# Stochastic Local Search for Falsification of Hybrid Systems*

Jyotirmoy Deshmukh[1], Xiaoqing Jin[1], James Kapinski[1], Oded Maler[2]

[1] Toyota Technical Center
{jyotirmoy.deshmukh, xiaoqing.jin, jim.kapinski}@tema.toyota.com,
[2] Verimag
Oded.Maler@imag.fr

**Abstract.** Falsification techniques for models of embedded control systems automate the process of testing models to find bugs by searching for model-inputs that violate behavioral specifications given by logical and quantitative correctness requirements. A recent advance in falsification is to encode property satisfaction as a cost function based on a finite parameterization of the (bounded-time) input signal, which allows formulating bug-finding as an optimization problem. In this paper, we present a falsification technique that uses a local search technique called *Tabu search* to search for optimal inputs. The key idea is to discretize the space of input signals and use the *Tabu list* to avoid revisiting previously encountered input signals. As local search techniques may converge to local optima, we introduce stochastic aspects such as random restarts, sampling and probabilistically picking suboptimal inputs to guide the technique towards a global optimum. Picking the right parameterization of the input space is often challenging for designers, so we allow dynamic refinement of the input space as the search progresses. We implement the technique in a tool called SITAR, and show scalability of the technique by using it to falsify requirements on an early prototype of an industrial-sized automotive powertrain control design.

## 1 Introduction

Embedded control governing safety-critical aspects in medical devices, avionics and automotive systems are increasingly being designed using the model-based development (MBD) paradigm. The early phase of MBD involves rapid iterations to check the correctness of the control software or to check the feasibility of a new control algorithm. Design models are usually closed-loop models, i.e., a plant model in a feedback loop with a controller model. Plant models represent the dynamic, physical behavior of the environment that is to be controlled (e.g., an engine, a powertrain system, a human heart, an avoinics power-distribution system). Controllers are modeled as a reactive computer program interacting in real-time with the plant, and are typically designed in a visual block-diagram based language such as Simulink®. The closed-loop system also has exogenous inputs, usually modeling user events or other disturbances from the environment. As fixing software issues in late design stages is expensive, identifying such bugs in the early phase of the design cycle is valuable.

* Oded Maler's research was supported in part by the ANR project CADMIDIA and Toyota.

Closed-loop models can be modeled as hybrid dynamical systems, and it is well-known that for even simple hybrid systems, the verification problem is highly undecidable [4, 11]. Prevalent practice in industry is extensive model-based testing, where control designers use heuristics and previous experience to pick a set of test input signals and system parameters to stimulate the system-under-test. Modeling environments usually have numerical simulation to estimate the system behavior for such inputs. Designers typically perform a finite number of such simulations over the chosen set of inputs and parameters, and identify undesirable behaviors by manual inspection of the resulting outputs. These techniques are incomplete in the sense that they provide no guarantees on whether a bug is found; however, they significantly increase the possibility of identifying problems early in the system design.

Recently developed falsification techniques seek to automate this process in many ways. First, they allow the designer to express correct behavior as a set of logical or quantitative requirements on the inputs and outputs in a machine-checkable format. Second, they allow formulating the search for errors as an optimization problem by encoding property satisfaction by a function that maps a given input/output signal and a logical requirement to a real number. A recent development is the advent of quantitative semantics for real-time temporal logics. Fainekos and Pappas define a robust semantics for metric temporal logic (MTL) [10], which allows to quantify *how much* a signal satisfies a specification. Similarly, Donzé and Maler define robust semantics for signal temporal logic (STL), which permits a similar analysis [8].

A core step in falsification is thus to find a (bounded-time) input signal that minimizes the cost function. Falsification tools typically rely on a global optimization algorithm to perform this task. Most falsification techniques, are often ineffective when the cost function is nonconvex or discontinuous, or if the underlying model has discontinuous dynamic behavior. The S-TaLiRo tool has several algorithms to perform global optimization including simulated annealing, ant-colony optimization, and the cross-entropy method among others [15, 2, 18, 3]. The Breach [7] tool provides a similar framework but uses nonlinear-simplex optimization, also known as the Nelder-Mead algorithm. Other falsification tools such as those based on the RRT algorithm [16, 17, 9], multiple-shooting [20], combined global and local search [14] and gradient descent-based local search [1] have also shown promise in exploring hybrid state-spaces and complex, nonlinear cost surfaces.

In this paper, we continue the quest to find an effective global optimization algorithm. In particular, we focus on addressing models whose structure reflects typical control designs found in industrial systems. Common features of these systems include: presence of Boolean combinations of predicates on real-valued input signals (typically to decide an operating mode of the system), discrete switching influenced strongly by the shape of the input signal and to a lesser extent by the state of the sytem, and highly nonlinear and occasionally discontinuous dynamics in the plant models. We present an adaptation of a discrete optimization technique known as Tabu search to address systems with such features, and make a case for the effectiveness of our technique with experimental evidence.

Tabu search is a *meta heuristic* applied to a lower-level heuristic method used to solve an optimization problem. It essentially restricts the search of a discrete decision

space so that particular valuations of the decision variables are not revisited once they have been evaluated using a data structure known as the *Tabu list*. The technique has been applied to integer programming problems and is mainly used to prevent cycling that would eventually occur while searching finite spaces [6].

In applying Tabu search, we handle the continuous input signals by discretizing the solution space. In Section 3, we explain how the algorithm locally searches in a neighborhood of an input signal (i.e., neighboring points on the discrete parameterization of the input signals) and follows a descent direction obtained by stochastically approximating the gradient of the cost function. Naïve local search techniques can converge to local optima. Also, an interesting case is when the output signals are Boolean or if their values are closely dependent on some internal Boolean conditions in the models. Here, the cost function has "plateaus" and "narrow valleys," i.e., the surface of the function to be optimized has regions that are flat with narrow regions where the cost is significantly lower. By adding stochastic aspects such as random restarts, and allowing the algorithm to pick sub-optimal inputs with a small probability, we allow the Tabu search to escape local optima and increase its chances of reaching the global optimum.

A key consideration for us is the ease of use of the tool by control designers; hence, we avoid techniques that seek user annotations to assist extraction of the underlying hybrid structure of the model [5]. Furthermore, when engineers use falsification tools like S-TaLiRo or Breach, they must parameterize the space of inputs to their models for the tools to work. This parameterization is usually in the form of a list of uniformly spaced control points, i.e., a list of times at which the optimizer is free to pick a value for the input signal. For the intermediate times, a suitable interpolation scheme is used to define a continuous input signal. A key challenge in using such tools is that the verification engineers need to have good insight into how many control points to choose. If too many control points are chosen, then the input search space becomes large, and the efficiency of the optimizer suffers. On the contrary, if too few control points are chosen, then the tool may be unable to find problematic behaviors that rely on having flexibility in the shape of the input signal. In Section 4, we show how we can allow designers to specify a coarse discretization of the input space, and we provide a mechanism to automatically refine the discretization to incrementally increase the accuracy of the optimization.

