CHALMERS
UNIVERSITY OF TECHNOLOGY

Testing Cyber-Physical Systems Using a Line-Search Falsification Method

Citation for the original published paper (version of record):
Ramezani, Z., Claessen, K., Smallbone, N. et al (2022). Testing Cyber-Physical Systems Using a Line-Search Falsification Method. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41(8): 2393-2406. http://dx.doi.org/10.1109/TCAD.2021.3110740

Testing Cyber–Physical Systems Using a Line-Search Falsification Method

Zahra Ramezani, Koen Claessen, Nicholas Smallbone, Martin Fabian, *Senior Member, IEEE*, and Knut Åkesson, *Associate Member, IEEE*

***Abstract*—Cyber–physical systems (CPSs) are complex and exhibit both continuous and discrete dynamics, hence it is difficult to guarantee that they satisfy given specifications, i.e., the properties that must be fulfilled by the system. Falsification of temporal logic properties is a testing approach that searches for counterexamples of a given specification that can be used to increase the confidence that a CPS does fulfill its specifications. Falsification can be done using random search methods or optimization methods, both of which have their own benefits and drawbacks. This article introduces two methods that exploit randomness to different degrees: 1) the optimization-free Hybrid-Corner-Random (HCR) and 2) the direct-search method Line-Search Falsification (LSF). HCR combines randomly chosen parameter values with extreme parameter values, which performs surprisingly well on benchmark evaluations. The gradient-free optimization-based LSF optimizes over line segments through a vector of inputs in the n-dimensional parameter space. The two methods are compared to the Nelder-Mead and SNOBFIT methods, using a well-known set of benchmark problems and LSF shows better performance than any of the evaluated methods.**

***Index Terms*—Cyber–physical systems (CPSs), simulation-based optimization, testing, falsification.**

I. Introduction

CYBER–PHYSICAL systems (CPSs), [1], bridge the cyber-world of communications and computing to the physical world. These systems exhibit both *continuous* and *discrete* dynamics. CPSs are often safety-critical systems, e.g., autonomous cars and medical devices, hence their correct functioning is crucial. Two commonly used methods for assessing the correctness of CPSs, see [2], are *formal verification* and *testing*. The main problem is to decide if it is possible to construct *counterexamples*, i.e., inputs that make the system under test (SUT) violate the specifications.

Formal verification is used to prove or disprove the correctness of the system with respect to a formal specification. This is typically done using model-checking or deductive verification [3]. However, for many industrial systems, formal verification is not a viable approach because of three reasons: 1) typically there are no formal models available to do verification on; 2) even if formal models were available, the time and space complexity of the verification algorithms makes them intractable for large systems; and 3) even if formal models were available and the algorithmic complexity tractable, verification of systems exhibiting a combination of discrete and continuous dynamics is, in general, an undecidable problem [4]. Though formal verification is both possible and practical for certain types of systems that are limited in their behavior and/or size, for large systems comprising other systems that all have different behavior, it is not a tractable approach.

Testing is a less formal approach that evaluates if the SUT behaves correctly with respect to given inputs. In testing, it is often assumed that the system can be simulated, but it is not assumed that a mathematical model is available. Contrary to formal verification, with testing, it is possible to prove the presence of counterexamples but not their absence. The main challenge with testing is to reduce the number of tests done so as to find *as many* counterexamples for a given specification as *early* as possible. Thus, there is a need for rigorous methods to cleverly search for inputs that break the specification.

Simulation-based falsification approaches [5] are a class of methods that can be used to test CPSs given only a black-box model of the SUT, but with formal specifications of the closed-loop behavior available. These specifications can, for example, be given in metric interval temporal logic (MITL) [6] or signal temporal logic (STL) [7], but the specification can also be defined using high-level languages. Eddeland *et al.* [8] presented how STL specifications can be automatically generated from Simulink charts, hence supporting engineers in the specification of formal specifications without their needing to be trained in temporal logic. The combination of black-box models for the SUT, together with white-box formal specifications, is a reasonable assumption in many industrial applications.

CPSs are typically developed using a model-based development paradigm. The models are often used for control design or optimization, but can also be used in simulation-based falsification. In *simulation-based falsification using optimization*, the goal is to reduce the number of tests

Manuscript received 22 March 2021; revised 17 June 2021; accepted 17 August 2021. Date of publication 6 September 2021; date of current version 19 July 2022. This work was supported in part by the Swedish Research Council (VR) Project SyTeC VR under Grant 2016-06204; in part by the Swedish Governmental Agency for Innovation Systems (VINNOVA) under Project TESTRON 2015-04893; and in part by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. This article was recommended by Associate Editor Q. Zhu. *(Corresponding author: Zahra Ramezani.)*

Zahra Ramezani, Martin Fabian, and Knut Åkesson are with the Department of Electrical Engineering, Chalmers University of Technology, 41718 Gothenburg, Sweden (e-mail: rzahra@chalmers.se; fabian@chalmers.se; knut@chalmers.se).

Koen Claessen and Nicholas Smallbone are with the Department of Computer Science and Engineering, Chalmers University of Technology, 41718 Gothenburg, Sweden (e-mail: koen@chalmers.se; nicsma@chalmers.se).

Digital Object Identifier 10.1109/TCAD.2021.3110740

by using an optimization method to decide on the next set of input signals from the evaluation of the simulation results from the previous simulations. A key challenge here is to choose the optimization method and the kind of information that the optimization method should use to decide on the next set of parameter values. Since the number of input signals might be large, and optimization methods are sensitive to the number of parameters, the allowed input signals are often parameterized using relatively few parameters. For example, for sinusoidal signals, the parameters might be the period and the amplitude.

The falsification process is performed using *quantitative semantics* [9], [10], of the specification. Quantitative semantics define an objective function that expresses a virtual distance to falsifying the specification. Given input and output traces from a simulation, the objective function is used to calculate an objective value that can guide the falsification process toward an input that is more likely to falsify the specification. Claessen *et al.* [11] introduced the concept of valued booleans (VBools) to express different quantitative semantics, e.g., *Max* and *Additive*.

The choice of optimization method affects the efficiency of the falsification. By assuming that the SUT is given as a black-box model, the optimization approach is restricted to black-box simulation-based optimization methods [12], [13], where gradients cannot be analytically computed. Black-box optimization is generally divided into *direct search methods* and *model-based methods*.

Direct-search methods are in [14] defined as the sequential examination of trial solutions generated by a certain strategy. A direct-search method uses and compares only objective function values at a collection of input parameters (points) to directly determine new candidate points for future exploration without any gradient approximation. Direct-search methods include the classic Nelder–Mead (NM) procedure [15].

Model-based search methods build a surrogate model of the objective function to guide the optimization process [13]. Bayesian optimization [16] is a global optimization method that has shown to be efficient for black-box optimization when the objective function is expensive to evaluate. Bayesian optimization is best suited for continuous domains and a relatively moderate number of input parameters. For CPSs, the system dynamics might switch between different discrete modes, each having different system dynamics and constraints, see for example [17]. This behavior of the system dynamics negatively affects the usefulness of a model-based approach for falsification of CPSs, and has to be explicitly handled for a Bayesian optimization approach to be successfully used for simulation-based falsification.

In [18], seventeen gradient-free optimization methods are reviewed and evaluated on a set of benchmark problems. These include both direct-search methods, such as NM and Mesh adaptive direct search algorithms [19], and model-based methods, like SNOBFIT [20]. The results show that no single optimization method consistently outperforms the others, but SNOBFIT was one of the best-evaluated methods.

Simulation-based falsification using different combinations of optimization algorithms and quantitative semantics were evaluated on a set of benchmark problems in [21]. The evaluation showed that quantitative semantics matter but also that the choice of optimization methods is very important. Furthermore, the study showed that NM and SNOBFIT have the best performance, with SNOBFIT as the overall top performer.

Three quantitative semantics, *Max*, *Additive*, and constant, were evaluated in [21]. The latter semantics works by assigning a constant positive value if the specification is fulfilled and a constant negative value if it is falsified. A constant objective value does not hold any information for the optimization methods about how far away from or close to falsifying the specification the current point is. In these experiments, it was observed that when SNOBFIT uses a constant semantics and thus does not get any hints in which direction to continue the search, it tends to explore new parameter values that are toward the extreme values of the allowed parameter ranges. We refer to these extreme values as the *corner points*. Surprisingly, SNOBFIT using a constant objective value performed as well as or better than NM using both *Max* and *Additive* semantics.

Random testing [22] is an optimization-free approach that, despite its simplicity, has proven to be useful for large systems. In [23], random testing is used to test space mission software and was shown to be a good complement to formal verification. In [24], the feedback-directed random test generation had better fault detection and coverage than the structured test generation used in the study. Extensions of random testing include *adaptive random testing* [25], where empirical observations show that many faults occur in contiguous areas of the input domain. In [26], the risks of using coverage-based criteria are discussed where completely randomized test-suites identified 13.5% more faults than test suites that were generated to specifically achieve coverage. Random search has also been explored as a strategy for hyper-parameter optimization in machine learning approaches. Bergstra and Bengio [27] showed that randomized trials are more efficient than uniform trials for hyper-parameter optimization. One reason for this is that only a few parameters really matter for many data sets, but which ones differ from case to case. Together, these observations from practical software systems highlight the benefits of randomness in testing applications.

This article focuses on CPSs with software components integrating with a physical environment typically described using differential equations. Evaluation results for falsification of CPSs in this article show that randomized testing is also an efficient strategy for CPSs. Hence, this article introduces two methods that explore the search space randomly to different degrees. The first one is *Hybrid-Corner-Random* (HCR), an optimization-free method that will serve as a baseline method to compare with. The second one is *Line-Search Falsification*, a direct-search method. The strong dependence on randomness in these methods is motivated by the complex dynamic behavior of the SUT, which complicates model-based optimization approaches. However, as is shown in this article for many falsification problems, it is also useful to consider corner points.

