}, \dots, \mathbf{u}_K \in I_1 \times \cdots \times I_M}} [\![\mathcal{M}(\mathbf{u}_1 \dots \mathbf{u}_K), \varphi]\!]$$
11: $a \leftarrow \big(a \in A$ such that $\mathbf{u}_{d+1} \in \textsc{Reg}(a)\big)$
12: $\mathcal{T} \leftarrow \mathcal{T} \cup \{wa\}$
13: $N(wa) \leftarrow 0;$ $R(wa) \leftarrow [\![\mathcal{M}(\mathbf{u}_1 \dots \mathbf{u}_K), \varphi]\!]$
14: **if** $R(wa) < 0$ **then**
15: $\vec{\mathbf{u}} \leftarrow \mathbf{u}_1 \dots \mathbf{u}_K$
16: **if** $R(wa) < R_{\min}$ **then**
17: $R_{\min} \leftarrow R(wa);$ $\vec{a}_{\min} \leftarrow a_1 \dots a_d a$
18: $R(w) \leftarrow \min_{a' \in A} R(wa')$ ▹ back-propagation

See Fig. 5 for an example. Smaller gray squares represent actions, and red dots represent input values (notice that they are chosen from the gray regions). The values $\mathbf{u}_1, \dots, \mathbf{u}_K$ are sampled repeatedly so that the robustness value $[\![\mathcal{M}(\mathbf{u}_1 \dots \mathbf{u}_K), \varphi]\!]$ becomes smaller.

An intuition of this playout operation is that we sample the best input signal $\mathbf{u}_1 \dots \mathbf{u}_K$, under the constraints imposed by the MCTS search tree (namely, the input regions prescribed by the actions). The least robustness value thus obtained is assigned to the newly expanded node wa as its reward (Line 26). If $R(wa) < 0$ then this means we have already succeeded in falsification (Line 28).

Back-propagation is an important operation in MCTS. Following the intuition that the reward $R(w)$ is the smallest robustness achievable at the node w, we define the reward of an internal node w by the minimum of its children's rewards. See Lines 21 and 31. Note that, via recursive calls of MCTSSample (Line 20), the result of playout is propagated to all ancestors.

A Two-Layered Framework. In Alg. 1, hill-climbing optimization occurs twice, in Lines 25 and 37. The first occurrence is in playout of MCTS—this way we interleave MCTS optimization (by growing a tree) and hill-climbing optimization. See Fig. 2. MCTS optimization is considered to be a *preprocessing* phase in Alg. 1 (Line 33): its principal role is to find an action sequence $\vec{a}_{\min}$, i.e. a sequence of input regions, that is most promising. In the remainder of the Main function, the second hill-climbing optimization is conducted for falsification, where we sample according to $\vec{a}_{\min}$.

The two occurrences of hill-climbing optimization (Lines 25 and 37) therefore have different roles. Given also the fact that the first occurrence is repeated every time we expand a new child, we choose to spend less time for the former than the latter. In our implementation, we set the timeout to be 5–15 seconds for the first hill-climbing sampling in Line 25 ($\mathrm{TO}_{\mathrm{po}}$ in §4), while for the second hill-climbing sampling in Line 37 the timeout is 300 seconds.

A falsifying input $\vec{\mathbf{u}}$ is often found already in the preprocessing phase. In this case the Main function simply returns $\vec{\mathbf{u}}$ (Line 35).

3.2 The Two-Layered Algorithm with Progressive Widening (Alg. 3)

Our second algorithm (Alg. 3) differs from the basic one (Alg. 1) in two points:

Progressive Widening. Alg. 3 uses *progressive widening* [10]; see Line 4. Unlike in the basic algorithm (Line 18 of Alg. 1), we do not always expand a new child even if there are unexpanded ones; the threshold $C \cdot N(w)^{\alpha}$ is computed using the visit count $N(w)$ and tunable parameters C, α.

Progressive widening is a widely employed technique in MCTS for coping with a large or infinite action set A—in such a case expanding all children incurs a lot of computational cost. See e.g. [31]. In our Alg. 3 the action set A can be quite large, depending on the numbers $L_1, \ldots, L_m$ of input range partitions.

Hill-Climbing Optimization for Expanding Children. In progressive widening, since we may not expand all the children, it makes sense to be selective about which child to expand. This is in contrast with random sampling in Alg. 1 (Line 23). See Line 10 of Alg. 3, where we first playout by hill-climbing optimization. The value $\mathbf{u}_{d+1}$ thus obtained is then used to determine which child wa to expand, in Line 11. In order to ensure that the new child wa is indeed previously unexpanded, the value $\mathbf{u}_{d+1}$ is sampled from the set $\bigcup_{wa' \notin \mathcal{T}} \mathrm{Reg}(a')$; in fact we restrict to its convex subset (Line 9) because many hill-climbing optimization algorithms work best in a convex domain. See Fig. 6 for illustration.

