2017 IEEE 56th Annual Conference on Decision and Control (CDC)
December 12-15, 2017, Melbourne, Australia

Grid-Based Temporal Logic Inference

Prashant Vaidyanathan, Rachael Ivison, Giuseppe Bombara, Nicholas A. DeLateur,
Ron Weiss, Douglas Densmore, and Calin Belta

Abstract— **This paper introduces a new algorithm to infer temporal logic properties of a system from data consisting of a set of finite time system traces. We propose an algorithm that generates a Signal Temporal Logic formula by discretizing the entire domain and codomain of the system traces. Unlike many popular inference algorithms which require labeled data that represents whether a trace exhibits a desired behavior (positive) or not (negative), this approach only requires positive traces to infer temporal logic properties. We present two case studies to illustrate the efficiency and accuracy of the proposed algorithm. The first is a biological network consisting of a genetic logic circuit in a bacterial cell. The second is a fault detection problem in automotive powertrain systems. We also compare the performance of the algorithm with an existing inference algorithm.**

I. INTRODUCTION

Testing and verification are very important steps in designing complex systems. Many verification techniques use a formal specification that describes admissible behaviors of a system during its execution in a mathematically unambiguous manner [1]. For this reason, specifications are expressed using appropriate logics, and Signal Temporal Logic (STL) has emerged as a rich specification language for describing continuous-time systems [2]. Traditionally, specifications are formulated by expert designers in a tedious and error-prone process. More recent methods involve learning the specifications directly from the execution traces of systems.

This challenge of inferring temporal logic properties from data procured from a system has been approached in many ways, including *mining parameters* for a given formula structure [3], and *temporal logic inference* (TLI), that is constructing both the formula structure and its parameters [4], [5]. In literature, the latter approach was cast within a two-class classification problem where a formula should distinguish and categorize a trace that belongs to one of two classes. In this setting, the learning is supervised, and the two classes are *positive* or *negative*. Positive traces represent normal working conditions, expected behaviors, or desired behaviors of a system, while negative traces indicate anomalies or nonconforming patterns of a system [5], [6].

This work was supported by the National Science Foundation under grant CPS Frontier 1446607.

Prashant Vaidyanathan (prash@bu.edu), Giuseppe Bombara (gbombara@bu.edu), and Douglas Densmore (dougd@bu.edu) are with the Electrical & Computer Engineering Department, Rachael Ivison (rivison@bu.edu) is with the Bioinformatics Program, and Calin Belta (cbelta@bu.edu) is with the Department of Mechanical Engineering at Boston University, Boston, MA, USA. Nicholas A. DeLateur (delateur@mit.edu) is with the Department of Chemistry and Ron Weiss (rweiss@mit.edu) is with the Department of Biological Engineering at Massachusetts Institute of Technology, Cambridge, MA, USA.

For some cases in which TLI would be useful, labeled data is difficult or impossible to procure. One such case is in complex biological systems, such as synthetic genetic circuits, where the ability to describe biological networks' behaviors over time would represent a powerful move forward in building complex and robust genetic circuits. The synthetic biology field currently relies on manual design of genetic circuits, sometimes aided by simulations. While the biological network construction paradigms [7], [8] allow for rapid genetic circuit assembly, they make no statement or guarantee to the behavior of the assembled DNA. Instead, these behaviors are captured empirically and describe specific properties of the genetic circuit in precise biochemical environments. While these environments could be varied to identify negative behaviors, the negative conditions for biological systems are so diverse that it is impractical to generate negative traces. In such situations, the two-class classification approach cannot be applied.

In this paper, we propose a grid-based approach, *GridTLI*, where we discretize the space of the finite-time system traces in the dataset, infer temporal properties, and express them as a formula in STL [2]. The proposed algorithm has some advantages over conventional two-class TLI approaches. First, GridTLI only requires positive traces and can guarantee zero false negatives for any trace in the training dataset. Second, the input parameters of the algorithm can be used to tune the complexity of the formula to fit the context of the data.

II. RELATED WORK

Two major tracks can be identified in the field of temporal logic inference. The first track addresses the so-called *Parameter Mining* problem [3], [9]: given a formula template and a set of traces generated by a normally-behaving system, find the optimal parameters for the template such that the resulting formula satisfies the input traces. Generally, the parameters are the time intervals for the temporal operators and the thresholds for the inequality predicates in the template formula. The approaches in these references differ in the way the underlying optimization problem is formulated and solved. It is also worth mentioning that the parameter optimization problem is cast within a more general active learning framework, where the original system is queried for new traces when deemed necessary.

