Decoding Output Sequences for Discrete-Time Linear Hybrid Systems.

Monal Narasimhamurthy
first.lastname@colorado.edu
University of Colorado Boulder
Boulder, CO, USA

Sriram Sankaranarayanan
first.lastname@colorado.edu
University of Colorado Boulder
Boulder, CO, USA

ABSTRACT

In this paper, we study the "decoding" problem for discrete-time, stochastic hybrid systems with linear dynamics in each mode. Given an output trace of the system, the decoding problem seeks to construct a sequence of modes and states that yield a trace "as close as possible" to the original output trace. The decoding problem generalizes the state estimation problem, and is applicable to hybrid systems with non-determinism. The decoding problem is NP-complete, and can be reduced to solving a mixed-integer linear program (MILP). In this paper, we decompose the decoding problem into two parts: (a) finding a sequence of discrete modes and transitions; and (b) finding corresponding continuous states for the mode/transition sequence. In particular, once a sequence of modes/transitions is fixed, the problem of "filling in" the continuous states is performed by a linear programming problem. In order to support the decomposition, we "cover" the set of all possible mode/transition sequences by a finite subset. We use well-known probabilistic arguments to justify a choice of cover with high confidence and design randomized algorithms for finding such covers. Our approach is demonstrated on a series of benchmarks, wherein we observe that relatively tiny fraction of the possible mode/transition sequences can be used as a cover. Furthermore, we show that the resulting linear programs can be solved rapidly by exploiting the tree structure of the set cover.

CCS CONCEPTS

• **Mathematics of computing** → *Probability and statistics*; • **Computing methodologies** → *Model development and analysis*; *Computational control theory*.

KEYWORDS

Hybrid Systems, Decoding, Randomized Algorithms, Observer Design, State Estimation.

ACM Reference Format:
Monal Narasimhamurthy and Sriram Sankaranarayanan. 2022. Decoding Output Sequences for Discrete-Time Linear Hybrid Systems.. In *25th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '22), May 4–6, 2022, Milan, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3501710.3519530

1 INTRODUCTION

In this paper, given a sequence of possibly noisy output values from a hybrid automaton we consider the problem of reconstructing a sequences of modes, transitions and continuous states for a hybrid automaton whose output is as close as possible to the given input sequence. Inspired by a similar problem of decoding for Hidden Markov models (HMMs) that involves constructing the most likely sequence of latent states given an output sequence, we call this the *decoding problem* for hybrid automata [20]. Specifically, we consider hybrid systems whose modes have discrete-time linear dynamics with polyhedral transition guards and linear updates on transitions. This class of hybrid automata has been shown to be very useful in modeling a variety of important applications [13]. The decoding problem is closely related to that of observer design with the key difference being that observers reconstruct the single state at the end of an output sequence, whereas decoding reconstructs an entire sequence of (latent) states. For deterministic systems, both problems are essentially equivalent. However, in the presence of non-determinism (common in many hybrid automata models), the sequence of states adds more information than a single state.

Our approach first proves that the problem of finding if there is a sequence of modes, transitions and continuous states whose outputs are ϵ-close to a given output sequence is NP-complete. Furthermore, it is possible to reduce this to solving a mixed integer linear program (MILP) along the lines of well-known results from Bemporad et al [1, 3]. However, solving this MILP is exponential in the size of the hybrid automaton and the length of the sequence, in the worst-case. In this paper, we consider a relaxation of this problem by first computing a finite *cover* consisting of a set of mode/transition sequences without the continuous state information. The desired property of such a cover is that for every given output sequence, there is a sequence in the cover which can produce an output sequence that is ϵ close to the given sequence under a suitable norm (we use L_∞ norm in this paper but our approach can handle L_1 or L_2 norms as well, with modifications). Once a cover is found, it is possible to solve the decoding problem approximately by solving as many LPs as the size of the cover. In practice, this drastically reduces the time to solve the decoding problem, since we find that the size of the cover is relatively small.

However, it is a computationally hard problem to verify if a given set C is a cover, let alone finding a cover. Our approach uses randomization by relaxing the notion of cover to be a probabilistic one. I.e, rather than covering all output sequences, we define a distribution over output sequences and ask that the set covers output sequences with probability at least $1-\alpha$ for a given α. In order to find a cover, we use a randomized algorithm based on sampling and verify a cover in a statistical sense. Our approach here borrows ideas from statistical model checking [10, 16, 25] and related ideas from randomized solution to control design problems [7, 22]. Our sampling-based verification algorithm grows a candidate cover set and in each iteration attempts to verify it by generating samples. If unsuccessful, the algorithm adds more samples to the candidate cover set. When successful, it produces a cover set with the required probabilistic guarantees.

We have empirically evaluated our approach on seven benchmark examples of hybrid automata with $4-10$ state variables and upto 400 modes. Our approach considers relatively long output sequences with upto 100 steps. We show that the MILP solver is often incapable of solving large problems within the given time budget. On the other hand, for each example, we discover relatively small cover sets whose sizes range from a few tens to a few thousands. Using an optimized tree-based decoding algorithm, we minimize the number of LPs that are needed to solve for the decoding. Thus, our approach achieves many orders of magnitude reduction in the decoding time when compared to the MILP.

Recent applications to runtime monitoring have used state estimation techniques to compute the state of a system under monitoring from sensor data in order to evaluate the possibility of runtime property violations [9, 17]. The decoded output can be naturally used for runtime verification, especially when the properties of interest are specified over the states of the system whereas observations are made on the outputs. Furthermore, we may use the model to predict future states, enabling predictive monitoring [23, 24, 26] or denoising, wherein the actual state sequence can be used to produce an idealized version of the output sequence.