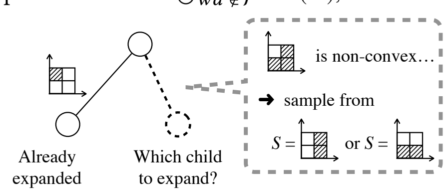

Figure 6: Lines 9–11 of Alg. 3

3.3 Discussion

Our algorithms interleave MCTS optimization and hill-climbing optimization: the latter is used in the playout operation of the former, for sampling and estimating the reward of a high-level input-synthesis strategy. This high-level strategy is concretely given by a sequence $a_1 a_2 \ldots a_d$ of input regions. Via the UCT tree search strategy, we ensure that our search in a search tree is driven not only by depth but also by width. This way we enhance exploration in search-based falsification, in the sense that different regions of the input space are sampled in a structured and disciplined manner. It is an interesting topic for future work to quantify the *coverage guarantees* that can potentially be achieved by our approach.

In falsification of hybrid systems, it is often the case that *simulation*, i.e. running a model $\mathcal{M}$ under a given input signal, is computationally the most expensive operation. In our algorithm it happens in Lines 25 and 37, since a hill-climbing optimization algorithm tries many samples of $\mathbf{u}_1, \ldots, \mathbf{u}_K$. Simplifying Line 25, e.g. by decimating the control points, can result in a useful variation of our algorithm.

Among the tunable parameters of the algorithm is the scaler c used for the UCB sampling (Line 2 of Alg. 2). Having this parameter is unique to our falsification framework in comparison to plain robustness guided optimization (with hill-climbing only). Specifically, the parameter c endows our algorithm with *flexibility* in the exploration-exploitation trade-off. Given the diversity of instances of the hybrid system falsification problem, it is unlikely that there is a single value of c that is optimal for all falsification examples. An engineer can then use her/his expert domain knowledge about the example to tune the parameter c.

4 EXPERIMENTAL EVALUATION

We have implemented our algorithms—the *basic* algorithm (Alg. 1, henceforth called "basic") and the variation with *progressive widening* (Alg. 3, henceforth called "P.W."). The implementation is in MATLAB, using Breach [13] as a front-end for hill-climbing optimization and for its implementation of the robust semantics.

The experiments have two goals. Firstly, in §4.2, we evaluate the falsification performance of our proposal in comparison to the state-of-the-art. Since our MCTS enhancement emphasizes coverage, our interest is in the success rate in hard problem instances rather than in execution time. Secondly, in §4.3, we evaluate the impact of different choices of parameters for our algorithms (such as the UCB scaler c in the Alg. 2).

4.1 Experiment Setup

The experiments are based on the following benchmarks.

The *automatic transmission* (AT) model is a Simulink model that was proposed as a benchmark for falsification in [24]. It has input signals *throttle* $\in [0, 100]$ and *brake* $\in [0, 325]$, and computes the car's speed *speed*, the engine rotation *rpm*, and the selected gear *gear*. We consider the following specifications, taken in part from [24].

S1 $\equiv \Box_{[0,30]}$ (*speed* < 120) can be falsified easily by hill-climbing with an input *throttle* $= 100$ and *brake* $= 0$ throughout.

S2 $\equiv \Box_{[0,30]}$ (*gear* $= 3 \rightarrow$ *speed* ≥ 20) states that in gear three, the speed should not get too low. The difficulty arises from the lack of guidance by robustness as long as *gear* $\neq 3$: we follow [24] and take *gear* $= 1, \ldots,$ *gear* $= 4$ as Boolean propositions, instead of taking *gear* as a numeric variable. In contrast to [24] we use a more difficult speed threshold of 20 instead of 30.

S3 $\equiv \Diamond_{[10,30]}$ ($speed \notin [53,57]$) states that it is not possible to maintain a constant speed after 10s. A falsifying trace needs precise inputs to hit and maintain the narrow speed range.

S4 $\equiv \Box_{[0,29]}(speed < 100) \vee \Box_{[29,30]}(speed > 65)$ is a specification designed to demonstrate the limitation of robustness-guided falsification by hill-climbing optimization only. Here, a falsifying trajectory has to reach high speed before braking down again. Similarly to S2, the speed 100 has to be reached much earlier than the indicated time bound of 29 to give sufficient time for deceleration. However, the robustness computation shadows either of the disjuncts by using the maximum as semantics for the $\vee$-connective.

S5 $\equiv \Box_{[0,30]}(rpm < 4770 \vee \Box_{[0,1]}(rpm > 600))$ aims to prevent systematic sudden drops from high to low *rpm*. It is falsified if an *rpm* peak above 4770 is immediately followed by a drop to $rpm \leq 600$.

