UNIVERSITÀ DEGLI STUDI DI TRIESTE

A robust genetic algorithm for learning temporal specifications from data

This is the peer reviewed (post-print) version of the following article:

Original
A robust genetic algorithm for learning temporal specifications from data / Nenzi, L., Silvetti, S., Bartocci, E., Bortolussi, L.. - ELETTRONICO. - 11024:(2018), pp. 323-338. (15th International Conference on Quantitative Evaluation of Systems, QEST 2018 Pechino, Cina 2018) [10.1007/978-3-319-99154-2_20].

Availability:
This version is available at: 11368/2931393 since: 2018-11-19T15:43:08Z

Publisher:
Springer Verlag

Published
DOI:10.1007/978-3-319-99154-2_20

Terms of use:

Some rights reserved. More details in the PostPrint rights section below.

License:
Copyright Editore

Publisher copyright

-.-

(Article begins on next page)

23 October 2026

arXiv:1711.06202v3 [cs.AI] 1 Aug 2018

A Robust Genetic Algorithm for Learning Temporal Specifications from Data

Laura Nenzi[1], Simone Silvetti[2,3], Ezio Bartocci[1], and Luca Bortolussi[4]

[1] TU Wien, Vienna, Austria
[2] DIMA, University of Udine, Udine, Italy
[3] Esteco S.p.A., Trieste, Italy
[4] DMG, University of Trieste, Trieste, Italy

Abstract. We consider the problem of mining signal temporal logical requirements from a dataset of regular (good) and anomalous (bad) trajectories of a dynamical system. We assume the training set to be labeled by human experts and that we have access only to a limited amount of data, typically noisy.
We provide a systematic approach to synthesize both the syntactical structure and the parameters of the temporal logic formula using a two-steps procedure: first, we leverage a novel evolutionary algorithm for learning the structure of the formula; second, we perform the parameter synthesis operating on the statistical emulation of the average robustness for a candidate formula w.r.t. its parameters. We compare our results with our previous work [9] and with a recently proposed decision-tree [8] based method. We present experimental results on two case studies: an anomalous trajectory detection problem of a naval surveillance system and the characterization of an Ineffective Respiratory effort, showing the usefulness of our work.

1 Introduction

Learning temporal logic requirements from data is an emergent research field gaining momentum in the rigorous engineering of cyber-physical systems. Classical machine learning methods typically generate very powerful black-box (statistical) models. However, these models often do not help in the comprehension of the phenomenon they capture. Temporal logic provides a precise formal specification language that can easily be interpreted by humans. The possibility to describe datasets in a concise way using temporal logic formulas can thus help to better clarify and comprehend which are the emergent patterns for the system at hand. A clearcut example is the problem of anomaly detection, where the input is a set of trajectories describing regular or anomalous behaviors, and the goal is to learn a classifier that not only can be used to detect anomalous behaviors at runtime, but also gives insights on what characterizes an anomalous behavior. Learning temporal properties is also relevant in combination with state of the art techniques for search-based falsification of complex closed-loop systems [6,22,23,28], as it can provide an automatic way to describe desired (or unwanted behaviors) that the system needs to satisfy.

In this paper, we consider the problem of learning a temporal logic specification from a set of trajectories which are labeled by human experts (or by any other method

which is not usable for real-time monitoring) as "good" for the normal expected behaviors and "bad" for the anomalous ones. The goal is to automatically synthesize both the structure of the formula and its parameters providing a temporal logic classifier that can discriminate as much as possible the bad and the good behaviors. This specification can be turned into a monitor that output a positive verdict for good behaviors and a negative verdict for bad ones.

Related Work Mining temporal logic requirements is an emerging field of research in the analysis of cyber-physical systems (CPS) [2, 5, 7–9, 14, 16, 17, 20, 26, 27]. This approach is orthogonal to active automata learning (AAL) such as $L*$ Angluin's algorithm [3] and its recent variants [15, 25]. AAL is suitable to capture the behaviours of black-box reactive systems and it has been successfully employed in the field of CPS to learn how to interact with the surrounding environments [10, 13]. Mining temporal logic requirements has the following advantages with respect to AAL. The first is that it does not require to interact with a reactive system. AAL needs to query the system in order to learn a Mealy machine representing the relation between the input provided and the output observed. Mining temporal logic requirements can be applied directly to a set of observed signals without the necessity to provide an input. The second is the possibility to use temporal logic requirements within popular tools (such as Breach [12] and S-TaLiRo [4]) for monitoring and falsification analysis of CPS models.

Most of the literature related to temporal logic inference from data focuses in particular on the problem of learning the optimal parameters given a specific template formula [5, 7, 14, 16, 20, 26, 27]. In [5], Asarin et al. extend *the Signal Temporal Logic* (STL) [18] with the possibility to express the time bounds of the temporal operators and the constants of the inequalities as parameters. They also provide a geometric approach to identify the subset of the parameter space that makes a particular signal to satisfy an STL specification. Xu et al. have recently proposed in [26] a temporal logic framework called *CensusSTL* for multi-agent systems that consists of *an outer logic STL formula* with a variable in the predicate representing the number of agents satisfying *an inner logic STL formula*. In the same paper the authors propose also a new inference algorithm similar to [5] that given the templates for both the *inner* and *outer* formulas, searches for the optimal parameter values that make the two formulas capturing the trajectory data of a group of agents. In [14] the authors use the same parametric STL extension in combination with the quantitative semantics of STL to perform a counter-example guided inductive parameter synthesis. This approach consists in iteratively generating a counterexample by executing a falsification tool for a template formula. The counterexample found at each step is then used to further refine the parameter set and the procedure terminates when no other counterexamples are found. In general, all these methods, when working directly with raw data, are potentially vulnerable to the noise of the measurements and they are limited by the amount of data available.

Learning both the structure and the parameters of a formula from a dataset poses even more challenges [7–9, 17]. This problem is usually addressed in two steps, learning the structure of the formula and synthesizing its parameters. In particular, in [17]

the structure of the formula is learned by exploring a directed acyclic graph and the method exploits *Support Vector Machine* (SVM) for the parameter optimization. In [8] the authors use instead a *decision tree* based approach for learning the formula, while the optimality is evaluated using heuristic impurity measures.

