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

Zhenya Zhang, Gidon Ernst, Sean Sedwards, Paolo Arcaini, and Ichiro Hasuo

Abstract—Few 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.

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

I. INTRODUCTION

A. 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.

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 $M$ (that takes an input signal $u$ and yields an output signal $M(u)$), and a specification $\varphi$ (a temporal formula)
- Find: an error input, that is, an input signal $u$ such that the corresponding output $M(u)$ violates $\varphi$

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 [1]. Instead of the Boolean satisfaction relation $v \models \varphi$, robust semantics assigns a quantity $[v, \varphi] \in \mathbb{R} \cup \{-\infty, -\infty\}$ that tells us, not only whether $\varphi$ 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 $u$) based on previous observations. One can use stochastic algorithms for optimization, such as simulated annealing (SA), globalized Nelder-Mead (GNM [2]) and covariance matrix adaptation evolution strategy (CMA-ES [3]), which turn out to be much more scalable than model checking algorithms that rely on exhaustive search. Note also that the system model $M$ can be black box: observing the correspondence between input $u$ and output $M(u)$ is enough. Observing an error $M(u')$ for some input $u'$ 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 [1] and has been actively pursued ever since [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]. See [15] for a recent survey. There are now mature tools, such as Breach [8] and S-Talio [4], which work with industry-standard Simulink models.

B. The Exploration-Exploitation Trade-off in Falsification

In optimization-based falsification, the important role of coverage is advocated by many authors [7], [6], [5], [10] (see
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 Table 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).

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). We integrate hill-climbing optimization in MCTS, and obtain a two-layered optimization framework.

MCTS is an expected outcome algorithm that searches a tree whose nodes are usually organized on a cause-effect basis, interleaving search (walking down the already expanded tree in a promising direction) with playout (expanding a new node and estimating its reward). One reason for the success of MCTS is that its search strategies balance exploration and exploitation. The most common search strategy, UCT (UCB applied to trees), is derived from the solid theoretical background of the UCB (upper confidence bounds) strategy for multi-armed bandit problems. Typical applications allowing such a structured search space are decision problems, such as games. In particular, MCTS is attracting a lot of attention thanks to its success in computer Go.

While MCTS is a relatively new methodology, it has already established its position in the rapidly growing community of machine learning. See [20] 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. [13]—is supervised by MCTS, with MCTS dictating which region to sample from. By expanding new children, MCTS can tell the hill-climbing optimization to try an input region that has not yet been explored, or to exploit and dig deep in a direction that seems promising. This 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 [1] for further discussion.

Our use of MCTS depends on our time-staged approach to falsification, in which we synthesize input segments one after another. Those input segments are for the time intervals \([0, \frac{T}{K}), (\frac{T}{K}, \frac{2T}{K}), \ldots, (\frac{(K-1)T}{K}, T]\), where \(T\) is the time horizon. The 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 [8]. Our experiments with benchmarks from [22], [23], [24] demonstrate the possible performance improvements, especially in the ability of finding rare counterexamples.

**Organization:** In [I] we formulate the falsification problem. In [II] we present our main contribution, namely a two-layered optimization framework for falsification that combines MCTS and hill-climbing. Our experimental results are in [IV].

In [V] 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 [VI] 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 as \([0, 2]\) and \((2, 3); [0, 2) = \{x \in \mathbb{R} | 0 \leq x < 2\}\) is a half-closed half-open interval. For a set \(X\), \(|X|\) denotes its cardinality.

## II. Problem: Hybrid System Falsification

We formulate the problem of hybrid system falsification. We also introduce robust semantics of temporal logics [1], [9] 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 $w: [0, T] \rightarrow \mathbb{R}^m$.

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

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

**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 $u: [0, T] \rightarrow \mathbb{R}^m$ and returns a signal $\mathcal{M}(u): [0, T] \rightarrow \mathbb{R}^n$. 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) [26]. We do so for simplicity of presentation; we can also use more expressive logics such as the one in [27].

Here are some intuitions and consequences of the definition. 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: $[w, \varphi] > 0$ implies $w \models \varphi$, and $[w, \varphi] < 0$ implies $w \not\models \varphi$, see [1] Prop. 16. Optimization-based falsification via robust semantics hinges on this refinement. Although the definitions so far are for time-bounded signals only, we note that the robust semantics $[w, \varphi]$, as well as the Boolean satisfaction $w \models \varphi$, 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 [2] In particular, its use of real-valued robust semantics enables hill-climbing optimization. See Fig. 1.

