Hybrid Systems:
From Verification to Falsification

Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi

Department of Computer Science, Rice University
Houston TX 77005, USA
{plakue, kavraki, vardi}@cs.rice.edu

Abstract. We propose HyDICE, Hybrid DIscrete Continuous Exploration, a multi-layered approach for hybrid-system testing that integrates continuous sampling-based robot motion planning with discrete searching. The discrete search uses the discrete transitions of the hybrid system and coarse-grained decompositions of the continuous state spaces or related projections to guide the motion planner during the search for witness trajectories. Experiments presented in this paper, using a hybrid system inspired by robot motion planning and with nonlinear dynamics associated with each of several thousand modes, provide an initial validation of HyDICE and demonstrate its promise as a hybrid-system testing method. Comparisons to related work show computational speedups of up to two orders of magnitude.

1 Introduction

Hybrid systems play an increasingly important role in transportation networks [1], manufacturing processes [2], robotics [3], and medicine and biology [4, 5]. Today we find hybrid systems in sophisticated embedded controllers used in the automotive and airplane industry, and also in medical devices that monitor serious health conditions. Recently, it has been shown that hybrid systems are a powerful tool for modeling biological processes and for analyzing how complex systems, such as living organisms, survive [5].

Hybrid systems are formal models that combine discrete and continuous dynamics by associating continuous dynamics with each operating mode, while using discrete logic to switch between operating modes. For example, a hybrid system may model a vehicle whose underlying motion dynamics varies discretely depending on terrain conditions. As another example, a hybrid system may model air-traffic control, where the modes correspond to the cruising of the planes and the discrete logic models conflict-resolution protocols.

As hybrid systems model more and more complex behaviors, and as they are often part of devices operating in safety-critical situations, the verification of safety properties becomes increasingly important. A hybrid system is considered safe if unsafe states cannot be reached from initial safe states. In general, hybrid-system verification consists of formally guaranteeing that a certain property is true for the system. A rich theory exists for this problem [6–10], as well as several tools, such as CheckMate[1], HyTech[2], and PHAVer[3].

Unfortunately, even for safety properties, where verification is equivalent to reachability checking, decidability holds only for hybrid systems with very simple continuous dynamics (essentially some types of linear dynamics) [11]. To handle more general hybrid systems, tools have to resort to overapproximation techniques, surveyed in [12, 13]. Such techniques are semi-decidable, since when a hybrid system is unsafe it may not be possible to show that unsafe states are reachable. Other recent approaches are semi-decidable in the opposite direction: capable of finding unsafe behaviors when the system is unsafe, but unable to determine that a system is safe [14, 15]. In essence, the focus in these recent approaches shifts from *verification* to *falsification*, which often is the main focus of model checking in industrial applications [16].

In this work we study the following problem: *Can we produce a hybrid-system trajectory from a safe state to an unsafe state when such trajectories exist?* This problem is commonly known as *hybrid-system testing*. When a hybrid system is safe, it may not be possible to prove that unsafe states are unreachable. Such an approach trades completeness for the ability to discover safety violations for complex systems that current verification methods cannot handle. Under appropriate conditions and for certain classes of algorithms, as discussed later in the paper, as the running time increases, we can also increase our confidence in the safety of the system, since the testing method has not been able to produce a trajectory that violates safety properties. An efficient framework for finding trajectories to unsafe states can shed light on the operation of the hybrid system and may suggest possible interventions. Such framework is particularly useful in the early stages of hybrid-system development, when errors in design are common.

This work approaches hybrid-system testing using a robotics-inspired method. Initially, we exploit the insight that hybrid-system testing is in many respects related to robot motion planning. The motion-planning problem consists of searching a continuous space for a trajectory for a robotic system from an initial to a final state, such that kinodynamic constraints on the robot motion are respected and collision with obstacles are avoided [17, 18]. Hybrid-system testing is also a reachability analysis on the state space of the hybrid system. In particular, finding a trajectory from a safe to an unsafe state in a hybrid system entails searching a high-dimensional state space with continuous and discrete components.

The connection with motion planning becomes deeper when we consider state-of-the art motion-planning algorithms as the starting point for the methods used for searching the continuous state space of a hybrid system. Recent advances in motion planning have made it possible to efficiently find trajectories from initial to final states even for continuous systems with hundreds of dimensions whose motion is governed by nonlinear dynamics [17–24]. The most success-

[1] `http://www.ece.cmu.edu/~webk/checkmate/`
[2] `http://embedded.eecs.berkeley.edu/research/hytech/`
[3] `http://www.cs.ru.nl/~goranf/`

ful planning methods are numerical, sampling-based methods. These methods generate samples in the state space and connect them with simple trajectories. (e.g., `PRM` [25], `RRT` [19], `EST` [20], `PDST` [22], `GRIP` [23], `DSLX-Plan` [24], and others [17, 18]). Motion planning methods, such as `RRT`, have already been used for hybrid-system testing for nonlinear hybrid systems with few modes [14, 15].

Departing from traditional robot motion planning, we introduce a discrete component to our work that is responsible for managing the potentially huge complexity of the discrete transitions. The contribution of this work is the development of a multi-layered framework for hybrid-system testing that blends sampling-based motion planning with discrete searching. The motivation and many of our design decisions come from our earlier work [24, 26]. In [26] we use discrete search to obtain a sequence of transitions that guides the generation of motions for a hybrid robotic system with 10–30 modes and mostly linear dynamics. In [24] we show that traditional motion-planning problems can be solved more efficiently by combining sampling-based motion planning with discrete search over an artificially imposed decomposition of the environment on which the robot moves (which in general can be regarded as a projection of its state space). The work presented in this paper combines and extends ideas developed in [24, 26] to obtain an effective testing method for hybrid systems with thousands of modes and nonlinear dynamics.