The second track is focused on constructing a formula– *both* the structure and its parameters–within the context of the supervised two-class classification problem [4], [5], [10], [11]. In this setting, given a set of labeled traces, e.g., either *normal* or *anomalous*, the goal is to build a formula that can distinguish between them. In [4], [10], the formula is constructed by exploring a fragment of STL that admits a partial ordering, whereas the parameter optimization is performed using an SVM-like (Support Vector Machine) optimization. [11] also tackled this problem by first building two generative probabilistic models, one for each class, and later obtaining a discriminative classifier by searching for a formula that maximizes the odds of being true for the first model and false for the other model. The formula structure is constructed through heuristics, while the parameter space is explored through Statistical Model Checking. Recently, [5] proposed a decision-tree based approach to solve the two-class problem. In this approach, a binary tree is constructed with simple formulae at each node, called primitives. The optimal primitives, along with their parameters, are chosen by optimizing some impurity measures which quantify how well a primitive partitions the traces by class at each node. A tree with this structure is in a one-to-one mapping with an STL formula, which can be used for classification of new traces or other purposes.

III. SIGNAL TEMPORAL LOGIC

For $t \in \mathbb{R}$, we write the interval $[t, \infty)$ as $\mathbb{R}_{\geq t}$. We represent the set of all continuous n-dimensional parameterized curves by $\mathcal{S} = \{s : \mathbb{R}_{\geq 0} \to \mathbb{R}^n\}$ for $n \in \mathbb{N}$, and we call an element $s \in \mathcal{S}$ a *signal*. The parameter of s is interpreted as *time*. The components of a signal s are denoted by s^x, $x \in \{1, \ldots, n\}$, and the projection operators from s to its components are represented by $\mathcal{F} = \{f^x : \mathbb{R}^n \to \mathbb{R},\ f^x(s) = s^x,\ x \in \{1, \ldots, n\}\}$.

The syntax of STL is defined in [2] as

$$\phi ::= \top \ \mid \ p_{f(y) \leq \mu} \ \mid \ \neg\phi \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ \phi_1 \mathcal{U}_{[a,b)} \phi_2 \ ,$$

where $\top$ is the Boolean *true* constant ($\perp$ for *false*); $p_{f(y) \leq \mu}$ is a predicate over $\mathbb{R}^n$ parameterized by $f \in \mathcal{F}$ and $\mu \in \mathbb{R}$ such that $p_{f(y) \leq \mu} = f(y) \leq \mu$; $\neg$ and $\wedge$ are the negation and conjunction Boolean operators, respectively; and $\mathcal{U}_{[a,b)}$ is the bounded temporal operator *until*. The temporal operators *eventually* and *globally* are defined as $\mathbf{F}_{[a,b)}\phi \equiv \top \mathcal{U}_{[a,b)} \phi$ and $\mathbf{G}_{[a,b)}\phi \equiv \neg \mathbf{F}_{[a,b)} \neg\phi$, respectively.

A signal s is said to *satisfy* an STL formula ϕ if and only if $s \models \phi$. The quantitative semantics of STL formally characterizes the degree of robustness [12] of a signal with respect to an STL formula. A signal is said to satisfy ϕ (in Boolean semantics) if and only if its robustness has a positive value. [1]

IV. PROBLEM DESCRIPTION

Given a *training* set of finite time signals S, which represents the desired behaviors of the system, we aim to find an STL formula ϕ such that all the signals in S satisfy the formula. The STL formula should *fit* the signals, and the fit should be neither too tight nor too loose. A loose fit results in an STL formula that is not very useful, since it might not grasp the peculiar characteristics of the signals. In contrast, a tight fit results in a long STL formula that is excessively specific to the training signals. Ideally, the granularity of the fit of the STL formula must be tunable based on the requirements and properties of the system.

V. GRID-BASED TEMPORAL LOGIC INFERENCE

Grid-based temporal logic inference, or GridTLI [2], is an algorithm that generates an STL formula from a given set of training signals. The algorithm uses three thresholds which affect the tightness of the produced formula to the training data. GridTLI operates on one-dimensional signals, for multi-dimensional signals the algorithm should be executed on each dimension separately (see Remark 3).

A. Notation and Definitions

R is a closed rectangular *region* in the 2-dimensional Euclidean space $\mathbb{R} \times \mathbb{R}_{\geq 0}$ given by the value codomain and time domain of one-dimensional signals. The region R is bounded by $[x_{min}, x_{max}] \times [0, t_{max}]$, where $x_{min}, x_{max} \in \mathbb{R}$, $x_{min} < x_{max}$, and $t_{max} \in \mathbb{R}_{>0}$. We will partition this region R into a grid made of rectangular cells.

A *cell* g is a rectangular bounding box in $\mathbb{R} \times \mathbb{R}_{\geq 0}$ represented with a tuple (x, t, x_{inc}, t_{inc}), where $(x, t) \in \mathbb{R} \times \mathbb{R}_{\geq 0}$ is the Cartesian coordinate of the bottom left corner of the box, and $x_{inc} \in \mathbb{R}_{>0}$ and $t_{inc} \in \mathbb{R}_{>0}$ are the lengths of the sides of the box along the value and time dimensions, respectively. Therefore, a cell g is bounded between $[x, x + x_{inc}]$ along the value dimension and between $[t, t + t_{inc}]$ along the time dimension. A set of cells is indicated with Γ.