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

### III. Two-Layered Optimization Framework with Monte Carlo Tree Search

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) [20]** for high-level planning in the upper layer; and hill-climbing optimization (such as SA, GNM [2] and CMA-ES [3]) for local input search in the lower layer. See Fig. 2 for a schematic overview. The upper layer steers the lower layer using the UCT strategy [17], an established method in machine learning for balancing exploration and exploitation.

We present two algorithms: the basic two-layered algorithm (Alg. 7), and a version enhanced with **progressive widening** (Alg. 5). The auxiliary functions used therein are presented in Alg. 2. Our algorithms work on an MCTS search tree, as illustrated in Fig. 4.

**A. The Basic Two-Layered Algorithm (Alg. 7)**

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

**a) Time Staging:** We search for a falsifying input signal, focusing on piecewise-constant signals (Fig. 3) left. The interval $[0, T]$ is divided into $K$ equal sub-intervals ($K$ is a tunable parameter). The time points $0, \frac{T}{K}, \frac{2T}{K}, \ldots, \frac{(K-1)T}{K}$ at which those intervals start are called **control points**. Our goal is therefore to find a sequence $u_1, \ldots, u_K$, where each
Algorithm 1 Basic Two-Layered Algorithm

Require: a system model $\mathcal{M}$, an STL formula $\varphi$, intervals $I_i = [u_i^{\text{min}}, u_i^{\text{max}}]$, $i \in \{1, \ldots, M\}$, for the ranges of input $u_1, \ldots, u_M$ 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 $[u_i^{\text{min}}, u_i^{\text{max}}]$ for each $i \in \{1, \ldots, M\}$, the scalar $c$ in Line 2 of Alg. 2 and an MCTS budget (the maximum number of MCTS samples, Line 9).

1: function MCTSPREPROCESS
2: $A \leftarrow \{1, \ldots, L_1\} \times \cdots \times \{1, \ldots, L_M\}$
3: $T \leftarrow \{\varepsilon\}$
4: $N \leftarrow (\varepsilon \mapsto 0)$
5: $R \leftarrow (\varepsilon \mapsto \infty)$
6: $\overline{u} \leftarrow \text{null}$
7: $R_{\text{min}} \leftarrow \infty$
8: $\overline{a}_{\text{min}} \leftarrow \text{null}$
9: while $R(\varepsilon) \geq 0$ and within the MCTS budget do
10:   MCTSSAMPLE($\varepsilon$)
11:   if $\overline{u} \neq \text{null}$ then
12:      return $\overline{u}$
13:   else
14:      return $\overline{a}_{\text{min}}$
15:   end if
16:   function MCTSSAMPLE($w$)
17:   $N(w) \leftarrow N(w) + 1$
18:   if $|w| < K$ then
19:      if $wa' \in T$ for all $a' \in A$ then
20:         $a \leftarrow \text{UCBSAMPLE}(w)$
21:         MCTSSAMPLE($wa$)
22:         $R(w) \leftarrow \min_{a' \in A} R(wa')$
23:      else
24:          randomly sample $a \in A$ from $\{a \mid wa \notin T\}$
25:          $T \leftarrow T \cup \{wa\}$
26:          $u_1, \ldots, u_K \leftarrow \arg \min_{u_1 \in \text{REG}(a_1), \ldots, u_K \in \text{REG}(a_K)} \mathbb{E}[M(u_1, \ldots, u_K), \varphi]$
27:          $N(wa) \leftarrow 0$
28:          $R(wa) \leftarrow \mathbb{E}[M(u_1, \ldots, u_K), \varphi]$
29:          if $R(wa) < 0$ then
30:             $\overline{u} \leftarrow u_1, \ldots, u_K$
31:          end if
32:      end if
33:   end if
34:   return $\overline{a}_{\text{min}}$
35: end function

36: function MAIN
37: $\overline{w} \leftarrow \text{MCTSPREPROCESS}$
38: if $\overline{w} = \overline{u}$, an input signal then
39:   return $\overline{u}$
40: else
41:   return $\arg \min_{u_1 \in \text{REG}(a_1), \ldots, u_K \in \text{REG}(a_K)} \mathbb{E}[M(u_1, \ldots, u_K), \varphi]$
42: end if

43: return $\overline{w} = a_1 a_2 \ldots a_{K'}$ with some $K' \leq K$, Line 13
Algorithm 2 Auxiliary Functions for Algs. [1 & 3]