In our previous works [7, 9] we have also addressed the problem of learning both the structure and the parameters of a temporal logic specification from data. In [7] the structure of the formula is learned using a heuristics algorithm, while in [9] using a genetic algorithm. The synthesis of the parameters is instead performed in both cases exploiting the *Gaussian Process Upper Confidence Bound* (GP-UCB) [24] algorithm, statistically emulating the satisfaction probability of a formula for a given set of parameters. In both these methodologies, it is required to learn first a statistical model from the training set of trajectories. However, the statistical learning of this model can be a very difficult task and this is one of the main reason for proposing our new approach.

Our contribution In this work, we consider the problem of mining the formula directly from a data set without requiring to learn a generative model from data. To achieve this goal, we introduce a number of techniques to improve the potentials of the genetic algorithm and to deal with the noise in the data in the absence of an underlying model.

First, instead of using the probability satisfaction as evaluator for the best formula, we design a discrimination function based on the quantitative semantics (or robustness) of STL and in particular the average robustness introduced in [6]. The average robustness enables us to differentiate among STL classifiers that have similar discriminating performance with respect to the data by choosing the most robust one. This gives us more information than just having the probability of satisfaction: for each trajectory, we can evaluate how much is it closed to the "boundary" of the classifier, instead of only checking whether it satisfies or not a formula. We then modify the discrimination function and the GP-UCB algorithm used in [7] and [9] to better deal with noisy data and to use the quantitative semantics to emulate the average robustness distribution with respect to the parameter space of the formula.

Second, we reduce the computational cost of the *Evolutionary Algorithm* (EA) by using a lightweight configuration (i.e., a low threshold of max number of iterations) of the GP-UCB optimization algorithm to estimate the parameters of the formulas at each generation. The EA algorithm generates, as a final result, an STL formula tailored for classification purpose.

We compare our framework with our previous methodology [9] and with the decision-tree based approach presented in [8] on an anomalous trajectory detection problem of a naval surveillance and on an Assisted Ventilation in Intensive Care Patients. Our experiments indicate that the proposed approach outperforms our previous work with respect to accuracy and show that it produces in general more compact, and easy to read, temporal logic specifications with respect to the decision-tree based approach with a comparable speed and accuracy.

Paper structure The rest of the paper is organized as follows: in the next section we present the *Signal Temporal Logic* and its robust semantics. We then introduce the prob-

lem considered in Section 3. In section 4, we describe our approach. The results are presented in Section 5. Finally, we conclude the paper in Section 6, by discussing the implications of our contribution, both from the practical and the methodological aspect and some directions of improvement.

2 Signal Temporal Logic

STL *Signal Temporal Logic* (STL) [18] is a formal specification language to express temporal properties over real-values trajectories with dense-time interval. For the rest of the paper, let be $\mathbf{x} : \mathbb{T} \to \mathbb{R}^n$ a trace/trajectory, where $\mathbb{T} = \mathbb{R}_{\geq 0}$ is the time domain, $x_i(t)$ is the value at time t of the projection on the ith coordinate, and $\mathbf{x} = (x_1, ..., x_n)$, as an abuse on the notation, is used also to indicate the set of variables of the trace considered in the formulae.

Definition 1 (STL syntax). *The syntax of STL is given by*

$$\varphi := \top \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \, \mathcal{U}_I \, \varphi_2$$

where $\top$ is the Boolean true constant, μ is an atomic proposition, inequality of the form $y(x) > 0$ ($y : \mathbb{R}^n \to \mathbb{R}$), negation $\neg$ and conjunction $\wedge$ are the standard Boolean connectives, and $\mathcal{U}_I$ is the Until temporal modality, where I is a real positive interval. As customary, we can derive the disjunction operator $\vee$ and the future eventually $\mathcal{F}_I$ and always $\mathcal{G}_I$ operators from the until temporal modality ($\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\mathcal{F}_I\varphi = \top\mathcal{U}_I\varphi$, $\mathcal{G}_I\varphi = \neg\mathcal{F}_I\neg\varphi$).

STL can be interpreted over a trajectory $\mathbf{x}$ using a qualitative (yes/no) or a quantitative (real-value) semantics [11, 18]. We report here only the quantitative semantics and we refer the reader to [11, 18, 19] for more details.

Definition 2 (STL Quantitative Semantics). *The* quantitative satisfaction function ρ *returns a value $\rho(\varphi, \mathbf{x}, t) \in \bar{\mathbb{R}}$,*[5] *quantifying the robustness degree (or satisfaction degree) of the property φ by the trajectory $\mathbf{x}$ at time t. It is defined recursively as follows:*

$$\begin{aligned}
\rho(\top, \mathbf{x}, t) &= +\infty \\
\rho(\mu, \mathbf{x}, t) &= y(\mathbf{x}(t)) \text{ where } \mu \equiv y(\mathbf{x}(t)) \geq 0 \\
\rho(\neg\varphi, \mathbf{x}, t) &= -\rho(\varphi, \mathbf{x}, t) \\
\rho(\varphi_1 \wedge \varphi_2, \mathbf{x}, t) &= \min(\rho(\varphi_1, \mathbf{x}, t), \rho(\varphi_2, \mathbf{x}, t)) \\
\rho(\varphi_1 \, \mathcal{U}_{[a,b]} \varphi_2, \mathbf{x}, t) &= \sup_{t' \in t+[a,b]} (\min(\rho(\varphi_2, \mathbf{x}, t'), \inf_{t'' \in [t,t')} (\rho(\varphi_1, \mathbf{x}, t''))))
\end{aligned}$$

Moreover, we let $\rho(\varphi, \mathbf{x}) := \rho(\varphi, \mathbf{x}, 0)$.

[5] $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$.