The proposed method, `HyDICE`, imposes a coarse-grained decomposition on the continuous state space associated with each mode. The decomposition and the discrete transitions of the hybrid system are used to construct a discrete search graph. Vertices of the search graph correspond to decomposition regions, while edges connect vertices corresponding to two adjacent decomposition regions or two decomposition regions that are connected by a discrete transition. The search graph is used to find *leads*, that is sequences of decomposition regions that constitute search directions and may be useful in finding a trajectory from a safe to an unsafe state. The search inside each of the continuous decomposition regions is done by a state-of-the-art sampling-based motion-planning technique. Information gathered during exploration, such as region coverage, exploration time and progress, and other quantities, are used to refine the discrete search and improve the quality of the lead computed for the next exploration step.

In contrast to previous work [14, 15], the multi-layered approach developed in this paper is well-suited for systems with many modes and transitions and offers considerable computational improvements over existing methods as demonstrated in this paper. Initial validation of `HyDICE` is provided by testing hybrid systems inspired by motion-planning problems that have thousands of modes and transitions and nonlinear dynamics associated with each mode. As indicated by the experiments, the tight integration of discrete search and exploration enables `HyDICE` to be up to two orders of magnitude faster than other related methods.

The rest of the paper is as follows. The hybrid-testing problem and the hybrid system used in the experiments are described in Section 2. Details of `HyDICE` are provided in Section 3. Experiments and results are presented in Section 4. We conclude in Section 5 with a discussion.

2 Problem Description

2.1 Hybrid Automata and Hybrid-Testing Problem

In this work, hybrid systems are modeled by hybrid automata [6]. A hybrid automaton is a tuple $H = (S, E, G, J, f, U, I, F)$, where $S = Q \times X$; Q is a discrete and finite set; each $X_q \in X$, $X_q \subseteq R^{\dim(\mathrm{X_q})}$, represents the continuous space associated with $q \in Q$; $E \subseteq Q \times Q$ indicates discrete transitions; each $G_{(q_i,q_j)} \in G$, $G_{(q_i,q_j)} \subseteq X_{q_i}$, and each $J_{(q_i,q_j)} \in J$, $J_{(q_i,q_j)} : X_{q_i} \rightarrow X_{q_j}$, represent the guard set and reset function associated with $(q_i, q_j) \in E$, respectively. The continuous dynamics of the system in each $q \in Q$ is governed by a set of differential equations $f_q : X_q \times U_q \rightarrow \mathrm{Tg}X_q$, where $f_q \in f$, $U_q \subseteq R^{\dim(\mathrm{U_q})}$ denotes the set of possible input controls, and $\mathrm{Tg}X_q$ denotes the tangent space of X_q. Each $X_q \in X$ usually includes derivatives of different orders, e.g., velocity and acceleration of a car, and thus f_q is typically nonlinear. The function f_q has the form $f_q(x, u_q(x))$, where the input $u_q(x) \in U_q$ associated with each $x \in X_q$ could represent continuous controls, nondeterminism, uncertainties, disturbances from the environment, or actions of other systems. In order to model these factors, the assignment $u_q(x) \in U_q$ could be non-deterministic or selected according to some probability distribution associated with U_q.

A hybrid system trajectory consists of one or more continuous trajectories interleaved with discrete transitions. Starting at some state $(q_0, x_0) \in I$, where $I \in S$ denotes the set of initial states, the system evolves according to f_{q_0} until it reaches $G_{(q_0,q_1)}$, for some $q_1 \in Q$. Then a discrete transition to q_1 occurs and the continuous state is reset by $J_{(q_0,q_1)}(x_0)$. The system evolution continues in such manner until the end of execution time.

A hybrid system is considered unsafe if a witness trajectory is produced that takes the hybrid system from some initial safe state $s_{\mathrm{safe}} \in I$ to some $s_{\mathrm{unsafe}} \in F$, where $F \subseteq S$ denotes a set of unsafe states.

2.2 A Hybrid System Inspired by Motion Planning Problems

The hybrid system used throughout this paper consists of an autonomous robotic car, whose underlying dynamics change discretely depending on terrain conditions. The choice of this specific system is to provide a concrete, scalable benchmark in which the competitiveness of our approach can be tested.

A given environment is divided into a number of terrains $\{R_1, \ldots, R_N\}$, where each R_i corresponds to an operating mode $q_i \in Q$. The motion of the robotic car inside each terrain is specified by a set of ordinary differential equations. A discrete transition (q_i, q_j) occurs when the robotic car enters a part of R_i designated as the guard set $G_{(q_i,q_j)}$. The discrete transition indicates necessary changes in the way the robotic car should be controlled to adapt to changes in terrain conditions. After entering R_j, the continuous state of the robotic car is reset as specified by $J_{(q_i,q_j)}$ and the underlying dynamics of the car is specified according to the motion equations associated with $q_j \in Q$. The robotic car is said to have entered some unsafe state, for example, if it is in a particular terrain

R_j and the speed is above a certain predefined threshold. The robotic car could behave as a kinematic (first-order) car (`KCar`), smooth (second-order) car (`SCar`), smooth Reeds-Shepp car (`RSCar`), smooth unicycle (`SUni`), or smooth differential drive (`SDDrive`). Detailed descriptions of these models can be found in [17, 18].