We have implemented stochastic local Tabu search with refinements in a tool named SITAR. In Section 5, we show how we use SITAR in a falsification framework and demonstrate its efficacy on a range of benchmarks, starting from toy models that pose challenges to existing falsification engines to industrial-sized benchmarks. Finally, we conclude with some promising future directions in Section 6.

## 2 Preliminaries

We denote the system under test $\mathcal{S}$. We assume that $\mathcal{S}$ is given by some model of an embedded control system and is equipped with a *simulator*, which is capable of computing *typed* output sequences generated by $\mathcal{S}$ under a given *typed* input sequence. A typed input sequence is a sequence of time-value pairs, where the values lie in a set known as the domain. We note that input domains can be finite, i.e., finite subsets of sets such as $\mathbb{Z}$ and $\mathbb{B}$, or could be compact (i.e., bounded and closed) subsets of sets

such as $\mathbb{R}$. Let $\mathcal{W} = \mathcal{W}_1 \times \ldots \times \mathcal{W}_m$, where each $\mathcal{W}_i \subseteq \mathbb{R}$ is a dense domain. Let $\mathcal{V}$ be a subset of the Cartesian product of some finite domains. Then, in general, the domain of values is $\mathcal{U} = \mathcal{W} \times \mathcal{V}$. Let $\mathcal{U}_T$ be the set of all input sequences over time bound $T$. An input sequence is defined as:

$$\mathbf{u} = (u_0, t_0), (u_1, t_1), \cdots, (u_N, t_N),$$

where $u_i \in \mathcal{U}$, $t_i \leq t_{i+1}$ for each $0 \leq i < N$, and $t_N \leq T$. Note that input sequences are often used to represent continuous-time signals; sequences can be used to represent a strictly discrete-time signal or can be used to represent the parameterization of a continuous-time signal[3]. We similarly define $\mathcal{Y}$ as an output set and $\mathcal{Y}_T$ as the set of all output sequences. Outputs are given as a sequence

$$\mathbf{y} = (y_0, \tilde{t}_0), (y_1, \tilde{t}_1), \cdots, (y_M, \tilde{t}_M),$$

where $y_i \in \mathcal{Y}$, $\tilde{t}_i \leq \tilde{t}_{i+1}$ for each $0 \leq i < M$, and $\tilde{t}_M \leq T$. We use the notation $\mathcal{S}(\mathbf{u})$ to mean the output sequence given by $\mathcal{S}$ under input $\mathbf{u}$. Note that the sequence of time instants associated with a given $\mathbf{u}$, $t_i$, does not necessarily match the time instants associated with the corresponding $\mathbf{y}$, $\tilde{t}_i$.

We define a property that should hold for $\mathcal{S}$ in terms of a property function. Let $\varphi : \mathcal{U}_T \times \mathcal{Y}_T \to \mathbb{R}$ be a function that maps an input and output sequence to a real value that determines whether system $\mathcal{S}$ under a given input sequence satisfies the desired property. Positive valuations of $\varphi$ indicate that the desired property is satisfied and negative values of $\varphi$ indicate that the desired property is not satisfied. Furthermore, the magnitude of the valuation indicates *how much* the property is satisfied. This function provides the cost used by an optimization tool to search for input sequences that give rise to output sequences that demonstrate incorrect behavior from $\mathcal{S}$. The falsification problem can then be cast as the following optimization problem.

$$\min_{\mathbf{u} \in \mathcal{U}_T} \varphi(\mathbf{u}, \mathbf{y}) \quad \text{subject to } \mathbf{y} = \mathcal{S}(\mathbf{u}) \tag{1}$$

Any assignment of the decision variable $\mathbf{u}$ that produces a negative value from the cost function is an input sequence that demonstrates a behavior from $\mathcal{S}$ that fails to satisfy the desired property defined by $\varphi$.

## 3 Stochastic Local Tabu Search

Local search is the discrete/combinatorial variant of steepest-descent, gradient-based methods for solving global optimization problems, such as the optimization problem indicated in (1). Local search is based on defining a distance function on the solution space, where typically the distance between two points is related to the number of modifications needed to transform one point into the other. The neighborhood of a point is typically the set of neighbors at distance 1 or some subset of this set. A key feature of

[3] For inputs representing continuous functions over dense time, the actual input signal to $\mathcal{S}$ is obtained by interpolating across the sequence $\mathbf{u}$ using a user-specified interpolation scheme.

Tabu search (a variant of local search) is to utilize a data structure called a *Tabu list*, to avoid repeated computations on visited points. In this section, we present a modification of the Tabu search procedure that allows systematic exploration of a finite parameterization of the input sequences of a model with the goal of falsifying a given quantitative property specification. We first introduce some required terminology.

### 3.1 Discretization and Neighborhoods

Note that, in general, the space of input sequences $\mathcal{U}_T$ is infinite. Without loss of generality, we can assume that the dense input domain ($\mathcal{W}$) is the set $[0, u_{\max}]^m$. For ease of exposition, we will start with a discretized representation of signals and a fixed discretization scheme. We assume a time step $\varepsilon$ and space increment $\delta$. See Fig. 1 (a) for an illustration.

**Definition 1 (Uniform $\delta$-Discretization of Input Domain).** *For a given $\delta$, let $\ell = \left\lfloor \frac{u_{\max}}{\delta} \right\rfloor$. Let $\Delta_{\mathrm{u}}(\delta, \mathcal{W})$ denote a uniform grid over $\mathcal{W}$, i.e., $\Delta_{\mathrm{u}}(\delta, \mathcal{W}) = \{j \cdot \delta \mid 0 \leq j \leq \ell\}^m$. A uniform $\delta$-discretization of the input domain $\mathcal{U}$ is then defined as $\Delta_{\mathrm{u}}(\delta, \mathcal{U}) = \Delta_{\mathrm{u}}(\delta, \mathcal{W}) \times \mathcal{V}$.*

**Definition 2 (Uniform $(\delta, \varepsilon)$-Discretization of Input Sequence Space).** *Let $\tau = \left\lfloor \frac{T}{\varepsilon} \right\rfloor$. We abuse notation and let $\Delta_{\mathrm{u}}(\varepsilon, T)$ denote a uniform discretization of time, i.e., $\Delta_{\mathrm{u}}(\varepsilon, T) = \{j \cdot \varepsilon \mid 0 \leq j \leq \tau\}$. A uniform $(\delta, \varepsilon)$-discretization of $\mathcal{U}_T$, denoted $\Delta_{\mathrm{u}}(\delta, \varepsilon, \mathcal{U}_T)$ is defined as the finite set of sequences $\hat{\mathbf{u}}$ of the form $(\hat{u_0}, \hat{t_0}), \ldots, (\hat{u_N}, \hat{t_N})$, where for each $i$, $\hat{u}_i \in \Delta_{\mathrm{u}}(\delta, \mathcal{U})$, and $\hat{t}_i \in \Delta_{\mathrm{u}}(\varepsilon, T)$.*

Having fixed the search-space, we define distance and neighborhoods in this space. For the finite input domain $\mathcal{V}$, we assume that there is function $N_{\mathcal{V}}(v)$ mapping each element $v \in \mathcal{V}$ to a set of neighbors in $\mathcal{V}$. For example, if $\mathcal{V}$ is the following set of integers: $\{1, 3, 5, 8\}$, then $N_{\mathcal{V}}(5)$ may be defined as $\{3, 8\}$.