The sign of $\rho(\varphi, \mathbf{x})$ provides the link with the standard Boolean semantics of [18]: $\rho(\varphi, \mathbf{x}) > 0$ only if $\mathbf{x} \models \varphi$, while $\rho(\varphi, \mathbf{x}) < 0$ only if $\mathbf{x} \not\models \varphi$[6]. The absolute value of $\rho(\varphi, \mathbf{x})$, instead, can be interpreted as a measure of the robustness of the satisfaction with respect to noise in signal $\mathbf{x}$, measured in terms of the maximal perturbation in the secondary signal $y(\mathbf{x}(t))$, preserving truth value. This means that if $\rho(\varphi, \mathbf{x}, t) = r$ then for every signal $\mathbf{x}'$ for which every secondary signal satisfies $\max_t |y_j(t) - y'_j(t)| < r$, we have that $\mathbf{x}(t) \models \varphi$ if and only if $\mathbf{x}'(t) \models \varphi$ (*correctness property*).

PSTL *Parametric Signal Temporal Logic* [5] is an extension of STL that parametrizes the formulas. There are two types of formula parameters: temporal parameters, corresponding to the time bounds in the time intervals associated with temporal operators (e.g. $a, b \in \mathbb{R}_{\geq 0}$, with $a < b$, s.t. $\mathcal{F}_{[a,b]}\mu$), and the threshold parameters, corresponding to the constants in the inequality predicates (e.g., $k \in \mathbb{R}, \mu = x_i > k$, where x_i is a variable of the trajectory). In this paper, we allow only atomic propositions of the form $\mu = x_i \bowtie k$ with $\bowtie \in \{>, \leq\}$. Given an STL formula φ, let $\mathbb{K} = (\mathcal{T} \times \mathcal{C})$ be the *parameter space*, where $\mathcal{T} \subseteq \mathbb{R}^{n_t}_{\geq 0}$ is the temporal parameter space (n_t being the number of temporal parameters), and $\mathcal{C} \subseteq \mathbb{R}^{n_k}$ is the threshold parameter space (n_k being the number of threshold parameters). A $\boldsymbol{\theta} \in \mathbb{K}$ is a parameter configuration that induces a corresponding STL formula; e.g., $\varphi = \mathcal{F}_{[a,b]}(x_i > k), \boldsymbol{\theta} = (0, 2, 3.5)$ then $\varphi_{\boldsymbol{\theta}} = \mathcal{F}_{[0,2]}(x_i > 3.5)$.

Stochastic Robustness Let us consider an unknown stochastic process $(\mathbf{X}(t))_{t \in T} = (X_1(t), ..., X_n(t))_{t \in T}$, where each vector $\mathbf{X}(t)$ corresponds to the state of the system at time t. For simplicity, we indicate the stochastic model with $\mathbf{X}(t)$. $\mathbf{X}(t)$ is a measurable also as a random variable $\mathbf{X}$ on the space D-valued *cadlag functions* $\mathcal{D}([0, \infty), D)$, here denoted by $\mathcal{D}$, assuming the domain $\mathbb{D}$ to be fixed. It means that the set of trajectories $\mathbf{x}$ of the stochastic process $\mathbf{X}$ is the set $\mathcal{D}$. Let us consider now an STL formula φ, with predicates interpreted over state variables of $\mathbf{X}$. Given a trajectory $\mathbf{x}(t)$ of a stochastic system, its robustness $\rho(\varphi, \mathbf{x}, 0)$ is a measurable functional R_φ [6] from the trajectories in $\mathcal{D}$ to $\mathbb{R}$ which defines the real-valued random variable $R_\varphi = R_\varphi(\mathbf{X})$ with probability distribution:

$$\mathbb{P}\left(R_\varphi(\mathbf{X}) \in [a, b]\right) = \mathbb{P}\left(\mathbf{X} \in \{\mathbf{x} \in \mathcal{D} \mid \rho(\varphi, \mathbf{x}, 0) \in [a, b]\}\right).$$

Such distribution of robustness degrees can be summarized by the average robustness degree. Fixing the stochastic process $\mathbf{X}$, $\mathbb{E}(R_\varphi|\mathbf{X})$, it gives a measure of how strongly a formula is satisfied. The satisfaction is more robust when this value is higher. In this paper, we will approximate this expectation by Monte Carlo sampling.

[6] The case $\rho(\varphi, \mathbf{x}) = 0$, instead, is a borderline case, and the truth of φ cannot be assessed from the robustness degree alone.

3 Problem formulation

In this paper, we focus our attention on learning the best property (or set of properties) that discriminates trajectories belonging to two different classes, say good and bad, starting from a labeled dataset of observed trajectories. Essentially, we want to tackle a supervised two-class classification problem over trajectories, by learning a temporal logic discriminant, describing the temporal patterns that better separate two sets of observed trajectories.

The idea behind this approach is that there exists an unknown procedure that, given a temporal trajectory, is able to decide if the signal is good or bad. This procedure can correspond to many different things, e.g., to the reason of the failure of a sensor that breaks when it receives certain inputs. In general, as there may not be an STL specification that perfectly explains/mimics the unknown procedure, our task is to approximate it with the most effective one.

Our approach works directly with observed data, and avoids the reconstruction of an intermediate generative model $p(\mathbf{x}|z)$ of trajectories x conditioned on their class z, as in [7, 9]. The reason is that such models can be hard to construct, even if they provide a powerful regularization, as they enable the generation of an arbitrary number of samples to train the logic classifier.

In a purely data-driven setting, to build an effective classifier, we need to consider that training data is available in limited amounts and it is typically noisy. This reflects in the necessity of finding methods that guarantee good generalization performance and avoid overfitting. In our context, overfitting can result in overly complex formulae, de facto encoding the training set itself rather than guessing the underlying patterns that separate the trajectories. This can be faced by using a score function based on robustness of temporal properties, combined with suitably designed regularizing terms.

We want to stress that the approach we present here, due to the use of the average robustness of STL properties, can be easily tailored to different problems, like finding the property that best characterise a single set of observations.