While HCR only relies on random and corner points, LSF combines random exploration with local search, by randomly generating lines in the n-dimensional parameter space for local

search. The implementation of LSF is very small and straightforward, and evaluation on benchmark falsification problems indicates that LSF performs as well as or better than more complex optimization methods.

For simulation-based falsification of CPSs, several toolboxes are available. S-TaLiRo [28] and Breach [29] are both MATLAB/Simulink toolboxes that can be used for the falsification of large-scale industrial systems. S-TaLiRo finds counterexamples using specifications expressed in MITL, and Breach performs falsification using specifications in STL. In this article, Breach is used to evaluate the proposed methods, but they are generic and could be implemented in most other tools.

The main contributions of this article are given in the following.

1) The introduction of LSF, an optimization-based method that exploits corners and random points, combined with local search. These techniques are well chosen for efficient falsification of CPSs.
2) The introduction of the optimization-free method HCR, that combines using corner points of the parameter space with random search. HCR is proposed as a baseline method to benchmark more sophisticated techniques against.
3) The benchmarking of the optimization-free methods, Corners, Random, and HCR with the optimization-based methods, LSF, NM, and SNOBFIT. The benchmarks are published CPS falsification problems [30], [31].

This article is organized as follows: STL and falsification of temporal logic are introduced in Section II. Section III proposes the suggested HCR method. Section IV introduces the LSF method. Section V evaluates the performance of the suggested methods on the chosen benchmark problems. Finally, Section VI summarizes the contributions.

II. Signal Temporal Logic for Falsification

Falsification of temporal specifications needs a *quantitative semantics* that defines an objective function to measure the distance of the specification to being falsified. In optimization-free methods, the objective function is used only to evaluate if the specification is falsified or not. In optimization-based methods, though, the objective function is used to guide the falsification process by choosing the next set of input values such that they are more likely to falsify the specification.

Temporal specifications are typically expressed in linear temporal logic (LTL) [32]. However, LTL cannot reason about time intervals, something that is necessary for many specifications of CPSs. For this, MITL [6] and STL [7] have been introduced. In our work, STL specifications are used, but the proposed methods work with any quantitative semantics, though the performance of the falsification process might depend on the particular quantitative semantics. Thus, the proposed methods can be evaluated using two different quantitative semantics defined for STL, *Max*, and *Additive*. Both of these can be expressed in terms of VBools [11].

A. Signal Temporal Logic

The syntax of STL [7] is defined as follows:

$$\varphi ::= \mu | \neg\mu | \varphi \wedge \psi | \varphi \vee \psi | \Box_{[a,b]}\varphi | \Diamond_{[a,b]}\varphi$$

where the predicate μ is $\mu \equiv \mu(x^s) > 0$ and x^s is a signal; φ and ψ are STL formulas; $\Box_{[a,b]}$ denotes the *globally* operator between times a and b (with $a < b$); and $\Diamond_{[a,b]}$ denotes the *finally* operator between a and b.

The satisfaction of the formula φ for the discrete signal x^s, consisting of both inputs and outputs to the SUT, at the discrete-time instant k is defined as

$$\begin{aligned}
(x^s, k) &\models \mu \Leftrightarrow \mu(x^s[k]) > 0\\
(x^s, k) &\models \neg\mu \Leftrightarrow \neg\big((x^s, k) \models \mu\big)\\
(x^s, k) &\models \varphi \wedge \psi \Leftrightarrow (x^s, k) \models \varphi \wedge (x^s, k) \models \psi\\
(x^s, k) &\models \varphi \vee \psi \Leftrightarrow (x^s, k) \models \varphi \vee (x^s, k) \models \psi\\
(x^s, k) &\models \Box_{[a,b]}\varphi \Leftrightarrow \quad \forall k' \in [k+a, k+b], (x^s, k') \models \varphi\\
(x^s, k) &\models \Diamond_{[a,b]}\varphi \Leftrightarrow \exists k' \in [k+a, k+b], (x^s, k') \models \varphi.
\end{aligned}$$

Instead of only checking the Boolean satisfaction of an STL formula, the notion of quantitative value, i.e., an objective value, will be defined to measure how far away a specification is from being falsified. A VBool [11] $\langle v, y\rangle$ is a combination of a Boolean value v (true $\top$, or false $\bot$) together with a real number y that is a measure of how true or false the VBool is. This value will be used as a measure of how convincingly a test passed, or how severely it failed, respectively. This value is defined by the quantitative semantics.

B. Quantitative Semantics

While *Max* is the most widely used quantitative semantics for STL, it has the disadvantage that for an and-clause only the smallest value will affect the final value, making the other part of the clause invisible. To allow both parts of the clause to affect the final value, *Additive* was introduced in [11], and later evaluated for falsification purposes in [8], [21], [31]. *Additive* is, in general, as good as or better than *Max* for falsification problems. In this paper we thus consider only the *Additive* in the evaluation part.

The *and* operator in *Additive* is defined using VBools as follows.

$$\begin{aligned}
(\top, y) \wedge (\top, s) &= \left(\top, \frac{1}{\frac{1}{y} + \frac{1}{s}}\right),\\
(\top, y) \wedge (\bot, s) &= (\bot, s),\\
(\bot, y) \wedge (\top, s) &= (\bot, y),\\
(\bot, y) \wedge (\bot, s) &= \Big(\bot, (y + s)\Big).
\end{aligned}$$

Using the de Morgan laws, the *or* operator can be defined in terms of *and*, as: $\langle v_y, y\rangle \vee \langle v_s, s\rangle = \neg_{\mathrm{v}}(\neg_{\mathrm{v}}\langle v_y, y\rangle \wedge \neg_{\mathrm{v}}\langle v_s, s\rangle)$, where VBool negation is defined as $\neg_{\mathrm{v}}\langle v_y, y\rangle = \langle \neg v_y, y\rangle$.

always is defined as $\Box_{[a;b]}\varphi = \bigwedge_{k=a}^{b} \varphi[k]\#\delta t$, where φ is a finite sequence of VBools defined for all the discrete time instants in $[a, b]$. δt is the simulation step size that makes the quantitative value independent of the simulation time, and # is defined as

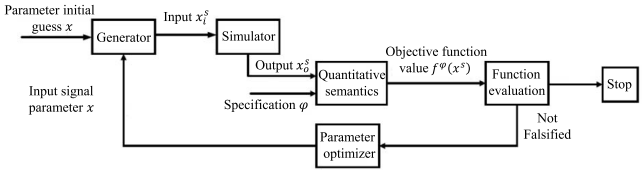

Fig. 1. Flowchart of optimization-based falsification. The input and output signals are denoted by x_i^s and x_o^s, respectively.

$$\langle \bot, y\rangle \# \delta t = \langle \bot, y \cdot \delta t\rangle,$$
$$\langle \top, y\rangle \# \delta t = \langle \top, y/\delta t\rangle.$$

Furthermore, the *eventually* operator is defined over an interval $[a, b]$ in terms of *always*, as: $\Diamond_{[a,b]}\varphi = \neg(\Box_{[a,b]}(\neg_{\mathrm{v}}\varphi))$.

C. Falsification

The main falsification procedure is shown in Fig. 1 for the optimization-based falsification approach. The possible input signals are parameterized with an n-dimensional vector, x, where each element is allowed to be within a defined range. A discrete sequence of inputs is generated by the *Generator* module that, given the parameters x, generates an input trace describing a sequence of input vectors, $x_i^s[k]$, for the input signals. Each element in the sequence is indexed by k, where k ranges from the start of the simulation to the end of the simulation, and we denote the full sequence by x_i^s. Note that the dimension of the input vector x_i^s is often lower than the dimension of x, since multiple parameters are used to parameterize each input signal. The SUT is simulated with the x_i^s as inputs, and its corresponding output x_o^s is generated by the *Simulator*. In the evaluation of specifications both the input and output signals are used, and we denote the combination of x_i^s and x_o^s by x^s.

The input x_i^s and output vectors x_o^s and the specification φ are used together with the objective function $f^\varphi(x^s)$ to evaluate whether the specification φ is falsified or not.

By using a quantitative semantics the specification can be determined to be satisfied or not. If it is satisfied, the semantics will also give a value of how convincingly the test passed. If the specification is not fulfilled, the current set of input and output traces is a counterexample, and thus the falsification process can terminate. However, in the sequel, we will consider the situation where the specification is fulfilled and the aim of the falsification is to lower the value in order to try to find counterexamples.

Let φ be the specification, possibly containing temporal operators. Given φ, the traces x^s, and the choice of quantitative semantics, the evaluation of x^s will return a VBool $\langle v, y\rangle$, where v denotes if φ is satisfied or not, and y is the measure.

Define $f^\varphi(x^s)$ such that

$$f^\varphi(x^s) = \begin{cases} y, & \text{if } v = \top \\ -y, & \text{if } v = \bot. \end{cases}$$

A negative objective function value, $f^\varphi(x^s) < 0$, means that the specification is falsified; thus, the falsification procedure may stop. A non-negative objective function value, $f^\varphi(x^s) \geq 0$, means that the specification is not falsified; it leads to new parameters being sampled, and the process is repeated. The parameter optimizer generates new parameter values within given ranges to find lower objective function values.

The optimization problem is formulated as follows. Let f^φ be the objective function defined above. The lower and upper bounds on the n-dimensional parameter vector x are defined as $l = (l_1, l_2, \ldots, l_n)^T$ and $u = (u_1, u_2, \ldots, u_n)^T$, respectively. The falsification problem is to check if $f^\varphi(x^*) < 0$ where x^* is defined by

$$x^* = \arg\min_{l \leq x \leq u} f^\varphi(x^s). \tag{1}$$

With slight abuse of notation we will write $f(x)$ as shorthand for $f^\varphi(x^s)$, where x_i^s is defined by the *Generator* by using the parameters x, and x_o^s is the output of the *Simulator* with x_i^s as the input trace.

It is important to note that for falsification of CPSs, the objective function value is defined by the input and output traces of the CPS. The output traces are defined by the inputs and the system dynamics. In the simulation-based falsification, the objective function is fully known, but the system dynamics is given as a black-box model.