**Definition 3 (Neighborhood in Input Domain).** *Consider a $u = (w^1, \ldots w^m, v) \in \Delta_{\mathrm{u}}(\delta, \mathcal{U})$. For a given $\delta$, the $(i, \delta, \mathcal{W})$-neighbor of $u$ is defined as:*

$$nb_{\mathcal{W}}(i, \delta, u) = (w^1, \ldots, w^i + \delta, \ldots, w^m, v).$$

*Let $D = \{-\delta, +\delta\}$. The $\mathcal{W}$-neighborhood of $u$ is defined as:*

$$N_{\mathcal{W},\delta}(u) = \bigcup_{i=1}^{m} \bigcup_{\delta \in D} nb_{\mathcal{W}}(i, \delta, u).$$

*We abuse notation and let the $\mathcal{V}$-neighborhood of $u$ be defined as:*

$$N_{\mathcal{V}}(u) = \{(w^1, \ldots, w^m, v') \mid v' \in N_{\mathcal{V}}(v)\}.$$

*Finally, we define the* neighborhood *of $u$, $N_{\mathcal{W},\delta,\mathcal{V}}(u) = N_{\mathcal{W},\delta}(u) \cup N_{\mathcal{V}}(u)$, and say that $u'$ is a* neighbor *of $u$ if $u' \in N_{\mathcal{W},\delta,\mathcal{V}}(u)$.*

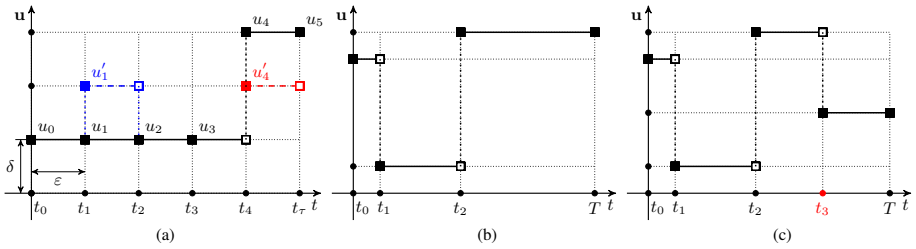


**Fig. 1.** Fig. (a) shows an instance of $\Delta_{\mathrm{u}}(\delta, \varepsilon, \mathcal{U}_T)$. For the input sequence $\mathbf{u} = (u_0, t_0),(u_1, t_1),(u_2, t_2),(u_3, t_3),(u_4, t_4),(u_5, t_\tau)$, the sequences $\mathbf{u}_1 = (u_0, t_0), (u'_1, t_1), (u_2, t_2), (u_3, t_3), (u_4, t_4), (u_5, t_\tau)$ and $\mathbf{u}_2 = (u_0, t_0), (u_1, t_1), (u_2, t_2), (u_3, t_3), (u'_4, t_4), (u_5, t_\tau)$, are two neighbors, shown in blue and red resp. Fig. (b) and (c) show an instance of $\Delta_{\mathrm{nu}}(\boldsymbol{\delta}, \boldsymbol{\varepsilon}, \mathcal{U}_T)$, where (c) shows a refinement of the space in (b) by adding a new time ($t_3$).

**Definition 4 (Neighborhood in Input Sequences).** *Given an input sequence* $\mathbf{u}$*, its* $j$*-neighborhood* $N_{\mathcal{U}_T}(j, \mathbf{u})$ *is the set of sequences:*

$$N_{\mathcal{U}_T}(j, \mathbf{u}) = \{(u_0, t_0), \cdots, (u'_j, t_j), \cdots, (u_N, t_N) \mid u'_j \in N_{\mathcal{W},\delta,\mathcal{V}}(u_j)\}.$$

*Given a time horizon* $T$*, the neighborhood of* $\mathbf{u}$ *is defined as:*

$$N_{\mathcal{W},\delta,\mathcal{V},\varepsilon}(\mathbf{u}) = \bigcup_{0 \le j \le \tau} N_{\mathcal{U}_T}(j, \mathbf{u}).$$

For an illustration of neighbors, see Fig. 1. Uniform discretizations provide a simple way to define a quantization of the decision space for the falsification problem, but it produces a decision space that increases exponentially as $\delta$ decreases, and it does not provide the flexibility to define a refinement of the decision space in specific regions. To address these deficiencies, we define a generalization of the uniform discretization notions that permits uneven discretizations. See Fig. 1(b) for an illustration.

**Definition 5 (Nonuniform $\boldsymbol{\delta}$-Discretization of the Input Domain).** *Let* $\boldsymbol{\delta}$ *be a nonuniform grid over the input domain, i.e.,* $\boldsymbol{\delta} = (\widehat{\mathcal{W}}_1, \ldots, \widehat{\mathcal{W}}_m)$*, where each* $\widehat{\mathcal{W}}_i$ *is a finite set of elements of* $\mathcal{W}_i$*. A nonuniform* $\boldsymbol{\delta}$*-discretization of* $\mathcal{U}$*, denoted as* $\Delta_{\mathrm{nu}}(\boldsymbol{\delta}, \mathcal{U})$ *is the set* $\widehat{\mathcal{W}}_1 \times \ldots \times \widehat{\mathcal{W}}_m \times \mathcal{V}$*.*

**Definition 6 (Nonuniform $(\boldsymbol{\delta}, \boldsymbol{\varepsilon})$-Discretization of the Input Sequence Space).** *Let* $\boldsymbol{\varepsilon}$ *denote a nonuniform discretization of the time domain, where* $\boldsymbol{\varepsilon}$ *is a finite set of elements from* $[0, T]$*. Given a nonuniform* $\boldsymbol{\delta}$*-discretization of* $\mathcal{U}$*, and* $\boldsymbol{\varepsilon}$*, a nonuniform* $(\boldsymbol{\delta}, \boldsymbol{\varepsilon})$*-discretization of* $\mathcal{U}_T$*, denoted* $\Delta_{\mathrm{nu}}(\boldsymbol{\delta}, \boldsymbol{\varepsilon}, \mathcal{U}_T)$*, is a finite set of input sequences* $\hat{\mathbf{u}} = (\hat{u_0}, \hat{t_0}), \ldots, (\hat{u_N}, \hat{t_N})$*, such that for all* $i$*,* $\hat{u_i} \in \Delta_{\mathrm{nu}}(\boldsymbol{\delta}, \mathcal{U})$ *and* $\hat{t_i} \in \boldsymbol{\varepsilon}$*.*

The notion of neighborhoods easily extends to nonuniformly distributed input spaces and input sequence spaces, and we denote them as $N_{\mathcal{W},\boldsymbol{\delta},\mathcal{V}}$ and $N_{\mathcal{W},\boldsymbol{\delta},\mathcal{V},\boldsymbol{\varepsilon}}(\mathbf{u})$ respectively.