We use the *object oriented* dot notation (".") to reference the properties of the region and the cells. The properties of the region R are its bounding values, which are $R.x_{min}$ and $R.x_{max}$ for the value dimension and $R.t_{max}$ for the time dimension. The properties of a cell g are the elements of the tuple, which are $g.x$, $g.t$, $g.x_{inc}$, $g.t_{inc}$, and $g.I_T$ which indicates the time interval $[t, t + t_{inc}]$ of g.

Given a cell g and a set of signals S, we define a Boolean function $\hat{g}(S)$ such that $\hat{g}(S) = \top$ if and only if at least one signal in $s \in S$ has a non-empty intersection with g. Such a cell is informally referred as *covered by* S.

GridTLI uses three thresholds x_t, t_t, and c_t, to determine its fitting and clustering behavior. Specifically, x_t is the *signal threshold* and defines the minimum value of $g.x_{inc}$ for any cell; t_t is the *temporal threshold* and defines the minimum value of $g.t_{inc}$ for any cell; c_t is the *cluster threshold* and is used to determine the number of clusters.

B. Algorithm

GridTLI produces an STL ϕ formula from a set S, made of one-dimensional positive-only signals, and the three thresholds x_t, t_t, and c_t. The formula is generated in four major steps: 1) the signal space is partitioned with a grid; 2) the signals are distributed into separate clusters; 3) simple STL primitives are fit to each cluster; and 4) the clusters are then mapped to the final STL formula. A high level overview of the algorithm is reported in Alg. 1.

[1]For definitions of the semantics of STL defined over signals and the robustness degree of a signal with respect to an STL formula, see [12].

[2]Code available at: https://github.com/CIDARLAB/GridTLI

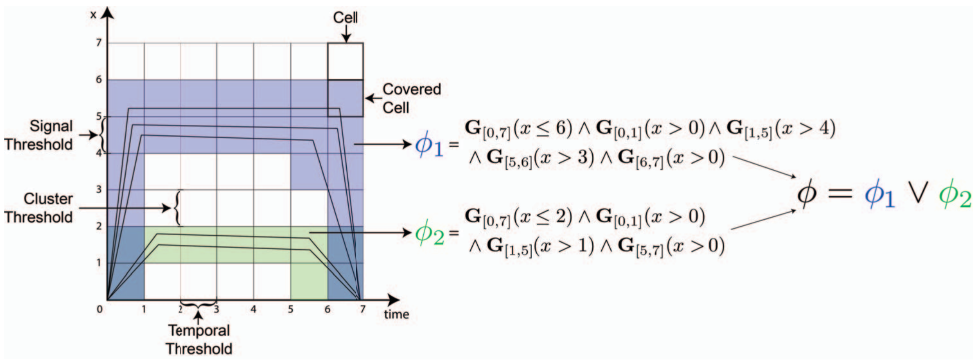
Cell
Covered Cell
Signal Threshold
Cluster Threshold
Temporal Threshold
x
time
$\phi_1 = \mathbf{G}_{[0,7]}(x \le 6) \wedge \mathbf{G}_{[0,1]}(x > 0) \wedge \mathbf{G}_{[1,5]}(x > 4) \wedge \mathbf{G}_{[5,6]}(x > 3) \wedge \mathbf{G}_{[6,7]}(x > 0)$
$\phi_2 = \mathbf{G}_{[0,7]}(x \le 2) \wedge \mathbf{G}_{[0,1]}(x > 0) \wedge \mathbf{G}_{[1,5]}(x > 1) \wedge \mathbf{G}_{[5,7]}(x > 0)$
$\phi = \phi_1 \vee \phi_2$

Fig. 1. Sample grid with cells and clusters. The five signals in this grid are split into two clusters (shaded in green and purple) based on the cluster threshold.

Algorithm 1: GRID-TLI

```
input : $S$, $x_t$, $t_t$, $c_t$
output: $\phi$
1: $\Gamma$ = ConstructGrid($S$, $x_t$, $t_t$)
2: $(S_1, \ldots, S_k)$ := ClusterSignals($S$, $c_t$)
3: $\Gamma_i$ := FindCoveredCells($\Gamma$, $S_i$),  $i = 1, \ldots, k$
4: $\phi_i$ := ClusterToFormula($\Gamma_i$),  $i = 1, \ldots, k$
   $\phi$ := $\bigvee_{i=1}^{k} \phi_i$
```

Algorithm 2: ConstructGrid()

```
input : $S$, $x_t$ , $t_t$
output: $\Gamma$
1: $R$ := FindBoundingRegion($S$)
2: for $i$:=0 to $R.t_{max}$ step $t_t$ do
       for $j$:=0 to $R.x_{max}$ step $x_t$ do
           $\Gamma$.addCell(($i$, $j$, $x_t$, $t_t$))
       for $j$:=0 to $R.x_{min}$ step $-x_t$ do
           $\Gamma$.addCell(($i$, $j$, $x_t$, $t_t$))
3: return $\Gamma$
```

The first step is implemented by `ConstructGrid()` in Alg. 2. This function first derives the bounds x_{min}, x_{max}, and t_{max} of the region of interest R from the location of the signals in S in the space $\mathbb{R} \times \mathbb{R}_{\geq 0}$. [3] The region R is then partitioned into a set of cells Γ using a 2D orthogonal grid with a granularity prescribed by thresholds x_t and t_t.