The optimization methods that are compared to in this article are NM and SNOBFIT, both briefly described below.

NM [15] is a direct-search method that starts with a simplex set of $n + 1$ points where n is the number of dimensions of the optimization problem. In each iteration, the points are sorted from the lowest to highest objective function value. NM attempts to replace the point with the highest objective function value with a new point obtained by reflection, expansion, or contraction. If all of these fail, the entire simplex shrinks toward the point with the lowest objective function value, n new points are generated, and the above process is continued.

SNOBFIT [20] is a model-based method that works by building surrogate models around each evaluated point. The optimization proceeds by processing of parameter values and the corresponding function values and recommending a new set of parameters values to evaluate. SNOBFIT typically has a fast local search and contains parameters that control the balance between the local and global search of the optimization.

III. Hybrid Corner-Random Method (HCR)

This section introduces the *Hybrid Corner-Random* (HCR) falsification method. This is an optimization-free method that explores corner points, i.e., extreme values within the allowed parameter ranges, in combination with evaluating random parameter values also within the allowed parameter ranges.

To clarify the meaning of a corner point, assume a system with n input parameters $x_1, \ldots, x_n$, where each parameter has a lower and upper bound. There are 2^n corner points, where a corner point only contains values that are at the lower or upper bound for each parameter.

For HCR, shown in Algorithm 1, the quantitative semantics used does not matter. It is merely evaluated whether the specification is falsified or not at the currently selected point.

Algorithm 1 Hybrid Corner-Random Method for Falsification

```
1: curr_simulation = 0;
2: Pick a corner point, x.
3: while curr_simulation < max_simulations do
4:    Simulate system with x as input.
5:    curr_simulation++;
6:    Evaluate spec at x.
7:    if the spec is falsified then
8:       Return ⟨Falsified, x⟩.
9:    else
10:      if curr_simulation mod 2 == 0 or corners exhausted then
11:         Pick a random point x.
12:      else
13:         Pick a corner point x, not previously selected.
14:      end if
15:   end if
16: end while
17: Return ⟨Not Falsified, the last evaluated x⟩.
```

Algorithm 1 outputs the tuple $\langle$*Falsified / Not Falsified*, $x\rangle$, where the first element describes if the specification was falsified or not, and the second element x is an n-dimensional parameter point with the following properties.

1) If the specification is falsified, $f(x) < 0$.
2) If the maximum number of simulations is reached and the specification is not falsified, x is the last evaluated point.

A. Description

Algorithm 1 starts with a corner point, x, at line 2. In each iteration, while the number of simulations, curr_simulation, is less than the given max_simulations, the algorithm picks a corner or random point x. Then, the system is simulated at the current point x to evaluate the specification. If the specification is falsified, the algorithm terminates, lines 7 and 8. Otherwise, if the specification is not falsified, for the next iteration, a new random or corner point x will be picked, lines 9–15.

It should be mentioned here that the number of corners depends on the dimension of the input parameters, x. Since the number of corners is constant, if there are no new corners to select, the algorithm will continue working with only random points, lines 10–14, until the maximum number of iterations has been executed.

Note that, the uniform-random (UR) method is used in this article as the random method. Other random search methods could be used in the HCR method.

IV. Line-Search Falsification (LSF)

In the previous section, HCR, an optimization-free approach was presented. From the evaluation in Section V we will see that while the HCR method is quite successful in falsifying many of the specifications, there are also harder problems where an optimization-based approach might be more efficient. In this section, *Line-Search Falsification*, LSF, is introduced. LSF is a gradient-free direct-search optimization-based approach. While HCR only relies on random and corner points, LSF combines random exploration with local search, by randomly generating lines in the n-dimensional parameter space. On these lines, corner points for different sets of parameters are evaluated together with a local search to find the minimal value of the objective function along the lines. When no improvements are reported for a number of iterations, a new line is generated for further exploration. LSF combines random search, with corner points, with a local search without being dependent on building a surrogate model of the objective function.

The method is presented in Algorithm 2, and has been implemented in Breach. The output of LSF is the tuple $\langle$*Falsified / Not Falsified*, $x\rangle$, where the first element describes if the specification was falsified or not, and the second element x is an n-dimensional parameter point where:

1) if the specification is falsified, $f(x) < 0$;
2) if the maximum number of simulations is reached and the specification is not falsified, x is the point with the minimum positive objective function value, $f(x) \geq 0$.

A. Description

LSF generates random lines through the parameter space and then does local search along the line but always within the allowed parameter range. Technically, the local search is done along a *line segment* since it has two end points, and not along a *line* that extends infinitely in both directions. However, in the description below *line* is used to refer to both line segments and lines when the exact meaning is clear from the context.

LSF (see Algorithm 2) consists of the following steps.

1) Initial Points: LSF needs three-parameter points during the optimization process, and a new point is generated in each iteration. The first one point has to be identified, line 1. This point is the middle point $x = [(l + u)/2]$ in each of the n dimensions. Note that, this point could instead be chosen randomly within the range (l, u). The SUT is then simulated with x as input parameters, that define x^s as the discrete input signals, and the corresponding objective function value $f(x)$ is computed. This is handled by the call to Eval(x), which returns $f(x)$. If $f(x) < 0$, then the specification is falsified and the algorithm terminates. If max_simulations have been done without falsifying the specification, *Not Falsified* is returned together with the point with the minimum positive objective function value, lines 4–6. Otherwise, if curr_simulations is larger than one, the heuristic H_1 is called, lines 7–9. When curr_simulations is not larger than 1, H_1 is not called, as will be discussed later. Then, SelectPoints(x) is called, line 10.

2) SelectPoints: The three points, x_M, x_L, and x_R, are defined and generated in the SelectPoints(x) function. SelectPoints(x) picks a random line that goes through x and computes end points x_L and x_R, lines 23 and 24. There are four options for how to pick these end points, described in the following paragraphs.

3) Pick Line Segments Through Point x: Let $x = (x_1, \ldots, x_n)^T$, where $l_i \leq x_i \leq u_i$, be an n-dimensional point within the given boundaries. Pick a random direction vector $d = (d_1, \ldots, d_n)^T$ where $d_i \neq 0$ for $1 \leq i \leq n$. Define the function $g{:}\mathbb{R}^n \to \mathbb{R}^n$ where each dimension i, $1 \leq i \leq n$, of g is defined by g_i as

$$g_i(z) = \begin{cases} u_i, & \text{if } x_i + z\ d_i > u_i \\ x_i + z\ d_i, & \text{if } l_i \leq x_i + z\ d_i \leq u_i \\ l_i, & \text{if } x_i + z\ d_i < l_i. \end{cases}$$

Algorithm 2 Line-Search Falsification

$x = \frac{(l+u)}{2}$;
global curr_simulation = 0;
while Eval$(x) \geq 0$ **do**
  if curr_simulation $\geq$ max_simulations **then**
    Return $\langle$Not Falsified, $x\rangle$
  end if
  if curr_simulations > 1 **then**
    x = **Heuristic** $H_1(x_{old}, x)$
  end if
  (x_L, x_M, x_R) = **SelectPoints** (x)
  $x_{old} = x$;
  x = **FalsifyLine**(x_L, x_M, x_R, x)
end while
Return $\langle$Falsified, $x\rangle$

$f(x)$ = **Eval**(x)

map$(x, f(x))$ evaluations;
if x is not in the evaluations map **then**
  Simulate with x as input, compute $f(x)$ using the quantitative semantics.
  Store $(x, f(x))$ in the evaluations map;
  curr_simulations++;
end if
Look up the value, $f(x)$, of x in the evaluations map.
Return $f(x)$

(x_L, x_M, x_R) = **SelectPoints**(x)

Pick a random line that goes through x.
Compute the end-points x_L and x_R by using one of Options 1-4.
Option 1: $x_L = x$, $x_M = \frac{(x_R+x_L)}{2}$, x_R is a point where one of the dimensions is on the boundary of l or u.
Option 2: $x_L = x$, $x_M = \frac{(x_R+x_L)}{2}$, x_R is a point where half of the dimensions are on the boundary of l and u.
Option 3: $x_L = x$, $x_M = \frac{(x_R+x_L)}{2}$, x_R is a corner point.
Option 4: $x_M = x$, x_R and x_L are the points where at least one of the dimensions is on the boundary of l or u, or is a corner points.
Return (x_L, x_M, x_R)

x = Heuristic H_1

if $f(x_{old}) < f(x)$ **then**
  $x =$ be the second point with lowest objective function value of the last call to **FalsifyLine.**
end if
Return x

x = **FalsifyLine**(x_L, x_M, x_R, x)

iterations_without_improvement = 0;
while iterations_without_improvement < max_iterations **do**
  if Eval$(x_L) < 0$ **then Return** x_L **end**
  if Eval$(x_R) < 0$ **then Return** x_R **end**
  if Eval$(x_M) < 0$ **then Return** x_M **end**
  if Eval$(x_L) <$ Eval(x_R) **and** Eval$(x_L) <$ Eval(x_M) **then**
    Let $x_{new} = \frac{(x_L+x_M)}{2}$,
    if Eval$(x_{new}) <$ Eval (x_L) **then**
      $x = x_{new}$
    end if
    $x_R = x_M$, $x_M = x_{new}$
  else if Eval$(x_R) <$ Eval(x_L) **and** Eval$(x_R) <$ Eval(x_M) **then**
    Let $x_{new} = \frac{(x_M+x_R)}{2}$,
    if Eval$(x_{new}) <$ Eval(x_R) **then**
      $x = x_{new}$
    end if
    $x_L = x_M$, $x_M = x_{new}$
  else
    if $\left(||x_L - x_M||_2 \geq ||x_M - x_R||_2\right)$ **then**
      Let $x_{new} = \frac{(x_L+x_M)}{2}$,
      if Eval$(x_{new}) <$ Eval(x_M) **then**
        $x = x_{new}$
        $x_R = x_M$, $x_M = x_{new}$
      else
        $x_L = x_{new}$
      end if
    else
      Let $x_{new} = \frac{(x_M+x_R)}{2}$,
      if Eval$(x_{new}) <$ Eval(x_M) **then**
        $x = x_{new}$
        $x_L = x_M$, $x_M = x_{new}$
      else
        $x_R = x_{new}$
      end if
    end if
  end if
  if $x == x_{new}$ **then**
    iterations_without_improvement = 0;
  else
    iterations_without_improvement++;
  end if
end while
Return x

Let $z_k^+ \in \mathbb{R}_{f>0}$ be the smallest positive value such that for at least k distinct dimensions

$$g_i\left(z_k^+\right) = u_i \text{ or } g_i\left(z_k^+\right) = l_i.$$

Let $z_k^- \in \mathbb{R}_{<0}$ for $1 \leq k \leq n$ be the smallest negative value such that for at least k distinct dimensions

$$g_i\left(z_k^-\right) = u_i \text{ or } g_i\left(z_k^-\right) = l_i.$$

In the sequel, we refer to the end points defined by z_1^+ as Option 1, $z_{\lceil (n/2) \rceil}^+$ as Option 2, and z_n^+ as Option 3. Let $z_{j,k} = (z_j^-, z_k^+)$, $1 \leq j, k \leq n$ for Option 4.