In the sequel, we describe an implementation of the stochastic local Tabu search method that can be configured to use either uniform or nonuniform discretizations of

**Algorithm 1:** Stochastic Local Tabu Search over given discretization of $\mathcal{U}_T$.

**Input**: Model $\mathcal{S}$, Input Sequence $\mathbf{u}$, grid over input domain $\boldsymbol{\delta}$, time domain discretization $\boldsymbol{\varepsilon}$, Property function $\varphi$, `maxLocalImprovements`, `maxRestarts`

**Output**: Minimum cost $c_{\min}$, Minimizing input $\mathbf{u}_{\min}$

```
1  c_min, c_prev := ∞ ; i, j := 0; TabuList := ∅
2  while i < maxRestarts do
3      while j < maxLocalImprovements do
4          TabuList.add(u)
5          y := Simulate(S, u) ; c := φ(u, y)
6          if c < c_min then c_min := c; u_min := u
7          neighborsVisited := 0
8          while neighborsVisited < maxNeighbors do
9              k := 0
10             do
11                 u_nb := pickNeighbor(δ, ε, u) ; k := k + 1
12             while u_nb ∈ TabuList ∧ k < |N_{W,δ,V,ε}(u)|
13             if u_nb = ∅ then break
14             neighborsVisited := neighborsVisited + 1
15             y_nb := Simulate(S, u_nb), c_nb := φ(u_nb, y_nb)
16             TabuList.add(u_nb)
17             if StoppingCondition(c_nb) then halt
18             if c_nb < c_min ∨ with probability P_subopt then
19                 c_min := c_nb ; u_min := u_nb
20         u := u_min
21         if slowConvergence(c_min, c_prev) ∨ localOptimum(c, c_min) then
22             break
23         j := j + 1; c_prev := c_min
24     do
25         u := pickRandomInput(δ, ε, U_T)
26     while u ∉ TabuList
27     i := i + 1
```

the decision space. In Algorithm 1, we show the basic steps of the search algorithm. The innermost while-loop (Lines 8-19) implements a stochastic local search scheme, augmented with a Tabu-list. A run of this while-loop to completion is called a single *local improvement*. The main steps executed in the loop are as follows: (1) Select a neighbor of the current input $\mathbf{u}$ (Line 11) that is not in the Tabu list. (2) Evaluate the cost of $\mathbf{u}_{\rm nb}$ (by running a simulation with $\mathbf{u}_{\rm nb}$ as input and computing the value of $\varphi(\mathbf{u}_{\rm nb}, \mathcal{S}(\mathbf{u}_{\rm nb}))$) (Line 15). (3) Add each $\mathbf{u}_{\rm nb}$ not previously in the Tabu list to the list (Line 16). (4) Once the desired number of neighbors have been visited, pick the minimal-cost neighbor as the next point in the search (Line 19). (5) Terminate when the stopping condition is reached (Line 17); typically this is $c_{\rm nb} < 0$. The while-loop from Line 3 to Line 23 iterates over the maximum number of local improvements permitted. The number `maxLocalImprovements` provides the user control over how much they wish to utilize the stochastic gradient-descent. A larger number is useful when the sur-

face of the cost function is smooth, while a smaller number is better to use when the cost function is highly discontinuous or with sharp valleys (as gradient-descent has less chance of success in this scenario). The outermost while-loop (Lines 2-27) iterates over the maximum number of random restarts permitted. We explain the purpose of random restarts in Sec. 3.3.

### 3.2 Local Search by Stochastic Gradient-Descent

The size of the neighborhood for a given point is $2m|\mathcal{V}|\tau$ and depending on $m$, $|\mathcal{V}|$, $\tau = \lfloor \frac{T}{\varepsilon} \rfloor$, and the cost of simulation, it might be too large to explore the entire neighborhood in each step. The procedure for selecting a subset of the neighborhood is stochastic: we sequentially sample up to `maxNeighbors` number of neighbors of $\mathbf{u}$ from $N_{\mathcal{W},\boldsymbol{\delta},\mathcal{V},\boldsymbol{\varepsilon}}(\mathbf{u})$. The actual sampling is effected by randomly selecting a $j$, and for the chosen $u_j$, randomly picking either a $\mathcal{V}$-neighbor or a $\mathcal{W}$-neighbor from $N_{\mathcal{V}}(u_j)$ or $N_{\mathcal{W},\delta}(u_j)$. If the latter is picked, the $\mathcal{W}$-neighbor $nb_{\mathcal{W}}(i, \delta, u_j)$ is obtained by randomly picking an $i \in [0, \lfloor \frac{u_{\max}}{\delta} \rfloor]$ and a $\delta \in \{+\delta, -\delta\}$. The random choice can be performed based on any distribution function on the set of $i, j$ indices or on the $\mathcal{V}$-neighborhood. In our implementation, we use uniform random sampling. This general technique of obtaining a stochastic approximation of the gradient is called the finite-difference stochastic approximation (FDSA) method [19], and is the technique currently implemented in our tool. The algorithm then moves to the neighbor with the lowest cost in the neighborhood (thus performing the steepest descent along the stochastically approximated gradient). If there is no neighbor with a lower cost, the algorithm infers that a local optimum has been reached (Line 21).

### 3.3 Random restarts and Stochasticity

Like any local search method there is a risk of getting stuck in a local optimum. We use two mechanisms to help the search procedure escape a local optimum:

1. We introduce jumps in the search procedure stochasitcally: If the algorithm detects a local optimum or slow convergence, or once it has exhausted the maximum permitted local improvements, it restarts the local search from a randomly chosen point in the input search space that is not in the Tabu list (Line 21).
2. In the phase where the algorithm is performing local search, we allow the algorithm to select a neighbor that is not optimum with a small probability (Line 18) in spirit of techniques such as simulated annealing [13].