1: function UCBSAMPLE(w)
2: \[ \text{return arg max}_{a \in A} \left( 1 - \frac{R(wa)}{\max_{w' \in T} R(w')} + \epsilon \sqrt{\frac{2 \ln N(w)}{N(wa)}} \right) \]
3: function REG(a) \quad \triangleright \text{The input region for an action } a \in A \text{ is of the form } (k_1, \ldots, k_M), \text{ see Line 2 of Alg. 1}
4: \[ \text{return } \prod_{i=1}^{M} u_i^{\min} + \frac{k_i-1}{L_i}(u_i^{\max} - u_i^{\min}), \quad u_i^{\min} + \frac{k_i}{L_i}(u_i^{\max} - u_i^{\min}) \]

Algorithm 3 Two-Layered Algorithm with Progressive Widening

Require: The same data as required in Alg. [1] and additionally, constants \( C, \alpha \) (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)
2: \[ N(w) \leftarrow N(w) + 1 \]
3: if \(|w| < K\) then
4: \[ a \leftarrow \text{UCBSAMPLE}(w) \quad \triangleright \text{progressive widening: all or enough children expanded} \]
5: \[ \text{MCTSSAMPLE}(wa) \]
6: \[ R(w) \leftarrow \min_{a' \in A} R(wa') \quad \triangleright \text{recursive call} \]
7: else
8: \[ S \leftarrow (\text{a maximal convex subset of } \bigcup_{w' \in T} \text{REG}(a')) \]
9: \[ u_1, \ldots, u_K \leftarrow \arg \min_{\text{HillClimb}} \left[ M(u_1 \ldots u_K), \varphi \right] \quad \triangleright \text{playout by hill-climbing} \]
10: \[ a \leftarrow (a \in A \text{ such that } u_{d+1} \in \text{REG}(a)) \]
11: \[ T \leftarrow T \cup \{wa\} \]
12: \[ N(wa) \leftarrow 0 \]
13: \[ R(wa) \leftarrow \left[ M(u_1 \ldots u_K), \varphi \right] \]
14: if \( R(wa) < 0 \) then
15: \[ \overrightarrow{u} \leftarrow u_1 \ldots u_K \]
16: \[ \text{if } R(wa) < R_{\text{min}} \text{ then} \]
17: \[ R_{\text{min}} \leftarrow R(wa) \]
18: \[ \overrightarrow{a}_{\text{min}} \leftarrow a_1 \ldots a_d \]
19: \[ R(w) \leftarrow \min_{a' \in A} R(wa') \quad \triangleright \text{back-propagation} \]
20: \[ \text{back-propagation} \]

An edge in our search tree represents a choice of an input region—from which we choose the input value \( u_i \)—for a single control point \( \frac{(-1)^T}{K} \). 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 \( w = a_1a_2 \ldots a_j \) over the alphabet \( A \), where \( j \leq K \). That is: the root is \( \varepsilon \) (the empty word), its child in the direction \( a_1 \in A \) is \( a_1 \), its children are \( a_1a_1, a_1a_2, \ldots \), 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).

b) 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 \( I_1 \times \ldots \times I_M \). We partition the input space into \( L_1 \times \ldots \times L_M \) hypercubes of equal size, according to predetermined parameters \( L_1, \ldots, L_M \), 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. 3, 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.
child means adding the child to $T$. We repeat MCTS sampling until a counterexample is found, or the MCTS budget is used up after the maximum number of iterations (Line 9).

The exploration-exploitation trade-off in MCTS comes in the choice of the node to add. In each MCTS sampling, we start from the root (Line 10), walk down the tree $T$, 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 [17], the most commonly used strategy in MCTS, to resolve the dilemma. UCT is based on the UCB strategy for the multi-armed bandit problems [18]. In Alg. 2, line 2 follows UCB, where the exploitation score

$$1 - \frac{R_1(w)}{\max_{w' \in T} R(w')}$$

and the exploration score $\sqrt{\frac{2 \ln N(w)}{N_1(w)}}$ are superposed using a scalar $c$. Recall that our rewards $R(w)$ 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 the exploitation score

$$1 - \frac{R_1(w)}{\max_{w' \in T} R(w')}$$

therefore, we normalize rewards to the interval $[0, 1]$, and reverse their order.