For Options 1–3, define end points for a given z_k^+ as $x_L = x$, $x_R = g\left(z_k^+\right)$, and $x_M = [(x_L + x_R)/2]$. For Option 4, define end points for a given $z_{j,k}$ as $x_L = g\left(z_j^-\right)$, $x_R = g\left(z_k^+\right)$, and $x_M = x$.

Note that which option to use is predetermined before Algorithm 2 starts, and the same option is used during the entire execution.

Fig. 2 exemplifies these four options in two dimensions $n = 2$, i.e., two input parameters with respective lower and upper bounds. We are allowed to take points within the input parameter box (l, u). Thus, the four corner points are

$$\begin{pmatrix} l_1 \\ l_2 \end{pmatrix}, \begin{pmatrix} l_1 \\ u_2 \end{pmatrix}, \begin{pmatrix} u_1 \\ l_2 \end{pmatrix}, \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}.$$

Fig. 2(a) shows an example for Option 1 where $k = 1$. Since this is a 2-D case, Options 1 and 2 will be the same. In this graph, the chosen line starts from x_L in the box and

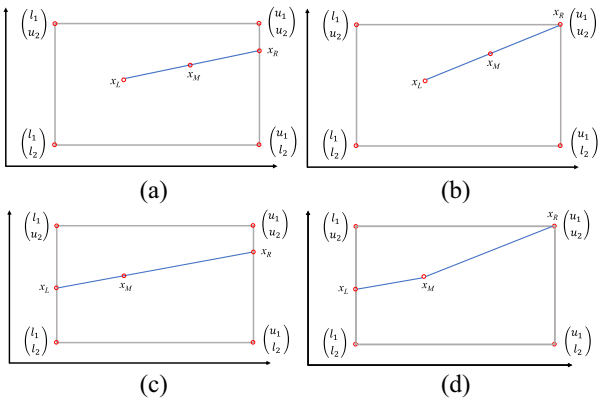

Fig. 2. Examples in two dimensions for the four options: (a) Example for Option 1 and Option 2, where $k = 1$ and $x = x_L$; (b) Example for Option 3, where $k = 2$ and $x = x_L$; (c) and (d) Examples for Option 4, where $k = 1$, $j = 1$ and $k = 2$, $j = 1$, respectively, and $x = x_M$.

ends up at the point x_R where it cuts off at the upper bound of dimension 1.

Fig. 2(b) shows an example for Option 3 where $k = 2$. In this graph, the chosen line starts from x_L in the box and ends up at the point x_R where it cuts off the corner point $(u_1, u_2)^T$.

To get a clear understanding of how to choose a line according to Option 4, Fig. 2(c) and (d) are given. In Fig. 2(c) where $k = 1$ and $j = 1$, we work with a line segment where x_R is on the upper bound of the first dimension, u_1, and x_L is on the lower bound of the first dimension, l_1. The line can be a line that passes through x_M, but it can also be two line segments that connect x_M to x_L and x_M to x_R as is shown in Fig. 2(d) where $k = 2$ and $j = 1$. In this graph, x_L is on the lower bound of the first dimension l_1, and x_R is a corner point $(u_1, u_2)^T$.

4) Comparison Among the Four Options to Choose Line: Which option performs best for a given SUT and specification is heavily application dependent, and thus comparing them has no clear outcome.

In Options 1 and 2, we work with lines that cut off at the upper or lower bound in one and half of the dimensions, respectively. In these options, one of the points, x_R, is on the bound of the box. In Option 3, one of the points, x_R, is on a corner point. On the other hand, in Option 4, x_R and x_L are either on a bound of the box or on corner points.

Options 1 and 2 search more points on the boundaries, while Option 3 searches more corner points and moves toward them if the optimization process guides in that direction. Since lines are chosen randomly, Options 1–3 may lead to using short lines that do not guide the process toward falsification or might get stuck for some iterations in a local area. On the other hand, Option 4 works with long lines, extending between the boundaries, which has a higher chance of finding a better point. A comparison from a practical perspective based on the examples evaluated in this article will be given in Section V.

Admittedly, there are many possible ways to pick the lines. What heuristics should be used to choose a line is an open question for future research.

5) Algorithm 2's Main Loop in FalsifyLine: Now it is time to perform the falsification process over a line by calling FalsifyLine, line 12. Before calling this function, we need to save the current point x in a variable x_{old} at line 11. FalsifyLine tries to find a new point with a lower objective function value, the ultimate goal being to find a point with a negative value.

The FalsifyLine algorithm is the local search loop in LSF. The SUT is simulated and evaluated at the three-parameter points x_M, x_L, and x_R by calling the Eval function, lines 32–34. If the specification is falsified, i.e., the objective function value is negative, FalsifyLine terminates. Otherwise, if the specification is not falsified at any of the points x_L, x_M, or x_R, the algorithm starts the local search by looking for new points that can potentially have lower objective function values.

The variable iterations_without_improvement is used to count the number of iterations that we stay with a line inside FalsifyLine. The local search along the line is completed when iterations_without_improvement has reached max_iterations.

There are three cases, depending on which of the points x_M, x_L, and x_R has the smallest objective function value. In each iteration, a new point is searched that is the middle point between x_M and x_L, or x_M and x_R. This new point will first be evaluated to decide if it falsifies the specification or not. If it does not, the new point will be replaced by one of the three points, x_L, x_M, or x_R, based on some rules presented below. The optimization will now be done on a shorter line segment than the initial line segment, defined by the selected end points, for the next iteration. The following cases determine the rule for selecting the new point and the shorter line segment.

Case 1: The point x_L has the lowest objective function value among the three points, lines 35–40, i.e.,

$$f(x_L) < f(x_M) \quad \text{and} \quad f(x_L) < f(x_R).$$

If the above condition is satisfied then let $x_{\text{new}} = ([x_L + x_M]/2)$ be defined as the point midway between x_L and x_M. Then, check whether x_{new} has a lower objective function value than x_L or not, if it does, then update $x = x_{\text{new}}$. For the next iteration, x_R will be omitted and replaced to work with a shorter line that connects x_M to x_L and passing from x_{new}. The new points are

$$x_R = x_M \quad \text{and} \quad x_M = x_{\text{new}}. \tag{2}$$

The point x_L, which was the point with the lowest objective function value, is not changed.

Case 2: The point x_R has the lowest objective function value among the three points, lines 41–46, i.e.,

$$f(x_R) < f(x_M) \quad \text{and} \quad f(x_R) < f(x_L).$$

In this case, the middle point $x_{\text{new}} = ([x_M + x_R]/2)$ is computed and checked to see whether x_{new} has a lower objective function value than x_R. If it does then $x = x_{\text{new}}$. For the next iteration, x_L will be omitted and replaced to work with a shorter line segment that connects x_M to x_R and passing through x_{new}. The three points for the next iteration are

$$x_L = x_M \quad \text{and} \quad x_M = x_{\text{new}} \tag{3}$$

while x_R, with the lowest objective function value, will not be changed.

Case 3: The point x_M has the lowest objective function value among the three points, lines 47–62, i.e.,

$$f(x_M) \leq f(x_L) \quad \text{and} \quad f(x_M) \leq f(x_R).$$

TABLE I
RESULTS FOR THE AUTOMATIC TRANSMISSION (AT) SYSTEM, INSTANCES 1 AND 2

Specifications	φ_1^{AT}		φ_2^{AT}		φ_3^{AT}		φ_4^{AT}		φ_5^{AT}	
Instances	Instance 1	Instance 2	Instance 1	Instance 2	Instance 1	Instance 2	Instance 1	Instance 2	Instance 1	Instance 2
Nelder-Mead	**100** (227)	0 (-)	**100** (8)	90 (234)	**100** (19)	**100** (5)	**100** (15)	**100** (1)	**100** (15)	**100** (1)
SNOBFIT	**100** (38)	0 (-)	**100** (6)	**100** (59)	**100** (8)	**100** (5)	**100** (30)	**100** (1)	**100** (9)	**100** (1)
Line-Search Falsification	**100** (40)	0 (-)	**100** (10)	**100** (96)	**100** (34)	**100** (13)	**100** (36)	**100** (3)	**100** (17)	**100** (2)
Corners	**100** (3)	0 (-)	**100** (2)	**100** (2)	0 (-)	**100** (5)	0 (-)	**100** (3)	0 (-)	**100** (3)
Uniform-Random	0 (-)	0 (-)	**100** (10)	**100** (288)	**100** (13)	**100** (5)	**100** (13)	**100** (2)	**100** (11)	**100** (1)
Hybrid Corner-Random	**100** (5)	0 (-)	**100** (3)	**100** (3)	**100** (27)	**100** (7)	**100** (25)	**100** (3)	**100** (23)	**100** (2)