Kinematic Car (KCar): A continuous state x is of the form $x = [p, \theta]$, where $p \in \mathbb{R}^2$ and $\theta \in (-\pi, \pi]$ denote the position and orientation of the robotic car. The motion equations are $\dot{p_0} = u_0(x)\cos(\theta); \dot{p_1} = u_1(x)\sin(\theta); \dot{\theta} = u_0(x)\tan(u_1(x))/L$, where $u_0(x) \in [-1, 1]$ and $u_1(x) \in [-1, 1]$ are the speed and steering wheel controls and L is the distance between the front and rear axles.

Smooth Car (SCar): The kinematic car model can be extended to a second-order model by expressing the velocity v and steering angle ϕ as differential equations of the acceleration $u_0(x)$ and the rotational velocity of the steering wheel $u_1(x)$ controls, as follows: $x = [p, \theta, v, \phi]$ and $\dot{p_0} = v\cos(\theta)$; $\dot{p_1} = v\sin(\theta)$; $\dot{\theta} = v\tan(\phi)/L; \dot{v} = u_0(x)$; $\dot{\phi} = u_1(x)$.

Smooth Reeds-Shepp Car (RSCar): A smooth Reeds-Shepp car is similar to a smooth car, but the acceleration control $u_0(x)$ is only allowed to be from the set $\{-\max, 0, \max\}$, where $\max \in \mathbb{R}$ is some predefined parameter.

Smooth Unicycle (SUni): The continuous state x is $x = [p, \theta, v, \omega]$, where p, θ, v are as in the smooth car model and ω indicates the rotational velocity. The motion equations are $\dot{p_0} = v\cos(\theta)$; $\dot{p_1} = v\sin(\theta)$; $\dot{\theta} = \omega$; $\dot{v} = u_0(x)$; $\dot{\omega} = u_1(x)$.

Smooth Differential Drive (SDDrive): The motion equations are $\dot{p_0} = 0.5r(\omega_\ell + \omega_r)\cos(\theta)$; $\dot{p_1} = 0.5r(\omega_\ell + \omega_r)\sin(\theta)$; $\dot{\theta} = r(\omega_r - \omega_\ell)/L$; $\dot{\omega_\ell} = u_0(x)$; $\dot{\omega_r} = u_1(x)$, where $x = [p, \theta, \omega_\ell, \omega_r]$ is the continuous state; ω_ℓ and ω_r are the rotational velocities of the left and right wheels, respectively; r is the wheel radius; and L is the length of the axis connecting the centers of the two wheels.

The controls $u_0(x)$ and $u_1(x)$ could be thought of as playing the role of the automatic driver. The objective of hybrid-system testing is then to test the safety of the automatic driver, i.e., the driver is unsafe if a witness trajectory is produced that indicates that it is possible for the robotic car to enter an unsafe state. Due to length limitations of this paper, we only provide high-level descriptions of the automatic drivers.[4] These driver models consist of simple if-then-else statements depending on the state values and motion equations. In the first model, `RandomDriver`, $u_0(x)$ and $u_1(x)$ are selected pseudo-uniformly at random from some $[-\max_0, \max_0]$ and $[-\max_1, \max_1]$, respectively. In a second model, `StudentDriver`, the driver follows an approach similar to stop-and-go. When the speed is close to zero, `StudentDriver` selects $u_0(x)$ and $u_1(x)$ as in the `RandomDriver` model. Otherwise, `StudentDriver` selects controls that reduce the speed. The third model, `HighwayDriver` attempts to maintain the speed within acceptable low and upper bounds and avoid sharp turns. When the speed is too low, `HighwayDriver` selects controls that increase the speed. When the speed is too high, `HighwayDriver` selects controls that slow down the robotic car. When the speed is between the low and upper bounds, `HighwayDriver` selects controls that do not change the speed too much.

[4] See `http://www.cs.rice.edu/CS/Robotics/CAV2007data/` for more details.

3 Methods

As discussed in the introduction, `HyDICE` constructs a discrete search graph based on the discrete transitions and a decomposition of the continuous state spaces. Observe that any witness trajectory from $s_{\text{safe}} \in I$ to $s_{\text{unsafe}} \in F$ passes through a sequence of decomposition regions. Although the converse does not hold, a sequence of connected decomposition regions, starting and ending in two regions containing states in I and F, respectively, *may* contain a witness trajectory. Such sequences of connected regions, referred to as *leads*, provide search directions which are used by the sampling-based motion-planning method as guides for the exploration of the state space of a given hybrid system.

The search for a witness trajectory proceeds iteratively. Throughout the search, `HyDICE` maintains an exploration tree $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$, which initially contains only s_{safe}, i.e., $V_{\mathcal{T}} = \{s_{\text{safe}}\}$ and $E_{\mathcal{T}} = \emptyset$. The vertices $V_{\mathcal{T}}$ are states of S, while an edge $(s', s'') \in E_{\mathcal{T}}$ indicates that a hybrid-system trajectory connects $s' \in S$ to $s'' \in S$. At each iteration, `HyDICE` uses the discrete search graph to compute a lead and then sampling-based motion planning to extend the branches of $\mathcal{T}$ in the direction specified by the lead. The branches of $\mathcal{T}$ are extended by adding new vertices and edges to $V_{\mathcal{T}}$ and $E_{\mathcal{T}}$, respectively. A witness trajectory is found when a state $s_{\text{unsafe}} \in F$ is added to $\mathcal{T}$. Otherwise, the search continues until an upper bound on computation time is exceeded. Pseudocode is provided in Algorithm 1. The discrete search and the sampling-based motion planning are described in Sections 3.1 and 3.2, respectively.