4 Methodology

Learning an STL formula can be separated in two optimization problems: the learning of the formula structure and the synthesis of the formula parameters. The structural learning is treated as a discrete optimization problem using an *Genetic Algorithm* (GA); the parameter synthesis, instead, considers a continuous parameter space and exploits an active learning algorithm, called *Gaussian Process Upper Confidence Bound (GP-UCB)*. The two techniques are not used separately but combined together in a bi-level optimization. The GA acts externally by defining a set of template formulae. Then, the GP-UCB algorithm, which acts at the inner level, finds the best parameter configuration such that each template formula better discriminates between the two datasets. For both optimizations, we need to define a score function to optimize, encoding the criterion to discriminate between the two datasets.

Algorithm 1 ROGE – RObustness GEnetic

Require: $\mathcal{D}_p, \mathcal{D}_n, \mathbb{K}, Ne, Ng, \alpha, s$
1: $gen \leftarrow$ GENERATEINITIALFORMULAE(Ne, s)
2: $gen_\Theta \leftarrow$ LEARNINGPARAMETERS$(gen, G, \mathbb{K})$
3: **for** $i = 1 \ldots Ng$ **do**
4: $\quad subg_\Theta \leftarrow$ SAMPLE(gen_Θ, F)
5: $\quad newg \leftarrow$ EVOLVE$(subg_\Theta, \alpha)$
6: $\quad newg_\Theta \leftarrow$ LEARNINGPARAMETERS$(newg, G, \mathbb{K})$
7: $\quad gen_\Theta \leftarrow$ SAMPLE$(newg_\Theta \cup gen_\Theta, F)$
8: **end for**
9: **return** gen_Θ

Our implementation, called *RObustness GEnetic* (ROGE) algorithm is described in Algorithm 1. First, we give an overview of it and then we described each specific function in the next subsections. The algorithm requires as input the dataset $\mathcal{D}_p$(good) and $\mathcal{D}_n$(bad), the parameter space $\mathbb{K}$, with the bound of each considered variable, the size of the initial set of formulae Ne, the number of maximum iterations Ng, the mutation probability α and the maximum initial formula size s. The algorithm starts generating (line 1) an initial set of PSTL formulae, called gen. For each of these formulae (line 2), the algorithm learns the best parameters accordingly to a discrimination function G. We call gen_Θ the generation for which we know the best parameters of each formula. During each iteration, the algorithm (line 4), guided by a fitness function F , extracts a subset $subg_\Theta$ composed by the best $Ne/2$ formulae, from the initial set gen_Θ. From this subset (line 5), a new set $newg$ composed of Ne formulae is created by employing the EVOLVE routine, which implements two *genetic* operators. Then (line 6), as in line 2, the algorithm identifies the best parameters of each formula belonging to $newg$. The new generation $newg_\Theta$ and the old generation gen_Θ are then merged together (line 7). From this set of $2Ne$ formulae the algorithm extracts, with respect to the fitness function F, the new generation gen_Θ of Ne formulae. At the end of the iterations or when the STOP criterion is true (lines 8-12), the algorithm returns the last generated formulae. The best formula is the one with the highest value of the discrimination function, i.e., $\varphi_{best} = \text{argmax}_{\varphi_{\boldsymbol{\theta}} \in gen_\Theta}(G(\varphi_{\boldsymbol{\theta}}))$. We describe below in order: the discrimination function algorithm, the learning of the parameters of a formula template and the learning of the formula structure.

4.1 Discrimination Function $G(\varphi)$

We have two datasets $\mathcal{D}_p$ and $\mathcal{D}_n$ and we search for the formula φ that better separates these two classes. We define a function able to discriminate between this two datasets, such that maximising this *discrimination function* corresponds to finding the best formula classifier. In this paper, we decide to use, as evaluation of satisfaction of each formula, the quantitative semantics of STL. As described in Section 2, this semantics computes a real-value (robustness degree) instead of only a yes/no answer.

Given a dataset $\mathcal{D}_i$, we assume that the data comes from an unknown stochastic process $\mathbf{X}_i$. The process in this case is like a black-box for which we observe only a subset of trajectories, the dataset $\mathcal{D}_i$. We can then estimate the averages robustness $\mathbb{E}(R_\varphi|\mathbf{X}_i)$ and its variance $\sigma^2(R_\varphi|\mathbf{X}_i)$, averaging over $\mathcal{D}_i$.

To discriminate between the two dataset $\mathcal{D}_p$ and $\mathcal{D}_n$, we search the formula φ that maximizes the difference between the average robustness of $\mathbf{X}_p$, $\mathbb{E}(R_\varphi|\mathbf{X}_p)$, and the average robustness of $\mathbf{X}_n$, $\mathbb{E}(R_\varphi|\mathbf{X}_n)$ divided by the sum of the respective standard deviation:

$$G(\varphi) = \frac{\mathbb{E}(R_\varphi|\mathbf{X}_p) - \mathbb{E}(R_\varphi|\mathbf{X}_n)}{\sigma(R_\varphi|\mathbf{X}_p) + \sigma(R_\varphi|\mathbf{X}_n)}. \tag{1}$$

This formula is proportional to the probability that a new point sampled from the distribution generating $\mathbf{X}_p$ or $\mathbf{X}_n$, will belong to one set and not to the other. In fact, an higher value of $G(\varphi)$ implies that the two average robustness will be sufficiently distant relative to their length-scale, given by the standard deviation σ.

As said above, we can evaluate only a statistical approximation of $G(\varphi)$ because we have a limited number of samples belonging to $\mathbf{X}_p$ and $\mathbf{X}_n$. We will see in the next paragraph how we tackle this problem.

4.2 GB-UCP: learning the parameters of the formula

Given a formula φ and a parameter space $\mathbb{K}$, we want to find the parameter configuration $\boldsymbol{\theta} \in \mathbb{K}$ that maximises the score function $g(\boldsymbol{\theta}) := G(\varphi_{\boldsymbol{\theta}})$, considering that the evaluations of this function are noisy. So, the question is how to best estimate (and optimize) an unknown function from observations of its value at a finite set of input points. This is a classic non-linear non-convex optimization problem that we tackle by means of the GP-UCB algorithm [24]. This algorithm interpolates the noisy observations using a stochastic process (a procedure called emulation in statistics) and optimize the emulated function using the uncertainty of the fit to determine regions where the true maximum can lie. More specifically, the GP-UCB bases its emulation phase on Gaussian Processes, a Bayesian non-parametric regression approach [21], adding candidate maximum points to the training set of the GP in an iterative fashion, and terminating when no improvement is possible (see [24] for more details).