Due to limited space, many details including the benchmarks used in our experimental evaluation are not presented here. An extended version of the paper with these details can be obtained from the authors' websites.

1.1 Related Work

Alur et al investigate a closely related problem of testing if a given sequence of labels can be accepted by a hybrid automaton with piecewise constant dynamics [2]. Although this model has more restrictive dynamics than the linear time dynamical systems studied in this paper, it is interesting to note that the NP-completeness of the decoding problem already applies to hybrid automata with piecewise constant dynamics.

As mentioned earlier, the decoding problem is closely related to that of designing observers. Observer design for hybrid systems is a hard problem and known conditions for linear and nonlinear systems are hard to generalize. For instance, Hwang et al present extensions of Luenberger observers for stochastic hybrid systems based on combining multiple observers, one for each mode in order to estimate the current latent mode and continuous state [15]. Another important line of work has involved mixed integer linear programs. This work, pioneered by Bemporad, Morari and their coworkers, models linear hybrid systems as so-called "mixed-logic dynamical systems". They encode the problem of finding state sequence as a mixed binary optimization problem and present combinations of SAT solvers with linear programming solvers [4], anticipating later work in the formal methods community on SAT-modulo theory solvers [11, 19] and SAT-modulo convex optimization solvers [21]. Nevertheless, these approaches are quite expensive for real-time state observations. An alternative solution involves explicitly computing the map from the output observations to the latent mode/state sequences so that it can be calculated efficiently during the deployment. This approach was first pioneered by Morari et al for model-predictive control [5] but also extended to the hybrid system observability problem [3]. A key drawback is that these approaches rely on solving parametric integer linear programming problem, which is often quite expensive to solve. The size of this problem is exponential in the length of the observation sequences. Therefore, such approaches are suitable for relatively short length observation sequences.

2 PRELIMINARIES AND PROBLEM FORMULATION

In this section, we will provide definitions of the linear stochastic hybrid automaton model and formulate our approach to the decoding problem. Let $X : \{x_1, \ldots, x_n\}$ represent state variables collectively denoted by a vector $\vec{x}$. Similarly, $Y : \{y_1, \ldots, y_k\}$ denote output variables denoted as vector $\vec{y}$ and $W : \{w_1, \ldots, w_l\}$ denote the disturbance input variables collectively written as vector $\vec{w}$.

Definition 2.1 (Linear Stochastic Hybrid Automaton). A linear stochastic hybrid automaton $\mathcal{H}$ over states X, outputs Y and disturbances W is represented by a tuple $\langle Q, \mathsf{dyn}, \mathcal{T}, \mathrm{G}, q_0, \mathcal{D}_0, \mathcal{D}_w\rangle$: (1) $Q : \{q_1, \ldots, q_m\}$ is a finite set of *modes*; (2) dyn maps each mode $q \in Q$ to its (discrete-time) dynamics $\mathcal{D}(q) : (A_q, B_q)$, denoting the dynamics: $\vec{x}(t+1) = A_q\vec{x}(t) + B_q\vec{w}(t)$. (3) $\mathcal{T} = \{\tau_1, \ldots, \tau_p\}$ is a finite set of transitions. Each $\tau_i : \left\langle q_i, q_i', \mathsf{guard}_i, \mathsf{update}_i\right\rangle$: (a) q_i, q_i' represent the pre and post-modes, respectively. (b) guard_i is a guard polyhedron $P_{\tau,i}\vec{x} \leq \vec{r}_{\tau,i}$. (c) update_i is an affine transformation $\vec{x}' := A_{\tau,i}\vec{x} + \vec{b}_{\tau,i}$. (4) G represents an affine output map of the form $\vec{y} = G\vec{x} + \vec{h}$ that maps each state to an output. (5) $q_0 \in Q$ is fixed initial mode. (6) $\mathcal{D}_0$ is a probability distribution over X, from which the initial states will be drawn. Its set of support is a polyhedron $\mathcal{X}_0$. (7) $\mathcal{D}_w$ is a probability distribution over W for the disturbance inputs. Its set of support is a polyhedron $\mathcal{W}$.

Additionally, for all $q \in Q$, if τ_i and τ_j are two different transitions with q as their pre-mode, then $\mathsf{guard}_i \cap \mathsf{guard}_j = \emptyset$.

Our approach can easily extend to a more general model that includes multiple initial modes, multiple enabled transitions from a given state and output maps that vary depending on the mode.

A *state* of a linear hybrid automaton is a tuple $\langle q, \vec{x}\rangle$ wherein $q \in Q$ is the current mode, $\vec{x}$ is the current continuous state (i.e, a real-value assigned to each state variable in X). A run of the hybrid automaton is a finite or infinite sequence of states and associated disturbance values: $(q(0), \vec{x}(0), \vec{w}(0)) \xrightarrow{\tau(1)} (q(1), \vec{x}(1), \vec{w}(1)) \cdots \xrightarrow{\tau(t)} (q(t), \vec{x}(t), \vec{w}(t)) \rightarrow \cdots$, wherein, (1) q_0 is the initial mode, and $\vec{x}(0)$ is a random initial state drawn according to $\mathcal{D}_0$. (2) $\tau(t)$ is either a transition $\tau \in \mathcal{T}$ or a special action nop, indicating that no transition is taken. (3) If $\tau(t) = \mathsf{nop}$, then $q(t-1) = q(t)$, $\vec{x}(t) = A_{q(t-1)}\vec{x}(t-1) + B_{q(t-1)}\vec{w}(t-1)$, where $\vec{w}(t-1)$ is a random vector drawn according to the distribution $\mathcal{D}_w$. (4) Otherwise, if $\tau(t) = \tau_j$, then $q(t-1)$ must be the source and $q(t)$ the target for transition τ_j. Furthermore, there exists "pre-transition" state $\vec{x}'(t-1)$ such that: (a) $\vec{x}'(t-1) = A_{q(t-1)}\vec{x}(t-1) + B_{q(t-1)}\vec{w}(t-1)$ (dynamics $q(t-1)$); (b) $\vec{x}'(t-1) \models \mathsf{guard}_{\tau,j}$ (must satisfy the transition guard); and (c) $\vec{x}(t) = \mathsf{update}_{\tau_j}(\vec{x}'(t-1))$ (state update).

