Test Coverage for Continuous and Hybrid Systems

Tarik Nahhal and Thao Dang

VERIMAG, 2 avenue de Vignate
38610 Gières, France

Abstract. We propose a novel test coverage measure for continuous and hybrid systems, which is defined using the star discrepancy notion. We also propose a test generation method guided by this coverage measure. This method was implemented in a prototype tool that can handle high dimensional systems (up to 100 dimensions).

1 Introduction

Hybrid systems have been recognized as a high-level model appropriate for embedded systems, since this model can describe, within a unified framework, the logical part and the continuous part of an embedded system. Due to the gap between the capacity of exhaustive formal verification methods and the complexity of embedded systems, testing is still the most commonly-used validation method in industry. Its success is probably due to the fact that testing suffers less from the 'state explosion' problem. Indeed, the engineer can choose the 'degree of validation' by the number of tests. In addition, this approach can be applied to the real system itself and not only to its model. Generally, testing of a reactive system is carried out by controlling the inputs and checking whether its behavior is as expected. Since it is impossible to enumerate all the admissible external inputs to the hybrid system in question, much effort has been invested in defining and implementing notions of *coverage* that guarantee, to some extent, that the finite set of input stimuli against which the system is tested is sufficient for validating correctness. For discrete systems, specified using programming languages or hardware design languages, some syntactic coverage measures can be defined, like exercising every statement or transition, etc. In this work, we treat continuous and hybrid systems that operate in a metric space (typically $\mathbb{R}^n$) and where there is not much inspiration coming from the syntax to the coverage issue. On the other hand, the metric nature of the state space encourages more *semantic* notions of coverage, namely that all system trajectories generated by the input test patterns form a kind of dense network in the reachable state space without too many big unexplored 'holes'.

In this work we adopt a model-based testing approach. This approach allows the engineer to perform validation during the design, where detecting and correcting errors on a model are less expensive than on an implementation. The main contributions of the paper can be summarized as follows. We propose a

W. Damm and H. Hermanns (Eds.): CAV 2007, LNCS 4590, pp. 449–462, 2007.

test coverage measure for hybrid systems, which is defined using the star discrepancy notion from statistics. This coverage measure is used to quantify the validation 'completeness'. It is also used to guide input stimulus generation by identifying the portions of the system behaviors that are not adequately examined. We propose an algorithm for generating tests from hybrid systems models, which is based on the RRT (Rapidly-exploring Random Tree) algorithm [8] from robotic motion planning and guided by the coverage measure. The rest of the paper is organized as follows. We first describe our test coverage measure and its estimation. We then present our test generation algorithm. In Section 5 we describe an implementation of the algorithm and some experimental results. Before concluding, we discuss related work.

2 Testing Problem

As a model for hybrid systems, we use hybrid automata. Note that a continuous system can be modeled as a hybrid automaton with only one discrete state. A hybrid automaton is an automaton augmented with continuous variables that evolve according to some differential equations. Formally, a *hybrid automaton* is a tuple $\mathcal{A} = (\mathcal{X}, Q, F, \mathcal{I}, \mathcal{G}, \mathcal{R})$ where $\mathcal{X} \subseteq \mathbb{R}^n$ is the continuous state space; Q is a (finite) set of locations (or discrete states); $E \subseteq Q \times Q$ is a set of discrete transitions; $F = \{F_q \mid q \in Q\}$ is a set of continuous vector fields such that for each $q \in Q$, $F_q = (U_q, f_q)$ where $U_q \subset \mathbb{R}^p$ is a set of inputs and $f_q : \mathbb{R}^n \times U_q \to \mathbb{R}^n$; $\mathcal{I} = \{\mathcal{I}_q \subseteq \mathbb{R}^n \mid q \in Q\}$ is a set of staying conditions; $\mathcal{G} = \{\mathcal{G}_e \mid e \in E\}$ is a set of guards such that for each discrete transition $e = (q, q') \in E$, $\mathcal{G}_e \subseteq \mathcal{I}_q$; $\mathcal{R} = \{\mathcal{R}_e \mid e \in E\}$ is a set of reset maps. For each $e = (q, q') \in E$, $\mathcal{R}_e : \mathcal{G}_e \to 2^{\mathcal{I}_{q'}}$ defines how x may change when $\mathcal{A}$ switches from q to q'. A *hybrid state* is a pair (q, x) where $q \in Q$ and $x \in \mathcal{X}$ and the hybrid state space is $\mathcal{S} = Q \times \mathcal{X}$. In location q, the evolution of the continuous variables is governed by $\dot{x}(t) = f_q(x(t), u(t))$. We assume that all f_q are Lipschitz continuous[1]. The admissible input functions $u(\cdot)$ are piecewise continuous. We denote the initial state of the automaton by (q_0, x_0). A state (q, x) of $\mathcal{A}$ can change in two ways as follows. By *continuous evolution*, the continuous state x evolves according to the dynamics f_q while the discrete state q remains constant. By *discrete evolution*, x satisfies the guard condition of an outgoing transition, the system changes the location by taking this transition and possibly changing the values of x according to the associated reset map. We assume that discrete transitions are instantaneous. It is important to note that this model allows to capture non-determinism in both continuous and discrete dynamics. This non-determinism is useful for describing disturbances from the environment and imprecision in modelling and implementation. The hybrid automata we consider are assumed to be non-blocking and non-Zeno.