After this optimization, we have found a formula that separates the two datasets, not from the point of view of the satisfaction (yes/no) of the formula but from the point of view of the robustness value. In other words, there is a threshold value α such that $\mathbb{E}(R_\varphi|\mathbf{X}_p) > \alpha$ and $\mathbb{E}(R_\varphi|\mathbf{X}_n) \leq \alpha$. Now, we consider the new STL formula φ' obtained translating the atomic predicates of φ by α (e.g., $y(x) > 0$ becomes $y(x) - \alpha > 0$). Taking into account that the quantitative robustness is achieved by combination of $\min$, $\max$, $\inf$ and $\sup$, which are linear algebraic operators with respect to translations (e.g, $\min(x, y) \pm c = \min(x \pm c, y \pm c)$), we easily obtain that $\mathbb{E}(R_{\varphi'}|\mathbf{X}_p) > 0$ and $\mathbb{E}(R_{\varphi'}|\mathbf{X}_b) < 0$. Therefore, φ' will divide the two datasets also from the point of view of the satisfaction.

4.3 Genetic Algorithm: learning the structure of the formula

To learn the formula structure, we exploit a modified version of the Genetic Algorithm (GA) presented in [9]. GAs belongs to the larger class of evolutionary algorithms (EA). They are used for search and optimization problems. The strategy takes inspiration from the genetic area, in particular in the selection strategy of species. Let us see now in detail the genetic routines of the ROGE algorithm.

$gen =$ **GENERATEINITIALFORMULAE**(Ne, s)**.** This routine generates the initial set of Ne formulae. A fraction $n_l < Ne$ of this initial set is constructed by a subset of the temporal properties: $\mathcal{F}_I\mu$, $\mathcal{G}_I\mu$, $\mu_1\mathcal{U}_I\mu_2$, where the atomic predicates are of the form $\mu = (x_i > k)$ or $\mu = (x_i \leq k)$ or simple boolean combinations of them (e.g. $\mu = (x_i > k_i) \wedge (x_j > k_j)$). The size of this initial set is exponential accordingly to the number of considered variables x_i. If we have few variables we can keep all the generated formulae, whereas if the number of variables is large we consider only a random subset. The remain $n_r = Ne - n_l$ formulae are chosen randomly from the set of formulae with a maximum size defined by the input parameter s. This size can be adapted to have a wider range of initial formulae.

$subg_\Theta =$ **SAMPLE**(gen_Θ, F)**.** This procedure extracts from gen_Θ a subset $subg_\Theta$ of $Ne/2$ formulae, according to a fitness function $F(\varphi) = G(\varphi) - S(\varphi)$. The first factor $G(\varphi)$ is the dicrimination function previously defined and $S(\varphi)$ is a size penalty, i.e. a function penalizes formulae with large sizes. In this paper, we consider $S(\varphi) = g \cdot p^{size(\varphi)}$, where p is heuristically set such that $p^5 = 0.5$, i.e. formulae of size 5 get a 50% penalty, and g is adaptively computed as the average discrimination in the current generation. An alternative choice of p can be done by cross-validation.

$newg =$ **EVOLVE**$(subg_\Theta, \alpha)$**.** This routine defines a new set of formulae ($newg$) starting from $subg_\Theta$, exploiting two *genetic* operators: the *recombination* and the *mutation* operator. The recombination operator takes as input two formulae φ_1, φ_2 (the parents), it randomly chooses a subtree from each formula and it swaps them, i.e. it assigns the subtree of φ_1 to φ_2 and viceversa. As a result, the two generated formulae (the children) share different subtrees of the parents' formulae. The mutation operator takes as input one formula and induce a random change (such as inequality flip, temporal operator substitution, etc.) on a randomly selected node of its tree-structure. The purpose of the genetic operators is to introduce innovation in the offspring population which leads to the optimization of a target function (G in this case). In particular, recombination is related to exploitation, whereas mutation to exploration. The EVOLVE routine implements an iterative loop that at each iteration selects which genetic operators to apply. A number $p \in [0, 1]$ is randomly sampled. If its value is lower than the mutation probability, i.e., $p \leq \alpha$, then the mutation is selected, otherwise a recombination is performed. At this point the input formulae of the selected genetic operator are chosen randomly between the formulas composing $subg_\Theta$ and the genetic operators are applied. In our implementation, we give more importance to the exploitation; therefore the mutation

acts less frequently than the recombination (i.e., $\alpha \leq 0.1$). The iteration loops will stop when the number of generated formula is equal to Ne, i.e. twice the cardinality of $subg_{\Theta}$.

5 Case studies and Experimental Results

5.1 Maritime Surveillance

As first case study, We consider the maritime surveillance problem presented in [8] to compare our framework with their *Decision Tree* (DTL4STL) approach. The experiments with the DTL4STL approach were implemented in `Matlab`, the code is available at [1]. We also test our previous implementation presented in [9] with the same benchmark. Both the new an the previous learning procedure were implemented in `Java` (JDK 1.8_0) and run on a Dell XPS, Windows 10 Pro, Intel Core i7-7700HQ 2.8 GHz, 8GB 1600 MHz memory.