Specifications	φ_6^{AT}		φ_7^{AT}		φ_8^{AT}		φ_9^{AT}	
Instances	Instance 1	Instance 2	Instance 1	Instance 2	Instance 1	Instance 2	Instance 1	Instance 2
Nelder-Mead	**100** (73)	**100** (2)	**100** (48)	5 (597)	**100** (103)	0 (-)	**100** (29)	0 (-)
SNOBFIT	**100** (87)	**100** (2)	**100** (38)	45 (304)	**100** (70)	30 (369)	**100** (47)	65 (243)
Line-Search Falsification	**100** (112)	**100** (6)	**100** (24)	**100** (122)	**100** (28)	**100** (214)	**100** (12)	**100** (41)
Corners	0 (-)	**100** (3)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)
Uniform-Random	**100** (59)	**100** (2)	**100** (65)	0 (-)	**100** (96)	0 (-)	**100** (31)	0 (-)
Hybrid Corner-Random	**100** (117)	**100** (3)	**100** (130)	0 (-)	**100** (178)	0 (-)	**100** (61)	0 (-)

Here, we need to decide on which line segment to continue the search, i.e., the line segment (x_M, x_L) or (x_M, x_R). We do an evaluation that depends on which interval (x_L, x_M) or (x_M, x_R) that is longer; for this purpose we use the Euclidean distance $(||\cdot||_2)$. x_{new} is selected to be the middle point on the longer of the two segments

$$x_{\text{new}} = \begin{cases} \frac{x_L+x_M}{2}, & \text{if } \|x_L - x_M\|_2 \geq \|x_M - x_R\|_2 \\ \frac{x_M+x_R}{2}, & \text{otherwise.} \end{cases} \tag{4}$$

When $||x_L - x_M||_2 \geq ||x_M - x_R||_2$ the end points are updated as (lines 48–55)

$$\begin{cases} x_R = x_M \text{ and } x_M = x_{\text{new}}, & \text{if } f(x_{\text{new}}) < f(x_M) \\ x_L = x_{\text{new}}, & \text{otherwise.} \end{cases} \tag{5}$$

When $||x_L - x_M||_2 < ||x_M - x_R||_2$ the end points are updated as (lines 56–63)

$$\begin{cases} x_L = x_M \text{ and } x_M = x_{\text{new}}, & \text{if } f(x_{\text{new}}) < f(x_M) \\ x_R = x_{\text{new}}, & \text{otherwise.} \end{cases} \tag{6}$$

Those points that are not explicitly updated in (5) and (6) will keep their previous value. Note that in this case, where x_M has the lowest objective function value, the decision about which line segment to continue on is based solely on the length of the line segments. The longest line segment is a sensible choice because this is the line segment that has the largest uncertainty in the sense of being least explored. Considering $f(x_L)$ and $f(x_R)$ would add complexity since it would require the algorithm to balance between the distance and objective function value.

Based on which of the above cases occurs, in each iteration, one of the new points $[(x_L + x_M)/2]$ or $[(x_M + x_R)/2]$ is searched, and one of the old three points will be replaced with the new point.

If we can improve the point x and find a better point with lower objective function value, where $x = x_{\text{new}}$, the iterations_without_improvement resets to zero, line 67. This leads us to work more with the lines that reach points with lower objective function values. On the other hand, if we cannot improve the point x, iterations_without_improvement is increased by 1, line 69. This leads us to work less with the lines that do not help reach points with lower objective function values. When iterations_without_improvement reaches max_iterations, we get out from the currently chosen line and pick a new one.

When iterations_without_improvement has reached max_iterations, the FalsifyLine process is finished and it returns a value for x. Now, we are at line 3. If the specification is not falsified yet and curr_simulation has not reached max_simulations and curr_simulation > 1, then Heuristic H_1 is called at line 8.

6) Heuristic H_1: In function Heuristic H_1, we compare the objective function value of x_{old} with x. If $f(x_{\text{old}}) < f(x)$, FalsifyLine could not give a better point, i.e., one with a lower objective function value than x_{old}. In this case, in order to force the algorithm to never return the same x as was given to FalsifyLine, we consider x to be the point with the *second lowest* objective function value of the last call to FalsifyLine. Our experimental evaluations show that this heuristic helps the algorithm to avoid getting stuck in local minima.

7) Stopping Condition: Algorithm 2 works until a falsified point is found, i.e., a point with a negative objective function value or until curr_simulation reaches max_simulations. If a falsified point is not found, the algorithm returns the point with the lowest objective function value found so far.

V. EXPERIMENTAL SETUP AND RESULTS

We consider the simulation-based falsification of all benchmark examples of the ARCH19 workshop [30], which is a friendly competition in falsification of temporal logic specifications over CPSs. For these benchmark problems, two variants of input signals are considered Instance 1 and Instance 2, respectively. Instance 1 allows arbitrary piece-wise continuous input signal while Instance 2 are restricted to constrained input signals. The problems AT, CC, NN, and SC have specifications both of Instance 1 and Instance 2 type. Additionally the benchmarks from [31] are included. Note that the example Automatic Transmission (AT′) presented in Table III has different specifications from the Automatic Transmission (AT) presented in Table I.

In this section, the Corners and UR methods are compared with the HCR approach. The LSF method is compared with NM, SNOBFIT, and HCR. Finally, all methods are compared together. The results are shown in Tables I–VII. Each falsification is set to have max_simulations $= 1000$. There are 20 falsification runs for each method and objective function to account for most algorithms' random nature. The *Additive* semantics is considered in this paper. The evaluation results for *Max* can be found in [34].

TABLE II
RESULTS FOR THE CHASING CARS (CC) SYSTEM, INSTANCES 1 AND 2

Specifications	φ_1^{CC}		φ_2^{CC}		φ_3^{CC}		φ_4^{CC}		φ_5^{CC}	
Instances	Instance 1	Instance 2	Instance 1	Instance 2	Instance 1	Instance 2	Instance 1	Instance 2	Instance 1	Instance 2
Nelder-Mead	**100** (11)	**100** (142)	**100** (5)	**100** (96)	**100** (21)	**100** (37)	0 (-)	0 (-)	**100** (23)	**100** (55)
SNOBFIT	**100** (5)	**100** (29)	**100** (3)	**100** (125)	**100** (5)	**100** (12)	60 (334)	0 (-)	**100** (16)	**100** (38)
Line-Search Falsification	**100** (9)	**100** (26)	**100** (10)	**100** (218)	**100** (19)	**100** (11)	35 (623)	10 (755)	**100** (37)	**100** (93)
Corners	**100** (3)	**100** (3)	**100** (1)	**100** (1)	**100** (3)	**100** (3)	0 (-)	0 (-)	0 (-)	**100** (21)
Uniform-Random	**100** (8)	**100** (67)	**100** (5)	**100** (157)	**100** (17)	**100** (24)	0 (-)	0 (-)	**100** (18)	**100** (96)
Hybrid Corner-Random	**100** (5)	**100** (5)	**100** (1)	**100** (1)	**100** (5)	**100** (5)	0 (-)	0 (-)	**100** (36)	**100** (38)

TABLE III
RESULTS FOR THE AUTOMATIC TRANSMISSION (AT′) SYSTEM

Specifications	$\varphi_1^{AT'}(T=20)$	$\varphi_1^{AT'}(T=30)$	$\varphi_1^{AT'}(T=40)$	$\varphi_2^{AT'}(T=10)$	$\varphi_3^{AT'}(T=4.5)$	$\varphi_3^{AT'}(T=5)$	$\varphi_4^{AT'}(T=1)$	$\varphi_4^{AT'}(T=2)$
Nelder-Mead	**100** (156)	95 (365)	65 (557)	**100** (14)	90 (323)	**100** (95)	35 (358)	**100** (20)
SNOBFIT	**100** (16)	**100** (34)	**100** (53)	**100** (10)	**100** (46)	**100** (24)	60 (328)	**100** (18)
Line-Search Falsification	**100** (56)	**100** (104)	**100** (219)	**100** (20)	**100** (151)	**100** (61)	75 (307)	**100** (24)
Corners	**100** (1)	**100** (1)	**100** (1)	**100** (2)	**100** (11)	**100** (11)	**100** (1)	**100** (1)
Uniform-Random	**100** (168)	60 (396)	25 (267)	**100** (20)	**100** (183)	**100** (79)	70 (407)	**100** (24)
Hybrid Corner-Random	**100** (1)	**100** (1)	**100** (1)	**100** (3)	**100** (21)	**100** (21)	**100** (1)	**100** (1)

Specifications	$\varphi_5^{AT'}(T=1)$	$\varphi_5^{AT'}(T=2)$	$\varphi_6^{AT'}(T=10)$	$\varphi_6^{AT'}(T=12)$	$\varphi_7^{AT'}$	$\varphi_8^{AT'}(\bar{\omega}=3000)$	$\varphi_8^{AT'}(\bar{\omega}=3500)$
Nelder-Mead	75 (516)	**100** (4)	45 (483)	**100** (215)	75 (383)	**100** (12)	90 (375)
SNOBFIT	**100** (59)	**100** (4)	0 (-)	90 (344)	95 (295)	**100** (9)	85 (341)
Line-Search Falsification	**100** (127)	**100** (7)	25 (317)	**100** (265)	75 (345)	**100** (9)	**100** (178)
Corners	**100** (71)	**100** (22)	0 (-)	0 (-)	0 (-)	**100** (3)	**100** (3)
Uniform-Random	95 (212)	**100** (5)	0 (-)	90 (328)	45 (421)	**100** (9)	30 (389)
Hybrid Corner-Random	**100** (120)	**100** (10)	0 (-)	60 (297)	30 (621)	**100** (5)	**100** (5)