A system under test often operates within some environment. In our testing problem, the tester plays the role of the environment. Given a hybrid automaton

[1] This ensures the existence and uniqueness of solutions of the differential equations.

modeling the behavior of the system under test, the tester can have the following controls on the system: first, it can control all the continuous inputs of the system; second, it can decide whether the system should take a given transition (among the enabled ones) or continue with the same continuous dynamics. Indeed, for simplicity of explanation, we do not include the control of the tester over the non-determinism in the reset maps. We also assume that the state of the system can be *fully observed* by the tester. Since we want to implement the tester as a computer program, we could assume that the continuous input functions generated by the tester are piecewise-constant with a fixed period h (i.e. they can change their values only after a fixed period of time), and h is called the *time step*. Hence, there are two types of control actions the tester can perform: continuous and discrete. A *continuous control action* denoted by a continuous dynamics and the value of the corresponding input, such as (f_q, v_q). It specifies that the system continues with the dynamics f_q under the input $u(t) = v_q$ for exactly h time. A *discrete control action* specifies a discrete transition to be taken by the system. We denote an action of this type by the corresponding transition, such as (q, q'). For a continuous action (f_q, v_q), we define its 'associated' transition as (q, q). A sequence of control actions is called *admissible* if after replacing all the continuous actions by their associated transitions, it corresponds to a path (i.e. a sequence of consecutive transitions) in the hybrid automaton $\mathcal{A}$ augmented with a self-loop at every location.

Definition 1 (Test case). *A test case is an admissible sequence of control actions $a_1, a_2, a_3, \ldots$ which is coherent with the initial state of the system, that is if $a_1 = (f_q, v_q)$ then $q = q_0$ and if $a_1 = (q, q')$ then $q = q_0$.*

Our testing problem can thus be stated as to automatically generate a set of test cases from the system model to satisfy a coverage criterion that we formally define in the following.

Test Coverage. Test coverage is a way to evaluate testing quality. More precisely, it is a way to relate the number of tests to carry out with the fraction of the system's behaviors effectively explored. As mentioned earlier, the classic coverage notions mainly used in software testing, such as statement coverage and if-then-else branch coverage, path coverage (see for example [16,14]), are not appropriate for the trajectories of continuous and hybrid systems defined by differential equations. However, geometric properties of the hybrid state space can be exploited to define a coverage measure which, on one hand, has a close relationship with the properties to verify and, on the other hand, can be efficiently computed or estimated. In this work, we are interested in state coverage and focus on a measure that describes how 'well' the visited states represent the reachable set of the system. This measure is defined using the *star discrepancy* notion in statistics, which characterises the uniformity of the distribution of a point set within a region. We first briefly recall the star discrepancy. The star discrepancy is an important notion in equidistribution theory as well as in quasi-Monte Carlo techniques (see for example [1]). Recently, it was also used in probabilistic motion planning to enhance the sampling uniformity [3].

Star Discrepancy. Let P be a set of k points inside $\mathcal{B} = [l_1, L_1] \times \ldots \times [l_n, L_n]$. Let Γ be the set of all sub-boxes J of the form $J = \prod_{i=1}^{n} [l_i, \beta_i]$ with $\beta_i \in [l_i, L_i]$ (see Figure 1 for an illustration). The local discrepancy of the point set P with respect to the subbox J is defined as follows: $D(P, J) = |\frac{A(P,J)}{k} - \frac{\lambda(J)}{\lambda(\mathcal{B})}|$ where $A(P, J)$ is the number of points of P that are inside J, and $\lambda(J)$ is the volume of the box J. The star discrepancy of P with respect to the box $\mathcal{B}$ is defined as:

$$D^*(P, \mathcal{B}) = sup_{J \in \Gamma} D(P, J) \tag{1}$$

Note that $0 < D^*(P, \mathcal{B}) \leq 1$. Intuitively, the star discrepancy is a measure for

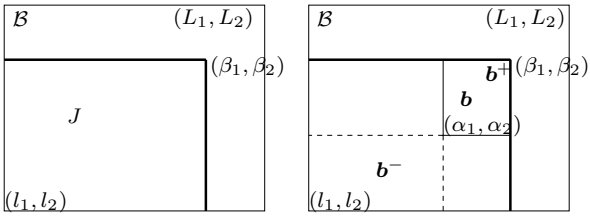

Fig. 1. Illustration of the star discrepancy notion

the irregularity of a set of points. A large value $D^*(P, \mathcal{B})$ means that the points in P are not much equidistributed over $\mathcal{B}$. When the region is a hyper-cube, the star discrepancy measures how badly the point set estimates the volume of the cube. Since a hybrid system can only evolve within the staying sets of the locations, we are interested in the coverage with respect to these sets. For simplicity we assume that all the staying sets are boxes.

Definition 2 (Hybrid System Test Coverage). *Let $\mathcal{P} = \{(q, P_q) \mid q \in Q \wedge P_q \subset \mathcal{I}_q\}$ be the set of states. The coverage of $\mathcal{P}$ is defined as: $Cov(\mathcal{P}) = \frac{1}{||Q||} \sum_{q \in Q} 1 - D^*(P_q, \mathcal{I}_q)$ where $||Q||$ is the number of locations in Q.*

If a staying set $\mathcal{I}_q$ is not a box, we can take the smallest oriented box that encloses it, and apply the star discrepancy definition in (1) to that box after an appropriate coordination change. We can see that a large value of $Cov(\mathcal{P})$ indicates a good space-covering quality. If $\mathcal{P}$ is the set of states visited by a set of test cases, our objective is to maximize $Cov(\mathcal{P})$.

3 Test Generation

Our test generation is based on a randomized exploration of the reachable state space of system. It is inspired by the Rapidly-exploring Random Tree (RRT) algorithm, which is a successful motion planning technique for finding feasible trajectories of robots in an environment with obstacles (see [8] for a survey). More

precisely, we extend the RRT algorithm to hybrid systems and combine it with a guiding tool in order to achieve a good coverage of the system's behaviors we want to test. In this context, we use the coverage measure defined in the previous section. Some preliminary results for continuous systems were described in [12].