Let us assume that $\mathcal{D}_0$ is defined by a probability density function $p_0(\vec{x}_0)$, and likewise, $p_w(\vec{w})$ is the PDF associated with $\mathcal{D}_w$. Note that $p_0(\vec{x}_0) > 0$ iff $\vec{x}_0 \in \mathcal{X}_0$, and likewise, $p_w(\vec{w}) > 0$ iff $\vec{w} \in \mathcal{W}$. For a finite run σ: $(q(0), \vec{x}(0), \vec{w}(0)) \xrightarrow{\tau(1)} (q(1), \vec{x}(1)) \cdots \xrightarrow{\tau(T)} (q(T), \vec{x}(T), \vec{w}(T))$, we define its *log-likelihood* as:

$$\mathsf{logL}(\sigma) : \log(p_0(\vec{x}(0))) + \sum_{j=0}^{T-1} \log(p_w(\vec{w}(j))) .$$

If the initial state $\vec{x}(0) \notin \mathcal{X}_0$, or any disturbance $\vec{w}(j) \notin \mathcal{W}$, the value of $\mathsf{logL}(\sigma)$ will be $-\infty$.

Whereas a run is a sequence of states of the hybrid automata, we define *mode-transition* sequences that consist purely of the discrete modes and transitions.

Definition 2.2 (Mode-Transition Sequences). A sequence of modes and transitions of length $T > 0$

$$\sigma : q(0) \xrightarrow{\tau(1)} q(1) \xrightarrow{\tau(2)} \cdots \xrightarrow{\tau(T-1)} q(T-1) ,$$

is a *mode-transition* sequence iff for all $t \in [0, T-1]$, (a) each $q(t) \in Q$, (b) each $\tau(t)$ is either nop or $\tau_j \in \mathcal{T}$, (c) if $\tau(t) = \text{nop}$ then $q(t-1) = q(t)$; otherwise; (d) if $\tau(t) = \tau_j \in \mathcal{T}$ then $q(t-1)$ is the source mode for τ_j while $q(t)$ is a destination mode. The mode-transition sequence is feasible iff there exists a finite run:

$$\sigma : (q(0), \vec{x}(0), \vec{w}(0)) \xrightarrow{\tau(1)} \cdots \xrightarrow{\tau(T-1)} (q(T-1), \vec{x}(T-1), \vec{w}(T-1)) ,$$

such that $\mathsf{logL}(\sigma) > -\infty$.

2.1 Decoding Problem Formulation

Let $\hat{Y} : (\hat{\mathsf{y}}_0, \hat{\mathsf{y}}_1, \ldots, \hat{\mathsf{y}}_{T-1})$ be a given sequence of outputs. Given two sequences $Y_1 : (\vec{y}_1(0), \ldots, \vec{y}_1(T-1))$ and $Y_2 : (\vec{y}_2(0), \ldots, \vec{y}_2(T-1))$ of the same length T, we fix a *discrepency metric*: $h(Y_1, Y_2)$. Examples of such a metric could include a norm such as:

$$||Y_1 - Y_2||_\infty : \max_{t=0}^{T-1}(||\vec{y}_1(t) - \vec{y}_2(t)||_\infty) .$$

Definition 2.3 (Optimal Decoding Problem). Given a sequence of outputs Y of length T and a linear stochastic hybrid automaton model $\mathcal{H}$, we wish to find a run σ:

$$(q(0), \vec{x}(0), \vec{w}(0)) \rightarrow \cdots \rightarrow (q(T-1), \vec{x}(T-1), \vec{w}(T-1)) .$$

such that $\mathsf{logL}(\sigma) > -\infty$ and the corresponding output sequence $\hat{Y} : (\mathsf{G}(\vec{x}(0)), \cdots, \mathsf{G}(\vec{x}(T-1)))$ minimizes the objective function:

$$-\mathsf{logL}(\sigma) + h(\hat{Y}, Y) .$$

The objective function has two parts, one term maximizes the likelihood of the trace and the other minimizes the discrepency between the actual observations and those produced by the trace. The decoding problem is closely related to that of designing an *observer* or a *state estimator*. The chief difference is that an observer (or state-estimator) typically recovers the state $(q(T-1), \vec{x}(T-1))$ at the last time step, and updates this state when a new input $\vec{y}(T)$ is received. The *decoding problem* on the other hand [1] reconstructs a sequence of states that produce a given output. The complexity of the optimal decoding problem depends on factors that include: (a) the form of the probability distributions $\mathcal{D}_0$ and $\mathcal{D}_w$; and (b) the discrepancy metric $h(Y, \hat{Y})$. Commonly encountered examples will treat $\mathcal{D}_0$ ($\mathcal{D}_w$) as uniformly distributed over some compact set in $\mathbb{R}^n$ (or $\mathbb{R}^l$); or a multivariate Gaussian with a given mean and standard deviation. Likewise, we assume the discrepancy metric is a norm such as the Euclidean norm $h(Y, \hat{Y}) = ||Y - \hat{Y}||_2^2$, L_1 norm $h(Y, \hat{Y}) = ||Y - \hat{Y}||_1$ or the L_∞ norm $h(Y, \hat{Y}) = ||Y - \hat{Y}||_\infty$. A detailed discussion of these choices will be made in the extended version.