Algorithm 1 Pseudocode for `HyDICE`

Input: $H = (S, Inv, E, G, J, f, U, I, F)$: hybrid system; $t_{\max} \in \mathbb{R}$: upper bound on overall computation time; $t_e \in \mathbb{R}$: short time allocated to each exploration step
Output: A witness trajectory or `FAILURE` if no witness trajectory is found

StartClock1
$\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$; $V_{\mathcal{T}} \leftarrow \{s_{\text{safe}}\}$; $E_{\mathcal{T}} \leftarrow \emptyset$
$D \leftarrow$ CoarseGrainedDecomposition(H)
$G_D = (V_D, E_D) \leftarrow$ DiscreteSearchGraph(D)
InitExplorationEstimates(G_D)
while ElapsedTime1 $< t_{\max}$ **do**
  $\sigma \leftarrow$ DiscreteLead(G_D)
  StartClock2
  while ElapsedTime2 $< t_e$ **do**
    $s \leftarrow$ SelectState($\mathcal{T}, \sigma$)
    $s_{\text{new}} \leftarrow$ PropagateForward($H, \mathcal{T}, s, \sigma$)
    $V_{\mathcal{T}} \leftarrow V_{\mathcal{T}} \cup \{s_{\text{new}}\}$; $E_{\mathcal{T}} \leftarrow E_{\mathcal{T}} \cup \{(s, s_{\text{new}})\}$
    if $s_{\text{new}} \in F$ **then return** WitnessTrajectory($\mathcal{T}, s_{\text{new}}$)
  UpdateExplorationEstimates(G_D, σ)
return `FAILURE`

3.1 Discrete Search

Coarse-grained Decomposition `HyDICE` constructs a coarse-grained decomposition $D = \{D_{q_1}, \ldots, D_{q_N}\}$, where $D_{q_i} = \{D_{q_i,1}, \ldots, D_{q_i,n_i}\}$ denotes the decomposition of X_{q_i} into $n_i \in \mathbb{N}$ different regions (line 3 of Algorithm 1). `HyDICE` does not impose any strict requirements on the decomposition. D_{q_i} is usually computed as a set of nonoverlapping regions in some low-dimensional projection of X_{q_i}. For the hybrid system used in this work, `HyDICE` projects each X_{q_i} onto $\mathbb{R}^2$ and constructs a cell-based decomposition. Other types of projections and decompositions are possible [17, 18].

Discrete Search Graph D is used to create a search graph $G_D = (V_D, E_D)$. A vertex $v_{q_i,j}$ is added to V_D for each $D_{q_i,j}$. An edge $(v_{q_i,j}, v_{q_i,k})$ is added to E_D for each two adjacent regions $D_{q_i,j}$ and $D_{q_i,k}$. Furthermore, an edge $(v_{q_i,j}, v_{q_\ell,k})$ is added to E_D for each two regions $D_{q_i,j}$ and $D_{q_\ell,k}$ such that there is a discrete transition from some state (q_i, x_1), $x_1 \in D_{q_i,j}$, to some state (q_ℓ, x_2), $x_2 \in D_{q_\ell,k}$. There are also two special vertices v_{safe} and v_{unsafe} added to V_D. An edge $(v_{\text{safe}}, v_{q_i,j})$ is added to E_D for each $D_{q_i,j}$ such that $D_{q_i,j} \cap I \neq \emptyset$. Similarly, an edge $(v_{q_i,j}, v_{\text{unsafe}})$ is added to E_D for each $D_{q_i,j}$ such that $D_{q_i,j} \cap F \neq \emptyset$. This operation is found in line 4 of Algorithm 1.

Importance of Leads A central issue is which lead to choose from the combinatorially large number of possibilities. This issue is addressed by associating a weight $w_{(v_{q_i,j}, v_{q_\ell,k})}$ with each $(v_{q_i,j}, v_{q_\ell,k}) \in E_D$, which estimates the importance of including $(v_{q_i,j}, v_{q_\ell,k}) \in E_D$ as an edge in the lead. Preference is given to leads associated with higher edge weights. For the moment assume that there is no distinction between edges corresponding to discrete transitions and edges connecting adjacent regions in the decomposition. At the end we discuss the possibility of using different weighting schemes depending on the edge type.