The second benchmark is the *Abstract Fuel Control* (AFC) model [27]. It takes two input signals, *pedal angle* and *engine speed*, and outputs the critical signal *air-fuel ratio* (*AF*), which influences fuel efficiency and car performance. The value is expected to be close to a reference value *AFref*. The pedal angle varies in the range $[0, 61.1]$ and the engine speed varies in the range $[900, 1100]$. According to [27], this setting corresponds to *normal mode*, where $AFref = 14.7$.

The basic requirement of the AFC is to keep the air-to-fuel ratio *AF* close to the reference *AFref*. However, changes to the pedal angle cause brief spikes in the output signal *AF* before the controller is able to regulate the engine. Falsification is used to discover amplitude and periods of such spikes.

The formal specification Sbasic is $\Box_{[11,30]}(\neg(|AF - AFref| > 0.05 * 14.7))$. It is violated when *AF* deviates from its *AFref* too much. Another specification is Sstable: $\neg(\Diamond_{[6,26]}\Box_{[0,4]}(|AF - AFref| > 0.01 * 14.7))$. The goal is to find spikes where ratio is off by a fraction 0.01 of the reference value for at least t' seconds during the interval $[6, 26]$.

The third benchmark model is called *Free Floating Robot* (FFR) that has been considered as a falsification benchmark in [12]. It is a robot vehicle powered by four boosters and moving in a two-dimensional plane. It is governed by the following second-order differential equations:

$$\begin{aligned}
\ddot{x} &= 0.1 \cdot (u_1 + u_3)\cos(\varphi) - 0.1 \cdot (u_2 + u_4)\sin(\varphi) \\
\ddot{y} &= 0.1 \cdot (u_1 + u_3)\sin(\varphi) + 0.1 \cdot (u_2 + u_4)\cos(\varphi) \\
\ddot{\varphi} &= 5/12 \cdot (u_1 + u_3) - 5/12 \cdot (u_2 + u_4)
\end{aligned}$$

Goal of the robot is to steer from $(x, y, \varphi) = (0, 0, 0)$ to $x = y = 4$ with a tolearance of 0.1 such that $\dot{x}$ and $\dot{y}$ are within $[-1, 1]$, given a time horizon of $T = 5$. The four inputs $u_i \in [-10, 10]$ range over the same domain. We run falsification on the negated requirement: Strap $\equiv \neg \Diamond_{[0,5]}\, x, y \in [3.9, 4.1] \wedge \dot{x}, \dot{y} \in [-1, 1]$.

The experiments ran Breach version 1.2.9 and MATLAB R2017b on an Amazon EC2 c4.large instance (March 2018, 2.9 GHz Intel Xeon E5-2666, 2 virtual CPU cores, 4 GB main memory).

4.2 Performance Evaluation

The results are shown in Table 1 and are grouped with respect to the method: uniform random sampling ("Random") as a baseline, Breach our basic algorithm ("Basic," Alg. 1) and its variation with progressive widening ("P.W.," Alg. 3), as well as the underlying hill-climbing optimization solver (CMA-ES, GNM and SA). Run times are shown in seconds. Since the algorithms are stochastic, we give the success rate out of a number of trials.

For all the experiments, input signals are chosen to be piecewise constant with $K = 5$ control points for AT and AFC, and $K = 3$ control points for FFR due to the shorter time horizon. These numbers coincide with the depth of the MCTS search trees. In Breach, this is achieved with the "UniStep" input generator with its `.cp` attribute set to K. The timeout for Breach was set to 900 seconds (which is well above all successful falsification trials) with no upper limit on the number of simulations. For our P.W. algorithm, we used the parameters $C = 0.7$ and $\alpha = 0.85$ (Line 4 of Alg. 3).

The choice of parameters for our two MCTS-based algorithms is as follows: for each combination with the hill-climbing optimization solvers, we present a set of parameters that gave good results over all the specifications. This is justified, because the performance is quite dependent on these parameters, and one choice that works for a given combination of a falsification algorithm and a hill-climbing solver might just not work for another combination. However, note that we do *not* change the settings across the specifications.

As we discussed at the end of §3.1, different timeouts are set for hill-climbing in playout (Line 25 of Alg. 1) and to hill-climbing in the end (Line 37 of Alg. 1). Specifically, the timeout for the former is $\mathrm{TO_{po}}$ in Table 1 (5–15 seconds) while the timeout for the latter is globally 300 seconds.

The results in Table 1 indicate, at a high-level, that for seemingly hard problems, the benefit of the extra exploration done by the MCTS layer significantly increases the falsification rate. This is most evident in S4 and S5, where Breach (with any of CMA-ES, GNM or SA) has at most 30–40% success rates. Our MCTS enhancements succeed much more often.