To provide the control over termination of the algorithm, we permit the user to limit the number of random restarts (`maxRestarts`). Together, `maxRestarts`, `maxNeighbors`, and `maxLocalImprovements` influence the number of simulations performed by the algorithm. The performance of Algorithm 1 depends on several factors. The *size* of the search space influences how much we can explore it in finite time. Given a time horizon $T$, the search space defined by $\Delta_{\mathrm{u}}(\delta, \varepsilon, \mathcal{U}_T)$ or $\Delta_{\mathrm{nu}}(\boldsymbol{\delta}, \boldsymbol{\varepsilon}, \mathcal{U}_T)$ is finite and of size $O(\tau^{m|\mathcal{V}|\ell})$, i.e., it depends exponentially on $m$, the dimension of the dense input domain, maximum number of neighbors in the finite input domain (which in the worst

**Algorithm 2:** Input Search Space Refinement Procedure

**Input**: Model $\mathcal{S}$, Initial grid over input domain $\boldsymbol{\delta}_0$, Initial grid over time domain $\boldsymbol{\varepsilon}_0$, Property function $\varphi$, User-defined parameters: `maxRefinements`, `maxLocalImprovements`, `maxRestarts`

**Output**: Minimum cost $c_{\min}$, Minimizing input $\mathbf{u}_{\min}$

$\texttt{refinementNum} := 0$

$\boldsymbol{\delta}, \boldsymbol{\varepsilon} := \boldsymbol{\delta}_0, \boldsymbol{\varepsilon}_0$

$\mathbf{u} := \mathsf{pickRandomInput}(\boldsymbol{\delta}, \boldsymbol{\varepsilon}, \mathcal{U}_T)$

**while** $\texttt{refinementNum} < \texttt{maxRefinements}$ **do**

$$(c_{\min}, \mathbf{u}_{\min}) := \mathsf{StochasticLocalTabuSearch}\begin{pmatrix}\mathcal{S}, \mathbf{u}, \boldsymbol{\delta}, \boldsymbol{\varepsilon}, \varphi, \\ \texttt{maxLocalImprovements}, \\ \texttt{maxRestarts}\end{pmatrix}$$

  **if** $c_{\min} < 0$ **then**

    Report violation found, violating input: $\mathbf{u}_{\min}$

  **else**

$$\boldsymbol{\delta}, \boldsymbol{\varepsilon}, \mathbf{u}, \texttt{TabuList}, \texttt{maxLocalImprovements} := \mathsf{Refine}\begin{pmatrix}\boldsymbol{\delta}, \boldsymbol{\varepsilon}, \mathbf{u}, \texttt{TabuList}, \\ \texttt{maxLocalImprovements}\end{pmatrix}$$

  $\texttt{refinementNum} := \texttt{refinementNum} + 1$

case is $|\mathcal{V}| - 1$), and exponentially in $\ell$ or the number of discretizations of the signal domain, and linearly in $\tau$, i.e., the length of the input sequence. The efficiency of the operations depends on the *inner-most loop* of the procedure, which includes application of the mutation operations, the cost of running a single simulation and the size of the neighborhood (which determines the number of simulations in each step).

## 4 Search Space Refinement

We intend our technique to be used by designers who may not have good insight into the optimal discretization of the input space needed to find a falsifying $(\mathbf{u}, \mathbf{y})$ pair. A conservative assumption is that designers start with a coarse discretization of the input space. There are distinct advantages to starting with a coarse level of discretization: as the cardinality of the decision space is given by $\tau^{m|\mathcal{V}|\ell}$, larger values of $\delta$ and $\varepsilon$ result in a smaller search space; however, in most cases, the initial discretization is too coarse to be able to find the $(\mathbf{u}, \mathbf{y})$ with the optimal cost. To address this, our tool supports automatic and heuristic refinement of the discretization of the input space. We describe this procedure in Algorithm 2. In Algorithm 2, the key procedure is Refine. This is a heuristic step to increase the number of elements in the quantized search space. The Refine procedure currently supports the following heuristics:

1. Naïve Refinement: For uniform discretization, in each refinement iteration, we can set $\delta = \frac{\delta}{2}, \varepsilon = \frac{\varepsilon}{2}$. Note this quadratically increases the search space for each input dimension in each iteration, and can cause an exponential blowup in the number of refinements.
2. Input Domain Random Refinement: For a given discretization $\Delta_{\mathrm{nu}}(\boldsymbol{\delta}, \boldsymbol{\varepsilon}, \mathcal{U}_T)$, we choose (at random) an index $i$, and for the corresponding dense input domain $\widehat{\mathcal{W}_i} \in$

$\boldsymbol{\delta}$, we add at random a new value in $[0, \mathbf{u}_{max}]$ to $\widehat{\mathcal{W}_i}$. In other words, we increase the number of quantization levels in the $i^{th}$ dense input domain.

3. Time Domain Largest Gap Refinement: For a given discretization $\Delta_{\text{nu}}(\boldsymbol{\delta}, \boldsymbol{\varepsilon}, \mathcal{U}_T)$, where $\boldsymbol{\varepsilon} = \{t_0, \ldots, t_N\}$ and $t_N = T$, we find the index $j$ corresponding to the largest time-gap in $\boldsymbol{\varepsilon}$, and add a time-point there. Formally, $j = \underset{0 \le j \le N-1}{\arg\max} \{t_{j+1} - t_j \mid t_j, t_{j+1} \in \boldsymbol{\varepsilon}\}$. Then, we add the time-point $\frac{1}{2}(t_j + t_{j+1})$. E.g., if $\boldsymbol{\varepsilon} = \{0.0, 1.1, 5.3, 7.3, 10.0\}$, then we add the time 3.2 to $\boldsymbol{\varepsilon}$. We can combine this heuristic with the heuristic above or the one below. (See Fig. 1 (c) for an illustration).
4. Input Domain Largest Gap Refinement: We can use a similar scheme as the above heuristic to refine the input domain for a particular dense input domain. We first choose (at random) an index $i$, and then for the corresponding input domain $\widehat{\mathcal{W}_i} \in \boldsymbol{\delta}$, we add a value to $\widehat{\mathcal{W}_i}$ where the distance between adjacent elements is the largest.

**Definition 7 (Dense Neighborhoods).** *Given a point $u = (w^0, \ldots, w^m, v)$ in the input domain $\mathcal{U}$ and a $\delta$, a dense neighborhood of $u$ is defined as follows:*

$$DN_{\mathcal{W},\delta}(u) = \{(w^0 + \delta^0, \ldots, w^m + \delta^m, v) \mid |\delta^i| \le \delta\}.$$

*Given an input sequence $\mathbf{u} = (u_0, t_0), \ldots, (u_N, t_N)$, and a time perturbation step $\varepsilon$, a dense neighborhood of the input sequence $\mathbf{u}$ is defined as follows:*

$$DN_{\mathcal{W},\delta,\varepsilon}(\mathbf{u}) = \{(u'_0, t_0 + \varepsilon_0), \ldots, (u'_N, t_N + \varepsilon_N) \mid u'_j \in DN_{\mathcal{W},\delta}(u_j) \text{ and } \forall j : |\varepsilon_j| \le \varepsilon\}.$$

**Definition 8 (Robust Violation).** *We say that an input $\mathbf{u} = (u_0, t_0), \cdots, (u_N, t_N)$ robustly violates a property $\varphi$ if the following conditions hold:*

1. *The model interprets $\mathbf{u}$ as a piecewise constant function[4] $\mathbf{u}_c$ over $[t_0, t_N]$, where $\mathbf{u}_c$ is defined s.t. $\forall j \in [0, N-1]$, $\forall t \in [t_j, t_{j+1})$, $\mathbf{u}_c(t) = u_j$, and $\mathbf{u}_c(t_N) = u_N$.*
2. *$\mathbf{y} = \mathcal{S}(\mathbf{u}) \wedge \varphi(\mathbf{u}, \mathbf{y}) < 0$,*
3. *$\exists \delta^* > 0, \varepsilon^* > 0$, s.t. $\forall \mathbf{u}' \in DN_{\mathcal{W},\delta^*,\varepsilon^*}(\mathbf{u})$ it is true that $\mathbf{y}' = \mathcal{S}(\mathbf{u}') \wedge \varphi(\mathbf{u}', \mathbf{y}') < 0$.*

In other words, a violation is robust if for a given input sequence that violates the property, all sufficiently nearby input sequences also violate the property. In the following theorem, we characterize the asymptotic behavior of Algorithm 2. The inputs to Algorithm 2 include user-defined constants `maxLocalImprovements`, `maxRestarts`, and `maxRefinements`; we show that as the user makes these constants arbitrarily large, the probability of finding a robust violation goes to 1.

