BUSTLE: A Versatile Tool for the Evolutionary Learning of STL Specifications from Data

Federico Pigozzi federico.pigozzi@phd.units.it
Laura Nenzi lnenzi@units.it
Eric Medvet emedvet@units.it
Department of Engineering and Architecture, University of Trieste, Trieste, Italy

Abstract

Describing the properties of complex systems that evolve over time is a crucial requirement for monitoring and understanding them. Signal Temporal Logic (STL) is a framework that proved to be effective for this aim because it is expressive and allows state properties as human-readable formulae. Crafting STL formulae that fit a particular system is, however, a difficult task. For this reason, a few approaches have been proposed recently for the automatic learning of STL formulae starting from observations of the system. In this paper, we propose BUSTLE (Bi-level Universal STL Evolver), an approach based on evolutionary computation for learning STL formulae from data. BUSTLE advances the state of the art because it (i) applies to a broader class of problems, in terms of what is known about the state of the system during its observation, and (ii) generates both the structure and the values of the parameters of the formulae employing a bi-level search mechanism (global for the structure, local for the parameters). We consider two cases where (a) observations of the system in both anomalous and regular state are available, or (b) only observations of regular state are available. We experimentally evaluate BUSTLE on problem instances corresponding to the two cases and compare it against previous approaches. We show that the evolved STL formulae are effective and human-readable: the versatility of BUSTLE does not come at the cost of lower effectiveness.

Keywords

Bi-level optimization, cyber-physical systems, anomaly detection.

1 Introduction

Explainability is of paramount importance in the field of Cyber-Physical Systems (CPS) (Bartocci, Mateis, et al., 2022b; Bartocci, Bortolussi, et al., 2022a). CPSs consist of computational models for which, usually, we do not have an exact description of the behavior, but just numerical trajectories derived by integration, simulations, or even only real observations of the systems. The increase in data availability led to a growing use of machine learning techniques to describe and analyze such systems. These methods typically generate powerful black-box models and work well in high dimensions; however, they suffer from a shortage of uncertainty measures surrounding point estimates and lack an understanding of the underlying mechanisms that give rise to the estimation results (Marcus, 2018). When dealing with safety-critical CPSs, where failure

Correspondence to federico.pigozzi@phd.units.it.

can cause high costs, these weaknesses become critical. In general, comprehending the phenomena CPSs capture and extracting interpretable information from their data are key points for the designers of such systems.

Learning temporal logic formulae has been recently explored as a way to tackle the problem of extracting human-interpretable information from data (Salamati et al., 2021). Temporal logic provides precise formal formulae that can be human-readable and verification algorithms that can check in an automatic way the value of satisfaction of interesting properties. They can describe properties as "the velocity will eventually reach 50 km h^{-1} in the next 10 min" or "it is always true that distance between two cars is more than 2 m." Temporal logic formulae can describe datasets in a concise way, and thus clarify and comprehend which are the emergent patterns for the system at hand. Additionally, an important application of this methodology is the problem of anomaly detection. Identifying the anomalous trajectories in an automatic way permits the designer to have an insight into what characterizes the anomalous behaviors.

We focus our attention on learning Signal Temporal Logic (STL) (Maler and Nickovic, 2004) formulae from data. STL is a linear-time temporal logic very suitable to specify properties over real-time trajectories, and it comes with efficient monitoring procedures to check in an automatic way whether the trajectories satisfy these properties. STL has been applied in many different contexts, from medical devices to automotive systems and robotics (Bartocci et al., 2018).

In this work, we propose a general framework to learn STL formulae from data, namely Bi-level Universal STL Evolver (BUSTLE), and validate it for two cases: (a) when data consist of both regular and anomalous trajectories (two-class learning), or (b) when data consists of only regular trajectories (one-class learning). With BUSTLE, the designer has the flexibility to choose between single-level optimization and bi-level optimization: in the former, we use genetic programming to learn both the structure of the formula and the parameters; in the latter, we use genetic programming to learn the structure and Bayesian optimization to learn the parameters. We frame the problem of learning the formula as an optimization problem minimizing a fitness function that has different shapes depending on the case.

First, we tackle the case of two-class learning from a labeled data set. We use an approach similar to Nenzi et al. (2018), but using a grammar-based form of genetic programming. In this way, we avoid the choice of the initial population by hand, which was one of the main limitations of Nenzi et al. (2018). We then show, using different case studies, that BUSTLE performs better or comparably with the most recent works in this field (Mohammadinejad et al., 2020a, 2020b; Bombara and Belta, 2021).

Then, we consider the case of one-class learning from a data set of only regular trajectories. To demonstrate the applicability of BUSTLE, we use the same case studies of the two-class learning case considering only the regular trajectories as the data set. We show that our BUSTLE learns STL formulae with a low misclassification rate for the regular trajectories and a low misclassification rate in detecting the anomalous trajectories.

Interestingly, BUSTLE achieves a good effectiveness in both scenarios (one-class and two-class) despite being based on a rather simple combination of existing building blocks: (a) a grammar-based representation for the solutions that allows limitation of their complexity, (b) a well-established evolutionary algorithm (genetic programming), (c) Bayesian optimization for refining solutions, and (d) a fitness function based on detection accuracy, yet applicable to both scenarios. We believe that this simplicity makes

BUSTLE more understandable; as the generated STL formulae are designed to be simple and human-interpretable (Virgolin et al., 2022), overall BUSTLE may be considered suitable for high-stakes applications where interpretability is a key goal (Virgolin, Alderliesten, et al., 2020).

