Sampling-Based and Gradient-Based Efficient Scenario Generation

Vidisha Kudalkar[1(✉)], Navid Hashemi[1], Shilpa Mukhopadhyay[2], Swapnil Mallick[1], Christof Budnik[3], Parinitha Nagaraja[3], and Jyotirmoy V. Deshmukh[1]

[1] University of Southern California, Los Angeles, USA
{kudalkar,navidhas,smallick,jdeshmuk}@usc.edu
[2] Texas A&M University, College Station, USA
shilpa2301@tamu.edu
[3] Siemens Corporation, Princeton, USA
{christof.budnik,parinitha.nagaraja}@siemens.com

Abstract. Safety-critical autonomous systems often operate in highly uncertain environments. These environments consist of many agents, some of which are being designed, and some represent the uncertain aspects of the environment. Testing autonomous systems requires generating diverse scenarios. However, the space of scenarios is very large, and many scenarios do not represent edge cases of the system. We want to develop a framework for automatically generating interesting scenarios. We propose to describe scenarios using a formal language. We show how we can extract interesting scenarios using scenario specifications from sampling-based approaches for scenario generation. We also introduce another technique for edge-case scenario generation using the gradient computation over STL. We demonstrate the capability of our framework in scenario generation in two case studies of autonomous systems involving the autonomous driving domain and the safety of human-robot systems in an industrial manufacturing context.

Keywords: Scenario Generation · Neurosymbolic Testing · Autonomous Systems · Temporal Logic

1 Introduction

Swift advancements in artificial intelligence have led to rapid growth of autonomous systems including self-driving vehicles, smart manufacturing systems, and unmanned aerial vehicles. These systems operate in open-world and real-time settings with pervasive uncertainty in their environment. For example, consider self-driving vehicles operating in urban environments; here, uncertainty includes extrinsic sources such as pedestrians, bicyclists, other cars, traffic lights, weather, and road conditions, and intrinsic sources such as uncertainties in correct object detection, trajectory prediction, localization, and scene understanding. As another example, consider a smart manufacturing system such as one

E. Ábrahám and H. Abbas (Eds.): RV 2024, LNCS 15191, pp. 70–88, 2025.
https://doi.org/10.1007/978-3-031-74234-7_5

described in [2], where humans and mobile robots cohabit on a factory floor with fixed manufacturing robotic manipulators. These systems often employ smart runtime monitoring to ensure safety by using sensors for real-time localization of various assets. The factory environment, though more well-structured than the open-world setting of cars, still has several sources of uncertainty; extrinsic sources such as human workers, lighting conditions, and electromagnetic interference, and intrinsic uncertainty related to the sensing errors in localization. Similar sources of uncertainty exist in other autonomous systems, such as unmanned aerial vehicles, smart warehouses, etc.

To assess the safety of such autonomous systems, a common mechanism is to construct a digital twin at an appropriate level of fidelity, and employ simulations to gauge the system behavior in many nominal and edge case *scenarios* [9,36]. Recently, modeling languages for specifying scenarios and tools for automatically generating scenarios have become a popular approach in testing autonomous cyber-physical systems [1,14,15,39]. The Scenic modeling language is one such formalism that has gained popularity. In Scenic, a scenario is defined as a collection of physical objects, with a probability distribution on their positions and other properties. A Scenic program is thus a probabilistic program that can generate many different concrete scenes by sampling from the distribution implied by the program. The overall approach of scenario generation in Scenic is based on rejection sampling. Scenic programs allow the user to specify *requirements* that constrain the set of scenes that will be generated (including temporal logic requirements specified in LTL). There are two main limitations to the current scenario generation approach in Scenic.

First, consider the situation where the high-level probabilistic program describes a very large collection of generic behaviors; for example, "up to ten workers walking through the smart factory," or "three cars traveling in three parallel lanes on a freeway." Scenic programs describing these scenarios would offer many degrees of freedom to each dynamic agent in the scene, leading to an astronomically large number of scenes in the implicit distribution. Furthermore, average scenarios (in a statistical sense) may only correspond to nominal system behaviors. For instance, "ten workers walking through the smart factory" may generate many scenarios where the workers walk in straight lines from their start to goal positions. In Scenic, requirements are used to constrain the generation of test scenarios to only interesting scenarios; for example, "at some time, two workers are within some δ distance of each other." However, the set of scenarios satisfying the requirement may have a very small measure in the overall distribution, making a rejection sampling-based approach inefficient as it is not requirement-driven.

The second challenge is that for broad scene descriptions, we may have several different sub-scenarios with differing requirements. For example, consider the Scenic program specifying "up to ten human workers walking through the factory floor", with the requirement "at least two workers enter the hazard region at the same time, and stay there for at least five seconds," and a different requirement: "at least three workers enter different hazard regions at the same time." If we adopt a requirements-driven sampling approach where we use

optimization-guided tools for scenario sampling, and then with every new requirement, the scenario generation tool will have to repeat the process of rejection sampling to identify scenarios.

To address these challenges, we investigate two different strategies for scenario generation. The first strategy seeks to solve the second challenge problem. We propose the use of incremental sampling-based path planning methods to generate a very large set of scenarios consistent with a given scenario description. These could include methods based on the rapidly exploring random trees (RRT) algorithm [29] and its variants [23], or probabilistic roadmaps (PRMs) [24] and its variants. We assume that the given scenario requirement is in discrete-time Signal Temporal Logic (STL). We show that extracting scenarios satisfying a given discrete-time STL requirement can be done by an adaptation of the CTL model checking algorithm [11].

The second strategy seeks to solve the first challenge problem of requirement-driven scenario generation using a *neuro-symbolic* method. We use the observation from [20] that the quantitative semantics (robustness) of a discrete-time STL requirement can be underapproximated by a feedforward neural network with smooth activation functions (called LB4TL). With this encoding, if we have a black-box simulator, then we can use derivative-free nonlinear optimization methods for scenario generation using the LB4TL approximation of robustness to efficiently generate scenarios. If the simulation environment itself is differentiable (for example, Waymax [18]), then scenario generation can be efficiently solved by stochastic gradient descent.

To summarize, our main contributions to this paper are:

- We propose a coverage sampling-based approach to generate a scenario database (represented either as a tree or a forest) and extract interesting scenarios using a model checker for discrete-time STL. Our method allows extracting scenarios corresponding to multiple requirements after a single upfront generation of the scenario database.
- We propose a property-guided method that utilizes the gradient of discrete-time STL robustness to propose optimization-based and gradient-based techniques to efficiently generate scenarios corresponding to requirements that may be of tiny measure in the overall space of the implied scenarios.
- We demonstrate the scenario generation techniques using two case studies: smart manufacturing and autonomous driving.
- We present a case study where we interface our scenario generation tool with a physics-based Unity simulator to test the generated scenarios in a 3D environment.