For easy problems, the increased exploration typically increases the falsification times somewhat, which is expected. One reason is that falsification is in general a hard problem that can only be tackled by heuristics. We note from Table 1 that the additional execution time is not prohibitively large, such as S1 and Sbasic. Actually there is generally no single algorithm that works on all instances equally well. For example, for Sstable, both Breach and our algorithms are even weaker than random testing. However, our algorithms still increase the falsification rate compared to Breach.

The choice of a hill-climbing optimization solver has a great influence on the outcome. CMA-ES has built-in support for some exploration before the search converges in the most promising direction. Nevertheless, we see that the upper-layer optimization by MCTS can improve success rates (S4, S5, Sstable). The Nelder-Mead variant GNM has very little support for exploration and furthermore, Breach's implementation is not stochastic (it uses deterministic low-discrepancy sequences as a source of quasi-randomness). For this reason, the method quickly converges to non-falsifying minima that are local and cannot be escaped without extra measures. Thus, using MCTS pays off especially with GNM; see for example S3 and S4. Conversely, SA heavily relies on exploration and keeps just a single good trace found so far, limiting its exploitation. In combination with MCTS, SA shows mixed performance. In some cases falsification time becomes longer (S1, S3), whereas for S4, MCTS is able to overcome this particular limitation, presumably as it maintains

Table 1: Comparison of uniform random sampling and Breach against Algs. 1 (Basic) and 3 (P.W.). For each specification, we show the success rate out of 10 trials, and the average run time (in seconds) of those trials which were successful. The respective parameters are shown in the leftmost columns: M_b (MCTS budget) is the maximum visit count for the root of the MCTS search tree (i.e. the maximum number of nodes of the tree); $\mathrm{TO_{po}}$ (in seconds) is the timeout (wall-clock) for each individual MCTS playout by hill-climbing optimization; c is the scaler for the exploration-exploitation trade-off in UCB (Alg. 2). The number K of control points is 5 for AT and AFC, and 3 for FFR. The partitioning L of the input space w.r.t. each dimension is 3×5 for AT (throttle, brake) and AFC (pedal, engine), and $2 \times 2 \times 2 \times 2$ for FFR (u_1, u_2, u_3, u_4). For progressive widening (Alg. 3) we use the parameters $C = 0.7$ and $\alpha = 0.85$. Timeout for the hill-climbing in the end (Line 37 of Alg. 1) is 300 seconds. For random testing, timeout is 900s. The cells with bold fonts are local best performers w.r.t. each hill-climbing solver, and green backgrounded cells are the global performers w.r.t. each property. Here, the ranking criterion takes success rate as first priority, and average time as second priority.

		Parameters			AT model										AFC model				FFR model	
					S1		S2		S3		S4		S5		Sbasic		Sstable		Strap	
	Algorithm	M_b	$\mathrm{TO_{po}}$	c	succ.	time	succ.	time	succ.	time	succ.	time	succ.	time	succ.	time	succ.	time	succ.	time
	Random				10/10	108.9	10/10	289.1	1/10	301.1	0/10	-	0/10	-	6/10	278.7	10/10	242.6	4/10	409.3
CMA-ES	Breach				10/10	21.9	6/10	30.3	**10/10**	**193.9**	4/10	208.8	3/10	75.5	**10/10**	**111.7**	3/10	256.3	**10/10**	**119.8**
	Basic	40	15	0.20	10/10	15.8	10/10	108.5	10/10	697.1	7/10	786.8	9/10	384.4.	10/10	182.0	7/10	336.9	10/10	338.0
	P.W.	40	15	0.20	**10/10**	**10.8**	**10/10**	**65.7**	10/10	728.6	**7/10**	**767.8**	**10/10**	**648.1**	10/10	177.1	**8/10**	**272.9**	10/10	473.9
GNM	Breach				**10/10**	**5.4**	**10/10**	**151.4**	0/10	-	0/10	-	0/10	-	**10/10**	**171.4**	0/10	-	0/10	-
	Basic	20	5	0.20	10/10	12.4	10/10	162.3	**10/10**	**185.6**	7/10	261.9	7/10	163.7	10/10	227.1	2/10	378.5	**10/10**	**162.2**
	P.W.	20	5	0.05	10/10	60.8	9/10	110.7	8/10	211.2	**8/10**	**313.0**	**10/10**	**178.7**	10/10	252.0	**6/10**	**153.2**	6/10	197.4
SA	Breach				**10/10**	**160.1**	0/10	-	3/10	383.7	0/10	-	3/10	80.4	0/10	-	6/10	307.0	3/10	92.8
	Basic	20	15	0.05	10/10	264.8	9/10	236.1	**8/10**	**385.6**	**8/10**	**505.3**	7/10	341.2	5/10	391.3	**8/10**	**273.8**	**10/10**	**273.2**
	P.W.	40	15	0.20	10/10	208.7	**10/10**	**377.6**	8/10	666.0	7/10	795.4	**10/10**	**624.2**	**8/10**	**665.7**	6/10	293.7	10/10	390.9

several good prefixes. For the free floating robot, we observe that our approach needs additional time in comparison to plain Breach with CMA-ES (within an order of magnitude), which is reasonable given the added exploration on the exponentially larger state space. However, it does increase the falsification rate with GNM and SA, and the reason is the same as we analyzed before.