The remainder of this paper is organized as follows. In Section 2, we briefly survey the recent literature that is relevant to this work. In Section 3, we provide the formal background concerning STL, including its syntax and semantics. In Section 4, we formally state the problem of learning STL formulae from data and define a few indices that can be used to assess the effectiveness of learned formulae. In Section 5, we describe BUSTLE in detail. In Section 6, we present the experiments we performed for validating BUSTLE and comment on the results. Finally, in Section 7, we draw the conclusions.

2 Related Work

Learning rules from data is a task that has been commonly framed as an optimization problem and often solved by means of evolutionary computation (EC). The applications go from road traffic rules (Medvet et al., 2017) to filtering rules for social network content (Bartoli et al., 2016), from active protocols (Pałka et al., 2017) to event processing rules (Bruns et al., 2019). When the systems for which the rules have to be learned involve time and quantities, as for CPSs, STL is a very practical choice for expressing the rules. Not surprisingly, learning STL formulae from data is thus an established field of research in CPS. We partition approaches for learning STL formulae into two sub-fields: template-based and template-free approaches.

Template-based approaches start from a parametric version of STL (PSTL) in which parameters substitute both time bounds and threshold values. Then, the user provides only the formula structure and learns its parameters. In Hoxha et al. (2018), the parameter mining problem is converted into a Pareto optimization problem, providing an approximate solution through a stochastic optimization method. In Jin et al. (2015), the authors exploit monotonicity in a fragment of PSTL to have an exponential saving when searching over the parameter space by using methods like binary search. Other interesting works on this field are Xu and Julius (2018), Asarin et al. (2012), Bartocci et al. (2014), and Nguyen et al. (2017). Although relevant, the applicability of template-based approaches is sometimes limited, since specifying formula structures can be hard.

Template-free approaches, on the other hand, attempt to learn both an optimal structure for the formula and its parameters. The vast majority of template-free approaches focus on two-class learning problems, learning an STL classifier for a labeled data set of trajectories (i.e., partitioned into regular and anomalous). In Bombara and Belta (2021), the authors devise a decision-tree-based framework that outputs an STL formula. The main limitation of the method is that it can produce very complex formulae for which the meaning is not readable. Furthermore, it is limited to a restricted fragment of STL. Mohammadinejad et al. (2020b) use an enumerative search, providing a way to systematically explore the space of all STL formulae, to learn the formula structure and logical binary classification, similar to Jin et al. (2015), to learn the parameters. The work that is the most similar to ours is Nenzi et al. (2018), which uses a genetic algorithm to learn the structure and Bayesian optimization to learn the parameters. Although groundbreaking, these works have limitations that we attempt to overcome. Our approach performs better or comparably with the most recent works while being generally faster. Finally, BUSTLE is one of the only two works that extend STL formula learning to the one-class learning problem, where data consist of only

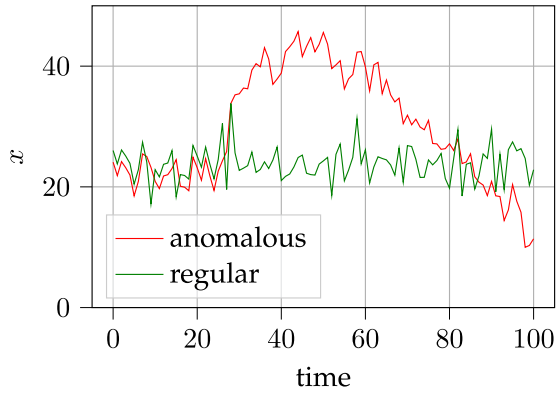

Figure 1: Example of a regular and an anomalous trajectory taken from the Train data set (see Section 6.1.2). This is an example for the sake of illustration: in general, what is anomalous or regular depends on the specific data set and problem.

regular trajectories; Indri et al. (2022) is the other, but it does not have the flexibility to choose between single-level and bi-level optimization.

In a previous work (Pigozzi et al., 2021), we applied EC for learning STL from data in a fully unsupervised context, that is, when the aim is not to use the STL formulae to tell apart anomalous and normal system behaviors, but simply to describe their properties. With respect to the cited work, here we (a) address the practically relevant case of anomaly detection in both the cases when anomalous observations are or are not available for learning and (b) employ a bi-level optimization strategy that improves both the effectiveness and efficiency of the learning process.

3 Background: Signal Temporal Logic

Signal Temporal Logic (STL) is a linear-time temporal logic (Clarke et al., 2018) suitable to describe the behavior of dynamical systems, in particular, to describe trajectories derived from the simulation of a system or from real-world data. Let X be a set of trajectories, with $\boldsymbol{x} : \mathbb{T} \to \mathbb{R}^n$ for every $\boldsymbol{x} \in X$, and $\mathbb{T} \subseteq \mathbb{R}_{\geq 0}$ a time domain. Each *trajectory* $\boldsymbol{x}$ is then an n-dimensional *signal* of real-valued attributes A, with $|A| = n$, and we denote by $x_i(t)$ the projection on the ith coordinate of $\boldsymbol{x} = (x_1, \ldots, x_n)$ at time $t \in \mathbb{T}$.

Definition 3.1 *(STL syntax): We define the syntax of an STL formula (i.e., the syntactical form of a formula) φ using the following grammar:*

$$\varphi \;:=\; \top \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathrm{U}_{[t_1,t_2]} \varphi_2$$

where $\top$ is the true value, μ is an atomic proposition in the form $y(t) \sim c$, with $y : \mathbb{R}^n \to \mathbb{R}$ projecting the n-dimensional signal onto a single variable, $\sim \in \{<, >\}$, $c \in \mathbb{R}^+$ being a threshold, $\neg$ and $\wedge$ are the usual Boolean connectives, and $\mathrm{U}_{[t_1,t_2]}$ is the Until temporal modality with $[t_1, t_2]$ being time interval with $t_1, t_2 \in \mathbb{T}$, with $t_1 \leq t_2$. One can easily derive the $\vee$ connective by composing $\neg$ and $\wedge$, namely $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$. We build more complex future temporal operators from Until; we define Eventually as $\mathrm{F}_{[t_1,t_2]}\varphi = \top \mathrm{U}_{[t_1,t_2]}\varphi$ and Globally as $\mathrm{G}_{[t_1,t_2]}\varphi = \neg\mathrm{F}_{[t_1,t_2]}\neg\varphi$.