**Theorem 1.** *If the given system $\mathcal{S}$ has an input $\mathbf{u}^*$ that robustly violates the property $\varphi$, then as the choice for the parameters* `maxLocalImprovements`, `maxRefinements`, *and* `maxRestarts` *tend to $\infty$, with a suitable refinement scheme, the probability that Algorithm 2 finds an input $\mathbf{u}'$ such that $\varphi(\mathbf{u}', \mathbf{y}') < 0$, where $\mathbf{y}' = \mathcal{S}(\mathbf{u}')$, tends to 1.*

*Proof.* Due to lack of space, we give only a proof sketch for the case when we choose the naïve refinement scheme. Let $\mathbf{u}^* = (u_0, t_0), \ldots, (u_L, t_L)$ for some $L$, where $t_L = T$, and let $\delta^*$ and $\varepsilon^*$ be values that satisfy Condition 3 from Def. 8. We first show that as we increase `maxRefinements`, there is some (large enough) `refinementNum` (Line 10) for which the refinement generates uniform discretization $\Delta_{\text{u}}(\delta, \varepsilon, \mathcal{U}_T)$, such that

[4] We can also use piecewise linear interpolation to define $\mathbf{u}_c$.

1. For every point $t_i$ in $\mathbf{u}^*$, there is a corresponding point $t_j$ in $\Delta_{\mathrm{u}}(\delta, \varepsilon, \mathcal{U}_T)$ such that $|t_j - t_i| < \varepsilon^*$. (Note that the number of time-points in $\Delta_{\mathrm{u}}(\delta, \varepsilon, \mathcal{U}_T)$ may be much larger than $L$);
2. Let $u_j = (w_j^1, \ldots, w_j^i, \ldots, w_j^m, v_j)$. Then, $\forall j, i$ there is a $(\omega^1, \ldots, \omega^m, v)$ in $\Delta_{\mathrm{u}}(\delta, \mathcal{U})$ such that $|w_j^i - \omega^i| < \delta^*$, and $v_j = v$.

Next we show that at this level of discretization, the algorithm can asymptotically find an input sequence $\mathbf{u}'$ in $DN_{\mathcal{W},\delta^*,\varepsilon^*}(\mathbf{u}^*)$, which guarantees that $\varphi(\mathbf{u}', \mathbf{y}') < 0$. The following observations are used to complete the proof: Under Condition 1 in Def. 8, any $L$-length sequence of the form $(u_0, t_0), \ldots, (u_L, t_L)$ in $\mathbf{u}^*$, can be found embedded in a sequence over more time points. For any given $\Delta_{\mathrm{u}}(\delta, \varepsilon, \mathcal{U}_T)$, there is a finite number of input sequences. The Tabu list ensures progress as it disallows visiting the same sequence twice; thus, the probability that a certain neighbor in $DN_{\mathcal{W},\delta^*,\varepsilon^*}$ is *not* picked goes to zero as the user-defined parameters `maxLocalImprovements` $\to \infty$ and `maxRestarts` $\to \infty$.

## 5 Experimental Results

| Model | Requirement | SITAR $\Delta$ | SITAR $\lvert\mathcal{U}\rvert$,$\lvert\varepsilon\rvert$ | SITAR Falsified? | SITAR Time (sec) | SITAR Sim. | S-TaLiRo $\lvert T\rvert$ | S-TaLiRo Falsified? | S-TaLiRo Time (sec) | S-TaLiRo Sim. |
|---|---|---|---|---|---|---|---|---|---|---|
| AFC | (2); $\zeta$=0.024 | U | 4,3 | y | 23 | 18 | 7 | y | 13 | 5 |
| | (2); $\zeta$=0.028 | U | 4,3 | y | 3 | 2 | 7 | y | 19 | 7 |
| | (2); $\zeta$=0.032 | U | 4,3 | y | 102 | 71 | 7 | n | 79 | 32 |
| MRS | (3) | NU | 35,3 | y | 50 | 233 | 40 | n | 745 | 1000 |
| | (3) | U | 35,3 | y | 241 | 2058 | 40 | n | 2121 | 3000 |
| RD | (5) | NU | 3,2$^*$ | y | 17 | 206 | 2 | n | 141 | 2000 |
| | (5) | U | 3,3$^*$ | y | 47 | 575 | 4 | n | 141 | 2000 |
| | (5) | U | 3,4$^*$ | y | 28 | 575 | 8 | y | 1 | 17 |
| PTAC | (6); $\zeta = \zeta_1$ | U | 3,3 | y | 3996 | 18 | 6 | y | 2448 | 6 |
| | (6); $\zeta = \zeta_2$ | U | 3,3 | y | 8424 | 31 | 6 | y | 21348 | 51 |
| | (6); $\zeta = \zeta_3$ | U | 3,3 | y | 8784 | 39 | 6 | y | 26568 | 71 |

**Table 1.** Results of comparison between SITAR approach and S-TaLiRo tool. For model RD, we allow refinement of the control points of the input state space, we report the initial control points in the table and mark these with a $*$.

We implemented Algorithm 2 with support for refinement heuristics and both uniform and nonuniform discretization in a tool named SITAR (**S**tochast**I**c **T**abu-search **A**nd **R**efinement). We present the results of comparing the performance of our method with a state-of-the-art falsification tool S-TaLiRo on multiple academic and industrial system models. For each model, we give requirements in temporal logic and compare the performance of SITAR and S-TaLiRo.

We pick S-TaLiRo for comparison as our tool SITAR shares several features with S-TaLiRo: (1) both tools use a property function $\varphi$ to guide the search over the input space (S-TaLiRo uses robustness degree of a requirement specified using Metric Temporal

Logic), (2) both tools support a finite parameterization of the input sequence space, (3) both tools use heuristic global optimization relying on black-box simulations of a given Simulink®model. A key difference is that S-TaLiRo allows a fixed parameterization of the input signals, where users choose a number uniformly spaced time-points in the time domain (known as *control points*), and the optimizer is free to pick any input value in the range $[0, u_{max}]$ for each dense input dimension. In contrast SITAR also uses a grid over the dense input domain, and search is restricted to be over the grid elements. Further, SITAR supports automatic refinement of the input discretization and allows nonuniform control points in the time domain.