The exploration score

$$\sqrt{\frac{2 \ln N(w)}{N_1(w)}}$$

is taken from UCB: the visit count $N(w)$ gives how many times the node $w$ has been visited, that is, how many offspring the node $w$ currently has in $T$. The scalar, for the trade-off, is a tunable parameter, as usual in MCTS.

d) Playout and Back-Propagation: In MCTS, the reward of a newly expanded node $a_1a_2\ldots a_d$ (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: $a_1\ldots a_d$, $a_1\ldots a_{d-1}$, ..., and finally $\varepsilon$.

In our MCTS algorithms for falsification we use hill-climbing optimization (e.g. SA, GNM and CMA-ES) for playout. See Line 25 where input values $u_1, u_2, \ldots, u_K$ are sampled by stochastic hill-climbing optimization, so that the resulting robustness value of the specification $\varphi$ becomes smaller. The regions from which to sample those values are dictated by the MCTS tree: $u_1 \in \text{REG}(a_1), \ldots, u_d \in \text{REG}(a_d)$ follow the actions $a_1, \ldots, a_d$ determined so far (here REG is from Alg. 2): $u_{d+1} \in \text{REG}(a)$ follows the newly chosen action $a$ (Line 23); and the remaining values $u_{d+2}, \ldots, u_K$ can be chosen from the whole input range $I_1 \times \cdots \times I_M$.

Fig. 5 illustrates an example of playout by hill-climbing optimization. Smaller gray squares represent actions, and red dots represent input values (notice that they are chosen from the gray regions). The values $u_1, \ldots, u_K$ are sampled repeatedly so that the robustness value $\|\mathcal{M}(u_1 \ldots u_K), \varphi\|$ becomes smaller.

An intuition of this playout operation is that we sample the best input signal, $u_1 \ldots u_K$, 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 27). If $R_1(wa) < 0$ then this means we have already succeeded in falsification (Line 29).

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 24 and 25. Note that, via recursive calls of MCTSSAMPLE (Line 20), the result of playout is propagated to all ancestors.

e) A Two-Layered Framework: In Alg. 1 hill-climbing optimization occurs twice, in Lines 25 and 39. 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 25): its principal role is to find an action sequence $a_{\min}$, 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 $a_{\min}$.

The two occurrences of hill-climbing optimization 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
Fig. 6. Lines 9–11 of Alg. 3

first hill-climbing sampling in Line 25 (TO_pe in IV), while for the second hill-climbing sampling in Line 39 the timeout is 300 seconds.

A falsifying input $\vec{u}$ is often found already in the preprocessing phase. In this case the MAIN function simply returns $\vec{u}$ (Line 37).

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

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

a) Progressive Widening: Alg. 3 uses progressive widening [28]; see Line 4. Unlike in the basic algorithm (Line 13 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 $C, \alpha$.

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. [29]. In our Alg. 3 the action set $A$ can be large, depending on the numbers $L_1, \ldots, L_m$ of input range partitions.

b) 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 to random sampling in Alg. 1 (Line 23). See Line 10 of Alg. 3 where we first playout by hill-climbing optimization. The value $u_{d+1}$ thus obtained is then used to determine which child $w_{a'}$ to expand, in Line 11. In order to ensure that the new child $w_{a'}$ is indeed previously unexpanded, the value $u_{d+1}$ is sampled from the set $\bigcup_{a' \in T} \text{REG}(a')$; in fact, we restrict to its convex subset (Line 9), because many hill-climbing optimization algorithms work best in a convex domain. See Fig. 4 for illustration.

C. 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 $a_1a_2 \ldots a_q$ 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 $M$ under a given input signal, is computationally the most expensive operation. In our algorithm this occurs in Lines 25 and 39 since a hill-climbing optimization algorithm tries many samples of $u_1, \ldots, u_K$. 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 scalar $c$, used for the UCB sampling (Line 2 of Alg. 2). Having this parameter is unique to our falsification framework, in comparison to simple robustness-guided optimization (with hill-climbing only). Specifically, the parameter $c$ endows our algorithm with flexibility in the exploration-exploitation tradeoff. 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 to tune the parameter $c$.

IV. EXPERIMENTAL EVALUATION

We have implemented our basic algorithm (Alg. 1 denoted “Basic”) and our progressive widening algorithm (Alg. 3 denoted “P.W.”) in MATLAB, using Breach [8] as a front-end for hill-climbing optimization and for its implementation of the robust semantics.

The experiments have two goals. Firstly, in IV-B 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 IV-C we evaluate the impact of different choices of parameters for our algorithms (such as the UCB scalar $c$ in Alg. 2).