The visited states are stored in a tree $\mathcal{T}$, the root of which corresponds to the initial state. The construction of the tree is summarized in Algorithm 1. In each iteration, a hybrid state $s_{goal} = (q_{goal}, x_{goal})$ is sampled to indicate the direction towards which the tree is expected to evolve. Expanding the tree towards s_{goal} is done by making a continuous step as follows:

- First, a starting state $s_{init} = (q_{init}, x_{init})$ for the current iteration is determined. It is natural to choose s_{init} to be a state near s_{goal}. The definition of the distance between hybrid states will be given later.
- Next, the procedure ContinuousStep tries to find the input $u_{q_{init}}$ such that, after one time step h, the current continuous dynamics at q_{init} takes the system from s_{init} towards s_{goal}, and this results in a new continuous state x_{new}. A new edge from s_{init} to $s_{new} = (q_{init}, x_{new})$, labeled with the associated input $u_{q_{init}}$, is then added to the tree.

Then, from s_{new}, we compute its successors by all possible discrete transitions. Each time we add a new edge, we label it with the associated control action. The algorithm terminates after reaching a satisfactory coverage value or after some maximal number of iterations. From the tree constructed by the algorithm we can then extract test cases. In addition, when applying such test cases to the system, the tree can be used to decide whether the system under test behaves as expected. In the classic RRT algorithms, which work in a continuous setting,

Algorithm 1. Test generation algotihm

$\mathcal{T}.init(s_0)$, $j = 1$ ▷ s_0: initial state
repeat
  $s_{goal} =$ Sampling($\mathcal{S}$) ▷ $\mathcal{S}$: hybrid state space
  $s_{init} =$ Neighbor($\mathcal{T}, s_{goal}$)
  $(s_{new}, u_{q_{init}}) =$ ContinuousStep(s_{init}, h) ▷ h: time step
  DiscreteSteps($\mathcal{T}, s_{new}$), $j++$
until $j \geq J_{max}$

only x_{goal} needs to be sampled, and a commonly used sampling distribution of x_{goal} is uniform over $\mathcal{X}$. In addition, the point x_{init} is defined as a nearest neighbor of x_{goal} in some usual distance, such as the Euclidian distance. In Algorithm 1, the function Sampling plays the role of guiding the exploration via a biased sampling of x_{goal}, which will be discussed in detail later. The computation of discrete successors in DiscreteSteps, which involves testing a guard condition and applying a reset map, is straightforward. In the following, we show how to compute the functions Neighbor and ContinuousStep.

Finding a Neighbor. To define the distance between two hybrid states, we first define the average length of a path. Given a path of two transitions $e = (q, q')$ and $e' = (q', q'')$, let $\sigma(e, e') = \overline{d}(\mathcal{R}_{(q,q')}(\mathcal{G}_{(q',q'')}), \mathcal{G}_{(q',q'')})$ where $\overline{d}$ is the average distance between two sets defined as the Euclidian distance between their centroids. Given a path $\gamma = e_1, e_2, \dots e_m$ where $e_i = (q_i, q_{i+1})$, we define its average length as: $len(\gamma) = \sum_{i=1}^{m-1} \sigma(e_i, e_{i+1})$. Note that from one location to another, there can be more than one path. Let $\Gamma(q, q')$ be the set of all paths from q to q'. Given two hybrid states $s = (q, x)$ and $s' = (q', x')$, if $q = q'$, we define the *hybrid distance* $d_H(s, s')$ from s to s' as the Euclidian distance between the continuous states x and x': $d_H(s, s') = ||x - x'||$. If $q \neq q'$,

$$d_H(s, s') = \begin{cases} \min\limits_{\gamma \in \Gamma(q,q')} \overline{d}(x, fG(\gamma)) + len(\gamma) + \overline{d}(x', lR(\gamma)), & \text{if } \Gamma(q, q') \neq \emptyset \\ \infty & \text{otherwise.} \end{cases}$$

where $fG(\gamma) = \mathcal{G}_{(q_1,q_2)}$, the first guard of γ, and $lR(\gamma) = \mathcal{R}_{(q_k,q_{k+1})}(\mathcal{G}_{(q_k,q_{k+1})})$, that is the set resulting from applying the reset map of the last transition to its guard set. Intuitively, $d_H(s, s')$ is obtained by adding to the average length of γ the distance from x to the first guard and the distance from the last 'reset set' to x'. This distance can be thought of as an average length of the trajectories from s to s'. The function NEIGHBOR can thus be computed using this hybrid distance as follows: $s_{init} = argmin_{s \in V} d_H(s_{goal}, s)$ where V is the set of states stored in the tree.

Continuous Step. If the states s_{init} and s_{goal} have the same discrete location component, we want to expand to tree from x_{init} towards x_{goal} as closely as possible. Otherwise, let γ be the path from q_{init} to q_{goal} with the shortest average length, we want to steer the system from x_{init} towards the first guard of γ. In both cases, this is an optimal control problem with the objective of minimizing the distance to some target point. This problem is difficult especially for systems with non-linear continuous dynamics. Thus, we can trade some optimality for computational efficiency. When the input set is not finite, we can sample a set of input values and pick from this set an optimal input. In addition, we can prove that by an appropriate sampling of the input set, the completeness property of the RRT algorithm is preserved [2].