2 Preliminaries

An environment is an abstract representation of the physical world containing mobile agents, sensors, and obstacles. We formalize an environment as follows:

Definition 1 (Environment). *An environment $\mathcal{E}$ is a tuple $\mathcal{L}, \mathcal{A}, \mathcal{Z}$, where: $\mathcal{L}$ is a finite or infinite set of locations in the environment, $\mathcal{A}$ is a set of dynamical, autonomous* agents*, and $\mathcal{Z}$ is a set of sensors that observe $\mathcal{E}$ and $\mathcal{A}$.*

For example, in a warehouse environment, we have multiple humans (agents) working in the factory around robot manipulators. The area around the robots is considered as hazard regions and these regions are not safe for humans to enter while the robots are operating. The environment can have a finite set of agents $\mathcal{A} = \{A_1, \ldots, A_n\}$. We define the agents as follows:

Definition 2 (Autonomous Agent). *An agent $\mathcal{A}_i$ is defined as a tuple $(S^i, U^i, \Delta^i, \pi^i, I^i)$ where S^i is a set of* states *of agent $\mathcal{A}_i$. At any time t, the agent is in state $s_t^i \in S^i$. U^i is the set of actions the agent can take, and we denote the action it takes at time t by u_t^i. Δ^i is a function mapping a state in S^i and an action in U^i to a next state (in S^i), π^i is a stochastic policy of the agents that represents a distribution over the set U^i conditioned by the current state, and I^i is a set of initial states s.t. $I^i \subset S^i$.*

Remark 1. We note that an agent can have several independent state variables, and the state space of the agent can be thought of as the Cartesian product of the domains of each of the state variables. We assume that the following state variables are always present: ℓ: the location of the agent in the environment, i.e., the value of this variable is in the set $\mathcal{L}$, and Θ: this variable defines the orientation of the agent in the environment; for 2-D environments, $\Theta = \{\theta\}$ or the angle between the agent's heading and the X-axis, and for 3-D environments, $\Theta = \{\phi, \theta, \psi\}$, the angle between the agent's heading with X, Y, Z axis respectively.

The environment can be equipped with sensors like cameras, RFID, lasers, etc. and we can observe the environment using these sensors. At every time t, the environment is updated as the agents transition into new states. We describe the environment at time t using the configuration. The configuration of the environment depends on the states of all agents $\mathcal{A}$.

Definition 3 (Configuration of an environment). *Formally, the configuration c_t at time t is defined as a vector $c_t = (s_t^1,\ s_t^2,\ \ldots,\ s_t^n)$.*

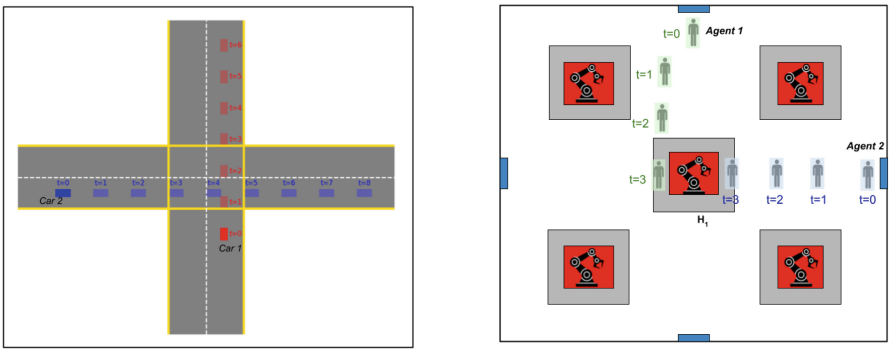
The environment is initialized using an initial configuration c_0 where each agent is in some initial state I^i. At every timestep t, the environment updates based on the actions of the agents $\mathcal{A}$. We denote this update as:

$$(s_t^1,\ s_t^2,\ \ldots,\ s_t^n) \xrightarrow{(u_t^1,\ u_t^2,\ \ldots,\ u_t^n)} (s_{t+1}^1,\ s_{t+1}^2,\ \ldots,\ s_{t+1}^n) \text{ s.t. } u_t^i \sim \pi^i(u \mid s = s_t^i)$$

We denote the states of all agents $(s_t^1,\ s_t^2,\ \ldots,\ s_t^n)$ by $\mathbf{s}_t$ and actions of all agents $(u_t^1,\ u_t^2,\ \ldots,\ u_t^n)$ by $\mathbf{u}_t$ at time t. Also, let $\mathcal{U} = U^1 \times \ldots \times U^n$.

Definition 4 (Scenario). *A scenario is defined as the finite sequence of configurations of length $L \in \mathbb{N}$ generated by the moves of the environment as $< c_0, c_1, c_2, \ldots, c_{L-1} >$. Formally, we can define a scenario of length $L \in \mathbb{N}$ as $\mathbf{s}_0 \xrightarrow{\mathbf{u}_0} \mathbf{s}_1 \xrightarrow{\mathbf{u}_1} \mathbf{s}_2 \xrightarrow{\mathbf{u}_2} \ldots \mathbf{s}_{L-1}$.*

We need a formal language to represent such scenario requirements. In this paper, we use Discrete-Time Signal Temporal Logic (DT-STL) to constrain the set of dynamic scenarios.



(a) car_1 *and* car_2 *cross the intersection*

(b) $human_1$ *and* $human_2$ *enter the hazard region simultaneously*

Fig. 1. Examples Scenarios: (a) Shows a scenario in the autonomous driving domain, and (b) shows a scenario in the industrial factory floor domain. (Color figure online)

Discrete-Time Signal Temporal Logic (STL). A DT-STL formula consists of atomic predicates connected using Boolean and temporal operators. An atomic predicate μ is written as $\mu = f(\mathbf{s}) \sim 0$, where f is a function over the states $\mathbf{s} = (s^1, s^2, \ldots, s^n)$, $\sim$ is a comparison operator such that $\sim \in \{<, \leq, >, \geq, =, \neq\}$. The temporal operators are always ($\mathbf{G}$), eventually ($\mathbf{F}$), until ($\mathbf{U}$) and release ($\mathbf{R}$) . Each temporal operator is associated with a bounded integer time interval $\mathcal{I} = [a, b]$ where $a < b$ and $a, b \in \mathbb{Z}^{\geq 0}$. DT-STL syntax is formally defined as:

$$\varphi := \top \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \, \mathbf{U}_{\mathcal{I}} \, \varphi_2 \mid \varphi_1 \mathbf{R}_{\mathcal{I}} \varphi_2$$

The remaining operators can be defined using these operators. The always operator is defined as $\mathbf{G}_{\mathcal{I}} \, \varphi \equiv \neg \, \mathbf{F}_{\mathcal{I}} \, \neg\varphi$ and eventually operator as $\mathbf{F}_{\mathcal{I}} \, \varphi \equiv \top \, \mathbf{U}_{\mathcal{I}} \, \varphi$.