Figure 1 shows an example of two one-dimensional trajectories $x : \mathbb{T} \to \mathbb{R}$, regular in green, and anomalous in red, with $\mathbb{T} = [0, 100]$. Examples of atomic proposition are

$\mu_1 = x < 35$ and $\mu_2 = x > 15$. Temporal operators permit the description of the temporal evolution of a system. An example of a temporal property is: $G_{[0,100]}(x < 35)$ which means that the value of x should be less than 35 for $t \in [0, 100]$. We remark that this is an example for the sake of illustration: in general, what is anomalous or regular depends on the specific data set and problem.

STL is interpreted pointwise over a trajectory $\mathfrak{x}$ using a qualitative (Boolean) or a quantitative (real-value) semantics (Maler and Nickovic, 2004; Donzé et al., 2013), which are both defined recursively on the operators of the logic. For the Boolean semantics, we write $(\boldsymbol{x}, t) \models \varphi$ if the trajectory $\boldsymbol{x}$ at time t satisfies φ and $(\boldsymbol{x}, t) \not\models \varphi$ if it does not satisfy the formula. The evaluation of the whole trajectory corresponds to the evaluation at time zero (i.e., $\boldsymbol{x} \models \varphi$ iff $(\boldsymbol{x}, 0) \models \varphi$), because it captures the system behavior over time and ensures the formula holds from the initial state; the evaluation at $t > 0$ corresponds to considering the portion of $\boldsymbol{x}$ starting at t. The quantitative semantics instead is defined considering the *robustness function* ρ which returns a real-value $\rho(\varphi, \boldsymbol{x}, t) \in \mathbb{R} \cup \{-\infty, +\infty\}$ quantifying the robustness degree of the property φ by the trajectory $\boldsymbol{x}$ at time t. Below, we report in detail the quantitative semantics and we refer the reader to Maler and Nickovic (2004, 2013) and Donzé et al. (2013) for more details.

Definition 3.2 *(STL Quantitative Semantics): Given a trajectory $\boldsymbol{x} : \mathbb{T} \to \mathbb{R}^n$, the robustness function is defined recursively as follows:*

$$
\begin{aligned}
\rho(\top, \boldsymbol{x}, t) &= +\infty \\
\rho(\mu, \boldsymbol{x}, t) &= \begin{cases} y(\boldsymbol{x}(t)) - c & \text{if } \mu \equiv y(\boldsymbol{x}(t)) > c \\ c - y(\boldsymbol{x}(t)) & \text{otherwise} \end{cases} \\
\rho(\neg\varphi, \boldsymbol{x}, t) &= -\rho(\varphi, \boldsymbol{x}, t) \\
\rho(\varphi_1 \wedge \varphi_2, \boldsymbol{x}, t) &= \min(\rho(\varphi_1, \boldsymbol{x}, t), \rho(\varphi_2, \boldsymbol{x}, t)) \\
\rho(\varphi_1 \, \mathrm{U}_{[t_1,t_2]} \varphi_2, \boldsymbol{x}, t) &= \sup_{t' \in [t_1+t, t_2+t]} \Big(\min(\rho(\varphi_2, \boldsymbol{x}, t'), \inf_{t'' \in [t, t'[} (\rho(\varphi_1, \boldsymbol{x}, t''))) \Big)
\end{aligned}
$$

where t_1 and t_2 are defined in terms of absolute time. Moreover, we let $\rho(\varphi, \boldsymbol{x}) := \rho(\varphi, \boldsymbol{x}, 0)$.

The sign of $\rho(\varphi, \boldsymbol{x})$ provides the link with the standard Boolean semantics of Maler and Nickovic (2004): if $\rho(\varphi, \boldsymbol{x}) > 0$ then $\boldsymbol{x} \models \varphi$, and if $\rho(\varphi, \boldsymbol{x}) < 0$ then $\boldsymbol{x} \not\models \varphi$. The case $\rho(\varphi, \boldsymbol{x}) = 0$, instead, is a borderline case, and the truth of φ cannot be assessed from the robustness degree alone.

When discussing the evaluation of temporal formulae, the necessary length concept is important (Maler and Nickovic, 2004). The *necessary length* $\|\varphi\|$ for a formula φ is defined recursively as:

$$
\begin{aligned}
\|\mu\| &= 0 \\
\|\neg\varphi\| &= \|\varphi\| \\
\|\varphi_1 \wedge \varphi_2\| &= \max(\|\varphi_1\|, \|\varphi_2\|) \\
\|\varphi_1 \mathrm{U}_{[t_1,t_2]} \varphi_2\| &= \max(\|\varphi_1\|, \|\varphi_2\|) + t_2
\end{aligned}
$$

Intuitively, the necessary length is the shortest trajectory length such that $\boldsymbol{x} \models \varphi$ is well-defined. For example, the formula $\varphi_1 \mathrm{U}_{[0,10]} \varphi_2$ cannot be evaluated on trajectories shorter than 10 (assuming φ_1 and φ_2 have a necessary length of 0) since this would imply looking at a future that is not part of the trajectory.