Initially, the weights are set to a fixed value (line 5) and are updated (line 14) after each exploration step. The weight $w_{(v_{q_i,j}, v_{q_\ell,k})}$ depends on the *coverage* of $D_{q_i,j}$ and $D_{q_\ell,k}$ by $\mathcal{T}$. The coverage $c(\mathcal{T}, D_{q_i,j})$ is computed by imposing an implicit uniform grid on $D_{q_i,j}$ and measuring the fraction of cells that contain at least one state from $\mathcal{T}$. Let $c_{\text{prev}}(\mathcal{T}, D_{q_i,j})$ denote $c(\mathcal{T}, D_{q_i,j})$ at the beginning of the current exploration step (before line 9) and let $c_{\text{after}}(\mathcal{T}, D_{q_i,j})$ denote $c(\mathcal{T}, D_{q_i,j})$ at the end of the exploration step (after line 13). Thus $\Delta c(\mathcal{T}, D_{q_i,j}) = c_{\text{after}}(\mathcal{T}, D_{q_i,j}) - c_{\text{prev}}(\mathcal{T}, D_{q_i,j})$ indicates the change in the coverage of $D_{q_i,j}$ by $\mathcal{T}$ as a result of the current exploration step. $c(\mathcal{T}, D_{q_\ell,k})$, $c_{\text{prev}}(\mathcal{T}, D_{q_\ell,k})$, $c_{\text{after}}(\mathcal{T}, D_{q_\ell,k})$, and $\Delta c(\mathcal{T}, D_{q_\ell,k})$ are defined similarly. Let t denote the computation time devoted to the exploration of $D_{q_i,j}$ and $D_{q_\ell,k}$ during the current exploration step and let $t_{\text{acc}}(D_{q_i,j}, D_{q_\ell,k})$ denote the accumulated exploration time devoted to $D_{q_i,j}$ and $D_{q_\ell,k}$. Then, the weight $w_{(v_{q_i,j}, v_{q_\ell,k})}$ is defined as $w_{(v_{q_i,j}, v_{q_\ell,k})} = (1-\epsilon)(\Delta c(\mathcal{T}, D_{q_i,j}) + \Delta c(\mathcal{T}, D_{q_\ell,k}))/(2t) + \epsilon / t_{\text{acc}}(D_{q_i,j}, D_{q_\ell,k})$, where $0 < \epsilon < 1$ is a normalization constant. Large values of $w_{(v_{q_i,j}, v_{q_\ell,k})}$ indicate promising leads, since such values are obtained when $\mathcal{T}$ in a short amount of time reaches previously uncovered parts of $D_{q_i,j}$ and $D_{q_\ell,k}$. $t_{\text{acc}}(i, j)$ is used to increase the weight of those regions that have been explored less frequently.

As described so far the same procedure determines $w_{(v_{q_i,j}, v_{q_\ell,k})}$, regardless of whether the edge from $D_{q_i,j}$ to $D_{q_\ell,k}$ corresponds to a discrete transition or adjacency in the decomposition. Intuitively, edges corresponding to discrete transitions may be more important, as they guide the sampling-based motion planner to extend branches of $\mathcal{T}$ from one continuous state space to another. For this reason, each $w_{(v_{q_i,j}, v_{q_\ell,k})}$ corresponding to a discrete transition is multiplied by some constant $w > 1$. Depending on the problem, it may also be beneficial to estimate the weights corresponding to discrete transitions differently.

Computation of Leads Leads associated with higher edge weights are selected more frequently. At the same time, each lead has a non-zero probability of being selected. In this way, `HyDICE` aims to obtain a balance between greedy and methodical search. The computation of leads is essentially a graph-search problem and there is extensive literature on the subject [27]. The approach undertaken in this work is to use combinations of different strategies, such as randomized depth-first search where the weights associated with each edge in E_D are used to select the successor vertices in the search process, Dijkstra's algorithm, and other graph-search methods [27]. For considerably larger problems, approaches from model checking, such as bounded model checking [28] or directed model checking [29], could also be used (see also discussion in Section 5).

DiscreteLead(G_D) (line 7) returns more frequently the most probable lead and the lead associated with the highest sum of edge weights and less frequently leads computed by randomized depth-first search. The most probable lead is computed using Dijkstra's algorithm and setting the weight function used in the graph search to $-\log(w_{(v_{q_i,j}, v_{q_\ell,k})}/w_{\text{total}})$ for $(v_{q_i,j}, v_{q_\ell,k}) \in E_D$, where $w_{\text{total}} = \sum_{(v',v'')\in E_D} w_{(v',v'')}$. The lead with the highest sum of edge weights is computed using Dijkstra's algorithm and setting the weight function to $w_{\max} - w_{(v_{q_i,j}, v_{q_\ell,k})}$ for $(v_{q_i,j}, v_{q_\ell,k}) \in E_D$, where $w_{\max}$ denotes the maximum weight.

3.2 Sampling-based Motion Planning

The objective of the exploration step (lines 9–13) is to use the lead σ to extend $\mathcal{T}$ toward F. The exploration step proceeds iteratively by selecting a state s from $\mathcal{T}$ and propagating forward from s to a new state s_{new}.

Conceptually, forward propagation provides the necessary mechanism for sampling-based motion planning to extend the branches of $\mathcal{T}$ and explore the state space. The forward propagation from s entails simulating the evolution of H starting at s and for a duration of t units of time, where t is selected pseudo-uniformly at random from $[\min_t, \max_t] \subset (0, \infty)$. The simulation can be computed by numerically integrating the motion equations for a short period of time and following the appropriate discrete transitions when guard conditions are met. The simulation terminates if at any point an unsafe state is reached (see [14] for more details). The new state s_{new} obtained at the end of the simulation and the edge (s, s_{new}) are added to the vertices and edges of $\mathcal{T}$, respectively.