Boolean Semantics of STL. Given a signal x and a time t, the Boolean semantics of any STL formula is inductively given by:

$$\begin{aligned}
(x,t) &\models \mu && \iff x \text{ satisfies } \mu \text{ at time } t \\
(x,t) &\models \neg\varphi && \iff (x,t) \not\models \varphi \\
(x,t) &\models \varphi_1 \wedge \varphi_2 && \iff (x,t) \models \varphi_1 \text{ and } (x,t) \models \varphi_2 \\
(x,t) &\models \varphi_1 \, U_{[a,b]} \, \varphi_2 && \iff \exists t' \in [t+a, t+b] \text{ s.t. } (x,t') \models \varphi_2 \\
& && \quad \text{and } \forall t'' \in [t, t'], (x,t'') \models \varphi_1
\end{aligned}$$

An example of the scenario in the autonomous driving domain is "*Eventually* car_1 *and* car_2 *cross the intersection*", and we write this requirement using STL as ψ_1 (Eq. (1)). Figure 1a shows this scenario where car_1 is shown using red boxes and car_2 is shown using blue boxes, and both cars cross the intersection.

$$\psi_1 : \mathbf{F}_{[0,T]} \left[(car_1 \in intersection) \right] \bigwedge \mathbf{F}_{[0,T]} \left[(car_2 \in intersection) \right] \qquad (1)$$

Another example in the industrial warehouse domain is $agent_1$ *and* $agent_2$ *eventually enter hazard region* H_1 *simultaneously*, and this is given by ψ_2 (Eq. (2)).

Figure 1b shows a scenario where $agent_1$ and $agent_2$ are highlighted in green and blue, resp., and both agents enter the hazard region H_1 at time $t = 3$.

$$\psi_2 : \mathbf{F}_{[0,T]} \left[(agent_1 \in H_1) \wedge (agent_2 \in H_1) \right] \tag{2}$$

Randomly Sampled Scenario: We can sample the scenarios randomly using Algorithm 1. First, we sample the initial configuration of the agents from the set of initial states (line 1). For each agent $\mathcal{A}_i$ where $i = 1 \ldots n$, we sample the states from time $t = 1 \ldots L-1$ using the agent's policy and transition into a new state (lines 2–4).

Algorithm 1: Randomly sampled scenario

1 Sample initial configuration $(s_0^1,\ s_0^2,\ \ldots,\ s_0^n) \in \mathcal{I}^1 \times \mathcal{I}^2, \ldots, \mathcal{I}^n$;
2 **for** $i \leftarrow 1$ **to** n **do**
3 **for** $t \leftarrow 0$ **to** $L-2$ **do**
4 $u_t^i \sim \pi^i(u \mid s = s_t^i)$, $s_{t+1}^i \leftarrow \Delta^i(s_t^i, u_t^i)$;
5 **end**
6 **end**

2.1 Problem Definition

In this paper, we propose algorithms to generate a set of scenarios that capture the desired behavior of the agents in the system. In this paper, our goal is to find the set of scenarios where each scenario can be described using the trajectories $\pi = (\pi_0, \pi_1, \ldots, \pi_n)$ for the agents $A_0, A_1, \ldots, A_n$ that satisfy the requirement φ. To find the scenarios, we focus on two problems:

1. Given a dataset $\mathcal{D}$ of randomly sampled scenarios and a DT-STL scenario requirement φ, we aim to find the set of scenarios $\models \varphi$.
2. Given a DT-STL scenario requirement φ, we find the actions $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{L-2}$ for all the agents $\mathcal{A}$ in the environment such that if we apply these actions we get a sequence of states, $\mathbf{s}_0 \xrightarrow{\mathbf{u}_0} \mathbf{s}_1 \xrightarrow{\mathbf{u}_1} \mathbf{s}_2 \xrightarrow{\mathbf{u}_2} \ldots \mathbf{s}_{L-1}$ and these sequence of states form a scenario s.t. $(\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_{L-1}) \models \varphi$.

For the first problem, we use a sampling-based path-planning approach to obtain the variety of agent behaviors in the environment and extract desired behaviors by specifying a scenario specification φ using DT-STL as detailed in Sect. 3. We solve the second problem by learning a neural network control policy π_0 or the parameters θ such that, for any initial state $s_0 \in S_{init}$, the trajectory obtained using the control policy π_0 satisfies a given DTL-STL specification φ as outlined in Sect. 4.

3 Scenario Generation Using Path Planning and Model Checking

Incremental sampling-based path-planning algorithms are commonly used in the field of robotics. The Rapidly-exploring Random Trees algorithm (RRT) developed in [29], is widely used due to its speed to rapidly identify paths, even in

Algorithm 2: Coverage-based RRT algorithm

Input: Initial configurations I^i, Maximum iterations $steps_{max}$

1 $T_i = (V_i = \phi, E_i = \phi)$;
2 Goal points $G = \phi$;
3 Divide $\mathcal{E}$ into $p \times q$ blocks B_{ij};
4 **for** *each block* B_{ij} **do**
5 Sample a set of goal points $G_{ij} \subset B_{ij}$;
6 $G = G \cup G_{ij}$;
7 $step \leftarrow 0$;
8 $s_0^i \leftarrow S_{init}^i.sample()$;
9 $V_i \leftarrow \{s_0^i\}$;
10 **while** $step < steps_{max}$ *and* !(*allGoalsReached*) **do**
11 $x_{sample} \leftarrow S^i$.sample();
12 $x_{near} \leftarrow$ NearestNeighbour(x_{sample});
13 $u \leftarrow$ findaction(x_{near});
14 $x_{new} \leftarrow f(x_{near}, u)$;
15 **if** *collisionfree*(x_{near}, x_{new}) **then**
16 $V_i.add(x_{new})$;
17 $E_i.add(x_{near}, x_{new})$;
18 $step \leftarrow step + 1$;

high-dimensional environments. The RRT algorithm incrementally constructs a tree starting at a randomly sampled initial location (within the set of permitted initial locations). Nodes of the tree are agent locations, and edges indicate the action that moves the agent from the parent node to the child node. RRT has three main steps: (1) randomly sample a goal point in the state-space, (2) find the nearest node n on the tree to the goal point, (3) use a local optimizer to find an action to move the agent from n to a node that is the closest to the goal point within one time unit. These three steps are repeatedly invoked till some computation budget is reached. The vanilla RRT algorithm focuses on rapidly finding a feasible path to a target region, but may not provide sufficient coverage over the state space. We propose a variation of RRT to improve coverage over the state space.