4 Problem Statement

We consider systems with real-valued attributes that can change over time. Given a system, the way its attributes change in a given time span is unequivocally described by a trajectory. We assume that any trajectory of a system is either *regular* or *anomalous*. Classifying trajectories into regular and anomalous is relevant in pattern recognition, anomaly detection, and decision making in several fields. How the process is carried out depends on the curators of the data set. Thus, BUSTLE leaves the system designer the choice on what kind of behavior is "regular" and what kind of behavior is "anomalous." BUSTLE also allows treatment of an n-class classification problem by iterating the two-class procedure $n-1$ times. For the sake of simplicity, we focus on the two-class problem and leave the multi-class one for a future work.

We aim to automatically learn an STL formula that classifies system trajectories as regular or anomalous starting from a collection of observed trajectories. The learned formula should (a) be satisfied only when the trajectory is regular, and not satisfied otherwise, and (b) be human-readable.

More precisely, let X be the set of possible trajectories for a system with attributes A and let X^+, X^-, with $X^+ \cup X^- = X$ and $X^+ \cap X^- = \emptyset$, the sets of regular and anomalous trajectories, respectively. Let Φ be the set of all STL formulae meaningful for the system, that is, defined over the attributes A. Ideally, we would like a methodology that, given a collection $X_{\text{learn}} \subseteq X$ of trajectories observed for the system, learns the STL formula $\varphi^\star$ that minimizes the classification error and is simple enough to be human-readable.

We further define the problem depending on the nature of the collection X_{learn} of observations available for learning, i.e., on the nature of the *learning data* X_{learn}.

- *Two-class*: X_{learn} consists of two sets X^+_{learn}, X^-_{learn} of observed trajectories that are known to be regular ($X^+_{\text{learn}} \subseteq X^+$) or anomalous ($X^-_{\text{learn}} \subseteq X^-$).
- *One-class*: X_{learn} consists of a single set X^+_{learn} of observed trajectories that are known to be regular.

The two cases differ in the granularity of details about the system provided by the learning data. In the first case, we have the most fine-grained information. In the second case, instead, we only know the regular behaviors; we learn from regular trajectories alone a formula that should be able to recognize anomalous trajectories as well. The second case is then a more complex problem. We present in Section 6 problem instances corresponding to real-world scenarios for both cases.

In the vast majority of practical cases, X^+, X^- are not actually known. We thus measure the effectiveness for a given problem instance, that is, a pair A, X_{learn}, using quantitative indices. For the two-class case, we use a pair X^+_{test}, X^-_{test} of sets of trajectories such that $X^+_{\text{test}} \subset X^+$ and $X^-_{\text{test}} \subset X^-$. To assess the accuracy of a learned formula $\hat{\varphi}$, we use the indices that are common in binary classification, namely the Accuracy (Acc), the False Positive Rate (FPR), and the False Negative Rate (FNR):

$$\text{Acc}(\hat{\varphi}; X^+_{\text{test}}, X^-_{\text{test}}) = \frac{\left|\left\{\boldsymbol{x} \in X^+_{\text{test}} : \boldsymbol{x} \models \hat{\varphi}\right\}\right|}{\left|X^+_{\text{test}}\right| + \left|X^-_{\text{test}}\right|} + \frac{\left|\left\{\boldsymbol{x} \in X^-_{\text{test}} : \boldsymbol{x} \not\models \hat{\varphi}\right\}\right|}{\left|X^+_{\text{test}}\right| + \left|X^-_{\text{test}}\right|} \tag{1}$$

$$\text{FPR}(\hat{\varphi}; X^-_{\text{test}}) = \frac{\left|\left\{\boldsymbol{x} \in X^-_{\text{test}} : \boldsymbol{x} \models \hat{\varphi}\right\}\right|}{\left|X^-_{\text{test}}\right|} \tag{2}$$

$$\text{FNR}(\hat{\varphi}; X^+_{\text{test}}) = \frac{\left|\left\{\boldsymbol{x} \in X^+_{\text{test}} : \boldsymbol{x} \not\models \hat{\varphi}\right\}\right|}{\left|X^+_{\text{test}}\right|} \tag{3}$$

For the one-class case, we only observe regular trajectories and thus cannot employ the indices of binary classification in the same way. To this end, we introduce a hyperparameter $\epsilon \in \mathbb{R}^+$ and classify a trajectory as regular if its robustness is greater than ϵ, while anomalous otherwise. Finally, we set $X^-_{\text{test}} = X^-$ and use the same X^+_{test} of the two-class case; thus, there is no X^- during evolution but we rather employ it to assess the one-shot performance of BUSTLE on anomalous trajectories during testing. The one-class indices are then:

$$\text{Acc}(\hat{\varphi}; X^+_{\text{test}}, X^-_{\text{test}}; \epsilon) = \frac{\left|\left\{\boldsymbol{x} \in X^+_{\text{test}} : \rho(\hat{\varphi}, \boldsymbol{x}) > \epsilon\right\}\right|}{\left|X^+_{\text{test}}\right| + \left|X^-_{\text{test}}\right|} + \frac{\left|\left\{\boldsymbol{x} \in X^-_{\text{test}} : \rho(\hat{\varphi}, \boldsymbol{x}) \leq \epsilon\right\}\right|}{\left|X^+_{\text{test}}\right| + \left|X^-_{\text{test}}\right|} \tag{4}$$

$$\text{FPR}(\hat{\varphi}; X^-_{\text{test}}; \epsilon) = \frac{\left|\left\{\boldsymbol{x} \in X^-_{\text{test}} : \rho(\hat{\varphi}, \boldsymbol{x}) > \epsilon\right\}\right|}{\left|X^-_{\text{test}}\right|} \tag{5}$$

$$\text{FNR}(\hat{\varphi}; X^+_{\text{test}}; \epsilon) = \frac{\left|\left\{\boldsymbol{x} \in X^+_{\text{test}} : \rho(\hat{\varphi}, \boldsymbol{x}) \leq \epsilon\right\}\right|}{\left|X^+_{\text{test}}\right|} \tag{6}$$

We remark that, by setting $\epsilon = 0$, we obtain the same indices of Equation (1) to (3) for the two-class case. Otherwise, the larger positive ϵ, the stricter the criterion for considering a trajectory to be satisfying the formula; the larger negative ϵ, the opposite.