A. 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 [22]. It has input signals $\text{throttle} \in [0, 100]$ and $\text{brake} \in [0, 325]$, and computes the car’s speed $\text{speed}$, the engine rotation $\text{rpm}$, and the selected gear $\text{gear}$. We consider the following specifications, taken in part from [22].

$S_1 \equiv \square_{[0,30]} (\text{speed} \leq 120)$ can be falsified easily by hill-climbing with an input $\text{throttle} = 100$ and $\text{brake} = 0$ throughout.

$S_2 \equiv \square_{[0,30]} (\text{gear} = 3 \rightarrow \text{speed} \geq 20)$ 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 $\text{gear} \neq 3$: we follow [22] and take $\text{gear} = 1, \ldots, \text{gear} = 4$ as Boolean propositions, instead of taking $\text{gear}$ as a numeric variable. In contrast to [22], we use a more difficult speed threshold of 20 instead of 30.

$S_3 \equiv \Diamond_{[10,30]} (\text{speed} \notin [53, 57])$ 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.

$S_4 \equiv \square_{[0,29]} (\text{speed} < 100) \lor \square_{[29,30]} (\text{speed} > 65)$ 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. 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, by using the maximum as semantics for the \( \lor \)-connective, the robustness computation can shadow either of the disjuncts.

\[ S5 \equiv [0.30] (rpm < 4770 \lor [0.1] (rpm > 600)) \]

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 [23]. 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 \( AF_{ref} \). The pedal angle varies in the range \( [0, 61.1] \) and the engine speed varies in the range \([900, 1100]\). According to [23], this setting corresponds to normal mode, where \( AF_{ref} = 14.7 \).

The basic requirement of the AFC is to keep the air-to-fuel ratio \( AF \) close to the reference \( AF_{ref} \). 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 the amplitude and periods of such spikes.

The formal specification Sbasic is \( []_{[11, 30]} \neg (|AF - AF_{ref}| > 0.05 * 14.7) \). It is violated when \( AF \) deviates from its \( AF_{ref} \) too much. Another specification is Sstable: \( \neg (\hat{\phi}_{[6,20]} []_{[0.4]} |AF - AF_{ref}| > 0.01 * 14.7) \). The goal is to find spikes where the 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 [24]. It is a robot vehicle powered by four boosters and moving in two spatial dimensions. It is governed by the following second-order differential equations:

\[
\begin{align*}
\ddot{x} &= 0.1 \cdot (u_1 + u_3) \cos(\phi) - 0.1 \cdot (u_2 + u_4) \sin(\phi) \\
\ddot{y} &= 0.1 \cdot (u_1 + u_3) \sin(\phi) + 0.1 \cdot (u_2 + u_4) \cos(\phi) \\
\dot{\phi} &= \frac{5}{12} \cdot (u_1 + u_3) - \frac{5}{12} \cdot (u_2 + u_4)
\end{align*}
\]

The goal of the robot is to steer from \((x, y, \phi) = (0, 0, 0)\) to \((x = y = 4, \text{ a tolerance of 0.1, such that } \dot{x} \text{ and } \dot{y} \) 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 required: \( \text{Strap } \equiv \neg \hat{\phi}_{[0, 5]} x, y \in [3.9, 4.1] \land \dot{x}, \dot{y} \in [-1, 1] \).

The experiments use 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).

B. Performance Evaluation

The results are shown in Table I and are grouped with respect to the method: uniform random sampling (“Random”) as a baseline, Breach, our “Basic” algorithm (Alg. 1) and our “P.W.” algorithm (Alg. 5), as well as with respect to 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. 1).

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 give 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 discuss at the end of III-A, different timeouts are set for hill-climbing in playout (Line 25 of Alg. 1) and to hill-climbing at the end (Line 59 of Alg. 1). Specifically, the timeout for the former is \(T_{cp} \) in Table II (5–15 seconds) while the timeout for the latter is globally 300 seconds.

The results in Table I 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, 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 I that the additional execution time is often not prohibitively large. We also note that 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 because it maintains several good prefixes. For
the free floating robot, we observe that our approach needs additional time in comparison to 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, for the same reasons as before.

The difference between the two variants, Algs. 1 and 2 (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 much 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.

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

The first concern is about the scalar \(c\) 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 significant 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 much less influence than the scalar \(c\). However, there are some differences, for example GNM seems to cope badly with the coarse partitioning \(2 \times 2\) 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 \(T_{OP}\), 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.