[1] Inspired by the Viterbi decoding algorithm for Hidden Markov Models.

Depending on these choices, the optimal decoding problem for a stochastic linear hybrid automaton can be formulated as a (binary) mixed-integer linear program or a (binary) mixed-integer quadratic program. For technical reasons we consider a *decision version* of the problem that asks if the optimal decoding problem has optimal value $\leq \epsilon$. We will call this the ϵ decoding problem.

For the rest of this paper, we will fix the following assumptions for the simplifying the presentation: (a) the discrepancy function $h(Y_1, Y_2) : ||Y_1 - Y_2||_\infty$ and (b) the distributions $\mathcal{D}_0$ and $\mathcal{D}_w$ are uniform over compact polyhedra $\mathcal{X}_0$ and $\mathcal{W}$, respectively. The latter assumption guarantees that for every finite trace σ of length T if $\mathsf{logL}(\sigma) > -\infty$ then $\mathsf{logL}(\sigma) = c_T$, for some fixed constant $c_T < 0$.

THEOREM 2.4. *The ϵ decoding problem given a stochastic linear hybrid automaton is NP-complete.*

Proof is provided in the extended version. Furthermore, in the extended version, we demonstrate a mixed integer formulation of the problem using binary variables to denote the choice of transitions. Continuous variables are used for encoding state $\vec{x}(t)$, output $\vec{y}(t)$ and disturbances $\vec{w}(t)$ at each time step. Likewise binary variables $w(q_j, t), w(\tau_j, t), w(\overline{T}, t)$ at each time step indicate whether the trace visits mode q_j at time t, takes the transition τ_j at time t or remains in the same mode at time t without any transitions, respectively. As a result, any feasible solution will yield a trace σ:

$$(q(0), \vec{x}(0), \vec{w}(0)) \rightarrow \cdots \rightarrow (q(T-1), \vec{x}(T-1), \vec{w}(T-1)) ,$$

with mode $q(t) = q_j$ iff $w(q_j, t) = 1$, $\tau(t) = \tau_k$ if $w(\tau_k, t) = 1$ or $\tau(t) = \text{nop}$ if $w(\overline{T}, t) = 1$, state $\vec{x}(t)$ and output $\vec{y}(t) = \mathsf{G}(\vec{x}(t))$. Likewise, the formulation guarantees that $\mathsf{logL}(\sigma) > -\infty$.

THEOREM 2.5. *The sequence of modes, transitions, states and outputs corresponding to any feasible solution of the MILP defined above constitutes a run of the hybrid automaton. Conversely, any run of length T corresponds to a feasible solution of the MILP.*

However, the MILP approach can be quite expensive. The number of binary variables is given by $O(T \times (|\mathcal{T}| + |Q|))$, wherein $|\mathcal{T}|$ is the number of transitions in the hybrid automaton and $|Q|$ is the number of modes. Also, the best known algorithms for solving (binary) MILPs require time exponential in the number of binary variables, in the worst case. This blow-up is seen in practice, as demonstrated by the empirical evaluation in Section 4.

3 PROPOSED APPROACH

Instead of solving an optimal decoding problem as a MILP in a *single shot*, we may solve it in two steps: (1) Iterate over all sequences of modes/transitions, $\sigma : q(0) \rightarrow \cdots \rightarrow q(T-1)$. (2) Once a sequence σ is fixed, we formulate an LP to find the continuous states

$\vec{x}(t)$, outputs $\vec{y}(t)$ and disturbances $\vec{w}(t)$. This LP is obtained by fixing the values of all the binary variables in the MILP formulation according to the sequence σ. The overall solution selects that sequence σ whose corresponding LP minimizes the objective function. However, this approach is explicitly equivalent to enumerating all assignments to the binary variables in the MILP and solving a LP for each such assignment. The number of such assignments is exponential in $T \times (|Q| + |\mathcal{T}|)$.

Our approach solves an approximate version of the optimal decoding problem by fixing a finite set of mode-transition sequences $C : \{\sigma_1, \ldots, \sigma_N\}$. Although there are no guarantees on the size of such a sequence, in practice, we will show that $N \ll 2^{T\times(|Q|+|\mathcal{T}|)}$. We will call the set C a covering set of sequences. Once such a sequence is obtained, we may solve $|C|$ linear programs to approximate the optimal decoding problem.

In what follows, we will first introduce the notion of ϵ-cover sequences. However, the problem of verifying if a given set C satisfies this notion is computationally expensive. Therefore, we will introduce a probabilistic notion of coverage with high confidence that is easy to check and synthesize.

3.1 ϵ-Cover sequences

Let us define the set of all possible output sequences of length T as

$$\mathcal{Y}_T : \{Y : (\vec{y}(0), \ldots, \vec{y}(T-1)) \mid \text{there exists a run of } \mathcal{H} \text{ with output } Y\}\,.$$

Let $\epsilon > 0$ be a given tolerance parameter. We say that a mode-transition sequence $\sigma : q(0) \xrightarrow{\tau(1)} \cdots \xrightarrow{\tau(T-1)} q(T-1)$, *$\epsilon$-covers* a given output sequence $Y : (\vec{y}(0), \ldots, \vec{y}(T-1))$ iff there exists a run

$$(q(0), \vec{x}(0), \vec{w}(0)) \rightarrow \cdots \rightarrow (q(T-1), \vec{x}(T-1), \vec{w}(T-1))\,,$$

of the automaton $\mathcal{H}$ with output sequence $\tilde{Y} : (\tilde{\vec{y}}(0), \ldots, \tilde{\vec{y}}(T-1))$; and the distance $||Y - \tilde{Y}||_\infty \leq \epsilon$. In other words, there is a run that follows the given mode-transition sequence σ whose outputs are ϵ close to Y.