We measure the human readability of a formula (a) in a qualitative way, by means of visual inspection, and (b) in a quantitative way, computing a measure of formula complexity. For the latter, we assume that larger formulae are less human-readable and use the number of nodes $n_{\text{nodes}}(\hat{\varphi})$ in the syntax tree of a formula $\hat{\varphi}$ as a proxy of formula complexity.

5 BUSTLE: Evolving STL Formulae

From the formulation of Section 4, it is clear that we are dealing with a search problem: we want a way to search in the space Φ of STL formulae for one that fits the learning data X_{learn}. The nature of the search space is peculiar—it consists of the set of STL formulae that are syntactically valid for a given set of attributes A—and hence the search method has to be capable of effectively searching this kind of space. We build our methodology, that is, BUSTLE, based on EC, since this family of population-based search methods, consisting of Evolutionary Algorithms (EAs), showed to be effective over a diversity of search spaces.

Since the STL syntax can be defined by means of the Context Free Grammar (CFG) of Definition 3.1, we rely on CFG Genetic Programming (CFGGP) (Whigham, 1995), a grammar-based version of genetic programming (Koza, 1992), a popular EA. In BUSTLE, candidate solutions are valid STL formulae, that is, strings of the language defined by the STL grammar, and they are subjected to variation through the application of genetic operators that operate on the corresponding derivation trees. We describe the EA of BUSTLE, including the representation of the solutions, in Section 5.1.1.

Since the attributes are real-valued, the thresholds in the atomic propositions (i.e., the c in the μ of Definition 3.1) play a relevant role in an STL formula; similarly, the time boundaries in the temporal operators are relevant too. We say that thresholds and boundaries are the *numerical parameters* of an STL formula (Asarin et al., 2012). The numerical parameters are subjected to search too, together with STL structure; however, there is evidence that constant creation in grammar-based EC is a difficult problem (Dempsey et al., 2007; Azad and Ryan, 2014). To address this peculiarity, we propose two variants of BUSTLE: a *single-level* variant that jointly optimizes the structure and

parameters with CFGGP, and a *bi-level* variant that separates the search for the optimal parameter values from the search for the optimal structure of the STL formula. In particular, the former relies on the Gaussian Process Upper Confidence Bound (GP-UCB) optimization algorithm (Srinivas et al., 2012). We describe this algorithm in Section 5.2. Summarizing, the bi-level variant performs a double-optimization loop: at the outer level (global search), BUSTLE optimizes the structure of the STL formula; at the inner level (local search), it optimizes the formula parameters. The quality of a solution at the outer level is given by the quality of that solution with the parameters optimized at the inner level.

A *fitness function* f measures the quality of candidate solutions and depends on the kind of problem at hand (two-class, one-class). In all cases, the fitness function is defined as $f : \Phi \to \mathbb{R}$ and is parametrized on the learning data X_{learn}. The semantics is that of a minimization problem: the lower $f(\varphi; X_{\text{learn}})$, the better the STL formula φ. We present the two formulations for the fitness function in Section 5.3.

5.1 Search for STL Formula Structures with CFGGP

In BUSTLE, we search for STL formula structures using CFGGP. In CFGGP, candidate solutions are represented as derivation trees of a grammar $\mathcal{G} = (N, T, s_0, R)$, where N is the set of non-terminal symbols, T is the set of terminal symbols (with $T \cap N = \emptyset$), $s_0 \in N$ is the starting symbol, and R is the set of derivation rules. Each derivation rule describes how a non-terminal symbol may be replaced by a sequence of symbols, either terminal or non-terminal. A *derivation tree* is a tree where nodes are symbols of the grammar: leaf nodes are terminal symbols, and non-leaf nodes are non-terminal symbols. The children of each node match one of the derivation rules for the corresponding non-terminal symbol. The string of the language defined by the grammar, corresponding to a given derivation tree, is the sequence of the leaves of the tree. Figure 2 shows an example of a derivation tree for the CFG of Figure 3.

We use an improved version of CFGGP that promotes diversity in the population. The lack of diversity in the population may result in premature convergence towards a local optimum (Squillero and Tonda, 2016), in particular when the search space is discrete, as in CFGGP (Bartoli et al., 2019). In this work, we promote diversity by simply enforcing the reapplication of the genetic operator whenever a generated individual is already part of the population.

5.1.1 Evolutionary Algorithm

Given a CFG $\mathcal{G}$ and a fitness function $f : L(\mathcal{G}) \to \mathbb{R}$, where $L(\mathcal{G})$ is the language defined by the CFG $\mathcal{G}$, CFGGP works as shown in Algorithm 1. After the initialization of the population P, CFGGP repeats n_{gen} times the following three steps.

(1) It builds the offspring population P', with $|P'| = n_{\text{pop}}$, by iteratively selecting one (mutation, with $1 - p_{\text{xover}}$ probability) or two (crossover, with p_{xover} probability) parents with tournament selection of size n_{tour} and then applying the genetic operator. If the resulting solution φ_c is already part of the offspring P' or parent P population, a new solution is generated, and the process is repeated for a maximum number n_{atts} of attempts; otherwise, φ_c is added to P' and its fitness $f(\varphi_c; X_{\text{learn}})$ is computed.