In the results reported in the sequel, we highlight the input sequence discretization method ($\Delta$), which is either uniform (U) or nonuniform (NU), size of the input sequence discretization ($|\mathcal{U}|$), number of control points in time domain ($|T|$), the result of whether faslficiation was successful, total computation time, and the total number of simulations (Sim.). For comparison between SITAR and S-TaLiRo, we use a fixed number of control points for three of the four cases and demonstrate the refinement capability of SITAR on one model. All requirements used are of the form $\Box_I(\mathbf{y} < \mathbf{c})$ (specifying that over the interval $I$, the output signal $\mathbf{y}$ remains less than $\mathbf{c}$). We use a property function similar to the robust satisfaction degree of STL [8].

### 5.1 Air-Fuel Control System (AFC)

Our first case study is an automotive air-fuel control (AFC) model [12]. The model is a representation of a closed-loop embedded control system, and contains several challenging features that are commonly found in industrial systems. It consists of a plant that describes physical phenomena such as fuel injection dynamics, exhaust gas transport dynamics, and sensor dynamics. While some aspects of the plant dynamics are derived using first principles, others aspects are captured with multi-dimensional lookup tables. One challenging aspect of the system dynamics from an analysis perspective is the presence of a *variable transport delay*; this effectively models dynamics containing a delay differential equation. The controller contains two parts: (1) an open-loop feedforward observer, and (2) a Proportional + Integral (PI) controller that regulates the air-fuel (A/F) ratio. For detailed description of this model, please refer to [12]. The controller has several modes, but for the purpose of this case study, we restrict ourselves to the normal mode of operation.

The paper [12] describes a safety requirement in the normal mode: in the time range $[T_{nom}, T_{hoz}]$ the normalized A/F ratio should remain within a given threshold $\zeta$.The requirement can be described in temporal logic as following:

$$\Box_{[T_{nom},T_{hoz}]}\mathbf{y} < \zeta. \tag{2}$$

We use $\min_{t_i \in [T_{nom},T_{hoz}]}(\zeta - \mathbf{y}(t_i))$ as the property function. The results shown in Table 1 show that both SITAR and S-TaLiRo can falsify the property for all three choices of the $\zeta$ parameter for the requirement. Two out of three requirements can be falsified by S-TaLiRo faster than SITAR. For the case when $\zeta = 0.028$, SITAR outperforms S-TaLiRo. The main lesson from this case study is that, although SITAR uses an essentially discrete optimization based technique, it can still search over a continuous state space relatively well.

### 5.2 Mode-specific Reference Selection Model (MRS)

In [9], the authors observe that current falsification tools became trapped at a local optimum due to the structure of the model, which contains complex discrete and temporal behaviors. The model selects an operating mode based on a region in the dense input state space. The mode is defined as a Boolean combination of conditions arising from $8$ input signals $w^1, \ldots, w^8$ being compared to a threshold. The output is some function of a ninth input $w^9$, with a different function for each operating mode. The range for inputs in $w^1, \ldots, w^8$, is $[0, 100]$ and the range for $w^9$ is $[-5, 5]$. The safety property requires the output $\mathbf{y}$ to remain above $-8$. The temporal logic requirement is defined as follows, and the property function can be derived as before:

$$\Box_{[\tau, T_{hoz}]}(y_i > -8). \tag{3}$$

In the above, we use $\tau = 5.1$ seconds, and $T = 10$ seconds. According to [9], in order to falsify the requirement, the system has to select the mode corresponding to the satisfaction of the following condition:

$$\bigwedge_{i \in [1..4]} \left( (w^{2i}(t) > 90) \wedge (w^{2i-1}(t) < 10) \right). \tag{4}$$

The probability of hitting the right combination of the $w^i$ values that falsify the property is $10^{-8}$ ($8$ inputs, and for each input there is a probability of $\frac{1}{10}$). Given the right combination, a tool such as S-TaLiRo can quickly falsify the requirement. However, it is not practical to expect that the designer to provide this insight to the tool, as a Boolean circuit of arbitrary complexity could be embedded in the given Simulink®model. In our opinion, the value of the falsification tool is to automatically explore the search space to identify the problematic input sequences with minimal user input.

As shown in Table 1, SITAR can falsify the requirement. When we discretize the input space in a nonuniform fashion, SITAR can falsify the requirement in less than one minute. Here, we choose the the partition for $w^i$ where $i \in [1, ..., 8]$ as $\{0, 10, 90, 100\}$ and $\{-5, 0, 5\}$ for $w^9$. One may claim that through this nonuniform discretization, we give extra information to the search algorithm; however, we argue that this information is related to the domain of the inputs and representative operating conditions, and such knowledge can be provided by design engineers with relative ease. Even with uniform gridding, SITAR is still able to falsify the requirement, with a moderate increase in computing time. For this example, S-TaLiRo (using the simulated annealing optimization heuristic) could not find a falsifying input sequence after $3000$ simulations.

### 5.3 Rate Detection (RD) System

Next, we describe the performance of the SITAR algorithm with refinement. We choose to use a simplified model originating from a rate detection system. The rate detection system checks if the rate of decrease of the signal is within a certain threshold $[\zeta_1, \zeta_2]$ in a given time window $[\tau_1, \tau_2]$, and if yes, causes the output $\mathbf{y}$ to be less than $-0.001$. For different values of $\zeta_1, \zeta_2, \tau_1, \tau_2$ the difficulty of finding the violating input varies.

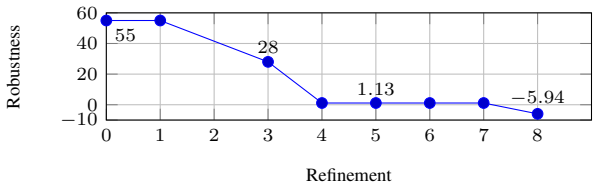


**Fig. 2.** The robustness value changes during the refinement process.

We picked the values $[\zeta_1, \zeta_2] = [2.2, 3.2]$ and $[\tau_1, \tau_2] = [5, 7]$, which present a reasonable level of difficulty for both SITAR and S-TaLiRo.

The safety requirement can be expressed as

$$\square_{[\tau, T_{hoz}]}(\mathbf{y} \geq -0.001). \tag{5}$$

Here, $\tau = 0.1$ is some initial time where the output behavior is ignored. For a falsifier like S-TaLiRo that uses a fixed parameterization of the input state space if the falsifier fails to find a falsifying input sequence using an initial input parameterization, the algorithm may be run again using an input parameterization corresponding to a finer resolution in the time domain. This approach is inefficient for this example, because it is not possible to find a falsifying input sequence unless discrete time instants exist near the required window of $[\tau_1, \tau_2]$, meaning computational effort is wasted on parameterizations of the time domain that are too coarse (and thus cannot possibly result in falsifying traces for this example). This explains why S-TaLiRo fails if the number of input parameters is less than 8, in Table 1. When using the appropriate number of input parameters, S-TaLiRo can trivially falsify the requirement; however, knowing the appropriate number of parameters requires significant user insight.