TABLE IV
RESULTS FOR $\Delta-\Sigma$ MODULATOR AND THE STATIC SWITCHED (SS) SYSTEM

Specifications	$\varphi_1^{\Delta-\Sigma}(U\in[-0.35,0.35])$	$\varphi_1^{\Delta-\Sigma}(U\in[-0.40,0.40])$	$\varphi_1^{\Delta-\Sigma}(U\in[-0.45,0.45])$	$\varphi_1^{SS}(thresh=0.7)$	$\varphi_1^{SS}(thresh=0.8)$	$\varphi_1^{SS}(thresh=0.9)$
Nelder-Mead	25 (320)	90 (399)	**100** (140)	**100** (68)	**100** (130)	95 (295)
SNOBFIT	95 (160)	**100** (47)	**100** (28)	**100** (22)	**100** (63)	75 (208)
Line-Search Falsification	**100** (254)	**100** (48)	**100** (50)	**100** (47)	**100** (119)	**100** (192)
Corners	0 (-)	**100** (3)	**100** (6)	**100** (2)	**100** (2)	**100** (2)
Uniform-Random	0 (-)	95 (313)	**100** (262)	**100** (42)	**100** (84)	90 (261)
Hybrid Corner-Random	0 (-)	**100** (5)	**100** (11)	**100** (3)	**100** (3)	**100** (3)

TABLE V
RESULTS FOR NEURAL NETWORK (NN), INSTANCES 1 AND 2

Specifications	φ_1^{NN}		φ_2^{NN}	
Instances	Instance 1	Instance 2	Instance 1	Instance 2
Nelder-Mead	**100** (19)	**100** (81)	5 (164)	10 (434)
SNOBFIT	**100** (20)	**100** (87)	85 (361)	20 (440)
Line-Search Falsification	**100** (18)	**100** (62)	90 (354)	25 (289)
Corners	**100** (36)	0 (-)	**100** (42)	0 (-)
Uniform-Random	**100** (29)	**100** (117)	5 (506)	0 (-)
Hybrid Corner-Random	**100** (39)	**100** (125)	**100** (83)	0 (-)

TABLE VI
RESULTS FOR THE AIRCRAFT GROUND COLLISION AVOIDANCE (F16), THE STEAM CONDENSER SC AND THE FUEL CONTROL OF AUTOMOTIVE POWER TRAIN (AFC) SYSTEM

Specifications	φ^{F16}	φ^{SC} (Instances 1, 2)	φ_1^{AFC}	φ_2^{AFC}
Nelder-Mead	20 (319)	0 (-)	95 (318)	**100** (13)
SNOBFIT	65 (420)	0 (-)	**100** (27)	**100** (7)
Line-Search Falsification	60 (399)	0 (-)	**100** (9)	**100** (3)
Corners	0 (-)	0 (-)	**100** (3)	**100** (3)
Uniform-Random	5 (759)	0 (-)	**100** (364)	**100** (16)
Hybrid Corner-Random	5 (767)	0 (-)	**100** (5)	**100** (5)

TABLE VII
RESULTS FOR THE WIND TURBINE (WT) SYSTEM.

Specifications	φ_1^{WT}	φ_2^{WT}	φ_3^{WT}	φ_4^{WT}
Nelder-Mead	**100** (1)	**100** (1)	**100** (1)	**100** (79)
SNOBFIT	**100** (1)	**100** (1)	**100** (1)	**100** (65)
Line-Search Falsification	**100** (3)	**100** (2)	**100** (2)	**100** (62)
Corners	**100** (3)	**100** (3)	**100** (3)	**100** (16)
Uniform-Random	**100** (2)	**100** (1)	**100** (1)	**100** (115)
Hybrid Corner-Random	**100** (3)	**100** (2)	**100** (2)	**100** (30)

1) *Corners Implementation Setup:* Each specification and example has a different number of corners. For some examples, the number of corners is less than the maximum number of simulations. Then, the Corners method terminates when all corners have been evaluated.

2) *Uniform-Random Implementation Setup:* For each of 20 falsification runs, uniformly distributed random points are generated from different seeds, i.e., 20 different seeds are considered.

3) *Hybrid Corner-Random Implementation Setup:* The HCR method starts with the Corners point, and the next point is the UR point. It switches between the Corners and UR until the maximum number of simulations, 1000 here, is reached or a falsified point is found. The number of corners is limited, and it depends on how many corner points the SUT has. If the maximum number of corners is reached, the HCR algorithm continues using only the UR points.

4) *Nelder-Mead Implementation Setup:* This algorithm is implemented as *fminsearch* [33] in MATLAB. Before starting NM, 100 random sampling points for all examples are generated. After evaluating 100 random sampling points, the point with the lowest objective value is picked as a starting point if none falsifies the specification. NM needs a simplex of $n+1$ points for n-dimensional vectors, while we have one point. The n points will be generated by MATLAB's *fminsearch* using that minimum point. The minimum point is similar to x_0 when calling *fminsearch* (f, x_0) in MATLAB, where it starts at the point x_0 and tries to find a local minimum. For *fminsearch* the default parameters are used.

5) *Line-Search Falsification Implementation Setup:* The value of max_iterations is set to 3.

In Tables I-VII, the first row denotes the specification, as defined in [30] and [31]. For the problems where specification of both Instance 1 and Instance 2 are available, this is indicated

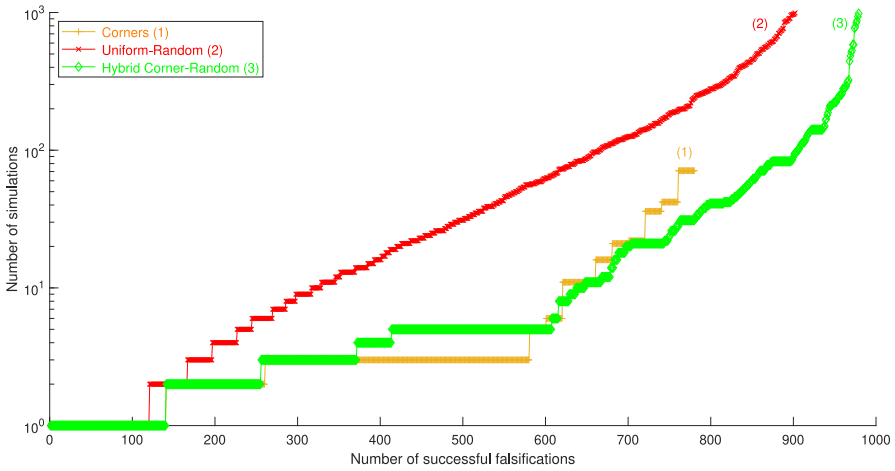

Fig. 3. Cactus plot showing the performance of Corners, UR, and HCR for all examples. The plotted values show how many successful falsifications (x-axis) were completed in less than 1000 simulations (y-axis, logarithmic scale).

by the second row. The following row contains the results for the different optimization methods, and the last row the results for the optimization-free methods. Two values are presented for all examples. The first value is the relative success rate of falsification in percent. There are 20 falsification runs for each parameter value and specification, thus the success rate will be a multiple of 5%. The second value, inside parentheses, is the average number of simulations (rounded) per successful falsification. The presented results for LSF use Option 4 presented above.

A. Optimization-Free Versus Optimization-Based Falsification

First, we compare the optimization-free methods Corners, UR, and HCR. Based on the results shown in Tables I–IV, for many specifications, one of Corners or UR succeeds. Hence, HCR outperforms the other two methods for many specifications. For a comparison between Corners, UR, and HCR, a cactus plot is shown in Fig. 3. The results presented in this plot relate to all examples. As can be seen in Fig. 3, while UR is more successful than Corners, for those cases that are falsified at the corner points, Corners falsifies much faster and with fewer simulations. Thus, HCR manages to falsify more examples than either of the other two.

To get a better understanding of the difference in performance between optimization-free and optimization-based methods, we go through some examples in more detail to compare HCR with NM, SNOBFIT, and LSF.

For φ_2^{AT} in Instance 2, Table I, NM is not 100 percent successful, and SNOBFIT and LSF need more simulations than Corners and HCR. On the other hand, HCR manages to falsify the property with just a few simulations. As can be seen in Table II, all optimization-based methods needed more simulations to falsify φ_2^{CC} for Instance 2. On the other hand, with only one simulation, HCR falsifies this specification. Similarly, HCR performs better than the optimization-based methods for specifications $\varphi_1^{AT'}(T = 30)$ and $(T = 40)$, $\varphi_3^{AT'}(T = 4.5)$, $\varphi_4^{AT'}(T = 1)$, $\varphi_8^{AT'}(\bar{\omega} = 3500)$ in Table III. There is a significant advantage for HCR for the specifications $\varphi_1^{\Delta-\Sigma}(U \in [0.40, 0.40]$ and $U \in [0.45, 0.45])$ and all specifications of SS system in Table IV, which HCR manages to falsify with fewer simulations.

To compare Corners, UR, and HCR with NM, SNOBFIT, and LSF, a cactus plot is shown in Fig. 4 for those examples where using an optimization-based approach to improve the falsification process is not needed. As can be seen in this figure, HCR beats all methods and manages to falsify more examples than either of the optimization methods and requires fewer simulations than all other methods.

On the other hand, there are specifications that neither Corners nor UR can falsify. Here, the HCR method will not be successful either. These are the specifications 1 and 7-9 of the AT example for Instance 2, Table I; specification 4 of both instances 1 and 2 of the CC example, Table II; specifications 6 for both T = 10, T = 12 and 7 of AT′, Table III; the first specification of the Modulator example, Table IV; specification 2 of the NN example for Instance 2, Table V; and the F16 example Table VI.

B. Line-Search Falsification Using the Four Options

Section IV introduced four different options for LSF to pick a line. A cactus plot comparing these options is given in Fig. 5.

Option 1, where we work with the lines that cut off at the upper or lower bound in one of the dimensions, does not work as well as the other options. Options 2 and 3 perform approximately the same, where Option 2 works with the lines that cut off at the half of the dimensions, and Option 3 searches more corner points. As can be seen in Fig. 5, Option 3 is more successful and requires fewer simulations. Option 3 is a good approach for those easy benchmark examples that can be falsified on the corners. On the other hand, Option 4 is more successful in the falsification process because it uses the combination of three options. Hence, in the tables, we just quote the results using Option 4 for LSF.