In the second step, the function `ClusterSignals()` uses the cluster threshold c_t to cluster the signals in S into k subsets $(S_1, S_2, \ldots, S_k)$, where k is at most $|S|$. Two signals $s_i, s_j \in S$ are in the same cluster if for all $t \in [0, t_{max}]$, there exist g_p and g_q, such that $t \in g_p.I_T$ and $\hat{g_p}(s_i) = \top$; $t \in g_q.I_T$ and $\hat{g_q}(s_j) = \top$; and

$$\begin{cases} g_p.x - (g_q.x + g_q.x_{inc}) \leq c_t & \text{if } g_p.x \geq g_q.x, \\ g_q.x - (g_p.x + g_p.x_{inc}) \leq c_t & \text{if } g_q.x > g_p.x. \end{cases} \quad (1)$$

The function `FindCoveredCells()`, simply maps every cluster S_i with a set of cells Γ_i that are covered by the signals in S_i, that is, for all $g \in \Gamma_i$, $\hat{g}(S_i) = \top$.

[3]The values of x_{min} and x_{max} are approximated to multiples of x_t. The value of t_{max} is approximated to a multiple of t_t.

In the fourth step, the function `ClusterToFormula()` maps each set of cells Γ_i to an STL formula ϕ_i. The formula is constructed by finding the top-most and bottom-most covered cells at each time step t, going from 0 to t_{max} in increments of t_t. This process is described in Alg. 3. Some simplifications to the formula are applied during this transformation process (see Example 1). Finally, the complete formula is assembled by disjunction of the formulae obtained for each cluster: $\phi = \phi_1 \vee \phi_2 \vee \ldots \vee \phi_k$.

Algorithm 3: ClusterToFormula()

```
input : $\Gamma_i$
output: STL formula ($\phi_i$)
1: $\phi_i := \top$
2: for $i$:=0 to $t_{max}$ step $t_t$ do
       find $g_m$ s.t. $\forall g \in \Gamma_i$ where $g.t, g_m.t = i, g_m.x \geq g.x$
       $\phi_i := \phi_i \wedge \mathbf{G}_{[i,i+g_m.t_{inc})}(x \leq (g_m.x + g_m.x_{inc}))$
       find $g_n$ s.t. $\forall g \in \Gamma_i$ where $g.t, g_n.t = i$, $g_n.x \leq g.x$
       $\phi_i := \phi_i \wedge \mathbf{G}_{[i,i+g_n.t_{inc})}(x \geq g_n.x)$
3: return $\phi_i$
```

Example 1: Consider the clusters in Fig. 1. In the cluster shaded in green for $t \in [0, 1]$, the covered cell with the minimum x value is $g(0, 0, 1, 1)$. The covered cell with the maximum x value is $g(1, 0, 1, 1)$. These two cells provide lower and upper bounds for the satisfaction region of the signals in the green cluster for $t \in [0, 1]$. This is formally represented in STL as $\phi_a = \mathbf{G}_{[0,1]}(x > 0) \wedge \mathbf{G}_{[0,1]}(x \leq 2)$. Similarly, the STL formula is $\phi_b = \mathbf{G}_{[1,2]}(x > 1) \wedge \mathbf{G}_{[0,1]}(x \leq 2)$ for $t \in [1, 2]$. Thus, for $t \in [0, 2]$, the STL formula for the green cluster is $\phi_a \wedge \phi_b$.

To simplify and reduce the length of the formula for a cluster, the temporal operators with adjacent time horizons and similar linear predicates can be combined. For example, $\mathbf{G}_{[0,1]}(x \leq 2) \wedge \mathbf{G}_{[1,2]}(x \leq 2)$ can be transformed to $\mathbf{G}_{[0,2]}(x \leq 2)$. The STL formula ϕ_1 for the green cluster is $\mathbf{G}_{[0,7]}(x \leq 2) \wedge \mathbf{G}_{[0,1]}(x > 0) \wedge \mathbf{G}_{[1,5]}(x > 1) \wedge \mathbf{G}_{[5,7]}(x > 0)$.

Remark 2: The clustering of signals can only be as precise as x_t. In fact, for any $z \in \mathbb{N}$ and for any $c_t \in \big[(z-1)x_t, zx_t\big)$, all other $c_t' \in \big[(z-1)x_t, zx_t\big)$ will result in the same clustering as that for c_t.