A set of mode-transition sequences $C : \{\sigma_1, \ldots, \sigma_N\}$ is said to ϵ-cover the output space $\mathcal{Y}_T$ iff for all output sequences $Y \in \mathcal{Y}_T$, there exists a mode-transition sequence $\sigma_j \in C$ such that σ_j ϵ-covers Y.

The set of all possible mode-transition sequences is a valid ϵ-cover for any $\epsilon \geq 0$. However, we seek a smaller covering set for a given ϵ. However, finding such as set is computationally hard. Let us first consider the *verification problem*, wherein given a set of mode-transition sequences C and tolerance $\epsilon > 0$, we wish to find out if C is a valid ϵ-cover for $\mathcal{Y}_T$.

Theorem 3.1. *The problem of checking given a set of sequences C, whether C ϵ-covers $\mathcal{Y}_T$ is co-NP-complete.*

The proof of membership in co-NP consists of formulating a mixed integer optimization problem whose infeasibility indicates that C covers $\mathcal{Y}_T$. Likewise, completeness is obtained by reducing the problem of checking if a hybrid automaton has no valid runs of length T, which can be separately proved to be co-NP-complete. Therefore, we may simply set $C = \emptyset$ and reduce to the ϵ-cover checking problem.

Although, co-NP complete problems are common in verification and solved for large instances by modern SAT/SMT solvers [11, 19], the complexity of checking whether a set C ϵ-covers $\mathcal{Y}_T$ is exponential in the time horizon length T, the size of the hybrid automaton $\mathcal{H}$ and also exponential in the size of the cover set $|C|$. Since we do not have a priori polynomial bounds on $|C|$, this becomes quite an expensive problem to solve exactly.

3.2 Probabilistic and Approximate ϵ-Covers

Rather than seek to cover $\mathcal{Y}_T$ exactly with a set of mode-transition sequences C, we seek to cover a subset of $\mathcal{Y}_T$ using a set C, wherein the probability that a randomly drawn sequence from $\mathcal{Y}_T$ will be ϵ-covered by some sequence in C will be bounded from below by probability $1 - \alpha$.

Probabilistic ϵ-Cover: Let $0 < \alpha \ll 1$ be a given *coverage* probability parameter. Let us assume that $\mathcal{D}$ is a probability distribution over the output space $\mathcal{Y}_T$. We say that a set C of mode-transition sequences is a probabilistic ϵ-cover with coverage parameter α iff for any randomly sampled sequence $\hat{Y} \sim \mathcal{D}$, there exists a sequence $\sigma \in C$ such that σ covers $\hat{Y}$ with probability at least $1-\alpha$. In order to statistically verify that a set C ϵ-covers $\mathcal{Y}_T$ with coverage parameter α, we implement a simple statistical test:

(1) Sample a fixed number $K > 0$ output sequences $Y_1, \ldots, Y_K$ from the distribution $\mathcal{D}$.
(2) Check if all the sampled output sequences are ϵ-covered by some mode-transition sequence in C.
  (a) If all sampled output sequences are covered, we declare that C ϵ-covers $\mathcal{Y}_T$ with coverage parameter α.
  (b) If not, we declare that C fails to cover $\mathcal{Y}_T$ with the desired coverage parameter.

Since our test is statistical in nature, there is the chance that it will erroneously accept a cover set C even if it does not actually satisfy the coverage criterion. This is called a type-I error in statistics [2]. Our goal is to choose the number of test samples K large enough to bound the probability of type-I error to be at most δ, where $0 < \delta \ll 1$ is a confidence parameter that is chosen by the user.

In what follows, we will consider how to choose a sample set size K that guarantees that whenever we verify that C is a cover with coverage parameter α, the probability of erroneous verification is bounded by at most δ.

Let $p(C)$ be the real underlying probability that a randomly chosen sample output sequence $Y \sim \mathcal{D}$ is covered by some mode-transition sequence in C. Thus, each sample's coverage is seen as a Bernoulli coin toss which will verify coverage (turn up "heads") with probability $p(C)$. The statistical test tells us that K such consecutive coin tosses all turn up heads. This happens with probability $p(C)^K$ (the sample space here is over imagined repetitions of this experiment of K consecutive coin tosses). Suppose $p(C) < 1 - \alpha$, but we turn up K heads anyways to pass the statistical test.

$$Pr(K \text{ consecutive "heads"}) = p(C)^K \leq (1-\alpha)^K\,.$$

In order to ensure $(1-\alpha)^K \leq \delta$, we need

$$K \geq \frac{\log(\delta)}{\log(1-\alpha)}\,. \tag{1}$$

[2]The other type of error where a valid cover set C is rejected is a type-II error, which we will ignore since it will not affect the design of the verification procedure.

Table 1: Overview of Linear Hybrid System Benchmarks

Benchmark	#state	#control	#output	#mode	#transition
Bouncing Ball	4	0	2	1	1
Water Tank	2	0	1	2	2
Room Heater 1	3	0	2	3	4
Room Heater 2	5	0	4	80	720
Vehicle Platoon	10	1	3	2	2
Geometric Shapes	7	0	2	47	47

The simple argument above assures us that as long as K is chosen to be at least $\frac{\log(\delta)}{\log(1-\alpha)}$, we can conclude with probability at least $1-\delta$ that the set C covers $\mathcal{Y}_T$ with coverage parameter α.