V. RELATED WORK

Formal verification approaches to correctness of hybrid systems employ a wide range of techniques, including model checking, theorem proving, rigorous numerics, nonstandard analysis, and so on [30], [31], [32], [33], [34], [35], [36]. These are currently not very successful in dealing with complex real-world systems, due to issues like scalability and black-box components.
Optimization-based falsification of hybrid systems has therefore attracted attention as a testing technique that adaptively searches for error input using algorithms, using recent advances in machine learning. An overview is given in [13].

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

Monte Carlo sampling is used in [27] 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 [11]. 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 rely on spacial metrics (such as A* search). In our current work, we allow arbitrary STL specifications and 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) has also attracted 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 [12], [13].

The value of exploration/coverage has been recognized in the falsification community [7], [6], [5], [10], 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 [5], 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 [5] there is no such staged structure in input spaces, and they use support vector machines (SVM) for identifying promising regions. Underminer [33] 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 [10] for falsification. They use rapidly-exploring random trees (RRT), a technique widely used for 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 [7], [10] 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 [27], [39]. This research direction is orthogonal to ours; we are able to investigate the use of such logics in our current framework. Other recent works with which our current results could be combined include [25], which mines parameter regions, and [14] 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 [40] uses MCTS combined with hill-climbing local optimization.

Use of MCTS for search-based testing of hybrid systems is pursued in [29]. We differ from [29] in the target systems: ours are deterministic, while [29] searches for random seeds for stochastic systems. We also combine robustness-guided hill-climbing optimization.

There are strong similarities between our falsification approach using MCTS and statistical model checking (SMC) using importance splitting [41]. The robustness semantics of STL can be seen as a “heuristic score function” [42], with both approaches using time staging and the notion of “levels” to iteratively guide the search to more promising subspaces. The principal difference is that importance splitting randomly explores a diverse set of traces that satisfy the property (in order to reduce the variance of the estimate of its probability).

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

| Solver | $c = 0.02$ | $c = 0.2$ | $c = 0.5$ | $c = 1.0$ |
|--------|------------|------------|------------|------------|
| CMA-ES | succ. time | succ. time | succ. time | succ. time |
| GNM | 6/10 826.1 | 7/10 728.7 | 8/10 725.7 | 9/10 744.3 |
| SA | 1/10 719.5 | 8/10 733.5 | 9/10 736.3 | 8/10 799.1 |

| Solver | $L = 2 \times 2$ | $L = 3 \times 3$ | $L = 5 \times 5$ | $L = 5 \times 5$ |
|--------|-----------------|-----------------|-----------------|-----------------|
| CMA-ES | succ. time | succ. time | succ. time | succ. time |
| GNM | 7/10 728.7 | 9/10 674.4 | 9/10 740.2 | 8/10 743.4 |
| SA | 8/10 733.5 | 6/10 755.7 | 8/10 832.0 | 6/10 832.8 |

| Solver | $T_{\text{po}} = 5$ | $T_{\text{po}} = 10$ | $T_{\text{po}} = 15$ | $T_{\text{po}} = 20$ |
|--------|-----------------|-----------------|-----------------|-----------------|
| CMA-ES | succ. time | succ. time | succ. time | succ. time |
| GNM | 8/10 431.8 | 7/10 728.7 | 9/10 776.2 | 7/10 1330.1 |
| SA | 8/10 710.5 | 4/10 807.3 | 4/10 809.4 | 2/10 1397.1 |

| Solver | $K = 3$ | $K = 5$ | $K = 7$ | $K = 10$ |
|--------|--------|--------|--------|--------|
| CMA-ES | succ. time | succ. time | succ. time | succ. time |
| 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 |
while our falsification approach finds a single falsifying input by optimizing (exploiting) the results of random exploration.

VI. 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 stochastic search method that effectively balances exploration and exploitation—and hill-climbing optimization—a local search method whose use in hybrid system falsification is established in the community. Our experiments demonstrate its promising performance.

In addition to the future work already outlined in [V], we add the following.

We have shown how systematic exploration can improve the chances of finding error input. Such exploration can also be used as a measure of confidence about a system’s validity, in the case that no error input is found (see [5], [10] and [V]). Concretely, it would be interesting to compute a quantitative coverage metric from the result of our MCTS algorithm.

Our choice of simple grid partitioning of actions in MCTS search trees achieves good performance. Other choices are also possible, such as using the extension of MCTS to continuous action sets in [28], [43].