Multi-agent RRTs for Building a Scenario Database. The Coverage-based RRT algorithm is outlined in Algorithm 2. We split the environment $\mathcal{E}$ into $p \times q$ blocks to systematically cover the working area of the agents. We define a block B_{ij} as the block located at row i and column j of $\mathcal{E}$ such that $0 \leq i \leq p-1$ and $0 \leq j \leq q-1$ and $B_{ij} \subset \mathcal{L}$. To ensure the coverage of the environment by the agents, we sample goal points from each block of the environment (lines 4–6). We defined the set of goal points $G = \cup_{i=0}^{p-1} \cup_{j=0}^{q-1} G_{ij}$ where G_{ij} is the set of goal points in B_{ij}. We incrementally construct a Coverage-based RRT $T_i = (V_i, E_i)$ for each agent $\mathcal{A}_i$ in the environment. We take a set of initial configurations I^i, and the maximum number of iterations $steps_{max}$ as input to the algorithm. The initial state s_0^i of an agent A_i is sampled from a set of initial configurations

(a) Yellow nodes shows RRT for $Agent_1$ and purple nodes shows an RRT for $Agent_2$

(b) Multi-agent RRT is generated by combining nodes at each level

Fig. 2. Combining single agent RRTs into a multi-agent RRT

$I^i \in S^i$ (line 8). We initialize $V_i = \{s_0^i\}$ and $E_i = \phi$ (line 9). This initial state s_0^i is assigned as the root node of the RRT of the agent A_i. At each time t, a random configuration x_{sample} is sampled from the state space S^i (line 11). Next, the node $x_{near} \in V_i$ is searched in the tree T_i that is closest to x_{sample} using Euclidean distance (line 12). An input $u \in \mathcal{U}$ is determined to move the system from x_{near} to x_{sample} (line 13). For our agents, velocity v is one of the inputs that can be selected from $[v_{min}, v_{max}]$ and the steering angle $\theta \in [0, 2\pi]$. The input u is applied to identify the new node x_{new} using the dynamics of the system $f(x, u)$ (line 14). The dynamics of the system can be of the form $x_{t+1} = x_t + v_t \cos(\theta_t)\delta t$ and $y_{t+1} = y_t + v_t \sin(\theta_t)\delta t$. The path between x_{near} and x_{new} is checked for collisions before adding it to the tree T_i (line 15). If the path is collision-free, the x_{new} is added to the vertex set V_i, and the edge (x_{near}, x_{new}) is added to the edge set E_i (lines 16–17). The tree is iteratively built until there are $steps_{max}$ nodes in T_i or all the goals have been reached. At each node, we store the state s, inputs u, and time t to reach x_{new} from the root. Intuitively, the timestamp t at each node is the depth of node in the tree.

Combine RRTs to Generate a Multi-agent RRT. We combine the generated RRT for each agent into a multi-agent RRT. We consider the RRT of agent $\mathcal{A}_i$ as $\mathcal{T}_i = (V_i, E_i)$ where V_i is the set of nodes and E_i is the set of edges in tree $\mathcal{T}_i$. Each node in $\mathcal{T}_i$ is associated with the state s^i, action u^i, and a timestamp t. We combine the agent trees $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ to generate a multi-agent RRT $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \times_i V_i$,

$$\left((s_t^1, \ldots, s_t^n) \xrightarrow{u_t^1, \ldots, u_t^n} (s_{t+1}^1, \ldots, s_{t+1}^n)\right) \in \mathcal{E} \text{ if } \forall j \; \exists (s_t^j \xrightarrow{u_t^j} s_{t+1}^j) \in E^j,$$

Figure 2 shows the process of generating a multi-agent RRT. This approach ensures the integration of all nodes across all trees $\mathcal{T}_i$ and facilitates a temporal synchronized representation of all agents $\mathcal{A}$ in the environment.

Extracting Scenarios Using Model Checking. To extract scenarios satisfying a given DT-STL property φ, we use a standard backward model checking procedure [4,11]. Bounded-horizon DT-STL can be viewed as essentially a bounded-horizon LTL formula. To determine if there exist paths in the multi-agent RRT $\mathcal{T}$ satisfying such a formula is equivalent to model checking $\mathcal{T}$ with

φ	$\rho(\varphi, k)$	φ	$\rho(\varphi, k)$
$h(s_k) \geq 0$	$h(s_k)$	$\mathbf{F}_{[a,b]}\psi$	$\max_{k' \in [k+a, k+b]} \rho(\psi, k')$
$\varphi_1 \wedge \varphi_2$	$\min(\rho(\varphi_1, k), \rho(\varphi_2, k))$	$\varphi_1 \mathbf{U}_{[a,b]} \varphi_2$	$\max_{k' \in [k+a, k+b]} \left(\min \left(\rho(\varphi_2, k'), \min_{k'' \in [k, k')} \rho(\varphi_1, k'') \right) \right)$
$\varphi_1 \vee \varphi_2$	$\max(\rho(\varphi_1, k), \rho(\varphi_2, k))$	$\varphi_1 \mathbf{R}_{[a,b]} \varphi_2$	$\min_{k' \in [k+a, k+b]} \left(\max \left(\rho(\varphi_2, k'), \max_{k'' \in [k, k')} \rho(\varphi_1, k'') \right) \right)$
$\mathbf{G}_{[a,b]}\psi$	$\min_{k' \in [k+a, k+b]} \rho(\psi, k')$		

Fig. 3. Quantitative Semantics (Robustness value) of STL

respect to an equivalent CTL* formula $\mathbf{E}\varphi$. Recall that a standard backward model checking algorithm returns the set of states satisfying the given formula; if this set does not contain the initial states, then there is no scenario satisfying the requirement φ, and we require the RRT algorithm to be run for several more iterations. Otherwise, we can extract the set of paths in $\mathcal{T}$ that satisfy φ using a standard witness extraction algorithm for CTL* [4].

4 Neurosymbolic Scenario Generation

In what follows, we describe a neuro-symbolic procedure for scenario generation. The basic idea is to represent the scenario requirement φ as a cost function to be maximized; such a cost-based encoding is easily accomplished by the standard quantitative semantics of DT-STL. We note that we do not need to actually maximize the robust satisfaction of DT-STL, but *satisficing* scenarios are enough (i.e. scenarios that satisfy φ). However, we can still use an optimization-guided procedure and the optimizer's exploration process to identify such satisficing scenarios.

Quantitative Semantics of STL (or Robustness Value): The quantitative semantics of STL define a signed distance of a given trajectory from the set of trajectories satisfying or violating the given STL formula. There are many alternative semantics proposed in the literature [3,8,12,35]. We use the semantics from [8] which computes an approximation of the quantitative semantics. The main idea is to define a function $\rho(\varphi, s, k)$ that returns a numerical satisfaction degree of the formula. This is obtained by replacing the Boolean satisfaction of a predicate of the form $f(s) > 0$ with the numeric value $f(s)$, all disjunctions in the Boolean semantics of an STL formula with max, and conjunctions with min. We note that if $\rho(\varphi, s, k) > 0$ the STL formula φ is satisfied at time k, and we say that the formula φ is satisfied by a trajectory if $\rho(\varphi, s, 0) > 0$. The quantitative semantics of DT-STL is essentially the same, but with the caveat that time indices can only be integers.

The main idea that we pursue in this scenario generation technique is to represent the STL constraint as a smooth function of the scenario generation parameters, and utilize optimization techniques to find scenarios satisfying the

constraint. There have been a number of methods for approximating STL semantics with smooth functions and neural networks [16,20,30]. There are two key differences in our approach: (1) we utilize the representation of a neural network-like computation graph representation of STL that guarantees that the value that it computes is a guaranteed lower bound for the actual STL robustness function first proposed in [20], (2) we leverage the most modern simulation environments are differentiable, and use stochastic gradient descent to find satisfying scenarios.