Coverage Estimation. To evaluate the coverage of a set of states, we need to compute the star discrepancy of a point set, which is not an easy problem (see for example [7]). Many theoretical results for one-dimensional point sets are not generalizable to higher dimensions, and among the fastest algorithms we can mention the one proposed in [7] of time complexity $\mathcal{O}(k^{1+n/2})$. In this work, we do not try to compute the star discrepancy but approximate it by estimating a lower and upper bound. These bounds as well as the information obtained from their estimation are then used to decide which parts of the state space have been 'well explored' and which parts need to be explored more. This estimation is done using a method published in [10]. Let us briefly describe this method for computing the star discrepancy $D^*(P, \mathcal{B})$ of a point set P w.r.t. a box $\mathcal{B}$.

Although in [10], the box $\mathcal{B}$ is $[0,1]^n$, we extended it to the case where $\mathcal{B}$ can be any full-dimensional box. Let $\mathcal{B} = [l_1, L_1] \times \ldots \times [l_n, L_n]$. We define a box partition of $\mathcal{B}$ as a set of boxes $\Pi = \{\boldsymbol{b}^1, \ldots, \boldsymbol{b}^m\}$ such that $\cup_{i=1}^m \boldsymbol{b}^i = \mathcal{B}$ and the interiors of the boxes $\boldsymbol{b}^i$ do not intersect. Each such box is called an *elementary box*. Given a box $\boldsymbol{b} = [\alpha_1, \beta_1] \times \ldots \times [\alpha_n, \beta_n] \in \Pi$, we define $\boldsymbol{b}^+ = [l_1, \beta_1] \times \ldots \times [l_n, \beta_n]$ and $\boldsymbol{b}^- = [l_1, \alpha_1] \times \ldots \times [l_n, \alpha_n]$ (see Figure 1 for an illustration).

For any finite box partition Π of $\mathcal{B}$, the star discrepancy $D^*(P, \mathcal{B})$ of the point set P with respect to $\mathcal{B}$ satisfies: $C(P, \Pi) \leq D^*(P, \mathcal{B}) \leq B(P, \Pi)$ where the upper and lower bounds are:

$$B(P, \Pi) = \max_{\boldsymbol{b} \in \Pi} \max\{\frac{A(P, \boldsymbol{b}^+)}{k} - \frac{\lambda(\boldsymbol{b}^-)}{\lambda(\mathcal{B})},\ \frac{\lambda(\boldsymbol{b}^+)}{\lambda(\mathcal{B})} - \frac{A(P, \boldsymbol{b}^-)}{k}\} \tag{2}$$

$$C(P, \Pi) = \max_{\boldsymbol{b} \in \Pi} \max\{|\frac{A(P, \boldsymbol{b}^-)}{k} - \frac{\lambda(\boldsymbol{b}^-)}{\lambda(\mathcal{B})}|,\ |\frac{A(P, \boldsymbol{b}^+)}{k} - \frac{\lambda(\boldsymbol{b}^+)}{\lambda(\mathcal{B})}|\} \tag{3}$$

The imprecision of this approximation is the difference between the upper and lower bounds, which can be bounded by $B(P, \Pi) - C(P, \Pi) \leq W(\Pi)$ where follows:

$$W(\Pi) = \max_{\boldsymbol{b} \in \Pi} (\lambda(\boldsymbol{b}^+) - \lambda(\boldsymbol{b}^-))/\lambda(\mathcal{B}) \tag{4}$$

Thus, one needs to find a partition Π such that this difference is small.

Coverage-Guided Sampling. We show how to use the estimation of the coverage measure to derive a guiding strategy. Recall that our goal is to achieve a good testing coverage quality, which is equivalent to a small value of the star discrepancy of the points visited at each discrete location. More concretely, in each iteration, we want to bias the goal state sampling distribution according to the current coverage of the visited states. To do so, we first sample a discrete location and then a continuous state. Let $\mathcal{P} = \{(q, P_q) \mid q \in Q \wedge P_q \subset \mathcal{I}_q\}$ be the current set of visited states. The discrete location sampling distribution depends on the current continuous state coverage of each location:

$$Pr[q_{goal} = q] = \frac{D^*(P_q, \mathcal{I}_q)}{\sum_{q' \in Q} D^*(P_{q'}, \mathcal{I}_{q'})}.$$

We now show how to sample x_{goal}, assuming that we have already sampled a discrete location $q_{goal} = q$. In the remainder of the paper, to give geometric intuitions, we often call a continuous state a point. In addition, since all the staying sets are assumed to be boxes, we denote the staying set $\mathcal{I}_q$ by the box $\mathcal{B}$ and denote the current set of visited points at location q simply by P instead of P_q. Let k be the number of points in P. Let Π be a finite box partition of $\mathcal{B}$ that is used to estimate the star discrepancy of P. The sampling process consists of two steps. In the first step, we sample an elementary box $\boldsymbol{b}_{goal}$ from the set Π; in the second step we sample a point x_{goal} in $\boldsymbol{b}_{goal}$ uniformly. The elementary box sampling distribution in the first step is biased in order to optimize the coverage. The intuition behind this guiding strategy is to favor the selection of

an elementary box such that a new point x added in this box results in a smaller star discrepancy of the new point set $P \cup \{x\}$. The strategy is determined so as to reduce both the lower bound $C(P, \Pi)$ and the upper bound $B(P, \Pi)$.