Finally, an extension of our framework to stochastic hybrid systems does not seem hard, following the MCTS approach in [29] that uses models with direct access to randomness seeds.

ACKNOWLEDGMENT

Thanks are due to Georgios Fainekos, Fuyuki Ishikawa, Masaaki Konishi and Akihisa Yamada for helpful discussions. The authors are supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST; and Grants-in-Aid No. 15KT0012, JSPS.

REFERENCES

[1] G. E. Fainekos and G. J. Pappas, “Robustness of temporal logic specifications for continuous-time signals,” Theor. Comput. Sci., vol. 410, no. 42, pp. 4262–4291, 2009.
[2] M. A. Luersen and R. Le Riche, “Globalized Nelder-Mead method for engineering optimization,” Computers & Structures, vol. 82, no. 23, pp. 2251–2260, 2004.
[3] A. Auger and N. Hansen, “A restart CMA evolution strategy with increasing population size,” in Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005. IEEE, 2005, pp. 1769–1776.
[4] Y. Annareddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, S-TalilRo: A Tool for Temporal Logic Falsification for Hybrid Systems. Springer, 2011, pp. 254–257.
[5] A. Adimoolam, T. Dang, A. Donzé, J. Kapinski, and X. Jin, “Classification and coverage-based falsification for embedded control systems,” in Computer Aided Verification. Springer, 2017, pp. 483–503.
[6] J. Deshmukh, X. Jin, J. Kapinski, and O. Maler, “Stochastic local search for falsification of hybrid systems,” in Automated Technology for Verification and Analysis. Springer, 2015, pp. 500–517.
[7] J. Kužík and S. Ratschan, “Combined global and local search for the falsification of hybrid systems,” in Formal Modeling and Analysis of Timed Systems. Springer, 2014, pp. 146–160.
[8] A. Donzé, “Breach: A toolbox for verification and parameter synthesis of hybrid systems,” in Computer Aided Verification, 22nd Int. Conf., CAV 2010, ser. LNCS, vol. 6174. Springer, 2010, pp. 167–170.
[9] A. Donzé and O. Maler, “Robust satisfaction of temporal logic over real-valued signals,” in Formal Modeling and Analysis of Timed Systems - 8th Int. Conf., FORMATS 2010, ser. LNCS, vol. 6246. Springer, 2010, pp. 92–106.
[10] T. Dresassi, T. Dang, A. Donzé, J. Kapinski, X. Jin, and J. V. Deshmukh, “Efficient guiding strategies for testing of temporal properties of hybrid systems,” in NASA Formal Methods. Springer, 2015, pp. 127–142.
[11] A. Zatshi, J. V. Deshmukh, S. Sankaranarayanan, and J. Kapinski, “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, 2014, pp. 5:1–5:10.
[12] T. Akazaki, Y. Kumazawa, and I. Hasuo, “Causality-aided falsification,” in Proceedings First Workshop on Formal Verification of Autonomous Vehicles, FV@iFM 2017, Turin, Italy, 19th September 2017., ser. EPTCS, vol. 257, 2017, pp. 3–18.
[13] S. Silvetti, A. Policriti, and L. Bortolussi, “An active learning approach to the falsification of black box cyber-physical systems,” in Integrated Formal Methods. Springer, 2017, pp. 3–17.
[14] T. Dresassi, A. Donzé, and S. A. Seshia, “Compositional falsification of cyber-physical systems with machine learning components,” in NASA Formal Methods. Springer, 2017, pp. 357–372.
[15] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts, “Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques,” IEEE Control Systems, vol. 36, no. 6, pp. 45–64, Dec 2016.
[16] B. Abramson, “Expected-outcome: a general model of static evaluation,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, no. 2, pp. 182–193, 1990.
[17] L. Kocsis and C. Szepesvári, “Bandit based Monte-Carlo planning,” in Machine Learning: ECML 2006. Springer, 2006, pp. 282–293.
[18] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” Machine Learning, vol. 47, no. 2, pp. 235–256, 2002.
[19] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” Nature, vol. 529, pp. 484–489, 2015.
[20] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavenner, D. P. Liebana, S. Samothrakis, and S. Colton, “A survey of Monte Carlo tree search methods,” IEEE Trans. Comput. Intellig. and AI in Games, vol. 4, no. 1, pp. 1–43, 2012.
[21] G. Ernst, I. Hasuo, Z. Zhang, and S. Sedwards, “Time-staging Enhancement of Hybrid System Falsification,” in Proc. of Symbolic and Numerical Methods for Reachability Analysis (SNR), ser. EPTCS, 2018, to Appear.
[22] B. Hoxha, H. Abbas, and G. E. Fainekos, “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, ARCH@CPSWeek 2015), ser. EPiC Series in Computing. 34, EasyChair, 2014, pp. 25–30.
[23] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts, “Powertrain control verification benchmark,” in Proc. of the 17th Int. Conf. on Hybrid Systems: Computation and Control, ser. HSCC ’14. ACM, 2014, pp. 255–262.
[24] J. V. Deshmukh, M. Horvat, X. Jin, R. Majumdar, and V. S. Prabhu, “Testing cyber-physical systems through bayesian optimization,” ACM Trans. Embedded Comput. Syst., vol. 16, no. 5, pp. 170:1–170:18, 2017.
[25] B. Hoxha, A. Dokhanchi, and G. E. Fainekos, “Mining parametric temporal logic properties in model-based design for cyber-physical systems,” STTT, vol. 20, no. 1, pp. 79–93, 2018.
[26] O. Maler and D. Nickovic, “Monitoring temporal properties of model-based design for cyber-physical systems,” STTT, vol. 20, no. 1, pp. 198–208, 2007.
[27] R. Lee, M. J. Kochenderfer, O. J. Mengshoel, G. P. Brat, and M. P. Owen, “Adaptive stress testing of airborne collision avoidance systems,” in Digital Avionics Systems Conference, 2015 IEEE/AIAA 34th. IEEE, 2015, pp. 6C2:1–6C2:13.
[28] X. Chen, E. Abrahám, and S. Sankaranarayanan, “Flow*: An analyzer for non-linear hybrid systems,” in Computer Aided Verification. Springer, 2013, pp. 258–263.
[29] S. Gao, J. Avigad, and E. M. Clarke, “5-complete decision procedures for satisfiability over the reals,” in Automated Reasoning. Springer, 2012, pp. 286–300.
[32] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, “SpaceEx: Scalable verification of hybrid systems,” in Computer Aided Verification. Springer, 2011, pp. 379–395.