C. Line-Search Falsification Versus Nelder-Mead and SNOBFIT

This part compares the results of the new LSF method with NM and SNOBFIT for those specifications and examples that

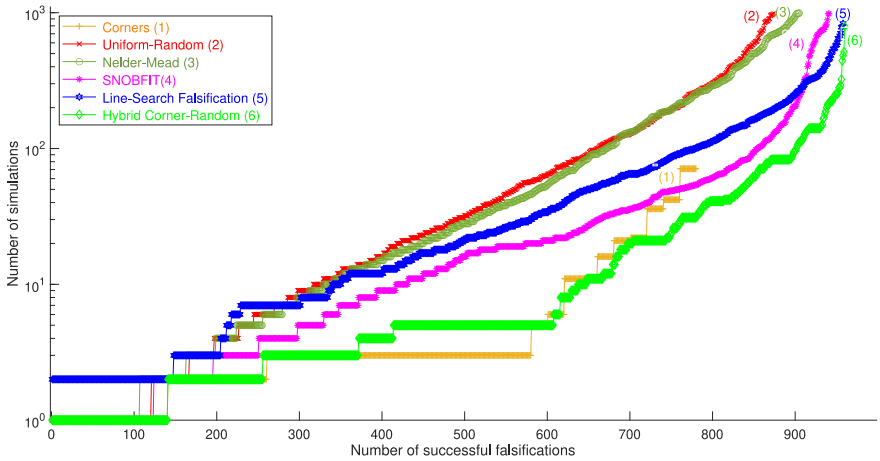

Fig. 4. Cactus plot showing performance of Corners, UR, and HCR against NM, SNOBFIT, and LSF for those examples where using an optimization-based approach to improve the falsification process is not needed. The plotted values show how many successful falsifications (x-axis) were completed in less than 1000 simulations (y-axis, logarithmic scale).

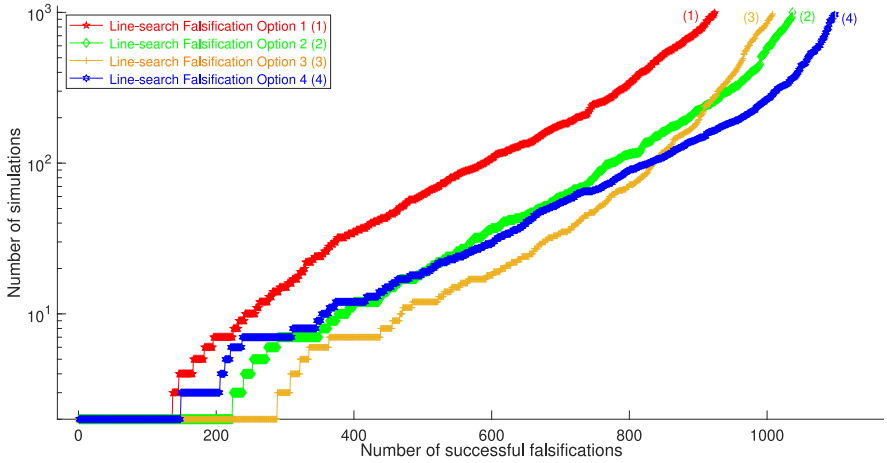

Fig. 5. Cactus plot showing the performance of LSF using different options for all examples. The plotted values show how many successful falsifications (x-axis) were completed in less than 1000 simulations (y-axis, logarithmic scale).

it is not possible to falsify by Corners or UR. It is only for a subset of the specifications and models that an optimization method can perform better than an optimization-free approach. These are specifications 1 and 7-9 of the AT example for Instance 2, Table I; the fourth specification of both instances 1 and 2 of the CC example, Table II; specifications 6 for both $T = 10$, $T = 12$ and 7 of AT$'$, Table III; the first specification of the Modulator example, Table IV; the second specification of the NN example for Instance 2, Table V; and the F16 example Table VI. For some specifications and systems, there is a clear tendency for a specific optimization method to perform better.

Instance 2 of the AT example, Table I, is an excellent example to show the performance of LSF. Except for φ_1^{AT}, LSF manages to falsify all specifications 100% of the runs, while NM cannot falsify any of the φ_i^{AT} with $i = 7, 8, 9$. There is also a significant advantage for LSF relative to SNOBFIT for these specifications. Note that, φ_1^{AT} can be falsified using a method not considered in this article [30].

For the CC example (Table II), in Instance 1 with φ_4^{CC}, while NM is not successful in the falsification process, SNOBFIT, and LSF are successful in some runs, and SNOBFIT works better than LSF. In Instance 2 of this example, LSF is able to falsify φ_4^{CC} in 10% of the runs, while neither of NM and SNOBFIT are successful. For the AT$'$ (Table III), only in the two specifications $\varphi_6^{\mathrm{AT}'}(\mathrm{T} = 10, \text{and} \mathrm{T} = 12)$, NM performs better than the others. Compared to SNOBFIT, which is not successful for $\varphi_6^{\mathrm{AT}'}(\mathrm{T} = 12)$, LSF performs better and manages to falsify this specification in some runs. $\varphi_7^{\mathrm{AT}'}$ is the only specification of this example where SNOBFIT works better than LSF. For the $\Delta - \Sigma$ Modulator example (Table IV), LSF manages to falsify to 100%, which is a slight improvement over SNOBFIT, and a big improvement over NM. For the NN example, (Table V), in Instance 2 of specification φ_2^{NN}, LSF and SNOBFIT perform similarly to and better than NM, respectively. For the F16 example (Table VI), LSF and SNOBFIT work similarly to and better than NM, respectively.

Note that the SC example, Table VI, is one example where it does not matter which the semantics or optimization method is used since none of them can falsify the specification for both instances. Yaghoubi and Fainekos [32] used a Simulated Annealing global search in combination with an optimal

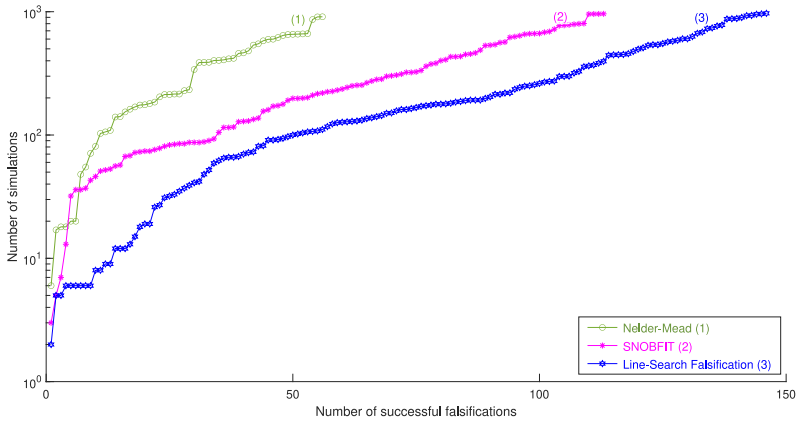

Fig. 6. Cactus plot showing the performance of each optimization-based methods for those examples where using an optimization-based approach to improve the falsification process is needed. The plotted values show how many successful falsifications (x-axis) were completed in less than 1000 simulations (y-axis, logarithmic scale).

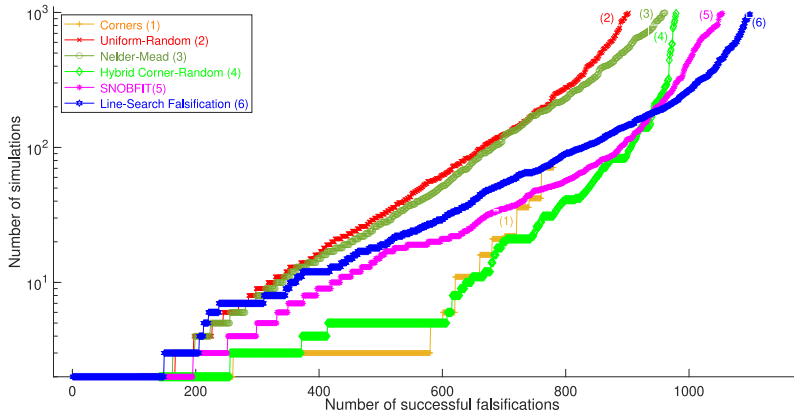

Fig. 7. Cactus plot showing the performance of all evaluated methods in this article for all examples. The plotted values show how many successful falsifications (x-axis) were completed in less than 1000 simulations (y-axis, logarithmic scale).

control-based local search on the infinite-dimensional input space to successfully falsify the specification for Instance 1.

An aggregated comparison is shown in Fig. 6. As can be seen, and is evident from the discussion above, LSF outperforms NM. LSF can falsify using fewer simulations, which is important as simulations are costly. LSF also works better than SNOBFIT while being simple.

D. Evaluation and Discussion

An aggregated comparison among all methods presented and discussed in this article for all examples is shown in Fig. 7. As can be seen, HCR not only manages to falsify more examples than Corners and UR but also has a better performance than NM.

LSF, compared to the other evaluated optimization free and optimization-based approaches, falsifies the most examples, as shown in Fig. 7. The LSF method covers the corner points and combines randomly generated lines in the n-dimensional parameter range with local search. This method has a simple strategy compared to SNOBFIT, but a performance that is on par with or better than SNOBFIT on the benchmark problems. LSF can falsify some examples that none of the other evaluated methods could successfully falsify. Instance 2 of CC is one example that shows the better performance of LSF.

The main focus in this article has been to develop and describe an approach that works well for falsification problems by investigating and exploiting the typical properties of these problems. It is beyond the scope of this article to discuss statistical guarantees or local and/or global convergence. However, we note that global convergence can typically be shown for random search algorithms, but local convergence requires assumptions on the underlying function. The evaluation is instead done using available benchmark problems. The nature of the complexity of the SUT present in many industrial applications, typically having discrete modes where the dynamics might change completely, invalidates the assumptions typically necessary for giving any guarantees.