Reducing the lower bound. We associate with each box $\mathcal{B} \subseteq \Pi$ a number $A^*(\boldsymbol{b})$ such that $\dfrac{\lambda(\boldsymbol{b})}{\lambda(\mathcal{B})} = \dfrac{A^*(\boldsymbol{b})}{k}$. We denote $\Delta_A(\boldsymbol{b}) = \frac{1}{k}(A(P, \boldsymbol{b}) - A^*(\boldsymbol{b}))$. Denote $c(\boldsymbol{b}) = \max\{|\Delta_A(\boldsymbol{b}^+)|, |\Delta_A(\boldsymbol{b}^-)|\}$, and the lower bound of the star discrepancy of the point set P over the bounding box $\mathcal{B}$ becomes: $C(P, \Pi) = \max_{\boldsymbol{b} \in \Pi}\{c(\boldsymbol{b})\}$. Note that in comparison with A^*, the negative (respectively positive) sign of $\Delta_A(\boldsymbol{b})$ indicates that in this box there is a lack (respectively an excess) of points; its absolute value indicates how significant the lack (or the excess) is. We now compare the values of $|\Delta_A(\boldsymbol{b}^+)|$ in two cases: the newly added point x_{new} is in $\boldsymbol{b}$ and x_{new} is not in $\boldsymbol{b}$. If $\Delta_A(\boldsymbol{b}^+)$ is positive, the value of $|\Delta_A(\boldsymbol{b}^+)|$ in the former case is smaller than or equal to that in the latter case; otherwise it is greater than or equal to. However, adding a new point in $\boldsymbol{b}$ does not affect the values of $A(P, \boldsymbol{b}^-)$ (see Figure 1). Thus, we define a function reflecting the potential influence on the lower bound as follows:

$$\xi(\boldsymbol{b}) = \frac{1 - \Delta_A(\boldsymbol{b}^+)}{1 - \Delta_A(\boldsymbol{b}^-)}, \tag{5}$$

and we favor the selection of $\boldsymbol{b}$ if the value $\xi(\boldsymbol{b})$ is large. Note that $1 - \Delta_A(\boldsymbol{b}) > 0$ for any box $\boldsymbol{b}$ inside $\mathcal{B}$. The intepretation of ξ is as follows. If $\Delta_A(\boldsymbol{b}^+)$ is negative and its absolute value is large, the 'lack' of points in $\boldsymbol{b}^+$ is significant. In this case, $\xi(\boldsymbol{b})$ is large, meaning that the selection of $\boldsymbol{b}$ is favored. On the other hand, if $\Delta_A(\boldsymbol{b}^-)$ is negative and its absolute value is large, then $\xi(\boldsymbol{b})$ is small, because it is preferable not to select $\boldsymbol{b}$ in order to increase the chance of adding new points in $\boldsymbol{b}^-$.

Reducing the upper bound. The upper bound in (2) can be rewritten as

$$B(P, \Pi) = \max_{\boldsymbol{b} \in \Pi} f_m(\boldsymbol{b}) \tag{6}$$

where $f_m(\boldsymbol{b}) = \max\{f_c(\boldsymbol{b}), f_o(\boldsymbol{b})\}$ and $f_c(\boldsymbol{b}) = \frac{1}{k}(A(P, \boldsymbol{b}^+) - A^*(\boldsymbol{b}^-))$ and $f_o(\boldsymbol{b}) = \frac{1}{k}(A^*(\boldsymbol{b}^+) - A(P, \boldsymbol{b}^-))$. Since the value of f_m is determined by comparing f_c with f_o. After straightforward calculations, the inequality $f_c(\boldsymbol{b}) - f_o(\boldsymbol{b}) \leq 0$ is equivalent to $f_c(\boldsymbol{b}) - f_o(\boldsymbol{b}) = \Delta_A(P, \boldsymbol{b}^+) + \Delta_A(P, \boldsymbol{b}^-) \leq 0$. Therefore,

$$f_m(\boldsymbol{b}) = \begin{cases} f_o(\boldsymbol{b}) & \text{if } \Delta_A(\boldsymbol{b}^+) + \Delta_A(\boldsymbol{b}^-) \leq 0, \\ f_c(\boldsymbol{b}) & \text{otherwise.} \end{cases} \tag{7}$$

Again, the value of $f_c(\boldsymbol{b})$ when the new point x_{new} is added in $\boldsymbol{b}$ is larger than that when x_{new} is not in $\boldsymbol{b}$, but the fact that x_{new} is in $\boldsymbol{b}$ does not affect $f_o(\boldsymbol{b})$. To reduce $f_o(\boldsymbol{b})$ we need to add points in $\boldsymbol{b}^-$. Hence, if $\boldsymbol{b}$ is a box in Π that maximizes f_m in (6), it is preferable not to add more points in $\boldsymbol{b}$ but in the boxes where the values of f_m are much lower than the current value of $B(P, \Pi)$ (in particular

those inside $\boldsymbol{b}^-$). Using the same reasoning for each box $\boldsymbol{b}$ locally, the smaller $|\Delta_A(P, \boldsymbol{b}^+)+\Delta_A(P, \boldsymbol{b}^-)|$ is, the smaller sampling probability we give to $\boldsymbol{b}$. Indeed, as mentionned earlier, if $f_m(\boldsymbol{b}) = f_c(\boldsymbol{b})$, increasing $f_c(\boldsymbol{b})$ directly increases $f_m(\boldsymbol{b})$. On the other hand, if $f_m(\boldsymbol{b}) = f_o(\boldsymbol{b})$, increasing $f_c(\boldsymbol{b})$ may make it greater than $f_o(\boldsymbol{b})$ and thus increase $f_m(\boldsymbol{b})$, because small $|\Delta_A(P, \boldsymbol{b}^+) + \Delta_A(P, \boldsymbol{b}^-)|$ implies that $f_c(\boldsymbol{b})$ is close to $f_o(\boldsymbol{b})$.