Remark 3: Algorithm 1 operates on one-dimensional signals. If the training set S contains multi-dimensional signals with dimensionality n, we derive the sets S^x, $x \in 1, \ldots, n$, from S by considering only the component s^x of each signal. For each set S^x, we use GridTLI to obtain a formula ϕ^x and the final formula ϕ is constructed by conjunction of all ϕ^x, that is $\phi = \phi^1 \wedge \phi^2 \wedge \cdots \wedge \phi^n$.

C. Properties of the Algorithm

The formula ϕ for S constructed using x_t, t_t, and c_t, will have the following properties:

Lemma 1: $\forall s \in S$, $s \models \phi$.

Lemma 2: For any subformula ϕ_a in ϕ, where $\phi_a \in \{\mathbf{G}_{[t_1,t_2)} p_{f(x)\leq\mu_1}, \mathbf{G}_{[t_1,t_2)} p_{f(x)>\mu_1}\}$, then $t_2 - t_1 \geq t_t$.

Lemma 3: For any two subformulae ϕ_a and ϕ_b within an STL formula ϕ_i for a cluster of cells Γ_i, if $\phi_a \in \{\mathbf{G}_{[t_1,t_2)} p_{f(x)\leq\mu_1}, \mathbf{G}_{[t_1,t_2)} p_{f(x)>\mu_1}\}$ and $\phi_b \in \{\mathbf{G}_{[t_1,t_2)} p_{f(x)\leq\mu_2}, \mathbf{G}_{[t_1,t_2)} p_{f(x)>\mu_2}\}$, then $\mid \mu_1 - \mu_2 \mid \geq x_t$.

Lemma 4: For any two signals $s_i, s_j \in S$ such that $s_i \models \phi_1$ and $s_j \models \phi_2$, where ϕ_1 and ϕ_2 are STL formulae for two separate clusters, then there exists $t \in [0, t_{max}]$ such that no g_p, g_q satisfy Eq. 1.

D. Complexity

In this section, we discuss the worst-case complexity to construct ϕ for a set of one-dimensional signals S using GridTLI. Since the steps of the algorithm are sequential, the overall asymptotic complexity can be obtained as the sum of the complexities of each step.

The complexity of `ConstructGrid()` is the same as the number of elements in Γ, that is:

$$\mid \Gamma \mid = (t_{max}/t_t) \cdot \Big((x_{max} - x_{min}) / x_t \Big) \qquad (2)$$

For the second step (function `ClusterSignals()`), the complexity is $\mid \Gamma \mid \cdot \mid S \mid$ since the clustering condition is checked for all signals and for all cells. The worst case complexity of the third step (function `FindCoveredCells()`) is $\mid \Gamma \mid \cdot \mid S \mid$, since all the cells in Γ could be covered and there can be at most $\mid S \mid$ clusters. The worst case complexity of the final step (function `ClusterToFormula()`) is $\mid S \mid$ since, each signal could form a unique cluster if the signals in S are too diverse or if c_t is too conservative.

The worst-case complexity of algorithm overall is

$$\big((\mid \Gamma \mid) + (\mid \Gamma \mid \cdot \mid S \mid) + (\mid \Gamma \mid \cdot \mid S \mid) + (\mid S \mid) \big) \qquad (3)$$

which can be simplified to $(\mid \Gamma \mid \cdot \mid S \mid)$, since this value dwarfs the other terms in Eq. 3. Using Eq. 2, we can rewrite the worst-case complexity as

$$\mathcal{O}\left(\frac{x_{max} - x_{min}}{x_t} \cdot \frac{t_{max}}{t_t} \cdot \mid S \mid \right). \qquad (4)$$

VI. CASE STUDIES

We present two case studies for GridTLI. Case study VI-A involves measurements of fluorescence from a synthetic genetic circuit in biological experiments with *Escherichia coli*. Case study VI-B is related to a fault detection problem in a fuel control system for a gasoline engine. The dataset corresponding to VI-B was developed and first investigated in [5] using their decision tree-based algorithm. Therefore, some comparisons will be made.

A. Biological Network

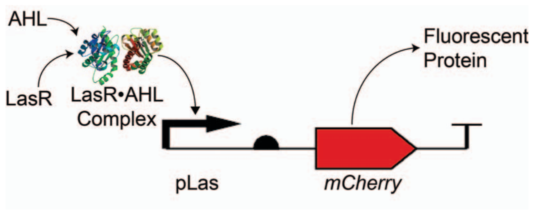

Fig. 2. SBOL Visual representation [13] of a genetic transcriptional unit capable of producing a protein in the presence of a protein complex formed by the small molecule AHL and protein LasR. We consider this protein complex to be the input for this genetic circuit and the produced protein, which can be any gene, the output. In this case, we used *mCherry*, a gene encoding for a red fluorescent protein, for the output. By varying the initial concentrations of AHL and LasR, various output traces can be generated for the expression of pLas. Experiments were conducted over several hours, and cells were harvested for measuring at regular intervals. Traces were generated by repeatedly sampling from the fluorescence measurements and calculating their geometric mean. Initial conditions of the inputs were varied to obtain a spectrum of potential expression levels.