SelectState($\mathcal{T}, \sigma$) (line 10) selects more frequently states s from $\mathcal{T}$, which, when propagated forward, bring $\mathcal{T}$ closer to F. Let $D_{q_{i_1},j_1}, \ldots, D_{q_{i_n},j_n}$ be the coarse-grained decomposition regions associated with the sequence of vertices $v_{\text{safe}}, v_{q_{i_1},j_1}, \ldots, v_{q_{i_n},j_n}, v_{\text{unsafe}}$ in σ. Since σ is a sequence of edges from v_{safe} to v_{unsafe}, the order $1 \leq k \leq n$ in which $v_{q_{i_k},j_k}$ appears in σ provides an indication of how close $D_{q_{i_k},j_k}$ is to F. Since the objective of the exploration step is to extend branches of $\mathcal{T}$ closer to F, `HyDICE` gives preference to regions $D_{q_{i_k},j_k}$ that are closer to F, i.e., k is close to n. To balance this greedy approach, `HyDICE` also takes into account the overall exploration time $t_{\text{acc}}(D_{q_{i_k},j_k})$ spent in each $D_{q_{i_k},j_k}$ and the coverage $c(\mathcal{T}, D_{q_{i_k},j_k})$. If $D_{q_{i_k},j_k}$ contains states from $\mathcal{T}$, let $w_k = \alpha k/n + \beta/c(\mathcal{T}, D_{q_{i_k},j_k}) + \gamma/t_{\text{acc}}(D_{q_{i_k},j_k})$, where α, β, and γ are normalization constants. Otherwise, let $w_k = 0$. $\text{SelectState}(\mathcal{T}, \sigma)$ selects a region $D_{q_{i_k},j_k}$ with probability $w_k / \sum_{h=1}^{m} w_h$. Each state s from $\mathcal{T}$ that is contained in $D_{q_{i_k},j_k}$ is selected with probability $1/\text{nsel}(s)$, where $\text{nsel}(s)$ is the number of times s has been previously selected. Preference is thus given to states that have been selected less frequently, since such states, when propagated forward, can cause $\mathcal{T}$ to extend in previously unexplored directions.

$\text{PropagateForward}(H, \mathcal{T}, s, \sigma)$ attempts to extend s toward $D_{q_{i_{k+1}},j_{k+1}}$ and thus bring $\mathcal{T}$ closer to F. Since the evolution of H can be nondeterministic, $\text{PropagateForward}(H, \mathcal{T}, s, \sigma)$ tries several times to propagate forward from s. Let s_{new_i} be the state obtained after simulating the evolution of H from s for a duration of t_i units of time, where t_i is selected pseudo-uniformly at random from $[\min_t, \max_t]$. $\text{PropagateForward}(H, \mathcal{T}, s, \sigma)$ computes s_{new} as the state s_{new_i} that is the closest to $D_{q_{i_{k+1}},j_{k+1}}$. A witness trajectory is found if $s_{\text{new}} \in F$. The witness trajectory is computed by reconstructing the evolution of the hybrid system from s_{safe} to s_{new} following the appropriate edges of $\mathcal{T}$ (line 13).

4 Experiments and Results

Experiments are performed using the hybrid robotic system described in Section 2.2. The hybrid robotic system is made increasingly complex by increasing the number of modes. This paper presents experiments with up to 10000 modes.

An important part of experiments is the comparison with previous related work. The closest work we can compare to is the application of `RRT` to hybrid systems [14, 15]. We also provide experiments that indicate the impact of the discrete search on the computational efficiency of `HyDICE`.

A problem instance is obtained by fixing the number N of operating modes of the hybrid robotic car. The continuous dynamics associated with each mode q_i (or terrain R_i) is selected pseudo-uniformly at random from `KCar`, `SCar`, `RSCar`, `SUni`, and `SDDrive`. The set of discrete transitions E is created as follows. Initially, discrete transitions are added between each pair R_i, R_j of neighboring terrains. A disjoint-set strategy, similar to maze creation, is then used to remove certain discrete transitions. Furthermore, each remaining discrete transition is kept with probability p. We experimented with different values of p and found that it has minimal impact on the comparisons between `HyDICE` and `RRT`. Experiments reported in this paper use $p = 0.1$. For each problem instance, we create 30 safety properties. Each safety property is created by selecting pseudo-uniformly at random one terrain as safe and another one as unsafe. As discussed in Section 2.2, the hybrid robotic car is said to have entered some unsafe state, if it is in an unsafe terrain and its speed is above a certain predefined threshold. In all the experiments, the systems were unsafe. For safe systems, since the hybrid-system testing problem is generally undecidable, all the tools used in the experiments would timeout.

Results For each problem instance, experiments are run using each of the driver models. Results are summarized in Table 1. We report the average computational time in seconds required by `RRT`, `HyDICE*`, and `HyDICE` to test 30 safety properties. `HyDICE*` refers to the version of `HyDICE` that does not use the discrete-search component, i.e., `HyDICE*` is the sampling-based motion planner of `HyDICE` with some minor modifications.[5] Comparisons include `HyDICE*` to investigate the importance of the discrete search on `HyDICE`. An entry marked with X indicates that the testing method timed out. The upper bound on time was set to 3000s for each safety property testing. The time allocated to each exploration step by `HyDICE` (t_e in Algorithm 1) was set to 1s. The Rice PBC Cluster and Rice Cray XD1 Cluster ADA were used for code development. Experiments were run on ADA, where each processor runs at 2.2GHz and has up to 8GB of RAM.

Table 1. Summary of experimental comparisons. Time is in seconds.