We define two functions reflecting the global and local potential influences on the upper bound: $\beta_g(\boldsymbol{b}) = B(P, \Pi) - f_m(\boldsymbol{b})$ and $\beta_l(\boldsymbol{b}) = \beta_g(\boldsymbol{b})|\Delta_A(P, \boldsymbol{b}^+) + \Delta_A(P, \boldsymbol{b}^-)|$. We can verify that $\beta_g(\boldsymbol{b})$ and $\beta_l(\boldsymbol{b})$ are always positive. Now, combining these functions with ξ in (5) that describes the potential influence on the lower bound, we define: $\kappa(\boldsymbol{b}) = \gamma_\xi \xi(\boldsymbol{b})+\gamma_g \beta_g(\boldsymbol{b})+\gamma_l \beta_l(\boldsymbol{b})$ where γ_ξ, γ_g, and γ_l are non-negative weights that can be user-defined parameters. Then, the probability of choosing the box $\boldsymbol{b}$ can be defined as follows $Pr[\boldsymbol{b}_{goal} = \boldsymbol{b}] = \dfrac{\kappa(\boldsymbol{b})}{\sum_{\boldsymbol{b}\in\Pi} \kappa(\boldsymbol{b})}$.

4 Implementation

In addition to the tree that is used to store the explored executions, to facilitate the computation of geometric operations, such as finding a neighbor, we store the points reachable by the dynamics at each location using a data structure similar to a k-d tree. Each node of the tree has exactly two children. Each internal node is associated with the information about a partitioning plane: its axis i and position c, and the partitioning plane is thus $x_i = c$ (where x_i is the i^{th} coordinate of x). The additional information associated with a leaf is a set of visited points. Each node thus corresponds to an elementary box resulting from a hierarchical box-partition of the state space. The box of the root of the tree is $\mathcal{B}$. The tree and the partition of a 2-dimensional example is shown in Figure 2, where the axes of the partitioning planes are specified by the horizontal and vertical bars inside the nodes.

Approximate Neighbors. Since the computation of exact nearest neighbors is expensive (even in a continuous setting), we approximate a neigbor of x as follows: find the elementary box $\boldsymbol{b}$ which contains at least one visited point and, in addition, is closest to x (note that some elementary boxes may not contain any visited points). Then, we find a point in $\boldsymbol{b}$ which is closest to x. It is easy to see that $\boldsymbol{b}$ does not necessarily contain a nearest neighbor of x. We use this approximation because, on one hand the sampling distribution reflects the boxes we want to explore, and on the other hand, it has lower complexity w.r.t. dimensions. In addition, as we will show later, this approximation preserves the completeness.

Update the discrepancy estimation. After adding a new point x, we need to update the estimation of the star discrepancy. More concretely, we need to find all the elementary boxes $\boldsymbol{b}$ such that the new point has increased the number of points in the corresponding boxes $\boldsymbol{b}^-$ and $\boldsymbol{b}^+$. These boxes are indeed those which intersect with the box $B_x = [x_1, L_1] \times \ldots \times [x_n, L_n]$. In addition, if $\boldsymbol{b}$ is a subset of B_x, the numbers of points in both $\boldsymbol{b}^+$ and $\boldsymbol{b}^-$ need to be incremented;

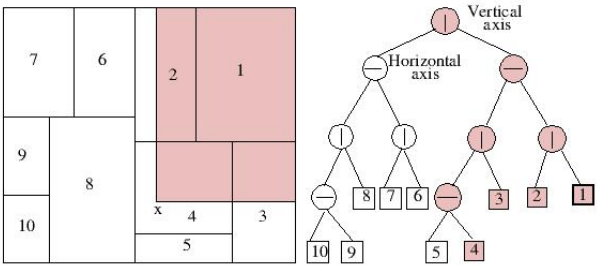

Fig. 2. Illustration of the update of the star discrepancy estimation

if $\boldsymbol{b}$ intersects with B_x but is not entirely inside B_x, only the number of points in $\boldsymbol{b}^+$ needs to be incremented. Searching for all the elementary boxes that are affected by x can be done by traversing the tree from the root and visiting all the nodes the boxes of which intersect with B_x. In the example of Figure 2, the box B_x is the dark rectangle, and the nodes of the trees visited in this search are drawn as dark cirles.

Box splitting. When the difference between the lower and upper bounds in the star discrepancy estimation is large, some boxes need to be split as indicated by (4). Additionally, splitting is also needed for efficiency of the neighbor computation.

Completeness Property. Probabilistic completeness is an important property of the RRT algorithm. Roughly speaking, it states that if the trajectory we seach for is feasible, then the probability that the algorithm finds it approaches 1 as the number k of iterations approaches infinity [8]. Although this property is mainly of theoretical interest, it is a way to explain a good space-covering property and the success of the RRT algorithm in solving practical robotic motion planning problems. We can prove that our test generation algorithm preserves this completeness property.