We fix a bound K on the number of samples for our verification procedure using (1). The synthesis procedure initializes the candidate cover set to the empty set. Next, it runs the verification procedure by drawing K independent samples from $\mathcal{D}$. If an output sequence is not covered, we add a corresponding mode/transition sequence back into the set C. This requires us to re-run the verification procedure from scratch. The process succeeds when the verification also succeeds.

Given a cover set C and an output sequence $Y : (\vec{y}(0), \ldots, \vec{y}(T-1))$, the decoding problem reduces to solving $|C|$ linear programs, one for each mode-transition sequence $\sigma \in C$. Each linear program is obtained from a mode-transition sequence σ by fixing the binary variables in the MILP formulation in accordance with the mode transition sequence. Linear programs can be solved relatively quickly with some of the fastest algorithms having a polynomial time complexity. Out of the $|C|$ linear program solutions, the solution with the least objective value is "closest to" the given output sequence Y as measured using the L_∞ norm distance. Subsequently, we choose this to be the solution of the ϵ decoding problem.

It is possible to improve the running time substantially by solving more LPs, but each of a smaller size. This is performed by organizing the cover set C as a tree with $|C|$ leaves, wherein each branch of the tree represents a sequence in C. The tree also serves to represent common prefixes between various sequences in C. This algorithm will be presented in our extended version.

4 EXPERIMENTAL EVALUATION

In this section, we present an experimental evaluation of the proposed approach. We compare the performance of the proposed approach and the mixed integer program (MILP) optimization approach on a set of seven benchmarks from different applications.

Table 1 provides an overview of the benchmarks used for evaluation. The benchmarks include a model of a bouncing ball, water tank from Lygeros et al [18], room heater (3 rooms and 5 rooms) from Fehnker and Ivancic [12], vehicle platoon with a control input from Makhlouf et al [6], and a geometric shapes recognition inspired by QuickDraw [8]. Descriptions of these benchmarks will be available in our extended version.

All of the algorithms were implemented in Python 3.6. The linear programs (MILP and LP) were solved using the Python extension of Gurobi [14].

Test Dataset: For each benchmark, we created a test dataset of n=1000 noisy output sequences to evaluate the two decoding approaches through simulation. The noise was uniformly distributed in the range $[-\epsilon, \epsilon]$ and simply added to the simulated outputs.

Synthesizing ϵ-Cover Set: We synthesized cover sets using the procedure outlined in Section 3. The values of ϵ used in the experiments are reported in Table 2, Column 1. The sample set size was chosen in accordance with seeing K=150 consecutive samples that were already covered corresponding to $\alpha = 0.03$ (97% coverage probability) and $\delta = 0.01$ (99% confidence). The number of mode-transition sequences found in the ϵ-cover set and the time taken to generate the set are reported in Table 2, Columns 2-5. After synthesizing the cover sets, we constructed ϵ-cover prefix trees for decoding output sequences in the test dataset.

Performance on Test Dataset: For every output sequence in the test dataset, we solve a mixed integer program to decode the state and mode sequence that best matches the output sequence. The average, minimum, and maximum time taken (in seconds) to decode the n=1000 noisy output sequences in the dataset are reported in Table 2, Columns 6-8. The average MILP objective is reported in Table 2, Column 9. Similarly, we report the performance of the our approach. The average, minimum, and maximum time taken (in seconds) is reported in Table 2, Columns 10-13 and the average LP objective is reported in Table 2, Column 14.

Figure 1 shows plots of the outputs of the proposed approach on some output sequenes. The solid lines and round markers indicate the decoder output. The dashed line and square markers shows the points in the provided noisy output sequence. We also simulate the system forward in time for some length after the decoding time horizon T and show this with solid lines in the figure.

Discussion of Results: In our experimental evaluation, we observed that a surprisingly small number of mode-transition sequences suffice to ϵ-cover with high confidence. This is demonstrated when we compare columns 2 and 3 of Table 2. The time taken to synthesize the set ranges from 14 seconds to around 30 hours for some of the larger benchmarks. However, note that the ϵ-cover set needs to be synthesized just once, offline, for a given time horizon T.

The remaining columns of Table 2 compares the MILP decoding approach and the ϵ decoding approach for different time horizons. The time taken to solve the two smaller benchmarks - bouncing ball and water tank - using the two approaches are comparable. We report values for time horizons up to T=100 time steps. For the other benchmarks, the MILP decoding approach times out (after 10,000 seconds), as the sequence length increases. This is indicated with a "t/o" label in Table 2.

In general, our approach is much faster than the MILP approach in terms of computation time, for most of the benchmarks. This shows that our approach of synthesizing a finite cover sequence is successful in practice. The average objective value of the LP-based decoding is slightly larger than ϵ, which is to be expected since the cover set accounts for about 97% of all output sequences.