In this case study, we wish to characterize the levels of protein expression for a biological network, such as the one seen in Fig. 2, over time using STL. However, two-class classification cannot be performed in this context because we do not have negative traces. This is largely because there is no practical way to define what data should be labeled as negative. Many biological networks have similar, or overlapping, behaviors, so we cannot, while characterizing one circuit, define the behavior of all other circuits as negative. The utility of GridTLI comes from its lack of reliance on negatively labeled data. We perform GridTLI on these data using a variety of values for the thresholds (x_t, t_t, c_t). We restrict their values using to the following intervals obtained from the data's region: $x_t \in \left(0, \frac{x_{max}-x_{min}}{2}\right], t_t \in \left(0, \frac{t_{max}-t_{min}}{2}\right], c_t \in \left[0, \frac{x_{max}-x_{min}}{2}\right]$. For these data, $\frac{x_{max}-x_{min}}{2} = 1843.5$ and $\frac{t_{max}-t_{min}}{2} = 3.45$, and we sample every 10% from these intervals for x_t and t_t. The values for c_t are chosen based on the given value for x_t, following from Remark 2. For convenience, we choose the lower bound of each interval $\big[(z-1)x_t,\ zx_t\big)$:

$$c_t \in \left\{ (z-1)x_t : \ z \in \mathbb{N} \text{ and } z \leq \frac{x_t + 1843.5}{x_t} \right\}.$$

In order to test the STL output from GridTLI, we employ k-fold cross-validation (CV), with $k = 10$. Since traces in this dataset were gathered from physically different experiments, using different initial conditions, we perform a *stratified* cross-validation by partitioning the data from each experiment evenly among the folds. The complete dataset contains 960 traces, so each training set has 894 traces, while the corresponding testing sets contain the remaining 96.

After fitting the training data, each trace in the dataset will either satisfy or not satisfy the STL formula. By design, every trace in the training set of a given formula will satisfy that formula. Since we only have positive traces, each trace in the testing set will either be a *true positive* (TP) or a *false negative* (FN), respective of whether it satisfies the formula or not. We use the false negative rate (FNR) and robustness degree of testing traces to assess the quality of our STL formulae. The FNR is defined as

$$\mathrm{FNR} = \frac{|\,\mathrm{FN}\,|}{|\,\mathrm{TP}\,| + |\,\mathrm{FN}\,|}.$$

The FNR over all chosen thresholds is between 0 and 0.0115.

We compare the mean robustness values of the false negative traces against the number of operators for each set of thresholds with the aim of identifying ideal threshold values. One of the primary challenges of fitting an STL formula to a dataset with only positive traces is that it is hard to quantify the quality of the fit. When very large thresholds are chosen, the entire region might be marked covered. This leads to an STL formula with fewer operators, higher robustness values, and a lower FNR for the testing set. Yet the formula might not specify any useful information about the characteristics of the behavior of the system that can be captured from the traces. In contrast, very conservative threshold values result in a tight fit, which leads to a higher FNR, lower robustness values, and a larger number of operators in the formula.

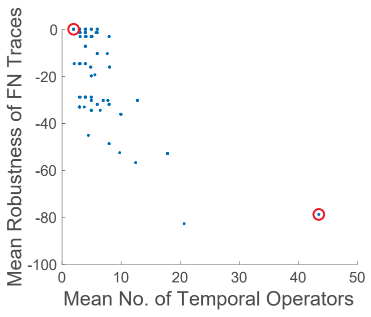

Fig. 3. Scatter plot of mean robustness of false negative traces against average number of operators in the STL formula generated by GridTLI for the biological traces. The data circled in red correspond to the cross-validation sets showcased in Fig. 4.

Fig. 3 shows a scatter plot of the mean robustness values against the number of temporal operators of the STL formula for the different threshold values used in this case study. The left most point marked in red corresponds to the STL formula with the fewest number of temporal operators (shortest formula) and highest mean robustness value. The threshold values for this point were ($x_t = 1843.5$, $t_t = 3.45$, $c_t = 1843.5$), which are the highest threshold values used in this case study. Fig. 4a shows an execution of GridTLI using these threshold values on a training set. The STL formula generated was: $\mathbf{G}_{[0,6.9]}(x \leq 3687) \wedge \mathbf{G}_{[0,6.9]}(x \geq 0)$. The testing dataset yielded 0 false negative rates (and the highest robustness) as illustrated in Fig. 4c. This STL formula does not convey any useful information about the system though. It merely states the upper and lower bounds of the traces over time.