	`RandomDriver`			`StudentDriver`			`HighwayDriver`		
$\|Q\|$	`RRT`	`HyDICE*`	`HyDICE`	`RRT`	`HyDICE*`	`HyDICE`	`RRT`	`HyDICE*`	`HyDICE`
100	22.30	4.34	2.68	74.01	6.74	2.20	21.29	4.92	2.82
225	117.79	14.02	6.24	336.85	32.44	5.24	230.88	21.64	6.30
525	295.88	75.60	6.87	792.45	65.40	15.52	668.67	106.56	16.31
900	504.93	175.96	13.74	X	120.48	17.06	2596.50	182.54	36.96
1600	2159.24	289.94	32.52	X	464.56	34.14	X	374.44	37.26
2500	X	910.86	60.18	X	699.66	62.30	X	929.36	71.44
10000	X	X	439.88	X	X	457.60	X	X	445.52

Table 1 shows that `HyDICE` is consistently more efficient than `RRT`. When the `RandomDriver` model is used, `HyDICE` is 8.32, 18.87, 43.06, and 66.39 times faster than `RRT`, as the number of modes is increased to 100, 225, 525, and 1600, respectively. Furthermore, `RRT` times out when $|Q| = 2500$, while `HyDICE` requires only 60.18s. Similarly, when the `StudentDriver` model is used, the computational speedups obtained by `HyDICE` vary from 33.64 to 51.05 on instances where `RRT` does not time out. Under the `StudentDriver` model, `RRT` times out on instances with $|Q| = 900$, while `HyDICE` requires only 17.06s. The `StudentDriver` model is particularly computationally challenging for `RRT` since the stop-and-go approach it uses makes it difficult for `RRT` to extend the exploration tree. On the other hand, since `HyDICE` relies on a discrete search component it successfully extends the exploration tree and quickly reaches unsafe states. Similar observations are made for the `HighwayDriver` model as well.

Table 1 indicates that `HyDICE` is up to two orders of magnitude computationally faster than `RRT`. Table 1 also shows that `HyDICE` scales up reasonably well with respect to $|Q|$. In fact, `RRT` timed out in all cases when $|Q| \geq 2500$, while `HyDICE` is shown to handle problems even with $|Q| = 10000$ quite efficiently.

[5] `HyDICE*` can be obtained from the implementation of `HyDICE` by computing the lead σ as $v_{\text{safe}}, \gamma, v_{\text{unsafe}}$, where γ is a random permutation of $V_D - \{v_{\text{safe}}, v_{\text{unsafe}}\}$, where, as described in Section 3, $G_D = (V_D, E_D)$ is the search graph.

The second set of experiments provides insight on the observed computational efficiency of HyDICE. In particular, we investigate the importance of the discrete search on HyDICE. Table 1 shows that although HyDICE* is still faster than RRT, it is considerably slower than HyDICE. (For a discussion on issues related to the computational efficiency of RRT and sampling-based motion planners similar to HyDICE* see [17,18,21,24].) For example, when $|Q| = 2500$, HyDICE* is 11–15 times slower than HyDICE. Furthermore, HyDICE* times out on instances with $|Q| = 10000$, while HyDICE handles such instances efficiently. These results highlight the importance of the discrete search and agree with observations made in [24]. The interplay between lead computations and sampling-based exploration has the desired effect of quickly improving the quality of future leads and explorations and bringing the search closer to obtaining a solution. By guiding the exploration, the discrete search significantly improves the computational efficiency of HyDICE.

5 Discussion

We have presented HyDICE, a multi-layered approach for hybrid-system testing that blends sampling-based motion planning with discrete searching. The discrete search, responsible for managing the potentially huge complexity of discrete transitions, also uses coarse-grained decompositions of the continuous state spaces or related projections to guide the motion planner during the search for witness trajectories. The motion planner feeds back to the discrete search information gathered during the exploration, which is then used to further refine the discrete search and guide the motion planner toward increasingly promising search directions. This tight integration of discrete search and motion planning in the framework of HyDICE offers considerable computational advantages over related work. Experiments presented in this paper, using a hybrid robotic car, different driving models, and nonlinear dynamics associated with each of the several thousand modes, provide initial validation of HyDICE and demonstrate its promise as a hybrid-system testing method. Comparisons to related work show computational speedups of up to two orders of magnitude.

Although HyDICE was shown to handle a system with thousands of modes and nonlinear dynamics, the scalability issue is relevant and remains open to further research. As the number of modes becomes significantly large, the simple graph-search strategies used in this work becomes a computational bottleneck and need to be replaced with state-of-the-art techniques developed in the verification community which can handle discrete systems with billions of modes [30].

Additionally, the search graph is based on a decomposition of the continuous state spaces or related projections and the ability to determine whether or not two decomposition regions are connected by a discrete transition. Depending on the hybrid system, guard sets, and reset functions it may be challenging to determine if such discrete transition exists. In such cases, a viable approach would be to resort to approximations of guard sets and reset functions, which also requires investigating the overall impact of approximations on HyDICE.

One important theoretical issue that is subject of ongoing research relates to guarantees `HyDICE` can offer for general hybrid-system testing. Although completeness cannot be guaranteed, since the problem is generally undecidable, our belief is that `HyDICE` offers a weaker form of completeness, referred to as *probabilistic completeness*. Guaranteeing probabilistic completeness means that, for unsafe systems, the probability of finding a witness trajectory goes to one as the running time approaches infinity [17]. Probabilistic completeness allows us to increase the confidence in the safety of the system as the running time increases. The work in [31] has already proven probabilistic completeness in a continuous setting for certain classes of motion-planning methods, such as the one used by `HyDICE`. The theoretical framework developed in [31] is also promising for showing probabilistic completeness in a hybrid-system setting and we are currently investigating such an approach.

We also intend in future work to experiment with `HyDICE` in other settings and apply it to increasingly realistic and complex hybrid systems.

Acknowledgment

This work has been supported in part by NSF CNS 0615328 (EP, LEK, MYV), NSF 0308237 (EP, LEK), a Sloan Fellowship (LEK), and NSF CCF 0613889 (MYV). Experiments were obtained on equipment supported by NSF CNS 0454333 and NSF CNS 0421109 in partnership with Rice University, AMD, and Cray.