The difference between the two variants, Algs. 1 and 3 (the latter with progressive widening), is not significant on most of the examples. However, progressive widening has a positive effect on the success rate and falsification time for S2 and S5.

In the experiments we set the MCTS budget (number of iterations of the main loop) to be 20–40. Note that the number of all possible nodes is way greater: it is $(1 + |A| + |A|^2 + \cdots + |A|^K)$. For AT and AFC (2 input signals, $L = 3 \times 5$ and $K = 5$), it is 813616; and for FFR (4 input signals, $L = 2 \times 2 \times 2 \times 2$ and $K = 3$), it is 4369. The overall success rates seem to suggest that, not only in computer Go but also in hybrid system falsification, MCTS is very effective in searching in a vast space with limited resources.

4.3 Evaluation of Parameter Choices

We evaluate the effect of the parameters using the specification S4 for the AT model, where the success of falsification varies strongly. For the experiments in this section we focus on Alg. 1 (Basic).

Table 2 contains 4 sub-tables, each showing the results for the different optimization solvers when varying a hyperparameter.

The first concern is about the scaler c for exploration/exploitation. We observe that there is a general trend that falsification rate improves with increased focus on exploration. It is particularly evident when comparing the results of $c = 0.02$ and $c = 0.5$. However, no essential performance gap is observed between $c = 0.5$ and $c = 1.0$, indicating that $c = 0.5$ is already sufficient for optimization solvers to benefit from exploration.

Next, consider the results for different partitioning of the input space, where $L = n \times m$ means that the throttle range is partitioned into n actions and the brake range into m actions (for the AT model; pedal and engine for the AFC model). We note that the different choices have by far less influence than the scaler c. However, there are some differences, for example GNM seems to cope badly with the coarse partitioning 2×2 in the first column, which could be attributed to its reliance on guidance by the MCTS layer.

With respect to the timeout for individual playouts $\mathrm{TO_{PO}}$, we observe that it is correlated with overall falsification time. This is expected, as we spend more time in non-falsifying regions of the input space as well.

Varying the number of control points K (and therefore the depth of the MCTS tree), shows that for the respective requirement, $K = 3$ is insufficient but the results for more control points are not clear. As more control points make the problem harder due to the larger search space, the falsification rate drops (specifically for $K = 10$). Note that purposely we kept the MCTS budget and playout time consistent to expose this effect, whereas in practice one might want to increase the limits when the problem is more complex.

5 RELATED WORK

Formal verification approaches to correctness of hybrid systems employ a wide range of techniques from model checking, theorem proving, rigorous numerics, nonstandard analysis, and so on [8, 16, 19–21, 23, 35]. Currently these are not very successful in dealing

Table 2: Parameter Variation for Alg. 1 (Basic) Success rate and average time (in seconds, only successful trials) for 4 parameter variations, respectively scaler c, input space partition L, playout timeout $\mathrm{TO}_{\mathrm{po}}$ and the number K of control points. The default parameter settings are: maximum tree size (MCTS budget) is 60, and $c = 0.2$, $L = 2$, $\mathrm{TO}_{\mathrm{po}} = 10$, $K = 5$ (gray headed columns). The green backgrounded cells are the best performers w.r.t. each solver.

	$c = 0.02$		$c = 0.2$		$c = 0.5$		$c = 1.0$	
Solver	succ.	time	succ.	time	succ.	time	succ.	time
CMA-ES	6/10	826.1	7/10	728.7	8/10	725.7	9/10	744.3
GNM	0/10	-	4/10	807.3	3/10	779.4	3/10	791.4
SA	1/10	719.5	8/10	733.5	9/10	736.3	8/10	799.1
	$L = 2 \times 2$		$L = 3 \times 3$		$L = 3 \times 5$		$L = 5 \times 5$	
Solver	succ.	time	succ.	time	succ.	time	succ.	time
CMA-ES	7/10	728.7	9/10	674.4	9/10	740.2	8/10	743.4
GNM	4/10	807.3	3/10	712.3	9/10	721.6	10/10	724.2
SA	8/10	733.5	6/10	755.7	8/10	832.0	6/10	832.8
	$\mathrm{TO}_{\mathrm{po}} = 5$		$\mathrm{TO}_{\mathrm{po}} = 10$		$\mathrm{TO}_{\mathrm{po}} = 15$		$\mathrm{TO}_{\mathrm{po}} = 20$	
Solver	succ.	time	succ.	time	succ.	time	succ.	time
CMA-ES	8/10	431.8	7/10	728.7	9/10	776.2	7/10	1330.1
GNM	3/10	502.6	4/10	807.3	4/10	809.4	2/10	1397.1
SA	7/10	510.5	8/10	733.5	7/10	1108.0	8/10	1342.5
	$K = 3$		$K = 5$		$K = 7$		$K = 10$	
Solver	succ.	time	succ.	time	succ.	time	succ.	time
CMA-ES	0/10	-	7/10	728.7	6/10	711.5	5/10	777.9
GNM	0/10	-	4/10	807.3	1/10	664.3	6/10	892.8
SA	0/10	-	8/10	733.5	8/10	709.7	3/10	750.9