Given an STL formula φ, in [20], the authors represent the robustness computation in Eq. (3) as a tree, which can in turn be viewed as a neural network like computation graph. Special care is taken during the construction of this graph to reduce its depth through the use of associativity of min and max. Thus, $\max(a, b, c, d)$ is represented as $\max(\max(a, b), \max(c, d))$ rather than as $\max(a, \max(b, \max(c, d)))$. Furthermore, all instances of $\max(a, b)$ are replaced with $b + \mathsf{swish}(a - b)$, and $\min(a, b)$ is replaced with $a - \mathsf{softplus}(a - b)$.[1]

Given the set of agents $(A_1, \ldots, A_n)$ in the simulation environment, let $\mathbf{u}(\cdot) = \mathbf{u}_0, \ldots, \mathbf{u}_{L-1}$ be the sequence of joint actions executed by the agents. Recall that a joint action $\mathbf{u}_t = (u_t^1, \ldots, u_t^n)$. Let the resulting scenario be denoted $\mathbf{s}^{\mathbf{u}}$. We wish to sample scenarios from different action sequences and find a set of scenarios $\mathcal{S} = \{\mathbf{s} \mid \rho(\mathbf{s}, \varphi, 0) > 0\}$, i.e. a set of scenarios each satisfying φ. Ideally, we would like $|\mathcal{S}|$ to be at least the allocated test budget, i.e., a minimum number of scenarios. Furthermore, we would like the set $\mathcal{S}$ to be ϵ-separated[2] under a suitable distance metric[3] between scenarios. Formally ensuring such ϵ-separation is computationally expensive, but in the procedures described below, we provide mechanisms to ensure sufficient diversity in scenarios.

Scenario Generation with Black-box Optimization: We can perform scenario generation by considering the following optimization problem:

$$\mathbf{u}^* = \arg\max_{\mathbf{u} \in \mathcal{U}} \mathsf{LB4TL}_{\varphi}(\mathbf{s}^{\mathbf{u}}) \tag{3}$$

For a general φ, $\mathsf{LB4TL}_{\varphi}$ is a non-convex function. Thus, we can use general blackbox optimization techniques such as CMAES [19], Particle Swarm Optimization [32], Nelder Mead [28] etc. to address problem (3). These optimizers typically have several random decision parameters, and iteratively approach the optimum. We record scenarios with positive robustness in a given optimizer run, and restart the optimizer with a different random seed to obtain a different scenario. While this does not guarantee that the scenarios identified will be sufficiently diverse, we empirically observe that it does give us diverse scenarios.

[1] The function $\mathsf{swish}(x; \gamma_1) = \frac{x}{1+e^{-\gamma_1 x}}$, and the function $\mathsf{softplus}(x; \gamma_2) = \frac{1}{\gamma_2}\log(1 + e^{\gamma_2 x})$. Here, $\gamma_1, \gamma_2 > 0$ are user-defined parameters trading off smoothness and the accuracy of approximation.

[2] Let Y be a metric space with the metric d. Then, $X \subseteq Y$ is called ϵ-separated if for all $x_1, x_2 \in X$, $d(x_1, x_2) > \epsilon$.

[3] A scenario can be viewed as a vector of length T, and any vector norm can be thus be used to define distance between scenarios.

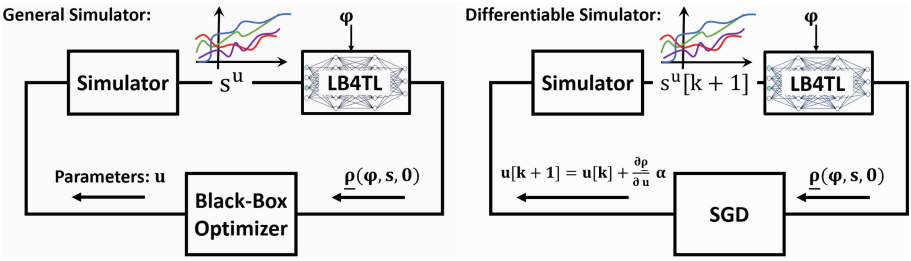


Fig. 4. Shows the computation graph utilized for neurosymbolic training process for satisficing parameters. In this paper, the learnable parameters $\mathbf{u}$, are open-loop controllers $\mathbf{u} = \{a_0, a_1, \ldots, a_{T-1}\}$ and the symbolic part is $\mathsf{LB4TL}_\varphi$ that generates a guaranteed lower-bound for the robustness. The neurosymbolic training process aims to increase the output of the symbolic part to make it positive. The left computation graph is utilized if we don't have access to the simulator and the right figure is for when we have access to a differentiable simulator where k is the k^{th} gradient step and α is the learning rate.

Scenario Generation for Differentiable Environments: Recently, there is a trend to design differentiable simulation environments. This means that the output of the simulator $\mathbf{s^u}$ is a differentiable function of the actions/parameters of the simulator $\mathbf{u}$. Examples of such environments include: Waymax [18], Brax [13], Tiny Differentiable Simulator [21], JAX [17]. With such environments, to generate the edge-case scenarios, we can utilize *stochastic gradient descent*. Let the initial guess for the sequence of actions (or the open-loop control policy) be denoted $\mathbf{u}[0]$. We then use the following updates to the input actions:

$$\mathbf{u}[k+1] := \mathbf{u}[k] + \alpha \frac{\partial \rho(\varphi, \mathbf{s^u}, 0)}{\partial \mathbf{u}} \approx \alpha \frac{\partial \mathsf{LB4TL}_\varphi(\mathbf{s^u})}{\partial \mathbf{s^u}} \cdot \frac{\partial \mathbf{s^u}}{\partial \mathbf{u}} \tag{4}$$

Here, $\mathbf{u}[k]$ denotes the sequence of actions taken in the k^{th} iteration, and α is a learning rate. For generating a diverse set of scenarios, we start with a different initial guess for the initial action sequence, which gives us a different scenario.

5 Case Studies

We evaluate our methods using two case studies: an industrial warehouse environment and an autonomous driving environment. In each case study, we demonstrate both of our techniques by generating the scenarios based on the requirements. The presented case studies would be challenging to existing test case generation techniques for several reasons: In the smart factory floor case study, the number of agents depends on the number of people allowed on the factory floor. This represents a unique challenge to test case generation tools that typically assume a fixed parameter space to explore. We argue that our RRT-based method allows composing scenarios of individual agents, thereby improving the scalability of scenario generation. In the autonomous driving case study, we

require the construction of challenging test scenarios that are constrained to satisfy temporal logic specifications. While Scenic has some capabilities to accomplish such a setting, we show that the rejection sampling-based approach that Scenic utilizes underperforms our proposed approaches.