References

1. Glover, W., Lygeros, J.: A stochastic hybrid model for air traffic control simulation. In Rajeev, A., Pappas, G.J., eds.: Hybrid Systems: Computation and Control. Volume 2993 of LNCS. (2004) 372–386
2. Pepyne, D., Cassandras, C.: Optimal control of hybrid systems in manufacturing. Proceedings of IEEE **88**(7) (2000) 1108–1123
3. Johansson, R., Rantzer, A., eds.: Nonlinear and Hybrid Systems in Automotive Control. Springer-Verlag, London, UK (2003)
4. Dounias, G., Linkens, D.A.: Adaptive systems and hybrid computational intelligence in medicine. Artificial Intelligence in Medicine **32**(3) (2004) 151–155
5. Piazza, C., Antoniotti, M., Mysore, V., Policriti, A., Winkler, F., Mishra, B.: Algorithmic algebraic model checking I: Challenges from systems biology. In: Computer Aided Verification. Volume 3576 of LNCS. (2005) 5–19
6. Alur, R., Courcoubetis, C., Henzinger, T., Ho, P.H.: Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In: Hybrid systems. Volume 736 of LNCS., Springer-Verlag (1993) 209–229
7. Henzinger, T., Kopke, P., Puri, A., Varaiya, P.: What's decidable about hybrid automata? In: STOC, ACM Press (1995) 373–382
8. Henzinger, T.: The theory of hybrid automata. In: Proc. 11th IEEE Symp. on Logic in Computer Science, DIMACS (1996) 278–292
9. Lafferriere, G., Pappas, G., Yovine, S.: A new class of decidable hybrid systems. In: Hybrid Systems: Computation and Control. Volume 1569 of LNCS. (1999) 137–151

10. Puri, A.: Theory of Hybrid Systems and Discrete Event Systems. PhD thesis, University of California, Berkeley (1995)
11. Tomlin, C.J., Mitchell, I., Bayen, A., Oishi, M.: Computational techniques for the verification and control of hybrid systems. Proc. of IEEE **91**(7) (2003) 986–1001
12. Chutinan, C., Krogh, B.H.: Computational techniques for hybrid system verification. IEEE Transactions on Automatic Control **48**(1) (2003) 64–75
13. Silva, B., , Stursberg, O., Krogh, B., Engell, S.: An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In: IEEE Conf. on Decision and Control. Volume 3. (2001) 2867–2874
14. Esposito, J.M., Kim, J., Kumar, V.: Adaptive RRTs for validating hybrid robotic control systems. In: WAFR, Zeist, Netherlands (2004) 107–132
15. Kim, J., Esposito, J.M., Kumar, V.: An RRT-based algorithm for testing and validating multi-robot controllers. In: RSS, Boston, MA (2005) 249–256
16. Copty, F., Fix, L., Fraer, R., Giunchiglia, E., Kamhi, G., Tacchella, A., Vardi, M.: Benefits of bounded model checking at an industrial setting. In: Computer Aided Verification. Volume 2102 of LNCS., Springer-Verlag (2001) 436–453
17. Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L.E., Thrun, S.: Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press, Cambridge, MA (2005)
18. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge, MA (2006)
19. LaValle, S.M., Kuffner, J.J.: Rapidly-exploring random trees: Progress and prospects. In Donald, B.R., Lynch, K., Rus, D., eds.: WAFR. (2000) 293–308
20. Hsu, D., Kindel, R., Latombe, J.C., Rock, S.: Randomized kinodynamic motion planning with moving obstacles. IJRR **21**(3) (2002) 233–255
21. Plaku, E., Bekris, K.E., Chen, B.Y., Ladd, A.M., Kavraki, L.E.: Sampling-based roadmap of trees for parallel motion planning. IEEE Trans. on Robotics **21**(4) (2005) 597–608
22. Ladd, A.M., Kavraki, L.E.: Motion planning in the presence of drift, underactuation and discrete system changes. In: RSS, Boston, MA (2005) 233–241
23. Bekris, K.E., Kavraki, L.E.: Greedy but safe replanning under kinodynamic constraints. In: IEEE ICRA, Rome, Italy (2007)
24. Plaku, E., Vardi, M.Y., Kavraki, L.E.: Discrete search leading continuous exploration for kinodynamic motion planning. In: RSS, Atlanta, GA (2007)
25. Kavraki, L.E., Švestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation **12**(4) (1996) 566–580
26. Plaku, E., Kavraki, L.E., Vardi, M.Y.: A motion planner for a hybrid robotic system with kinodynamic constraints. In: IEEE ICRA, Rome, Italy (2007)
27. Zhang, W.: State-space Search: Algorithms, Complexity, Extensions, and Applications. Springer Verlag, New York, NY (2006)
28. Biere, A., Cimatti, A., Clarke, E., Fujita, M., Zhu, Y.: Symbolic model checking using SAT procedures instead of BDDs. In: Proc. 36th Design Automation Conference, IEEE Computer Society (1999) 317–320
29. Edelkamp, S., Jabbar, S.: Large-scale directed model checking ltl. In: Inter. SPIN Work. on Model Checking Software. Volume 3925 of LNCS., Springer (2006) 1–18
30. Burch, J., Clarke, E., McMillan, K., Dill, D., Hwang, L.: Symbolic model checking: 10^{20} states and beyond. Information and Computation **98**(2) (1992) 142–170
31. Ladd, A.M.: Motion Planning for Physical Simulation. PhD thesis, Rice University, Houston, TX (2006)