with complex real-world systems, due to issues like scalability and black-box components.

Optimization-based falsification of hybrid systems therefore attracts attention, as a testing technique that adaptively searches for error input using algorithms from recent advances in machine learning. An overview is given in [28].

We now discuss the relationship between the current work and existing works in the context of falsification.

Monte Carlo sampling is used in [33] for falsification. Our thesis is that Monte Carlo *tree* search—an extension of Monte Carlo methods—yields a powerful guiding method in optimization-based falsification.

The so-called *multiple-shooting* approach to falsification is studied in [38]. It consists of: an upper layer that searches for an *abstract error trace* given by a succession of cells; and a lower layer where an abstract error trace is concretized to an actual error trace by picking points from cells. This two-layered framework differs from ours: they focus on safety specifications (avoiding an unsafe set); this restriction allows search heuristics that relies on spacial metrics (such as A^* search). In our current work, we allow arbitrary STL specifications, and we use robustness values as guidance. Our framework can be seen as an integration of multiple-shooting (the upper layer) and single-shooting (the lower layer); they are interleaved in the same way as search and sampling are interleaved in MCTS.

Besides MCTS, *Gaussian process learning (GP learning)* attracts attention in machine learning as a clean way of balancing exploitation and exploration. The GP-UCB algorithm is a widely used strategy there. Its use in hybrid system falsification is pursued e.g. in [3, 37].

The value of exploration/coverage has been recognized in the falsification community [1, 11, 15, 30], not only for efficient search for error inputs, but also for correctness guarantees in case no error input is found. In this line, the closest to the current work is [1] in which search is guided by a coverage metric on input spaces. The biggest difference in the current work is that we structure the input space by time, using time stages (see Fig. 3). We explore this staged input space in the disciplined manner of MCTS. In [1] there is no such staged structure in input spaces, and they use support vector machines (SVM) for identifying promising regions. Underminer [6] is a falsification tool that learns the (non-)convergence of a system to direct falsification and parameter mining. It supports STL formulas, SVMs, neural nets, and Lyapunov-like functions as classifiers.

Tree-based search is also used in [15] for falsification. They use *rapidly-exploring random trees (RRT)*, a technique widely used in path planning in robotics. Their use of trees is geared largely towards exploration, using the coverage metric called *star discrepancy* as guidance. In their algorithm, robustness-guided hill-climbing optimization plays a supplementary role. This is in contrast to our current framework, where we use MCTS and systematically integrate it with hill-climbing optimization.

Many works in coverage-guided falsification [15, 30] use metrics in the space of output or internal states, instead of the input space. A challenge in such methods is that, in a complex model, the correlation between input and output/state is hard to predict. It is hard to steer the system's output/state to a desired region.

There have been efforts to enhance expressiveness of MTL and STL, so that engineers can express richer intentions—such as time robustness and frequency—in specifications [2, 34]. This research direction is orthogonal to ours; we shall investigate use of such logics in the current framework. Other recent works with which our current results could be combined include [25], which mines parameter regions, and [17] that aims to exploit features of machine learning components of system models for the sake of falsification.

We believe that the combination of MCTS and application-specific lower-layer optimization—an instance of which is the proposed falsification framework—is a general methodology applicable to a variety of applications. For example, for the MaxSAT problem, the work [22] uses MCTS combined with hill-climbing local optimization.

Use of MCTS for search-based testing of hybrid systems is pursued in [31]. We differ from [31] in the target systems: ours are deterministic, while [31] searches for random seeds for stochastic systems. We also combine robustness-guided hill-climbing optimization.

Our use of time-ordered MCTS search trees can be seen as a form of *importance sampling*, where we iteratively narrow down a search space to more promising subspaces. Importance sampling is used in [26] for rare events in statistical model checking.

6 CONCLUSIONS AND FUTURE WORK

In this work we have presented a two-layered optimization framework for hybrid system falsification. It combines Monte Carlo tree search—a widely used method in machine learning for effective stochastic search, balancing exploration and exploitation—and hill-climbing optimization—a local search method whose use in hybrid system falsification is established in the community. Our experiments suggest its promising performance.