Theorem 1. [Reachability completeness] *Let V^k be the set of states visited after k iterations of Algorithm 1. Given $\varepsilon > 0$ and a* **reachable** *state (q, x), the probability that there exists a state $(q, x') \in V^k$ such that $||x - x'|| \leq \varepsilon$ approaches 1 when k approaches infinity, i.e. $lim_{k\to\infty} Pr[\exists (q, x') \in V^k : ||x - x'|| \leq \varepsilon] = 1$.*

Sketch of Proof. The proofs of the completeness of RRTs are often established for the algorithms where the goal point sampling distribution is uniform and all the operations are exactly computed (see for example [8]). We first identify the following condition: at each iteration k, $\forall s \in V^k : Pr[s_{init} = s] > 0$. We can prove that this condition is sufficient for the completeness proofs to remain valid, even when the sampling distribution is non-uniform and the operations are not exactly computed. The proof of this is rather technical and thus omitted (see [2]). We now give a sketch of proof that our guided sampling method and nearest neighbor approximation satisfy this condition. We first observe that,

both the location and elementary box sampling distributions guarantee that all the locations and all the boxes have non-null probability of being selected. We consider only the case where the elementary box $\boldsymbol{b}$ within which we search for a neighbor of x_{goal} contains x_{goal} (the other case can be handled similarly). Let $P_{\boldsymbol{b}}$ be the set of visited points that are inside $\boldsymbol{b}$. Let $\mathcal{V}_{\boldsymbol{b}}$ be the Voronoi diagram of $P_{\boldsymbol{b}}$ restricted to $\boldsymbol{b}$ and $\mathcal{C}_p$ the corresponding Voronoi cell of a visited point $p \in P_{\boldsymbol{b}}$. (Recall that the Voronoi cell of a point p is the set of all points that are closer to p than to any other point). We can prove that the volume of $\mathcal{C}_p$ is strictly positive. Since the goal point sampling distribution within $\boldsymbol{b}$ is uniform, $Pr[x_{goal} \in \mathcal{C}_p] > 0$, and hence the probability that p is the approximate neighbor is also positive. It then follows that any visited point has a positive probability of being selected to be x_{init}. This implies that at each iteration, any visited state has a positive probability to be s_{init}. ■

5 Experimental Results

We implemented the test generation algorithm using C++ in a prototype tool, and the results reported here were obtained by running the tool on a 1.4 GHz Pentium III. First, to demonstrate the performance of our algorithm, we use a set of examples of linear systems $\dot{x} = Ax + u$ in various dimensions. In this experiment, we did not exploit the linearity of the dynamics and the tested systems were randomly generated: the matrix A is in Jordan canonical form, each diagonal value of which is randomly chosen from $[-3, 3]$ and the input set U contains 100 values randomly chosen from $[-0.5, 0.5]^n$. We fix a maximal number $K_{max} = 50000$ of visited states. In terms of coverage, the star discrepancy of the results obtained by our algorithm and the classic RRT algorithm are shown in Table 1 (left), which indicates that our algorithm has better coverage quality. These discrepancy values were computed for the final set of visited states, using a partition optimal w.r.t. to the imprecision bound in (4). Note that in each iteration of our test generation algorithm we do not compute such a partition because it is very expensive. The results obtained on a 2-dimensional system are visualized in Figure 3. Table 1 (right) shows the time efficiency of our algorithm for linear systems of dimensions up to 100.

To illustrate the application of our algorithm to hybrid systems, we use the well-known aircraft collision avoidance problem [11]. The dynamics of each

Table 1. Discrepancy results and computation time for some linear systems

dim n	Lower bound		Upper bound	
	Algo 1	RRT	Algo 1	RRT
3	0.451	0.546	0.457	0.555
5	0.462	0.650	0.531	0.742
10	0.540	0.780	0.696	0.904

dim n	Time (min)
5	1
10	3.5
20	7.3
50	24
100	71

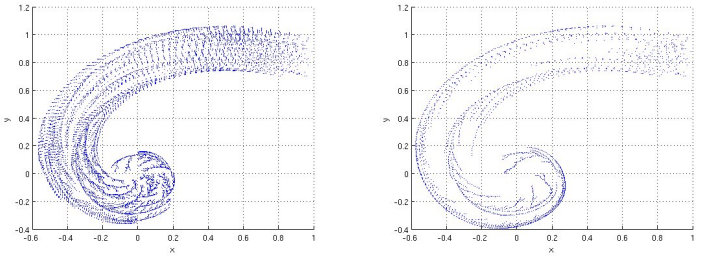

Fig. 3. Results obtained using Algorithm 1 (left) and the RRT algorithm (right)

aircraft is as follows: $\dot{x_i} = vcos(\theta_i) + d_1 sin(\theta_i) + d_2 cos(\theta_2)$, $\dot{y_i} = vsin(\theta_i) - d_1 cos(\theta_i) + d_2 sin(\theta_2)$, $\dot{\theta_i} = \omega$ where x_i, y_i describe the position and θ_i is the relative heading. The continuous inputs are d_1 and d_2 describing the external disturbances on the aircrafts and $-\delta \leq d_1, d_2 \leq \delta$. There are three discrete modes. At first, each aircraft begins in straight flight with a fixed heading (mode 1). Then, as soon as two aircrafts are within the distance between each other, they enter mode 2, at which point each makes an instantaneous heading change of 90 degrees, and begins a circular flight for π time units. After that, they switch to mode 3 and make another instantaneous heading change of 90 degrees and resume their original headings from mode 1. Thus for N aircrafts, the system has $3N + 1$ continuous variables (one for modeling a clock). For the case of $N = 2$ aircrafts, when the collision distance is 5 no colission was detected after visiting 10000 visited states, and the computation time was 0.9 min. The result for $N = 8$ aircrafts with the disturbance bound $\delta = 0.06$ is shown in Figure 4. For this example, the computation time for 50000 visited states was 10 min and a collision was found. For a similar example with $N = 10$ aicrafts, the computation time was 14 minutes and a collision was also found.