The synthetic dataset of naval surveillance reported in [8] consists of 2-dimensional coordinates traces of vessels behaviours. It considers two kind of anomalous trajectories and regular trajectories, as illustrated in Fig. 1. The dataset contains 2000 total trajectories (1000 normal and 1000 anomalous) with 61 sample points per trace. We

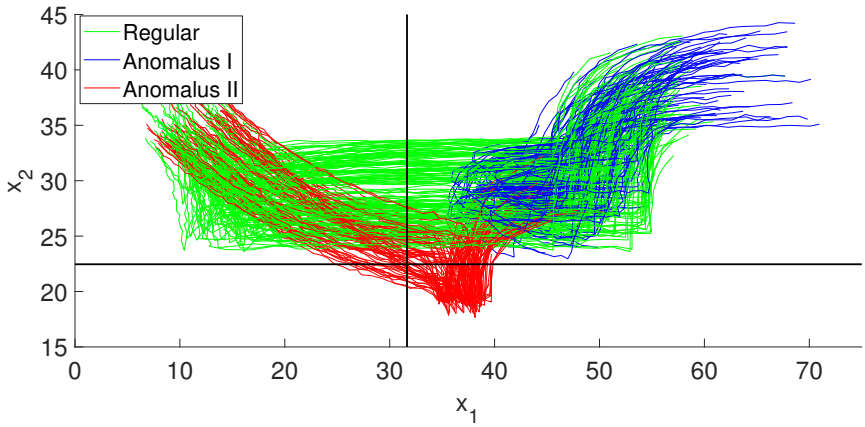

Fig. 1: The regular trajectories are represented in green. The anomalous trajectories which are of two kinds, are represented respectively in blue and red.

run the experiments using a 10-fold cross-validation in order to collect the mean and variance of the misclassified trajectories of the validation set. Results are summarized in Table 1, where we report also the average execution time. We test also our previous implementation [9] without a generative model from data. It performs so poorly on

	ROGE	DTL4STL	DTL4STL$_p$
Mis. Rate	0	0.01 ± 0.013	0.007 ± 0.008
Comp. Time	73 ± 18	144 ± 24	-

Table 1: The comparison of the computational time (in sec.), the mean missclassification rate and its standard deviation between the learning procedure using the RObust GEnetic algorithm, the Decision-Tree (DTL4STL) Matlab code provided by the authors and the results reported in [8] (DTL4STL$_p$).

the chosen benchmark that is not meaningful to report it: the misclassification rate on the validation set is around 50%. In Table 1, we also report DTL4STL$_p$ the DTL4STL performance declared in [8], but we were not be able to reproduce them in our setting.

In terms of accuracy our approach is comparable with respect to the performance of the DTL4STL. In terms of computational cost, the decision tree approach is slightly slower but it is implemented in `Matlab` rather than `Java`.

An example of formula that we learn with ROGE is

$$\varphi = ((x_2 > 22.46)\,\mathcal{U}_{[49,287]}\,(x_1 \leq 31.65)) \tag{2}$$

The formula (2) identifies all the regular trajectories, remarkably resulting in a misclassification error equal to zero, as reported in Table 1. The red anomalous trajectories falsify the predicate $x_2 > 22.46$ before verifying $x_1 \leq 31.65$, on the contrary the blue anomalous trajectories globally satisfy $x_2 > 22.46$ but never verify $x_1 \leq 31.65$ (consider that all the vessels start from the top right part of the graphic in Figure 1). Both these conditions result in the falsification of Formula(2), which on the contrary is satisfied by all the regular trajectories. In Figure 1, we have reported the threshold $x_2 = 22.46$ and $x_1 = 31.65$.

An example instead of formula found by the DTL4STL tool using the same dataset is the following:

$$\begin{aligned}\psi = (((\mathcal{G}_{[187,196)}x_1 < 19.8) \wedge (\mathcal{F}_{[55.3,298)}x_1 > 40.8)) \vee ((\mathcal{F}_{[187,196)}x_1 > 19.8) \wedge \\ ((\mathcal{G}_{[94.9,296)}x_2 < 32.2) \vee ((\mathcal{F}_{[94.9,296)}x_2 > 32.2) \wedge (((\mathcal{G}_{[50.2,274)}x_2 > 29.6) \wedge \\ (\mathcal{G}_{[125,222)}x_1 < 47)) \vee ((\mathcal{F}_{[50.2,274)}x_2 < 29.6) \wedge (\mathcal{G}_{[206,233)}x_1 < 16.7))))))\end{aligned}$$

The specific formula generated using ROGE is simpler than the formula generated using DTL4STL and indeed it is easier to understand it. Furthermore DTL4STL does not consider the until operator in the set of primitives (see [8] for more details).

5.2 Ineffective Inspiratory Effort (IIE)

The Ineffective Inspiratory Effort (IIE) is one of the major problems concerning the mechanical ventilation of patients in intensive care suffering from respiratory failure.

The mechanical ventilation uses a pump to help the patient in the process of ispiration and expiration, controlling the $flow$ of air into the lugs and the airway pressure (paw). Inspiration is characterised by growth of pressure up to the selected paw value and positive $flow$, while expiration has a negative $flow$ and a drop paw. The exact dynamics of these respiratory curves stricly depends on patient and on ventilatory modes. We can see an example in Fig. 2 of two regular (blue regions) and one ineffective (red region) breaths. An IIE occurs when the patients tries to inspire, but its effort is not sufficient to trigger the appropriate reaction of the ventilator. This results in a longer expiration phase. Persistence of IIE may cause permanent damages of the respiratory system.

An early detection of anomalies using low-cost methods is a key challenge to prevent IIE conditions and still an open problem. In [9], starting with a dataset of discrete

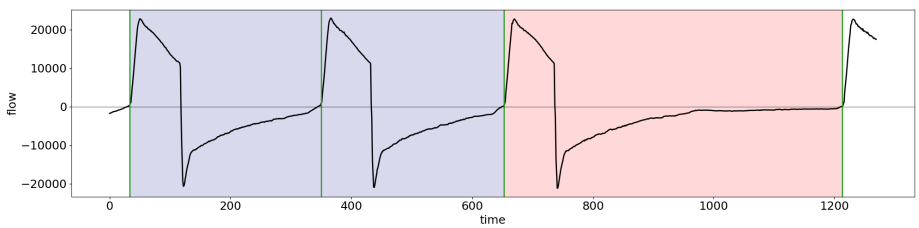

Fig. 2: Example of regular (blue regions) and ineffective (red region) breaths.

time series and sampled $flow$ values (labeled good and bad) from a single patient, the authors statically designed generative models of $flow$ and of its derivative $flow'$, for regular and ineffective signals. Then they used the simulations of such models to identify the best formula/formulae discriminating between them. In that contribution trajectories with different lengths are considered, treating as false trajectories that are too short to verify a given formula, and use this to detect the length of trajectories and separate anomalous breaths which last longer than regular ones. However, using the information about the duration of a breath is not advisable if the goal is to monitor the signals at runtime and detect the IEE at their onset. For this reason, in our approach we consider a new dataset, cutting the original trajectories used to generate the stochastic model in Bufo et al. [9] to a maximum common length of the order of 2 seconds. We also apply a moving average filter to reduce the noise in the computation of $flow'$.