(2) It merges the parent and offspring populations.

(3) It shrinks the resulting new population P, until its size is n_{pop}, by iteratively removing the worst solution.

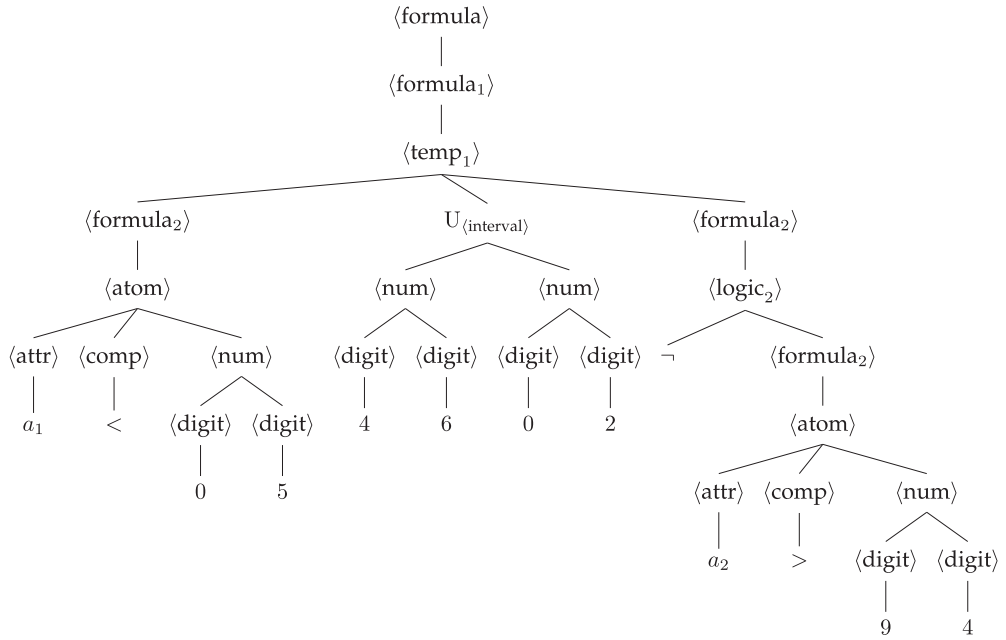

Figure 2: A derivation tree of the grammar of Figure 3 for the formula $(a_1 < 0.05)\ \mathrm{U}_{[46,48]}\neg(a_2 > 0.94)$, as in single-level optimization. For bi-level optimization, we substitute ⟨interval⟩ and ⟨num⟩ with placeholders I and c, respectively, to be optimized by local search. In both cases, we assume data to be normalized in [0, 1].

$$
\begin{aligned}
\langle \text{formula} \rangle &::= \langle \text{formula}_1 \rangle \\
\langle \text{formula}_i \rangle &::= \begin{cases} \langle \text{atom} \rangle \mid \langle \text{logic}_i \rangle \mid \langle \text{temp}_i \rangle & \text{if } i < i_{\max} \\ \langle \text{atom} \rangle \mid \langle \text{logic}_i \rangle & \text{otherwise} \end{cases} \\
\langle \text{logic}_i \rangle &::= \neg \langle \text{formula}_i \rangle \mid \langle \text{formula}_i \rangle \wedge \langle \text{formula}_i \rangle \\
\langle \text{temp}_i \rangle &::= \langle \text{formula}_{i+1} \rangle \mathrm{U}_{\langle \text{interval} \rangle} \langle \text{formula}_{i+1} \rangle \mid \\
&\qquad \mathrm{G}_{\langle \text{interval} \rangle} \langle \text{formula}_{i+1} \rangle \mid \\
&\qquad \mathrm{F}_{\langle \text{interval} \rangle} \langle \text{formula}_{i+1} \rangle \\
\langle \text{interval} \rangle &::= [\langle \text{num} \rangle, \langle \text{num} \rangle] \\
\langle \text{atom} \rangle &::= \langle \text{attr} \rangle \langle \text{comp} \rangle 0. \langle \text{num} \rangle \\
\langle \text{attr} \rangle &::= a_1 \mid a_2 \mid \ldots \mid a_{|A|} \\
\langle \text{comp} \rangle &::= < \mid > \\
\langle \text{num} \rangle &::= \langle \text{digit} \rangle \langle \text{digit} \rangle \\
\langle \text{digit} \rangle &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

Figure 3: The CFG for describing STL formulae structure and parameters, as in single-level optimization. Non-terminal symbols are enclosed in angle brackets: the topmost non-terminal symbol, ⟨formula⟩, is the starting symbol s_0 of the grammar. The derivation rules for the symbols ⟨formula$_i$⟩, ⟨logic$_i$⟩, ⟨temp$_i$⟩ are parametric on i, which represents the nesting level: that is, for $i_{\max} = 3$, the concrete version of this CFG would have three $7 + 3 \cdot 2 + 2$ non-terminal symbols, including ⟨formula$_1$⟩, ⟨formula$_2$⟩, ⟨formula$_3$⟩, ⟨logic$_1$⟩, ⟨logic$_2$⟩, ⟨logic$_3$⟩, ⟨temp$_1$⟩, and ⟨temp$_2$⟩. The derivation rule for ⟨attr⟩ is the one that makes the grammar tailored to a given system with attributes $A = \{a_1, a_2, \ldots, a_{|A|}\}$.