Similarly, for the rightmost point marked in red in Fig.3 which corresponds to the STL formula with the highest number of temporal operators (longest formula) and lowest mean robustness value, the threshold values correspond to the smallest values used in the case study: ($x_t = 184.35$, $t_t = 0.345$, $c_t = 0$). Fig. 4b shows an execution of GridTLI using these low threshold values. It is evident that the covered cells in the region form a very tight fit. However, this leads to a longer STL formula and low robustness values for some signals in the testing set (as shown by the red trace in Fig. 4d.

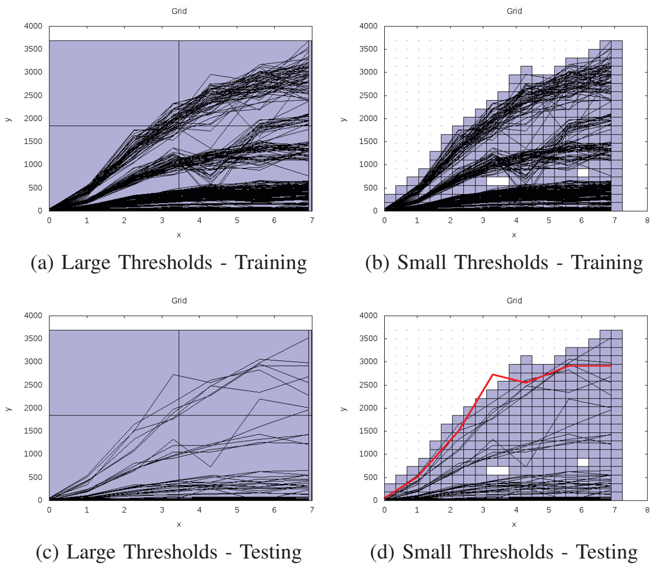

Fig. 4. 4a & 4b show covered cells for different thresholds using GridTLI on the same training data of pLas Expression. 4c & 4d show cross-validation for the testing data. The trace colored red in 4d violates the STL formula learned from the training set in 4b.

B. Fuel Control System

This dataset (composed of 1200 traces, with 600 normal traces and 600 anomalous traces) was constructed by modifying a built-in model from Simulink for the fuel control system of a gasoline engine. Every trace contains 200 samples from two sensors, and examples of traces are shown in Fig. 5.

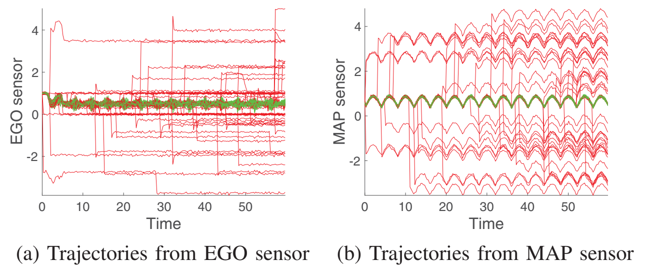

Fig. 5. Example testing set for case study VI-B. Two core sensors provide the feedback information necessary to control the system: the EGO sensor, which reports the amount of residual oxygen present in the exhaust, and the MAP sensor, which reports the (intake) manifold absolute pressure. The normal traces were obtained while the system was operating properly, and the anomalous traces were obtained when a fault was artificially injected into one or both sensors. Every trace in the set consists of a trajectory in (a) and a corresponding trajectory in (b). Trajectories colored *green* are positively labeled traces, and all trajectories in *red* are labeled negative.

As in the first case study, we use 10-fold CV to evaluate the output of GridTLI. We use stratified sampling to insure each partition contains an equal number of positive and negative traces. Although the training sets contain negatively labeled traces, GridTLI uses only the positive traces. The negative traces in the training sets will be used for training in the comparison. Each training set has 1080 traces with a corresponding testing set of 120 traces. Since these traces have two dimensions, GridTLI is run once for each. The resulting STL formula has the form $\phi = \phi^{\text{EGO}} \wedge \phi^{\text{MAP}}$.

Each subformula, ϕ^{EGO} and ϕ^{MAP}, is found using its own set of threshold values, $(x_t^{\text{EGO}}, t_t^{\text{EGO}}, c_t^{\text{EGO}})$ and $(x_t^{\text{MAP}}, t_t^{\text{MAP}}, c_t^{\text{MAP}})$, respectively. We again restrict the possible values for the thresholds using the data: $x_t^{\text{EGO}} \in (0, 0.5512]$, $t_t^{\text{EGO}} \in (0, 29.85]$, $c_t^{\text{EGO}} \in [0, 0.5512]$, $x_t^{\text{MAP}} \in (0, 0.3537]$, $t_t^{\text{MAP}} \in (0, 29.85]$, $c_t^{\text{MAP}} \in [0, 0.3537]$, and sample uniformly from these intervals in the same fashion as the first case study.