On the other hand, SITAR dynamically refines the discretization of the input space as needed. If combined with nonuniform input discretization, the SITAR algorithm can falsify the requirement in less than 20 seconds. In Fig. 2, we show how the robustness value decreases over the 8 refinement steps. Although SITAR could be trapped in a local optimum (in refinements 4 to 7), it eventually escapes from the local optimum and falsifies the requirement. Even with a uniform input discretization, as SITAR introduces time discretization, it can find a falsifying input sequence.

### 5.4 Powertrain Air Control (PTAC) System

To illustrate the scalability of our algorithm, we consider a powertrain air control subsystem (PTAC). This is a prototype model, developed during the early design phase of an advanced powertrain project. It has a complex, high fidelity plant model that is able to generate accurate simulation results, which correlates closely with data collected from the actual powertrain components. The controller contains logic for two of the system's electronic control units (ECU). Because of the complexity of the model, simulations are computationally expensive (simulating 1 second of real operation requires almost 5 seconds, so 5x slower). Although simulation is usually light-weight, in this case, it

is crucial for the falsification tool to use fewer simulations. The safety requirement for this system, obtained from design engineers, is expressed as follows in temporal logic:

$$\Box_{[\tau,T]}(\mathbf{y} < \zeta). \tag{6}$$

Due to proprietary reasons, we suppress the values of $\zeta$. As shown in the last few rows in Table 1, both S-TaLiRo and SITAR can falsify all requirements successfully. We use three values of threshold $\zeta_1$, $\zeta_2$, and $\zeta_3$, where $\zeta_1 < \zeta_2 < \zeta_3$. Note that falsifying the requirement becomes harder with increasing $\zeta$ value. For the last two values of $\zeta$, SITAR can falsifiy the requirements much faster, using less than half the time and a lower number of simulations. When $\zeta = \zeta_1$, S-TaLiRo can falsify the requirement faster, partly because SITAR performs thrice the number of simulations.

## 6 Conclusions and Future Work

**Conclusion:** Given a model of an embedded control system, and a (quantifiable) property over its input/output behaviors, we present a technique to find robust violations of the property. The key idea is to transform the problem of searching over an infinite set of timed input sequences to the system to a finite search over a discretized version of the input space using a Tabu list to avoid repeated computations. The search proceeds in the fashion of a stochastic gradient descent, with random restarts to perturb the system away from local optima. We wrap the search procedure in an outer loop that dynamically refines the input space discretization. This removes the burden of providing a clever discretization from the system designers, shifting it to the refinement heuristics employed by the tool. Our technique shows promise on industrial-sized benchmark problems, as well as toy problems that pose a challenge to existing falsification tools.

**Future Work:** In this paper, the technique to perform stochastic gradient descent involves computing finite differences of a point's cost with its neighbors' costs ($k+1$ valuations for a $k$-dimensional space). Each cost computation requires a simulation with the neighboring sequence as input. We can instead use the simultaneous perturbation stochastic approximation technique (SPSA), where the effect of a gradient descent is converged upon using a stochastic result [19]. In the SPSA approach, only 3 valuations are made, regardless of the dimension. Each computation of a cost function requires a simulation; thus, the SPSA scheme would help reduce the number of simulations.

In our current implementation, the Tabu list is stored as a simple list data structure in MATLAB®; an alternative is to use spatial data structures such as $k$-$d$ trees. Currently, the random restarts in our technique are chosen based on a uniform random distribution over the points in the discretization of the input space. An interesting direction to pursue is that of coverage metrics such as the star-discrepancy metric [9] to cover the input sequence space.

**Acknowledgments:** The authors would like to thank the anonymous reviewers for constructive feedback that helped improve this paper.

## References


1. H. Abbas and G. Fainekos. Linear hybrid system falsification through local search. In *Automated Technology for Verification and Analysis*, pages 503–510. 2011.
2. Y. S. R. Annapureddy and G. E. Fainekos. Ant Colonies for Temporal Logic Falsification of Hybrid Systems. In *Proc. of IECON*, pages 91–96, 2010.
3. Y. S. R. Annapureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257, 2011.
4. E. Asarin and O. Maler. Achilles and the tortoise climbing up the arithmetical hierarchy. *JCSS*, 57(3):389–398, 1998.
5. H. A. Bardh Hoxha and G. Fainekos. Using S-TaLiRo on industrial size automotive models. In *Worskhop on Applied Verification for Continuous and Hybrid Systems*, 2014.
6. J.-F. Cordeau, G. Laporte, A. Mercier, et al. A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows. *J. Oper. Res. Soc.*, 52(8):928–936, 2001.
7. A. Donzé. Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In *Proc. of Computer Aided Verification*, pages 167–170, 2010.
8. A. Donzé and O. Maler. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *Proc. of Formal Modeling and Analysis of Timed Systems*, pages 92–106, 2010.
9. T. Dreossi, T. Dang, A. Donzé, J. Kapinski, X. Jin, and J. V. Deshmukh. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *Proc. of NASA Formal Methods*, pages 127–142. 2015.
10. G. E. Fainekos and G. J. Pappas. Robustness of Temporal Logic Specifications for Continuous-Time Signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.
11. T. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What's Decidable about Hybrid Automata? In *Proc. of the Symposium on Theory of Computing*, pages 373–382, 1995.
12. X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts. Powertrain Control Verification Benchmark. In *Proc. of Hybrid Systems: Computation and Control*, pages 253–262, 2014.
13. S. Kirkpatrick, M. Vecchi, et al. Optimization by simmulated annealing. *Science*, 220(4598):671–680, 1983.
14. J. Kuřátko and S. Ratschan. Combined global and local search for the falsification of hybrid systems. In *Formal Modeling and Analysis of Timed Systems*, pages 146–160. 2014.
15. T. Nghiem, S. Sankaranarayanan, G. E. Fainekos, F. Ivancic, A. Gupta, and G. J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Proc. of Hybrid Systems: Computation and Control*, pages 211–220, 2010.
16. E. Plaku, L. E. Kavraki, and M. Y. Vardi. Hybrid systems: from verification to falsification by combining motion planning and discrete search. *Formal Methods in System Design*, 34(2):157–182, 2009.
17. E. Plaku, L. E. Kavraki, and M. Y. Vardi. Falsification of ltl safety properties in hybrid systems. *Software Tools for Technology Transfer*, 15(4):305–320, 2013.
18. S. Sankaranarayanan and G. E. Fainekos. Falsification of Temporal Properties of Hybrid Systems using the Cross-Entropy Method. In *Proc. of Hybrid Systems: Computation and Control*, 2012.
19. J. C. Spall. *Introduction to Stochastic Search and Optimization*. John Wiley & Sons, Inc., New York, NY, USA, $1^{st}$ edition, 2003.
20. A. Zutshi, S. Sankaranarayanan, J. V. Deshmukh, and J. Kapinski. Multiple shooting, cegar-based falsification for hybrid systems. In *Proceedings of the 14th International Conference on Embedded Software*, page 5, 2014.