Specifically, the new dataset consists of 2-dimensional traces of $flow$ and its derivative, $flow'$, containing a total of 288 trajectories (144 normal and 144 anomalous) with 134 sample points per trace. The time scale is in hundredths of a second. We run the experiments using a 6-fold cross-validation and report our results and comparison on the new dataset with DTL4STL [8] in Table 2.

An example of formula that we learn with ROGE is:

$$\varphi = \mathcal{F}_{[31,130]}((flow \geq -670) \vee (flow' \leq -94)) \tag{3}$$

	ROGE	DTL4STL
Mis. Rate	0.17 ± 0.01	0.23 ± 0.07
False. Pos	0.20 ± 0.02	0.23 ± 0.07
False. Neg	0.14 ± 0.02	0.20 ± 0.15
Comp. Time	65 ± 7	201 ± 7

Table 2: The comparison of the computational time (in sec.), the mean missclassification rate and its standard deviation between the learning procedure using the RObust GEnetic algorithm,

Formula φ identifies the anomalous trajectories, stating that at a time between 31 sec and 130 seconds either $flow$ is higher than -670 or $flow'$ is below -94. This is in accordance with the informal description of an IIE, principally caused by an unexpected increment of the $flow$ during the expiration phase followed by a rapid decrease. Indeed, one of the main characteristic of an IIE is the presence of a small hump in the $flow$ curve during this phase. In Fig. 3 we report some of the trajectories of the dataset, showing the areas where the property is satisfied.

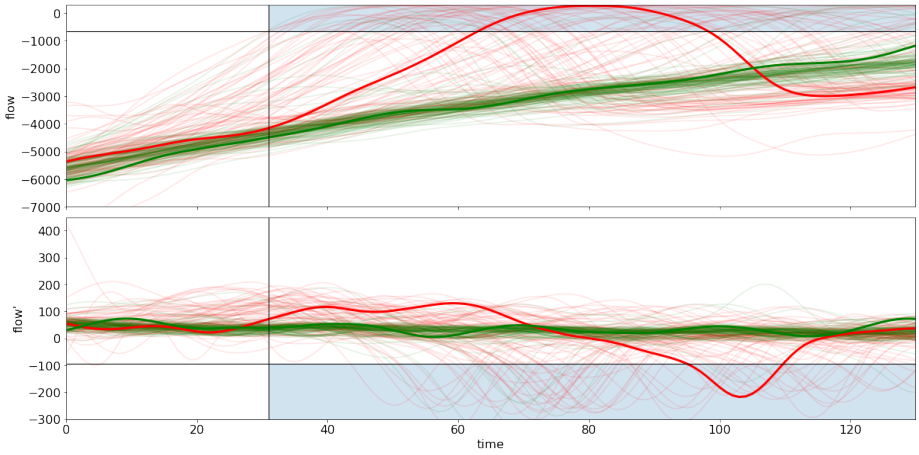

Fig. 3: 100 regular (green) and 100 ineffective (red) flow and $flow'$ trajectories during the expiration phase. The light blue rectangles correspond to the satisfaction area of formula (3). One regular (green) and one ineffective (red) trajectories are showed thicker.

On average, formulae found by the DTL4STL tool on the new dataset are disjunctions or conjunctions of 10 eventually or always subformulae, which are not readily interpretable.

Our approach on the new dataset is better in term of accuracy and computation time with an improvement of 22% and 67%, respectively.

Similarly to the previous test case, we compare our approach with [9], performed directly on the dataset, and not on the generative model. That approach performs so poorly also in this benchmark, obtaining a misclassification rate of 0.47, which is comparable with a random classifier. The problem here is that the methods proposed in [9], differently from the one presented here, relies on a large number of model simulations, and it is not suited to work directly with observed data.

If we give up to online monitoring and consider only full breaths, we can improve classification by rescaling their duration into [0,1], so that each breath lasts the same amount of time. In this case, we learn a formula with misclassification rate of 0.05 ± 0.01, while DTL4STL reaches a misclassification rate of 0.07 ± 0.02. This suggests that the high variability of durations of ineffective breaths has to be treated more carefully.

6 Conclusion

We present a framework to learn from a labeled dataset of normal and anomalous trajectories a *Signal Temporal Logic* (STL) specification that better discriminates among them. In particular, we design a Robust Genetic algorithm (ROGE) that combines an *Evolutionary Algorithm* (EA) for learning the structure of the formula and the *Gaussian Process Upper Confidence Bound* algorithm (GP-UCB) for synthesizing the formula parameters. We compare ROGE with our previous work [9] and with the Decision Tree approach presented in [8] on an anomalous trajectory detection problem of a maritime surveillance system and on an Ineffective Inspiratory Effort example.

A significant difference concerning our previous approach [7, 9] is that we avoid reconstructing a generative statistical model from the dataset. Furthermore, we modified both the structure and parameter optimization procedure of the genetic algorithm, which now relays on the robustness semantics of STL. This structural improvement was necessary considering that a naive application of our previous approach [9] directly on datasets performs very poorly.

We compare our method also with the Decision Tree (DTL4STL) approach of [8] obtaining a low misclassification rate and producing smaller and more interpretable STL specifications. Furthermore, we do not restrict the class of the temporal formula to only *eventually* and *globally* and we allow arbitrary temporal nesting.

Our genetic algorithm can get wrong results if the formulae of the initial generation are chosen entirely randomly. We avoid this behavior by considering simpler formulae from the beginning as a result of the GENERATEINITIALFORMULAE routine. This approach is a form of regularization and resembles the choice of the set of primitive of DTL4STL.