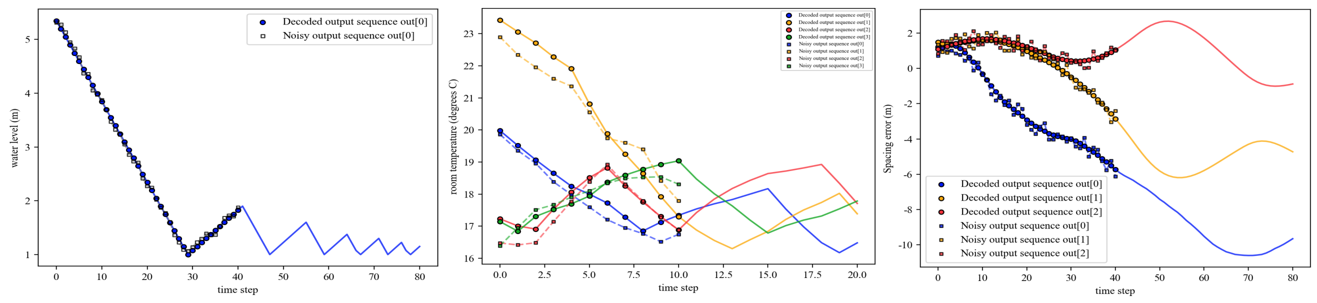

Figure 1: Simulations showing data with noise and the decoded sequence run forward in time to predict future outputs.

Table 2: Candidate set generation, Performance of MILP vs LP decoder on 1000 noisy output sequence samples. All of the reported times are from experiments performed on a Linux server running Ubuntu 18.04 OS with 24 cores and 64 GB RAM. "t/o" denotes timeout after 10,000 seconds.

	ϵ-Cover Set Generation				Mixed-Integer Linear Program				ϵ Decoder			
Benchmark	T	# possible	# found	time	Avg. time(s)	Min. time(s)	Max. time(s)	Avg. obj.	Avg. time(s)	Min. time(s)	Max. time(s)	Avg. obj.
Bouncing Ball $\epsilon = 0.05$	20	2^{20}	95	01m 19s	0.31	0.04	1.12	0.05	0.72	0.28	2.40	0.11
	40	2^{40}	411	22m 12s	0.92	0.12	4.34	0.05	2.15	1.06	6.08	0.06
	60	2^{60}	705	1h 25m 53s	1.98	0.19	8.99	0.05	4.27	2.01	11.65	0.06
	80	2^{80}	888	2h 50m 14s	4.08	0.32	38.15	0.05	6.67	3.39	13.05	0.05
	100	2^{100}	983	4h 07m 38s	7.30	0.64	44.55	0.05	10.40	5.41	34.33	0.06
Water Tank $\epsilon = 0.05$	20	$2 \cdot 3^{20}$	40	14s	0.19	0.04	1.62	0.05	0.39	0.11	0.85	0.05
	40	$2 \cdot 3^{40}$	486	15m 36s	0.75	0.08	14.80	0.05	1.31	0.37	3.75	0.05
	60	$2 \cdot 3^{60}$	1668	1h 53m 10s	2.52	0.29	14.84	0.05	2.80	0.84	9.06	0.06
	80	$2 \cdot 3^{80}$	2916	5h 20m 01s	4.61	0.58	21.65	0.05	5.04	1.34	16.29	0.05
	100	$2 \cdot 3^{100}$	4613	11h 31m 00s	6.77	0.68	29.46	0.05	6.89	2.20	23.37	0.07
Room Heater 1 $\epsilon = 1.0$	10	$3 \cdot 5^{10}$	11	22s	3.28	0.58	26.03	0.83	0.16	0.05	0.43	0.99
	20	$3 \cdot 5^{20}$	113	12m 26s	29.41	1.94	175.85	0.91	1.57	0.24	6.32	1.14
	30	$3 \cdot 5^{30}$	944	17h 13m 21s	191.38	10.47	7776.37	0.94	10.01	0.77	103.49	1.07
Room Heater 2 $\epsilon = 1.0$	5	$80 \cdot 721^{5}$	37	5m 24s	-	t/o	t/o	-	0.48	0.13	1.26	0.99
	10	$80 \cdot 721^{10}$	606	22h 50m 39s	-	t/o	t/o	-	16.77	4.72	31.47	1.15
Vehicle Platoon $\epsilon = 0.05$	10	$2 \cdot 3^{10}$	1	38s	5.66	1.91	12.49	0.05	0.54	0.27	0.69	0.05
	20	$2 \cdot 3^{20}$	2	02s	35.60	23.75	56.92	0.05	3.57	1.86	4.49	0.06
	30	$2 \cdot 3^{30}$	11	4m 18s	135.74	83.91	233.48	0.05	21.36	8.27	36.44	0.05
	40	$2 \cdot 3^{40}$	16	4m 46s	424.09	259.45	659.43	0.05	30.68	7.65	94.49	0.06
	50	$2 \cdot 3^{50}$	42	31m 53s	-	t/o	t/o	-	38.77	11.22	195.27	0.06
Geometric Shapes $\epsilon = 0.1$	10	$61 \cdot 121^{10}$	45	1m 39s	-	t/o	t/o	-	1.30	0.31	4.40	0.09
	20	$61 \cdot 121^{20}$	161	19m 46s	-	t/o	t/o	-	5.20	0.94	15.59	0.09
	30	$61 \cdot 121^{30}$	500	2h 20m 10s	-	t/o	t/o	-	10.05	1.96	35.80	0.10
	40	$61 \cdot 121^{40}$	1049	11h 10m 05s	-	t/o	t/o	-	13.85	3.31	72.38	0.10
	50	$61 \cdot 121^{50}$	1525	30h 11m 45s	-	t/o	t/o	-	10.10	3.78	76.99	0.10

5 CONCLUSIONS

To conclude, we have demonstrated an approach to the decoding problem for discrete time linear hybrid automata along with an approach based on synthesizing a cover set with probabilistic guarantees. Our approach is evaluated over benchmark examples and shows promise by synthesizing relatively small cover sets that allow rapid solution to the decoding problem even for large hybrid systems with long output sequences.

ACKNOWLEDGMENTS

The authors thank Dr. Guillaume Berger for helpful discussions. This work was supported by the US National Science Foundation (NSF) under award numbers 1836900 and 1932189. All opinions are those of the author and not necessarily of NSF.