5.1 An Industrial Factory Floor Environment

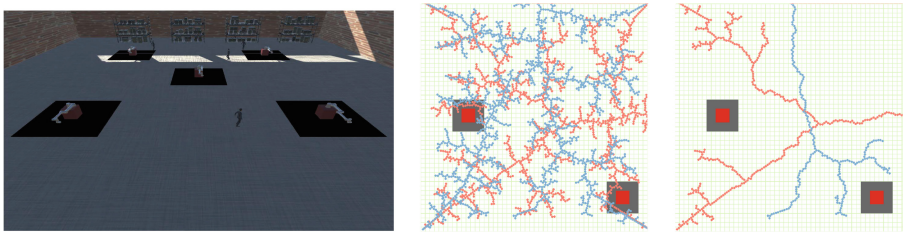
In many industrial factory floors, mobile robots and fixed robotic manipulators are employed to perform various tasks like delivering packages to stations, assembling manufacturing parts, etc. Such robots are mostly operated in fenced areas because of the safety hazards they could cause to the humans around the factory. If the humans navigate close to the robots, the robots are turned off hampering the productivity. Monitoring the human activities near the working region of the robots could help avoid accidents and ensure the safety of humans working in factories. We derived our case study based on this use case.

In this case study, we have considered a $M \times M$ warehouse with n agents that maneuver in a squared area in the presence of k hazardous regions. Figure 1b shows a top-down view of the factory floor. The red regions represent the robots, and the gray regions surrounding the robots are the hazard regions. The agents can only enter through the doors of the warehouse (annotated in blue). Figure 5a shows our 3D warehouse simulation environment designed using Unity, which is used to test the generated scenarios. The agents in our environment follow the dynamics of the Dubins cars i.e. $x^i_{t+1} = x^i_t + v^i_t \cos(\theta^i_t)\delta t$ and $y^i_{t+1} = y^i_t + v^i_t \sin(\theta^i_t)\delta t$ for $i \in \{1, 2, \ldots, n\}$.

Example 1: RRT-MC Scenario Generation. In the first example, we have two agents in the warehouse environment with two manipulator robots, denoted in red regions in Fig. 5b and 5c. The robots are surrounded by hazard regions (marked with gray color). There are four doors to the warehouse, and the agents can enter only through these entrances into the warehouse. We want scenarios where all the agents walk only in safe regions and we represent this using $\varphi_1 = \bigwedge_{j=1}^{2} \mathbf{G}_{[0,T]} \left[\bigwedge_{i=1}^{2} p^i \notin H_j \right]$. Figure 5b shows the complete RRTs for both the agents, which are generated using Algorithm 2, and Fig. 5c shows all the scenarios that satisfy the requirement φ_1. The scenarios are extracted from the multi-agent RRT using the model-checking approach. In Fig. 6, we show the five scenarios that satisfy φ_1. We can see that the trajectories for both agents do not intersect with the hazard regions.

Example 2: Neurosymbolic Scenario Generation. In this case, we have five agents and four hazard regions. The agents are planned to satisfy the following specification formalized in Signal Temporal Logic, which represents the edge case. In other words, the agents meet in the vicinity of one of the hazard areas.

$$\varphi_2 = \bigvee_{j=1}^{4} \mathbf{F}_{[0,T]} \left[\bigwedge_{i,k \in [1,5]} (\mathbf{d}(p_i, p_k) < \epsilon \wedge \mathbf{d}(p_i, H_j) < \epsilon) \right] \wedge \bigwedge_{j=1}^{4} \mathbf{G}_{[0,T]} \left[\bigwedge_{i=1}^{5} p_i \notin H_j \right]$$



(a) Camera view of the factory floor Unity simulator

(b) Generation of RRTs for two agents

(c) Trajectories of agents that satisfy the requirement φ_1

Fig. 5. (a) shows Unity simulator for warehouse environment, (b) shows the generated RRT for two agents, and (c) shows the extracted trajectories from RRTs that satisfy the requirement φ_1.

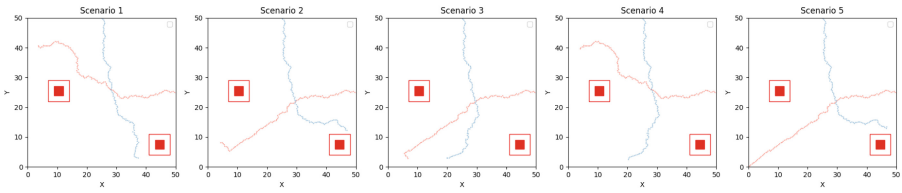


Fig. 6. The figure shows five scenarios that satisfy φ_1. In each scenario, we can see the trajectories of two agents that are always safe. These scenarios are generated using RRT and the model-checking approach.

p_i indicates the position of the agent $i \in \{1, 2, 3, 4, 5\}$ and H_j represents the j^{th} hazardous area $j \in \{1, 2, 3, 4\}$. The specification implies the agents should meet together in the vicinity of one of the hazardous areas, while they are always avoiding all the hazardous regions. The agents will also start from one of the four doors provided for the room. Here the velocities are bounded within the interval $v^i \in [0, 3]$ and the horizon of the specification is $T = 50$ time-steps with sampling time $\delta t = 0.1$ s. Figure 7 shows five scenarios generated to satisfy φ_2. We have generated 25 scenarios. Assuming the sup-norm[4] as the distance between the scenarios, the scenarios are ϵ-separated with $\epsilon = 3.0683$. The maximum distance and the average distance are also 12.1254 and 8.2053 respectively.

5.2 Autonomous Driving

Autonomous driving is another popular area for scenario generation, as on-road testing of critical scenarios is not always possible. Our environment for this case study consists of a two-lane road, as shown in Fig. 8, where agents can drive from left to right. The car agents follow the dynamics of the Dubins cars. We consider that we have two car agents in the environemnt.

[4] For two vectors $V_1, V_2 \in \mathbb{R}^n$, we have: $\text{sup-norm}(V_1, V_2) = \max_i(|\, V_1(i) - V_2(i)\,|)$.

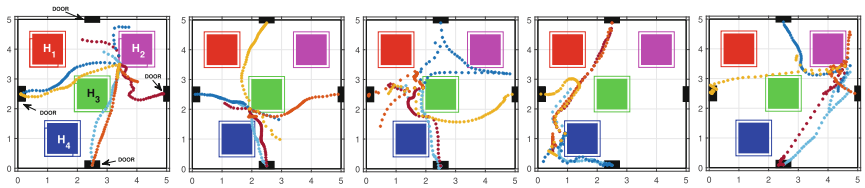
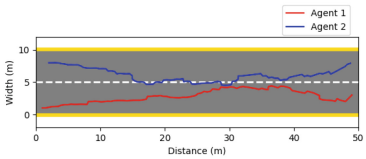
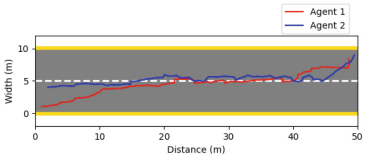


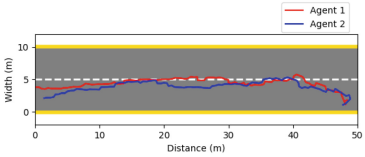
Fig. 7. This figure shows 5 different scenarios generated via Neurosymbolic technique. The scenario is the collection of 5 traces from the 5 agents, where the agents together satisfy the STL specification. The agents start from one of the doors and meet each other in the ϵ vicinity of one of the hazardous regions.