As future work, we plan to develop an iterative method which uses the proposed genetic algorithm and the robustness of STL to reduce the misclassification rate of the generated formula. The idea is to use the robustness value of a learned formula φ to identify the region of the trajectory space $\mathcal{D}$ where the generated formula φ has a high accuracy, i.e. trajectories whose robustness is greater than a positive threshold h^+ or smaller than a negative threshold h^-. These thresholds can be identified by reducing the number of false positives or false negatives. We can then train a new STL classifier φ' on the remaining trajectories, having small robustness for φ. This will create a hierarchical classifier, that first tests on φ, if robustness is too low it tests on φ', and so on. The depth of such classification is limited only by the remaining data at each step. The method can be further strengthen by relying on bagging to generate an ensemble of classifiers at each level, averaging their predictions or choosing the best answer, i.e. the one with larger robustness.

Acknowledgment

E.B. and L.N. acknowledge the partial support of the Austrian National Research Network S 11405-N23 (RiSE/SHiNE) of the Austrian Science Fund (FWF). E.B., L.N. and S.S. acknowledge the partial support of the ICT COST Action IC1402 (ARVI).

References

1. DTL4STL. `http://sites.bu.edu/hyness/dtl4stl/` (2016)
2. Ackermann, C., Cleaveland, R., Huang, S., Ray, A., Shelton, C.P., Latronico, E.: Automatic requirement extraction from test cases. In: Proc. of RV 2010. pp. 1–15 (2010)
3. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. 75(2), 87–106 (1987)
4. Annpureddy, Y., Liu, C., Fainekos, G.E., Sankaranarayanan, S.: S-taliro: A tool for temporal logic falsification for hybrid systems. In: Proc. of TACAS 2011. pp. 254–257. Springer (2011)
5. Asarin, E., Donzé, A., Maler, O., Nickovic, D.: Parametric identification of temporal properties. In: Proc. of RV. pp. 147–160 (2012)
6. Bartocci, E., Bortolussi, L., Nenzi, L., Sanguinetti, G.: System design of stochastic models using robustness of temporal properties. Theor. Comput. Sci. 587, 3–25 (2015)
7. Bartocci, E., Bortolussi, L., Sanguinetti, G.: Data-driven statistical learning of temporal logic properties. In: Proc. of FORMATS. pp. 23–37 (2014)
8. Bombara, G., Vasile, C.I., Penedo, F., Yasuoka, H., Belta, C.: A Decision Tree Approach to Data Classification Using Signal Temporal Logic. In: Proc. of HSCC. pp. 1–10 (2016)
9. Bufo, S., Bartocci, E., Sanguinetti, G., Borelli, M., Lucangelo, U., Bortolussi, L.: Temporal logic based monitoring of assisted ventilation in intensive care patients. In: Proc. of ISoLA. pp. 391–403 (2014)
10. Chen, Y., Tumova, J., Ulusoy, A., Belta, C.: Temporal logic robot control based on automata learning of environmental dynamics. The International Journal of Robotics Research 32(5), 547–565 (2013)

11. Donzé, A., Ferrer, T., Maler, O.: Efficient robust monitoring for stl. In: Proc. of CAV. pp. 264–279 (2013)
12. Donzé, A.: Breach, A toolbox for verification and parameter synthesis of hybrid systems. In: Proc. of CAV 2010. pp. 167–170. Springer (2010)
13. Fu, J., Tanner, H.G., Heinz, J., Chandlee, J.: Adaptive symbolic control for finite-state transition systems with grammatical inference. IEEE Trans. Automat. Contr. 59(2), 505–511 (2014)
14. Hoxha, B., Dokhanchi, A., Fainekos, G.E.: Mining parametric temporal logic properties in model-based design for cyber-physical systems. STTT 20(1), 79–93 (2018)
15. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: A redundancy-free approach to active automata learning. In: Proc. of RV 2014. pp. 307–322 (2014)
16. Jin, X., Donzé, A., Deshmukh, J.V., Seshia, S.A.: Mining requirements from closed-loop control models. IEEE Trans. on CAD of Integrated Circuits and Systems 34(11), 1704–1717 (2015)
17. Kong, Z., Jones, A., Belta, C.: Temporal Logics for Learning and Detection of Anomalous Behavior. IEEE Transactions on Automatic Control 62(3), 1210–1222 (Mar 2017)
18. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Proc. of FORMATS. LNCS, vol. 3253, pp. 152–166 (2004)
19. Maler, O., Nickovic, D.: Monitoring properties of analog and mixed-signal circuits. STTT 15(3), 247–268 (2013)
20. Nguyen, L.V., Kapinski, J., Jin, X., Deshmukh, J.V., Butts, K., Johnson, T.T.: Abnormal Data Classification Using Time-Frequency Temporal Logic. In: Proc. of HSCC. pp. 237–242 (2017)
21. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press (2006)
22. Sankaranarayanan, S., Kumar, S.A., Cameron, F., Bequette, B.W., Fainekos, G.E., Maahs, D.M.: Model-based falsification of an artificial pancreas control system. SIGBED Review 14(2), 24–33 (2017)
23. Silvetti, S., Policriti, A., Bortolussi, L.: An active learning approach to the falsification of black box cyber-physical systems. In: Proc. of IFM. pp. 3–17 (2017)
24. Srinivas, N., Krause, A., Kakade, S.M., Seeger, M.W.: Information-theoretic regret bounds for gaussian process optimization in the bandit setting. IEEE Transactions on Information Theory 58(5), 3250–3265 (2012)
25. Steffen, B., Howar, F., Isberner, M.: Active automata learning: From dfas to interface programs and beyond. In: Proc. of ICGI 2012. pp. 195–209 (2012)
26. Xu, Z., Julius, A.A.: Census signal temporal logic inference for multiagent group behavior analysis. IEEE Transactions on Automation Science and Engineering 15(1), 264–277 (2018)
27. Zhou, J., Ramanathan, R., Wong, W., Thiagarajan, P.S.: Automated property synthesis of odes based bio-pathways models. In: Proc. of CMSB. pp. 265–282 (2017)
28. Zutshi, A., Sankaranarayanan, S., Deshmukh, J.V., Kapinski, J., Jin, X.: Falsification of safety properties for closed loop control systems. In: Proc. of HSCC. pp. 299–300 (2015)