[33] C. Fan, B. Qi, S. Mitra, M. Viswanathan, and P. S. Duggirala, “Automatic reachability analysis for nonlinear hybrid models with C2E2,” in Computer Aided Verification. Springer, 2016, pp. 531–538.

[34] T. Dreossi, T. Dang, and C. Piazza, “Paralleloptope bundles for polynomial reachability,” in Proc. of the 19th International Conference on Hybrid Systems: Computation and Control, ser. HSCC ’16. ACM, 2016, pp. 297–306.

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

[36] I. Hasuo and K. Suenaga, “Exercises in nonstandard Static Analysis of hybrid systems,” in Computer Aided Verification. Springer, 2012, pp. 462–478.

[37] T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivancić, A. Gupta, and G. J. Pappas, “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, ser. HSCC ’10. ACM, 2010, pp. 211–220.

[38] A. Balkan, P. Tabuada, J. V. Deshmukh, X. Jin, and J. Kapinski, “Underminer: A framework for automatically identifying nonconverging behaviors in black-box system models,” ACM Trans. Embedded Comput. Syst., vol. 17, no. 1, pp. 20:1–20:28, 2018.

[39] L. V. Nguyen, J. Kapinski, X. Jin, J. V. Deshmukh, K. Butts, and T. T. Johnson, “Abnormal data classification using time-frequency temporal logic,” in Proc. of the 20th International Conference on Hybrid Systems: Computation and Control, ser. HSCC ’17. ACM, 2017, pp. 237–242.

[40] J. Goffinet and R. Ramanujan, “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, ser. LNCS, vol. 9892. Springer, 2016, pp. 251–267.

[41] ——, “An effective heuristic for adaptive importance splitting in statistical model checking rare properties,” in Computer Aided Verification, ser. LNCS. Springer, 2013, vol. 8044, pp. 576–591.

[42] ——, “An effective heuristic for adaptive importance splitting in statistical model checking,” in Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications, ser. LNCS. Springer, 2014, vol. 8803, pp. 143–159.

[43] A. Couëtoux, J.-B. Hoock, N. Sokolovská, O. Teytaud, and N. Bonnard, “Continuous upper confidence trees,” in Learning and Intelligent Optimization. Springer, 2011, pp. 433–445.