In §5 we already indicated some directions for future work. Further future directions are as follows.

In this work we demonstrated by experiments how systematic exploration can improve the chances of finding error input. Another use of exploration, namely as the confidence measure about systems' validity in case no error input is found (see [1, 15] and §5), should be further investigated. Concretely, we are interested in computing a quantitative coverage metric from the result of our MCTS algorithm.

Our choice of grid partitioning for actions in MCTS search trees, although simple, achieves good performance. Other choices are possible, using the extension of MCTS to continuous action sets [9, 10]. The effect of taking those other choices shall be investigated.

An extension to stochastic hybrid systems does not seem hard, following the MCTS approach in [31] that uses models with direct access to randomness seeds (see §5).

REFERENCES

[1] Arvind Adimoolam, Thao Dang, Alexandre Donzé, James Kapinski, and Xiaoqing Jin. 2017. Classification and Coverage-Based Falsification for Embedded Control Systems. In *Computer Aided Verification*. Springer International Publishing, Cham, 483–503.

[2] Takumi Akazaki and Ichiro Hasuo. 2015. Time Robustness in MTL and Expressivity in Hybrid System Falsification. In *Computer Aided Verification*. Springer International Publishing, Cham, 356–374.

[3] Takumi Akazaki, Yoshihiro Kumazawa, and Ichiro Hasuo. 2017. Causality-Aided Falsification. In *Proceedings First Workshop on Formal Verification of Autonomous Vehicles, FVAV@iFM 2017, Turin, Italy, 19th September 2017. (EPTCS)*, Vol. 257. 3–18.

[4] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. 2011. *S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems*. Springer, Berlin, Heidelberg, 254–257.

[5] Anne Auger and Nikolaus Hansen. 2005. A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005*. IEEE, 1769–1776.

[6] Ayca Balkan, Paulo Tabuada, Jyotirmoy V. Deshmukh, Xiaoqing Jin, and James Kapinski. 2018. Underminer: A Framework for Automatically Identifying Nonconverging Behaviors in Black-Box System Models. *ACM Trans. Embedded Comput. Syst.* 17, 1 (2018), 20:1–20:28.

[7] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intellig. and AI in Games* 4, 1 (2012), 1–43.

[8] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. 2013. Flow*: An Analyzer for Non-linear Hybrid Systems. In *Computer Aided Verification*. Springer Berlin Heidelberg, Berlin, Heidelberg, 258–263.

[9] Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. 2011. Continuous Upper Confidence Trees. In *Learning and Intelligent Optimization*. Springer, Berlin, Heidelberg, 433–445.

[10] Rémi Coulom. 2007. Computing "Elo Ratings" of Move Patterns in the Game of Go. *ICGA Journal* 30, 4 (2007), 198–208.

[11] Jyotirmoy Deshmukh, Xiaoqing Jin, James Kapinski, and Oded Maler. 2015. Stochastic Local Search for Falsification of Hybrid Systems. In *Automated Technology for Verification and Analysis*. Springer International Publishing, Cham, 500–517.

[12] Jyotirmoy V. Deshmukh, Marko Horvat, Xiaoqing Jin, Rupak Majumdar, and Vinayak S. Prabhu. 2017. Testing Cyber-Physical Systems through Bayesian Optimization. *ACM Trans. Embedded Comput. Syst.* 16, 5 (2017), 170:1–170:18.

[13] Alexandre Donzé. 2010. Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In *Computer Aided Verification, 22nd Int. Conf., CAV 2010 (LNCS)*, Vol. 6174. Springer, 167–170.

[14] Alexandre Donzé and Oded Maler. 2010. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *Formal Modeling and Analysis of Timed Systems - 8th Int. Conf., FORMATS 2010 (LNCS)*, Vol. 6246. Springer, 92–106.

[15] Tommaso Dreossi, Thao Dang, Alexandre Donzé, James Kapinski, Xiaoqing Jin, and Jyotirmoy V. Deshmukh. 2015. Efficient Guiding Strategies for Testing of Temporal Properties of Hybrid Systems. In *NASA Formal Methods*. Springer International Publishing, Cham, 127–142.

[16] Tommaso Dreossi, Thao Dang, and Carla Piazza. 2016. Parallelotope Bundles for Polynomial Reachability. In *Proc. of the 19th International Conference on Hybrid Systems: Computation and Control (HSCC '16)*. ACM, New York, NY, USA, 297–306.

[17] Tommaso Dreossi, Alexandre Donzé, and Sanjit A. Seshia. 2017. Compositional Falsification of Cyber-Physical Systems with Machine Learning Components. In *NASA Formal Methods*. Springer International Publishing, Cham, 357–372.