REFERENCES

[1] Alessandro Alessio and Alberto Bemporad. 2009. A Survey on Explicit Model Predictive Control. In *Nonlinear Model Predictive Control: Towards New Challenging Applications*. Springer, Berlin, Germany, 345–369.

[2] R. Alur, R. P. Kurshan, and M. Viswanathan. 1998. *Membership questions for timed and hybrid automata*. IEEE.

[3] A. Bemporad, G. Ferrari-Trecate, and M. Morari. 2000. Observability and controllability of piecewise affine and hybrid systems. *IEEE Trans. Autom. Control* 45, 10 (Oct 2000), 1864–1876.

[4] Alberto Bemporad and Nicolò Giorgetti. 2004. A SAT-Based Hybrid Solver for Optimal Control of Hybrid Systems. In *Hybrid Systems: Computation and Control (HSCC'04)*. Springer, 126–141.

[5] Alberto Bemporad and Manfred Morari. 1999. Control of systems integrating logic, dynamics, and constraints. *Automatica* 35, 3 (Mar 1999), 407–427.

[6] Ibtissem Ben Makhlouf and Stefan Kowalewski. [n.d.]. Networked Cooperative Platoon of Vehicles for Testing Methods and Verification Tools. 30–37. https://doi.org/10.29007/zvkb

[7] G.C. Calafiore and M.C. Campi. 2006. The scenario approach to robust control design. *IEEE Trans. Automat. Control* 51, 5 (2006), 742–753. https://doi.org/10.1109/TAC.2006.875041

[8] Salman Cheema, Sumit Gulwani, and Joseph LaViola. 2012. QuickDraw: improving drawing experience for geometric diagrams. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Austin Texas USA, 1037–1064.

[9] Yi Chou, Hansol Yoon, and Sriram Sankaranarayanan. 2020. Predictive Runtime Monitoring of Vehicle Models Using Bayesian Estimation and Reachability Analysis. In *IROS 2020*. IEEE, 2111–2118.

[10] Edmund Clarke, Alexandre Donze, and Axel Legay. 2009. Statistical Model Checking of Analog Mixed-Signal Circuits With An application to a third order $\Delta - \Sigma$ modulator. In *Hardware and Software: Verification and Testing (LNCS, Vol. 5394/2009)*. 149–163.

[11] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS (LNCS, Vol. 4963)*. Springer, 337–340.

[12] Ansgar Fehnker and Franjo Ivancic. 2004. Benchmarks for Hybrid Systems Verification.. In *Hybrid Systems: Computation and Control (LNCS, Vol. 2993)*. Springer, 326–341.

[13] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *Proc. CAV'11 (Lecture Notes in Computer Science, Vol. 6806)*. Springer-Verlag, 379–395.

[14] Gurobi Optimization, LLC. 2021. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[15] Inseok Hwang, Hamsa Balakrishnan, and Claire Tomlin. 2003. Observability criteria and estimator design for stochastic linear hybrid systems. In *2003 European Control Conference (ECC)*. 3317–3322.

[16] Sumit Kumar Jha, Edmund M. Clarke, Christopher James Langmead, Axel Legay, André Platzer, and Paolo Zuliani. 2009. A Bayesian Approach to Model Checking Biological Systems. In *CMSB (Lecture Notes in Computer Science, Vol. 5688)*. Springer, 218–234.

[17] Kenan Kalajdzic, Ezio Bartocci, Scott A. Smolka, Scott D. Stoller, and Radu Grosu. 2013. Runtime Verification with Particle Filtering. In *Proceedings of Runtime Verification (RV) (Lecture Notes in Computer Science, Vol. 8174)*. Springer, 149–166.

[18] John Lygeros, Shankar Sastry, and Claire Tomlin. 2012. *Hybrid Systems: Foundations, advanced topics and applications*.

[19] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 2006. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM* 53, 6 (2006), 937–977.

[20] Stuart J Russell, Peter Norvig, Ernest Davis, and Douglas Edwards. 2016. *Artificial intelligence: a modern approach*. Pearson, 566–610.

[21] Yasser Shoukry, Pierluigi Nuzzo, Alberto Sangiovanni-Vincentelli, Sanjit A. Seshia, George J. Pappas, and Paulo Tabuada. 2017. SMC: Satisfiability Modulo Convex Optimization. In *Hybrid Systems: Computation and Control (HSCC'17)*. 19–28.

[22] Roberto Tempo, Fabrizio Dabbene, and Giuseppe Calafiore. 2005. *Randomized Algorithms for Analysis and Control of Uncertain Systems*. Springer London, London, 117–130.

[23] Hansol Yoon, Yi Chou, Xin Chen, Eric Frew, and Sriram Sankaranarayanan. 2019. Predictive runtime monitoring for linear stochastic systems and applications to geofence enforcement for UAVs. In *International Conference on Runtime Verification*. Springer, 349–367.

[24] Hansol Yoon and Sriram Sankaranarayanan. 2021. Predictive Runtime Monitoring for Mobile Robots using Logic-Based Bayesian Intent Inference. In *IEEE International Conference on Robotics and Automation, ICRA 2021*. 8565–8571.

[25] Håkan L. S. Younes and Reid G. Simmons. 2006. Statistical Probabilitistic Model Checking with a Focus on Time-Bounded Properties. *Information & Computation* 204, 9 (2006), 1368–1409.

[26] Xian Zhang, Martin Leucker, and Wei Dong. 2012. Runtime Verification with Predictive Semantics. In *NASA Formal Methods*. Springer Berlin Heidelberg, 418–432.