Algorithm 1 The EA for the outer optimization of BUSTLE

```
1  function evolve():
2      P ← initialize(G, n_pop)
3      foreach i ∈ {1, ..., n_gen} do
4          P' ← ∅
5          while |P'| ≤ n_pop do
6              i ← 0
7              repeat
8                  if ∼ U(0,1) ≤ p_xover then
9                      (φ_{p,1}, f_{p,1}) ← select(P)
10                     (φ_{p,2}, f_{p,2}) ← select(P)
11                     φ_c ← crossover(φ_{p,1}, φ_{p,2}; G)
12                 else
13                     (φ_p, f_p) ← select(P)
14                     φ_c ← mutate(φ_p; G)
15                 end
16                 i ← i + 1
17             until (φ_c ∉ P ∪ P') ∧ (i ≤ n_atts)
18             P' ← P' ∪ {(φ_c, f(φ_c; X_learn))}
19         end
20         P ← P ∪ P'
21         while |P| ≥ n_pop do
22             P ← P \ {worst(P)}
23         end
24     end
25     return best(P)
26 end
```

The initial population is built with the ramped half-and-half method (Luke, 2009). Let a range $\{d_{\min}, \ldots, d_{\max}\}$ for the depth of the derivation trees be given and let n_{pop} be the number of trees to be generated. For each d in the range, we build k random approximately full derivation trees (i.e., where each leaf node is at depth d) and k random trees with the deepest leaf at depth d, with $k = \frac{n_{\text{pop}}}{2(d_{\max} - d_{\min} + 1)}$. We write "approximately" because it is not possible, in general, to build a derivation tree of a grammar $\mathcal{G}$ where each leaf is exactly at depth d. This procedure ensures that the size, and hence, the complexity of the generated formulae is evenly distributed in a predefined range.

The genetic operators are defined over the space of derivation trees of the grammar $\mathcal{G}$. We used the standard CFGGP mutation and crossover. The former "replaces" a random subtree of the derivation tree with a randomly generated subtree that is appropriate according to $\mathcal{G}$. The crossover "replaces" a random subtree of one parent with an appropriate random subtree of the other parent. Both operators ensure that the resulting derivation tree is at most $d_{\max}$ deep.

After preliminary experiments and exploiting our previous knowledge, we set the following values for the parameters: population size $n_{\text{pop}} = 500$, number of iterations (generations) $n_{\text{gen}} = 50$ (corresponding to 25,000 fitness evaluations), tournament size $n_{\text{tour}} = 5$, crossover probability $p_{\text{xover}} = 0.8$, number of attempts $n_{\text{atts}} = 100$, min and max depth $d_{\min} = 1$, and $d_{\max} = 12$. We thus include all the details necessary to replicate the results of BUSTLE.

$$\langle \text{temp}_i \rangle ::= \langle \text{formula}_{i+1} \rangle \text{U}_I \langle \text{formula}_{i+1} \rangle \mid \text{G}_I \langle \text{formula}_{i+1} \rangle \mid \text{F}_I \langle \text{formula}_{i+1} \rangle$$
$$\langle \text{atom} \rangle ::= \langle \text{attr} \rangle \langle \text{comp} \rangle c$$

Figure 4: The CFG for describing STL formulae structure, as in bi-level optimization. Here, we show just the rules that vary with respect to single-level optimization (see Figure 3) for the sake of conciseness. I and c are placeholders that BUSTLE optimizes by local search.

5.1.2 Grammar for STL Formula Structures

We need a grammar $\mathcal{G}$ for the language of STL formulae that is customizable for a given problem, i.e., for its attributes A. Moreover, we want $\mathcal{G}$ to define a language suitable for single-level optimization and bi-level optimization. Finally, for the sake of human readability, we want $\mathcal{G}$ to explicitly limit the depth of nesting of temporal operators. We remark that the overall size of STL formulae is limited by the maximum tree depth $d_{\max}$ used by CFGGP (in the operators and in the initialization of the population). However, we believe that posing a further limit on the nesting of the temporal operators makes the STL formulae more readable, and not just smaller. The findings of Virgolin, De Lorenzo, et al. (2020) for mathematical expressions corroborate our belief: some operators, such as log and sin, make the expressions less interpretable than others, for example, $+$ and $\div$.

Figure 3 shows our grammar $\mathcal{G}$, with limited nesting of the temporal operators, for single-level optimization. The figure adopts the common Backus–Naur form: the non-terminal symbols are enclosed in angle brackets, whereas the terminal symbols are shown as literals ($\neg$, $\wedge$, ..., a_1, a_2, ..., $<$, $>$, 0, ..., 9); for each non-terminal, derivation rules are separated by |; the starting symbol is the topmost non-terminal, that is, $s_0 = \langle \text{formula} \rangle$. The terminals $a_1, a_2, \ldots$, derived from the non-terminal $\langle \text{attr} \rangle$, represent the attributes of the problem at hand: in this way, the grammar is tailored to a specific problem. We express some of the non-terminals using a parameter i that represents the maximum nesting. The only derivation rule that increases i is the one for $\langle \text{temp}_i \rangle$, which represents temporal operators. The limit to nesting is enforced by the parametric definition of the derivation rule of $\langle \text{formula}_i \rangle$, that does not expand to $\langle \text{temp}_i \rangle$ if $i \geq i_{\max}$. We set the maximum nesting to $i_{\max} = 3$. This means that CFGGP operates on a grammar $\mathcal{G}$ that is the realization of the grammar of Figure 3 with $i_{\max} = 3$ and a given set of attributes A.