Traces in the testing set will now have four possible characterizations: TP or FP (*false positive*), for positive or negative traces, respectively, that satisfy the formula; or FN or TN (*true negative*), for positive or negative traces, respectively, that do not satisfy the formula. This also allows us to use the *misclassification rate* (MCR) of testing traces to evaluate our STL formulae. The MCR is defined as:

$$\text{MCR} = \frac{|\text{ FP }| + |\text{ FN }|}{|\text{ TP }| + |\text{ FP }| + |\text{ FN }| + |\text{ TN }|}.$$

The MCR over all chosen thresholds is between 0.0183 and 0.0342. Fig. 6 shows all MCRs against the number of temporal operators in the corresponding formula.

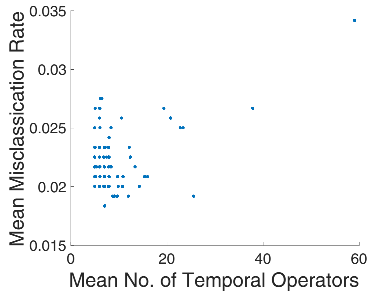

Fig. 6. Scatter plot of mean MCR against mean number of operators in the STL formula generated by GridTLI for the fuel control system data.

In an attempt to find threshold values that can give formulae with acceptable number of operators and effective MCR values, we investigate the threshold values associated with the lowest MCRs and smallest number of temporal operators. In this case, we arbitrarily chose to limit the number of operators to 15 and MCR to 0.05. There are 323 points from Fig. 6 within these limits. The average threshold values for these points are $(x_t^{\text{EGO}} = 0.2211, x_t^{\text{MAP}} = 0.1419, t_t^{\text{EGO}} = 18.22, t_t^{\text{MAP}} = 18.22, c_t^{\text{EGO}} = 0.2514, c_t^{\text{MAP}} = 0.1613)$. We use these threshold values and compare the results of GridTLI with a previous approach. Decision Tree TLI

TABLE I
COMPARISON OF GRIDTLI AND DECISION TREE TLI [5]

	Testing MCR	Operators	Runtime (seconds)
Grid-based	0.0242 (0.0186)	7.1 (0.32)	0.12 (0.015)
Decision tree	0.0217 (0.0131)	4.40 (0.70)	379.80 (33.17)

from [5] was used to learn STL formulae for the same 10-fold CV training/testing sets. Table I shows statistics–in the form 'mean (std)'–used for comparison. *Training FPR* is the false positive rate over all training sets. These are interesting numbers to consider since GridTLI is guaranteed to have zero false negatives among the traces in the training set but provides no guarantee or bound on false positives. *Testing MCR* is the mean MCR over all testing sets, and while the mean MCR for the testing sets of the STL formulae from GridTLI is higher than that of Decision Tree TLI, it is still reasonably low. *Operators* is the mean number of temporal operators over all learned STL formulae, which is significantly higher in GridTLI than in Decision Tree TLI. *Runtime* is the mean algorithm runtime in seconds over all training sets, where we see that GridTLI drastically outperforms Decision Tree TLI.

VII. CONCLUSION

This paper presents two case studies, a biological network (where only positive traces were available) and a fuel control system (where both positive and negative traces were available). GridTLI generated efficient STL formulae with acceptable misclassification rates for the testing data. We showed that GridTLI is an effective algorithm that works well, especially when only positive data is available, and is fast (over 3000 times faster than a previous two-class approach) with results comparable to an inference algorithm that uses a decision-tree approach.

Future work includes optimizing, and automating, the choice of threshold values by dividing available data into two sets: one for optimization and another for testing.

REFERENCES

[1] C. Baier *et al.*, *Principles of Model Checking*. MIT press, 2008.
[2] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.
[3] X. Jin *et al.*, "Mining Requirements from Closed-Loop Control Models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2015.
[4] Z. Kong *et al.*, "Temporal Logic Inference for Classification and Prediction from Data," in *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '14. New York, NY, USA: ACM, 2014, pp. 273–282.
[5] G. Bombara *et al.*, "A Decision Tree Approach to Data Classification Using Signal Temporal Logic," in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '16. New York, NY, USA: ACM, 2016, pp. 1–10.
[6] V. Chandola *et al.*, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.
[7] T. Knight, "Idempotent vector design for standard assembly of biobricks," DTIC Document, Tech. Rep., 2003.
[8] E. Weber *et al.*, "A modular cloning system for standardized assembly of multigene constructs," *PloS one*, vol. 6, no. 2, p. e16765, 2011.
[9] B. Hoxha *et al.*, "Mining parametric temporal logic properties in model-based design for cyber-physical systems," *International Journal on Software Tools for Technology Transfer*, pp. 1–15, Feb. 2017.
[10] A. Jones *et al.*, "Anomaly detection in cyber-physical systems: A formal methods approach," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference On*. IEEE, 2014, pp. 848–853.
[11] E. Bartocci *et al.*, "Data-driven statistical learning of temporal logic properties," in *Formal Modeling and Analysis of Timed Systems*. Springer, 2014, pp. 23–37.
[12] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2010, pp. 92–106.
[13] J. Quinn *et al.*, "Synthetic biology open language visual (SBOL Visual), version 1.0. 0," Tech. Rep., 2013.