(a) Scenario for maintaining safe distance between agents



(b) Scenario for lane change



(c) Scenario showing vehicle crash

Fig. 8. Autonomous driving case study: A two-lane road where cars can navigate from left to right. A white dotted line separates the two lanes. The lane shown on the top is considered the left lane, and the bottom is the right lane.

In the first example, $\varphi_3 = \mathbf{G}_{[0,T]}\left[\bigwedge_{i,k\in\{1,\ldots,n\}}(\mathbf{d}(c_i,c_j) > d_{safe})\right]$, we want the agents to always maintain at least d_{safe} distance from each other. We generated scenarios for this behavior using the requirement φ_3 and Fig. 8a shows an example scenario that satisfies φ_3. Next, we demonstrate the scenario where car agents change lanes. We assume that the agents start in the right lane, eventually merge into the left lane, and stay in the left lane. We represent this requirement by $\varphi_4 = \mathbf{F}_{[0,b]}\left[\mathbf{G}_{[b,L-b]}\left[\bigwedge_{i\in\{1,\ldots,n\}}(lane(c_i) = left)\right]\right]$. We describe this scenario using φ_4, and Fig. 8b shows a scenario for this case. This figure shows the left lane on the top and the right lane on the bottom. Lastly, we show a crash scenario in Fig. 8c using $\varphi_5 = \mathbf{F}_{[0,T]}\left[\bigwedge_{i,k\in\{1,\ldots,n\}}(\mathbf{d}(c_i,c_j) < \epsilon)\right]$. In this scenario, two car agents collide with each other at some time t.

Table 1. Shows the runtime and the distance between the scenarios. Note: For the RRT and model checking (RRT-MC) approach, the time for RRT generation is not included in the runtimes. Scenic could not generate any scenarios for φ_2 and φ_4 for 15 min.

Case Study	Requirement	Appraoch	Average Runtime (seconds)	Distance between scenarios	
				Mean	Std. Deviation
Industrial factory floor	φ_1	RRT-MC	0.02	0.50	0.31
		Neuro-symbolic	32.96	50.21	10.98
		Scenic	3.7	28.36	10.50
	φ_2	RRT-MC	14.87	2.63	2.17
		Neuro-symbolic	135.54	8.20	1.30
		Scenic	TIMEOUT	—	—
Autonomous Driving	φ_3	RRT-MC	0.11	0.11	0.07
		Neuro-symbolic	80.15	7.86	2.12
		Scenic	6.54	4.59	1.39
	φ_4	RRT-MC	0.27	3.18	0.70
		Neuro-symbolic	69.35	9.98	2.68
		Scenic	TIMEOUT	—	—
	φ_5	RRT-MC	0.07	0.09	0.07
		Neuro-symbolic	53.62	7.16	1.85
		Scenic	7.92	3.60	0.92

5.3 Results

We have performed our experiments for scenario generation using both of the proposed approaches. We also performed the same experiments with Scenic and to compare our results. We outlined our quantitative results in Table 1 by reporting runtimes and distance between scenarios for five requirements. In the RRT approach, there is an upfront cost for generating the multi-agent RRT. Once the RRT is constructed, it can be reused for generating several scenarios based on various requirements and this scenario extraction is quite fast. To show this, we have used the same RRT for generating scenarios for for φ_3, φ_4 and φ_5 because the environment is exactly the same (Fig. 8). The time for generating RRT for autonomous driving case study is 1.49 seconds. The time to generate RRT for φ_1 is 26.14 seconds and φ_2 is 82.56 seconds. We do not reuse the multi-agent RRT here because the environments and number of agents are different. We observe from the Table 1 that the scenarios generated using path planning and model checking method are much faster than the Neurosymbolic technique but the Neurosymbolic methods generates much diverse scenarios.

The RRT approach is beneficial when there are several scenarios under independent constraints that need to be identified for testing, and reuse of generated trajectories is possible. The Neuro-symbolic approach is useful when there is a complex scenario constraint that needs to be satisfied and is more useful when we have a differentiable simulation environment. These are complementary approaches, and their application depends on the use case. While RRT-based

method is better for reusing the generated trajectories for different scenario constraints, the Neuro-Symbolic approach is better for complex specifications with differentiable environments.

6 Related Work and Conclusions

Generating safety-critical scenarios is highly important for adaptive stress testing [41] and analyzing corner cases [34] in the research and development of autonomous cyber-physical systems [40].

Scenario Description Languages: Several scenario description languages have been proposed to define complex scenarios [1,31,37]. OpenScenario [1] offers a standardized approach and an XML format for modeling autonomous vehicle scenarios, where the motion of an agent is characterized through actions and triggers. These triggers are linked to the activities of other agents and their corresponding actions. GeoScenario [33], built on the foundation of OpenScenario, defines the behavior of a dynamic agent through its position and speed profiles, along with responsive triggers. Initially created to describe various scenes, Scenic [14], a probabilistic programming language, was later adapted to handle dynamic scenarios through actions specified by simulators, which may depend on the behavior of other traffic participants on the road. While Scenic relies on a rejection sampling approach to generate scenarios, we focus on a requirement-driven approach.

Scenario Based Testing: Originally developed for robot path and motion planning issues [22,23], Rapidly-exploring Random Trees (RRT) [29] have also shown promising results in the test generation domain [6,7,10,26] due to their capability to efficiently search high-dimensional spaces. Previous work the use of RRT for testing multi-robot controllers [25], autonomous racing maneuvers [5], vehicle control [27,38]. Since its introduction, numerous variants of RRT have been developed. One variant, known as Transition-based RRT (T-RRT) [22], merges the exploratory capabilities of the RRT algorithm, which swiftly expands random trees into unexplored areas, with the effectiveness of stochastic optimization methods, that employ transition tests to either accept or reject a new potential state. Another variant, RRT* [23], is demonstrably asymptotically optimal, meaning the cost of the solution it produces almost surely converges to the optimum.

Conclusion. In this paper, we consider the problem of scenario generation where the dynamic scenarios generated are required to satisfy given Discrete-Time STL requirements. We explore two different strategies for scenario generation. In the first, we use incremental sampling-based path-planning approaches to build a database of scenarios consistent with a given scenario description program (for example, in the probabilistic programming language Scenic). We then use model checking to extract scenarios matching the given requirement. In the second strategy, we explore a neuro-symbolic approach, where the given STL

requirement is encoded with a smooth neural network and we use optimization-based methods to efficiently generate scenarios satisfying the requirement. We demonstrate our scenario generation technique on two different domains: industrial warehouse and autonomous driving.