[18] Georgios E. Fainekos and George J. Pappas. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.* 410, 42 (2009), 4262–4291.

[19] Chuchu Fan, Bolun Qi, Sayan Mitra, Mahesh Viswanathan, and Parasara Sridhar Duggirala. 2016. Automatic Reachability Analysis for Nonlinear Hybrid Models with C2E2. In *Computer Aided Verification*. Springer International Publishing, Cham, 531–538.

[20] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *Computer Aided Verification*. Springer, Berlin, Heidelberg, 379–395.

[21] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. 2012. δ-Complete Decision Procedures for Satisfiability over the Reals. In *Automated Reasoning*. Springer, Berlin, Heidelberg, 286–300.

[22] Jack Goffinet and Raghuram Ramanujan. 2016. Monte-Carlo Tree Search for the Maximum Satisfiability Problem. In *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings (Lecture Notes in Computer Science)*, Vol. 9892. Springer, 251–267.

[23] Ichiro Hasuo and Kohei Suenaga. 2012. Exercises in Nonstandard Static Analysis of Hybrid Systems. In *Computer Aided Verification*. Springer, Berlin, Heidelberg, 462–478.

[24] Bardh Hoxha, Houssam Abbas, and Georgios E. Fainekos. 2014. Benchmarks for Temporal Logic Requirements for Automotive Systems. In *1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems, ARCH@CPSWeek 2014, Berlin, Germany, April 14, 2014 / ARCH@CPSWeek 2015, Seattle, USA, April 13, 2015. (EPiC Series in Computing)*, Vol. 34. EasyChair, 25–30.

[25] Bardh Hoxha, Adel Dokhanchi, and Georgios E. Fainekos. 2018. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *STTT* 20, 1 (2018), 79–93.

[26] Cyrille Jegourel, Axel Legay, and Sean Sedwards. 2012. Cross-Entropy Optimisation of Importance Sampling Parameters for Statistical Model Checking. In *Computer Aided Verification*. Springer, Berlin, Heidelberg, 327–342.

[27] Xiaoqing Jin, Jyotirmoy V. Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. 2014. Powertrain Control Verification Benchmark. In *Proc. of the 17th Int. Conf. on Hybrid Systems: Computation and Control (HSCC '14)*. ACM, NY, USA, 253–262.

[28] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts. 2016. Simulation-Based Approaches for Verification of Embedded Control Systems: An Overview of Traditional and Advanced Modeling, Testing, and Verification Techniques. *IEEE Control Systems* 36, 6 (Dec 2016), 45–64.

[29] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML 2006*. Springer, Berlin, Heidelberg, 282–293.

[30] Jan Kuřátko and Stefan Ratschan. 2014. Combined Global and Local Search for the Falsification of Hybrid Systems. In *Formal Modeling and Analysis of Timed Systems*. Springer International Publishing, Cham, 146–160.

[31] Ritchie Lee, Mykel J Kochenderfer, Ole J Mengshoel, Guillaume P Brat, and Michael P Owen. 2015. Adaptive stress testing of airborne collision avoidance systems. In *Digital Avionics Systems Conference, 2015 IEEE/AIAA 34th*. IEEE, 6C2–1.

[32] Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, Berlin, Heidelberg, 152–166.

[33] Truong Nghiem, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancić, Aarti Gupta, and George J. Pappas. 2010. Monte-carlo Techniques for Falsification of Temporal Properties of Non-linear Hybrid Systems. In *Proc. of the 13th ACM Int. Conf. on Hybrid Systems: Computation and Control (HSCC '10)*. ACM, NY, USA, 211–220.

[34] Luan Viet Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy V. Deshmukh, Ken Butts, and Taylor T. Johnson. 2017. Abnormal Data Classification Using Time-Frequency Temporal Logic. In *Proc. of the 20th International Conference on Hybrid Systems: Computation and Control (HSCC '17)*. ACM, New York, NY, USA, 237–242.

[35] André Platzer. 2010. *Logical Analysis of Hybrid Systems - Proving Theorems for Complex Dynamics*. Springer.

[36] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2015. Mastering the game of Go with deep neural networks and tree search. *Nature* 529 (2015), 484–489.

[37] Simone Silvetti, Alberto Policriti, and Luca Bortolussi. 2017. An Active Learning Approach to the Falsification of Black Box Cyber-Physical Systems. In *Integrated Formal Methods*. Springer International Publishing, Cham, 3–17.

[38] Aditya Zutshi, Jyotirmoy V. Deshmukh, Sriram Sankaranarayanan, and James Kapinski. 2014. Multiple shooting, CEGAR-based falsification for hybrid systems. In *2014 International Conference on Embedded Software, EMSOFT 2014, New Delhi, India, October 12-17, 2014*. ACM, 5:1–5:10.