We have evaluated the approach on a set of benchmark problems that are available within the falsification community. We encourage researchers to post more problems, problems of larger complexity, and problems with more involved specifications, since this would increase the quality of the evaluations. However, it is important to note that the problems used are not cherry-picked to show the benefits of the proposed approach, but they are the set of problems available to us.

VI. Conclusion

In this article, two optimization methods have been proposed to enhance the falsification of CPSs. The first method is the combination of Corners and UR, *Hybrid Corner-Random*. For systems that are falsified on the parameter corners or by randomly choosing parameters, this method manages to falsify in the fewest number of simulations. On the other hand, there are specifications and models for which neither Corners nor UR are efficient, and then optimization-based methods are needed.

The second proposed method is a gradient-free optimization-based method, called *Line-Search Falsification*, that optimizes over line segments through a vector of inputs in the n-dimensional parameter space. The method is designed by using insights from the optimization-free approaches in order to combine randomness, corner points, and local search. This method is compared to the NM and SNOBFIT methods. The proposed method has been evaluated on standard benchmark problems. LSF has better performance than all other evaluated methods in this article.

A conclusion that can be drawn from the combinations of methods and quantitative semantics is that the choice of the optimization method appears to matter more than the choice of quantitative semantics [34]. This conclusion is counter-intuitive, but the evaluation of the benchmark problems shows that for falsification of CPSs the problem is not only about solving the optimization problem as efficiently as possible but also about exploring corner cases and the parameter space.

Testing benefits from running a large number of simulations. If the closed-loop system can be simulated, testing can take advantage of computing clusters to run the models in parallel, using thousands of computing nodes. The proposed methods can easily be adapted to execute on computing clusters, thus raising the confidence in the correctness of the SUT.

LSF uses heuristics in its internal process. For future work, it would be interesting to evaluate the efficiency of using different heuristics. In particular, it would be interesting to evaluate the suggested LSF method on several large-scale industrial applications.

Acknowledgment

The simulations of all benchmark problems in this article were performed on resources at High Performance Computing Center North (HPC2N), Umeå University, a Swedish national center for Scientific and Parallel Computing.

References

[1] R. Alur, *Principles of Cyber–Physical Systems*. Cambridge, MA, USA: MIT Press, 2015.

[2] S. Mitra, *Verifying Cyber–Physical Systems: A Path to Safe Autonomy*. Cambridge, MA, USA: MIT Press, 2021.

[3] E. M. Clarke, E. A. Emerson, and J. Sifakis, "Model checking: Algorithmic verification and debugging," *Commun. ACM*, vol. 52, no. 11, pp. 74–84, 2009.

[4] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" in *Proc. 27th Annu. ACM Symp. Theory Comput.*, 1995, pp. 373–382.

[5] E. Bartocci *et al.*, "Specification-based monitoring of cyber–physical systems: A survey on theory, tools and applications," in *Lectures on Runtime Verification*. Cham, Switzerland: Springer, 2018, pp. 135–175.

[6] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Syst.*, vol. 2, no. 4, pp. 255–299, 1990.

[7] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Heidelberg, Germany: Springer, 2004, pp. 152–166.

[8] J. L. Eddeland, K. Claessen, N. Smallbone, Z. Ramezani, S. Miremadi, and K. Åkesson, "Enhancing temporal logic falsification with specification transformation and valued booleans," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 5247–5260, Dec. 2020.

[9] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proc. 18th Int. Conf. Hybrid Syst. Comput. Control*, 2015, pp. 239–248.

[10] 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.

[11] K. Claessen, N. Smallbone, J. Eddeland, Z. Ramezani, and K. Åkesson, "Using valued booleans to find simpler counterexamples in random testing of cyber-physical systems," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 408–415, 2018.

[12] S. Amaran, N. Sahinidis, B. Sharda, and S. Bury, "Simulation optimization: A review of algorithms and applications," *Ann. Oper. Res.*, vol. 240, no. 1, pp. 351–380, 2016.

[13] C. Audet and W. Hare, *Derivative-Free and Blackbox Optimization* (Operations Research and Financial Engineering). Cham, Switzerland: Springer, 2017.

[14] R. Hooke and T. A. Jeeves, "'Direct search' solution of numerical and statistical problems," *J. ACM*, vol. 8, no. 2, pp. 212–229, 1961.

[15] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Comput. J.*, vol. 7, no. 4, pp. 308–313, 1965.

[16] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2016.

[17] J. Eddeland, J. G. Cepeda, R. Fransen, S. Miremadi, M. Fabian, and K. Åkesson, "Automated mode coverage analysis for cyber-physical systems using hybrid automata," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9260–9265, 2017.

[18] L. Rios and N. Sahinidis, "Derivative-free optimization: A review of algorithms and comparison of software implementations," *J. Global Optim.*, vol. 56, no. 3, pp. 1247–1293, 2013.

[19] C. Audet and J. E. Dennis, Jr., "Mesh adaptive direct search algorithms for constrained optimization," *SIAM J. Optim.*, vol. 17, no. 1, pp. 188–217, 2006.

[20] W. Huyer and A. Neumaier, "Snobfit—Stable noisy optimization by branch and fit," *ACM Trans. Math. Softw.*, vol. 35, no. 2, pp. 1–25, 2008.

[21] J. L. Eddeland, S. Miremadi, and K. Åkesson, "Evaluating optimization solvers and robust semantics for simulation-based falsification," in *Proc. 7th Int. Workshop Appl. Verification Continuous Hybrid Syst.*, 2020, pp. 259–266.

[22] A. Arcuri, M. Z. Iqbal, and L. Briand, "Random testing: Theoretical results and practical implications," *IEEE Trans. Softw. Eng.*, vol. 38, no. 2, pp. 258–277, Mar./Apr. 2012.

[23] A. Groce, G. Holzmann, and R. Joshi, "Randomized differential testing as a prelude to formal verification," in *Proc. 29th Int. Conf. Softw. Eng. (ICSE)*, 2007, pp. 621–631.

[24] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball, "Feedback-directed random test generation," in *Proc. 29th Int. Conf. Softw. Eng. (ICSE)*, 2007, pp. 75–84.

[25] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. Tse, "Adaptive random testing: The art of test case diversity," *J. Syst. Softw.*, vol. 83, no. 1, pp. 60–66, 2010.

[26] G. Gay, M. Staats, M. Whalen, and M. P. E. Heimdahl, "The risks of coverage-directed test case generation," *IEEE Trans. Softw. Eng.*, vol. 41, no. 8, pp. 803–819, Aug. 2015.

[27] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 10, pp. 281–305, 2012.

[28] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-TaLiRo: A tool for temporal logic falsification for hybrid systems," in *Tools and Algorithms for the Construction and Analysis of Systems*. Heidelberg, Germany: Springer, 2011, pp. 254–257.

[29] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds. Heidelberg, Germany: Springer, 2010, pp. 167–170.

[30] G. Ernst *et al.*, "ARCH-COMP 2019 category report: Falsification," in *Proc. 6th Int. Workshop Appl. Verification Continuous Hybrid Syst.*, vol. 61, 2019, pp. 129–140.

[31] Z. Ramezani, J. L. Eddeland, K. Claessen, M. Fabian, and K. Åkesson, "Multiple objective functions for falsification of cyber-physical systems," *IFAC-PapersOnLine*, vol. 53, no. 4, pp. 417–422, 2020.
[32] S. Yaghoubi and G. Fainekos, "Gray-box adversarial testing for control systems with machine learning components," in *Proc. 22nd ACM Int. Conf. Hybrid Syst. Comput. Control*, 2019, pp. 179–184.
[33] J. Lagarias, J. Reeds, M. Wright, and P. Wright, "Convergence properties of the Nelder–Mead simplex method in low dimensions," *SIAM J. Optim.*, vol. 9, no. 1, pp. 112–147, 1998.
[34] Z. Ramezani, K. Claessen, N. Smallbone, M. Fabian, and K. Åkesson, "Testing cyber-physical systems using a line-search falsification method," Jul. 2021. [Online]. Available: 10.36227/techrxiv.14555826.v3.

Zahra Ramezani received the B.Sc. and M.Sc. degrees in electrical engineering from the Iran University of Science and Technology, Tehran, Iran, in 2011 and 2013, respectively. She is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden.

Her research interest is testing of cyber–physical systems.

Koen Claessen received the Ph.D. degree in computer science from the Chalmers University of Technology, Gothenburg, Sweden, in 2001.

He has been a Full Professor of Computer Science with the Chalmers University of Technology, since 2010. His research interests include software testing (particularly random property-based testing), formal verification, automated reasoning, and functional programming.

Nicholas Smallbone received the Ph.D. degree in computer science from the Chalmers University of Technology, Gothenburg, Sweden, in 2013.

Then, he has been employed with the Chalmers University of Technology first as a Postdoctoral Researcher and currently as a Researcher. His research interests include software testing (particularly random property-based testing), formal verification, automated reasoning, and functional programming.

Martin Fabian (Senior Member, IEEE) received the Ph.D. degree in control engineering from the Chalmers University of Technology, Gothenburg, Sweden, in 1995.

He is a Full Professor of Automation and the Head of the Automation Research Group, Department of Electrical Engineering, Chalmers University of Technology. He has authored more than 200 publications, and is a Co-Developer of the formal methods tool Supremica, which implements several state-of-the-art algorithms for supervisory control synthesis. His research interests include formal methods for automation systems in a broad sense, merging the fields of control engineering and computer science.

Knut Åkesson (Associate Member, IEEE) received the Master of Science degree in computer science and technology from the Lund Institute of Technology, University of Lund, Lund, Sweden, and the Ph.D. degree in control engineering from the Chalmers University of Technology, Gothenburg, Sweden, in 1997.

He is a Professor with the Department of Electrical Engineering, Chalmers University of Technology. His main research is in using rigorous methods for analysis of cyber–physical systems.