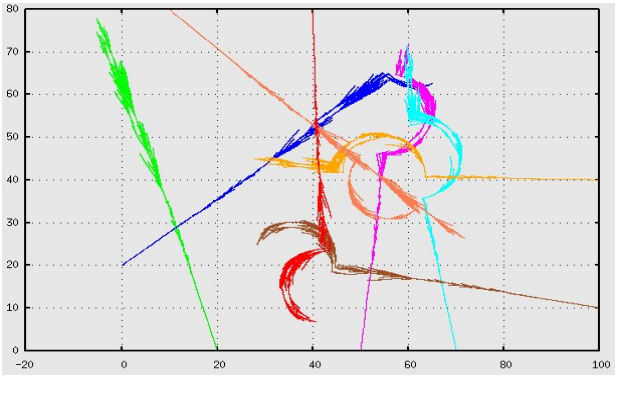

Fig. 4. Eight-aircraft collision avoidance (50000 visited states, computation time: 10 min)

6 Related Work and Conclusion

Classical model-based testing frameworks use Mealy machines or finite labeled transition systems and their applications include testing of digital circuits, communication protocols and software. Recently, these frameworks have been extended to real-time systems and hybrid systems. Here we only discuss related works in hybrid systems testing. The paper [13] proposed a framework for generating test cases from simulation of hybrid models specified using the language CHARON. A probabilistic test generation approach, similiar to ours, was presented in [5]. In this paper, the authors also proposed a coverage measure based on a discretized version of dispersion. This measure is defined over a set of grid points with a fixed size δ. The spacing s_g of a grid point g is the distance from g to the tree if it is smaller than δ, and $s_g = \delta$ otherwise. Let S be the sum of the spacings of all the grid points. This means that the value of S is the largest when the tree is empty. Then, the coverage measure is defined in terms of how much the vertices of the tree reduce the value of S. While in our work, the coverage measure is used to guide the simulation, in [5] it is used as a termination criterion. The RRT algorithms have also been used to solve other hybrid systems analysis problems such as hybrid systems control and verification [6,4]. Biasing the sampling process is guided by geometric constraints (such as avoiding sampling near the obstacles) in [15] and by the exploration history (to predict unreachable parts) in [5]. The difference which is also the novelty in our method for guiding test generation is that we use the information about the current coverage in order to improve it.

To conclude, in this paper we described a test coverage measure for continuous and hybrid systems and a test generation algorithm. The originality of our paper is a way to guide the test generation process by the coverage measure. The experimental results obtained using an implementation of the test generation algorithm show its scalability to high dimensional systems and good coverage quality. A number of directions for future research can be identified. First, we are interested in defining a measure for trace coverage. Partial observability also needs to be considered. Convergence rate of the exploration in the test generation algorithm is another interesting theoretical problem to tackle. This problem is particular hard especially in the verification context where the system is subject to uncontrollable inputs. Finally, we intend to apply the results of this research to validation of analog and mixed-signal circuits, a domain where testing is a widely used technique.

Acknowledgement. This research is supported by a fund ANR-06-SETI-018 of Agence Nationale de la Recherche.

References

1. Beck, J., Chen, W.: Irregularities of Distribution. Cambridge Univ. Press, Cambridge (1987)
2. Dang, T., Nahhal, T.: Randomized simulation of hybrid systems. In: Technical report, Verimag, IMAG (May 2006)

3. LaValle, S.M., Branicky, M.S., Lindemann, S.R.: On the relationship between classical grid search and probabilistic roadmaps. Intl. Journal of Robotics Research 23(7-8), 673–692 (2004)
4. Branicky, M.S., Curtiss, M.M., Levine, J., Morgan, S.: Sampling-based reachability algorithms for control and verification of complex systems. In: Proc. Thirteenth Yale Workshop on Adaptive and Learning Systems, New Haven, CT (2005)
5. Esposito, J.M., Kim, J., Kumar, V.: Adaptive RRTs for validating hybrid robotic control systems. In: Int. Workshop on the Algorithmic Founddations of Robotics (2004)
6. Bhatia, A., Frazzoli, E.: Incremental Search Methods for Reachability Analysis of Continuous and Hybrid Systems. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 142–156. Springer, Heidelberg (2004)
7. Dobkin, D., Eppstein, D.: Computing the discrepancy. In: Proceedings of the Ninth Annual Symposium on Computational Geometry, pp. 47–52 (1993)
8. LaValle, S.M., Kuffner, J.J.: Rapidly-exploring random trees: Progress and prospects. In: Algorithmic and Computational Robotics: New Directions, pp. 293–308. A K Peters, Wellesley, MA (2001)
9. Lee, D., Yannakakis, M.: Principles and Methods of Testing Finite State Machines - A Survey. In: Proceedings of the IEEE, vol. 84 (1996)
10. Thiémard, E.: An algorithm to compute bounds for the star discrepancy. J. Complexity 17(4), 850 (2001)
11. Mitchell, I., Tomlin, C.: Level Set Methods for Computation in Hybrid Systems. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, Springer, Heidelberg (2000)
12. Nahhal, T., Dang, T.: Guided randomized simulation. In: HSCC. LNCS, Springer, Heidelberg (2007)
13. Tan, L., Kim, J., Sokolsky, O., Lee, I.: Model-based Testing and Monitoring for Hybrid Embedded Systems. In: IRI, pp. 487–492 (2004)
14. Tretmans, J.: Testing Concurrent Systems: A Formal Approach. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, Springer, Heidelberg (1999)
15. Yershova, A., Jaillet, L., Simeon, T., LaValle, S.M.: Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In: Proc. IEEE International Conference on Robotics and Automation, IEEE Computer Society Press, Los Alamitos (2005)
16. Zhu, H., Hall, P.A.V., May, J.H.R.: Software Unit Test Coverage and Adequacy. In: ACM Computing Surveys, vol. 29(4), ACM Press, New York (1997)