References

1. Asam openscenario. https://www.asam.net/standards/detail/openscenario/. Accessed 19 May 2024
2. AG, S.: Siemens offers real-time locating system for a safe production environment and optimized production processes (2020). https://press.siemens.com/global/en/pressrelease/siemens-offers-real-time-locating-system-safe-production-environment-and-optimized
3. Akazaki, T., Hasuo, I.: Time robustness in MTL and expressivity in hybrid system falsification. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9207, pp. 356–374. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21668-3_21
4. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
5. Bak, S., Betz, J., Chawla, A., Zheng, H., Mangharam, R.: Stress testing autonomous racing overtake maneuvers with RRT. In: 2022 IEEE Intelligent Vehicles Symposium (IV), pp. 806–812. IEEE (2022)
6. Branicky, M.S., Curtiss, M.M., Levine, J., Morgan, S.: Sampling-based planning, control and verification of hybrid systems. IEE Proc.-Control Theory Appl. **153**(5), 575–590 (2006)
7. Dang, T., Donzé, A., Maler, O., Shalev, N.: Sensitive state-space exploration. In: 2008 47th IEEE Conference on Decision and Control, pp. 4049–4054. IEEE (2008)
8. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15297-9_9
9. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: Carla: an open urban driving simulator. In: Conference on Robot Learning, pp. 1–16. PMLR (2017)
10. Dreossi, T., Dang, T., Donzé, A., Kapinski, J., Jin, X., Deshmukh, J.V.: Efficient guiding strategies for testing of temporal properties of hybrid systems. In: Havelund, K., Holzmann, G., Joshi, R. (eds.) NFM 2015. LNCS, vol. 9058, pp. 127–142. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17524-9_10
11. Emerson, E.A., Lei, C.L.: Modalities for model checking (extended abstract) branching time strikes back. In: Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, pp. 84–96 (1985)
12. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications. In: Havelund, K., Núñez, M., Roşu, G., Wolff, B. (eds.) FATES/RV -2006. LNCS, vol. 4262, pp. 178–192. Springer, Heidelberg (2006). https://doi.org/10.1007/11940197_12
13. Freeman, C.D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., Bachem, O.: Brax–a differentiable physics engine for large scale rigid body simulation. arXiv preprint arXiv:2106.13281 (2021)
14. Fremont, D.J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: a language for scenario specification and scene generation. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 63–78 (2019)

15. Fremont, D.J., et al.: Scenic: a language for scenario specification and data generation. Mach. Learn. **112**(10), 3805–3849 (2023)
16. Fronda, N., Abbas, H.: Differentiable inference of temporal logic formulas. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **41**(11), 4193–4204 (2022)
17. Frostig, R., Johnson, M.J., Leary, C.: Compiling machine learning programs via high-level tracing. Syst. Mach. Learn. **4**(9) (2018)
18. Gulino, C., et al.: Waymax: an accelerated, data-driven simulator for large-scale autonomous driving research. Adv. Neural Inf. Process. Syst. **36** (2024)
19. Hansen, N., Müller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evol. Comput. **11**(1), 1–18 (2003)
20. Hashemi, N., Hoxha, B., Prokhorov, D., Fainekos, G., Deshmukh, J.: Scaling learning based policy optimization for temporal tasks via dropout. arXiv preprint arXiv:2403.15826 (2024)
21. Heiden, E., Millard, D., Coumans, E., Sheng, Y., Sukhatme, G.S.: NeuralSim: augmenting differentiable simulators with neural networks. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) (2021). https://github.com/google-research/tiny-differentiable-simulator
22. Jaillet, L., Cortés, J., Siméon, T.: Transition-based RRT for path planning in continuous cost spaces. In: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2145–2150. IEEE (2008)
23. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. J. Robot. Res. **30**(7), 846–894 (2011)
24. Kavraki, L.E., Kolountzakis, M.N., Latombe, J.C.: Analysis of probabilistic roadmaps for path planning. IEEE Trans. Robot. Autom. **14**(1), 166–171 (1998)
25. Kim, J., Esposito, J.M., Kumar, V.: An RRT-based algorithm for testing and validating multi-robot controllers. In: Robotics: Science and Systems, pp. 249–256. Boston, MA (2005)
26. Kim, J., Esposito, J.M., Kumar, V.: Sampling-based algorithm for testing and validating robot controllers. Int. J. Robot. Res. **25**(12), 1257–1272 (2006)
27. Koschi, M., Pek, C., Maierhofer, S., Althoff, M.: Computationally efficient safety falsification of adaptive cruise control systems. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC), pp. 2879–2886. IEEE (2019)
28. Lagarias, J.C., Reeds, J.A., Wright, M.H., Wright, P.E.: Convergence properties of the nelder-mead simplex method in low dimensions. SIAM J. Optim. **9**(1), 112–147 (1998)
29. LaValle, S.: Rapidly-exploring random trees: a new tool for path planning. Research Report, no. 9811 (1998)
30. Leung, K., Aréchiga, N., Pavone, M.: Backpropagation through signal temporal logic specifications: infusing logical structure into gradient-based methods. Int. J. Robot. Res. **42**(6), 356–370 (2023)
31. Majumdar, R., Mathur, A., Pirron, M., Stegner, L., Zufferey, D.: Paracosm: a test framework for autonomous driving simulations. In: FASE 2021. LNCS, vol. 12649, pp. 172–195. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-71500-7_9
32. Marini, F., Walczak, B.: Particle swarm optimization (PSO). A tutorial. Chemom. Intell. Lab. Syst. **149**, 153–165 (2015)
33. Queiroz, R., Berger, T., Czarnecki, K.: Geoscenario: an open dsl for autonomous driving scenario representation. In: 2019 IEEE Intelligent Vehicles Symposium (IV), pp. 287–294. IEEE (2019)

34. Riedmaier, S., Ponn, T., Ludwig, D., Schick, B., Diermeyer, F.: Survey on scenario-based safety assessment of automated vehicles. IEEE Access **8**, 87456–87477 (2020)
35. Rodionova, A., Lindemann, L., Morari, M., Pappas, G.J.: Combined left and right temporal robustness for control under STL specifications. IEEE Control Syst. Lett. (2022)
36. Rong, G., et al.: LGSVL simulator: a high fidelity simulator for autonomous driving. In: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), pp. 1–6. IEEE (2020)
37. Schütt, B., Braun, T., Otten, S., Sax, E.: SceML: a graphical modeling framework for scenario-based testing of autonomous vehicles. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp. 114–120 (2020)
38. Tuncali, C.E., Fainekos, G.: Rapidly-exploring random trees for testing automated vehicles. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC), pp. 661–666. IEEE (2019)
39. Vin, E., et al.: 3D environment modeling for falsification and beyond with scenic 3.0. In: Enea, C., Lal, A. (eds.) Computer Aided Verification. CAV 2023. LNCS, vol. 13964, pp. 253–265. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-37706-8_13
40. Zhang, L., Peng, Z., Li, Q., Zhou, B.: Cat: closed-loop adversarial training for safe end-to-end driving. In: Conference on Robot Learning, pp. 2357–2372. PMLR (2023)
41. Zhong, Z., Tang, Y., Zhou, Y., Neves, V.D.O., Liu, Y., Ray, B.: A survey on scenario-based testing for automated driving systems in high-fidelity simulation. arXiv preprint arXiv:2112.00964 (2021)