For single-level optimization, when mapping a derivation tree into the corresponding STL formula, we apply the following adjustments for the numerical parameters. To map a non-terminal interval symbol $\langle \text{interval} \rangle$ into the corresponding time interval $[t_1, t_2] \subset \mathbb{T}$, we first obtain the interval $[a, b]$, with $a, b \in \{0, \ldots, 99\}$ according to the derivation rules for $\langle \text{num} \rangle$; then, we set $t_1 = a$ and $t_2 = a + b$. This way, we enforce valid time intervals without imposing constraints on a and b. Moreover, we remark that, since there can be at most two nested temporal operators, the maximum necessary length of any formula will be 198, corresponding to the necessary length of a formula with two nested temporal operators with intervals [0, 99]. Note that, according to the definition of $\langle \text{atom} \rangle$, when mapping a $\langle \text{num} \rangle$ in an atomic proposition, we assume a leading 0. to precede the two-digit numerical value. As a result, numeric constants lie in [0,1], and, for normalized data, we can express thresholds in the entire relevant domain.

For bi-level optimization, as shown in Figure 4, we substitute $\langle \text{interval} \rangle$ with a placeholder I and substitute $\langle \text{num} \rangle$ with a placeholder c. I and c represent the placeholders for the formula parameters, which BUSTLE will optimize with the local search, as detailed in the next subsection.

5.2 Search for Formula Parameters with GP-UCB

For the bi-level variant of BUSTLE, we perform a local search to find the optimal values for its numerical parameters. From the point of view of the global search of Algorithm 1, the local search is simply part of the computation of the fitness:

$$f_{\text{opt}}\left(\overline{\varphi}; X_{\text{learn}}\right) = \min_{\boldsymbol{\theta}\in[0,1]^p} f\left(\overline{\varphi}|_{\boldsymbol{\theta}}; X_{\text{learn}}\right) \tag{7}$$

where f is the actual fitness function, $\overline{\varphi}$ is a formula structure, $\boldsymbol{\theta} \in [0,1]^p$ is the vector of normalized numerical parameters of $\overline{\varphi}$, and $\varphi = \overline{\varphi}|_{\boldsymbol{\theta}}$ is the STL formula obtained by evaluating $\overline{\varphi}$ in $\boldsymbol{\theta}$.

For the purpose of computing $f\left(\overline{\varphi}|_{\boldsymbol{\theta}}; X_{\text{learn}}\right)$, we map the normalized parameters $\boldsymbol{\theta}$ into the actual parameters as follows. Let p_c be the number of c placeholders (i.e., thresholds in atomic propositions) in $\overline{\varphi}$ and let p_I be the number of I placeholders (i.e., bounds in temporal operators), then $p = p_c + 2p_I$. For each c_j, we take the corresponding θ_j element in $\boldsymbol{\theta}$ and de-normalize c_j by considering the minimum $v_{a,\min}$ and maximum $v_{a,\max}$ values observed in X_{learn} for the attribute in the atomic proposition containing c_j in $\overline{\varphi}$:

$$c_j = v_{a,\min} + \theta_j(v_{a,\max} - v_{a,\min}) \tag{8}$$

For each I_j, we take $\theta_{j'}, \theta_{j'+1}$, with $j' = p_r + 2j - 1$, and de-normalize as $I_j = \left[t_{j,\text{inf}}, t_{j,\text{sup}}\right]$:

$$t_{j,\text{inf}} = \left\lfloor (\tau - 1)\theta_{j'} \right\rfloor \tag{9}$$

$$t_{j,\text{sup}} = t_{j,\text{inf}} + \left\lfloor (\tau - t_{j,\text{inf}})\theta_{j'+1} \right\rfloor \tag{10}$$

where τ is a parameter representing the maximum span of an interval; this way we enforce proper constraints for the intervals.

Computing f_{opt} corresponds to solving a numerical optimization problem. The fitness function can be treated as an unknown objective function with variables for the parameters to optimize, $f_{\overline{\varphi}} : [0,1]^p \to \mathbb{R}$. In BUSTLE, we solve this optimization problem using the Gaussian Process Upper Confidence Bound (GP-UCB) algorithm introduced in Srinivas et al. (2012), taking inspiration from Nenzi et al. (2018).

The general idea of the approach is to treat the unknown function as a Gaussian process (Rasmussen and Williams, 2005). This means that it can be completely described by its mean $\boldsymbol{\mu}(\boldsymbol{\theta})$ and covariance matrix $\boldsymbol{\Sigma}(\boldsymbol{\theta})$. We can then approximate it considering the posterior of the distribution, and use the UCB to balance the exploration and exploitation in the parameter space.

First, we collect as training data set a number of observations (evaluations of $f_{\overline{\varphi}}$ for some fixed number of samples for a fixed number of parameter values). Then, for a number of iterations, we use Gaussian processes to probabilistically estimate the values of $f_{\overline{\varphi}}$ for the rest of the parameter space, computing the posterior $\text{GP}(\boldsymbol{\mu}_k(\boldsymbol{\theta}), \boldsymbol{\Sigma}_k(\boldsymbol{\theta}))$, where k is the kth iteration. More specifically, in a regression task, a prior is combined with observations to provide a posterior. $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ are prior distributions. At iteration k, GP-UCB samples $\boldsymbol{\theta}_k \in [0,1]^p$ from $f_{\overline{\varphi}}$ in order to maximize the upper confidence bound:

$$\boldsymbol{\theta}_k = \underset{\boldsymbol{\theta}\in[0,1]^p}{\text{argmax}} \left(\boldsymbol{\mu}_{k-1}(\boldsymbol{\theta}) + \sqrt{\beta}\boldsymbol{\Sigma}_{\boldsymbol{k-1}}(\boldsymbol{\theta}) \right) \tag{11}$$

to balance exploitation (first addend of Equation (11)) and exploration (second addend of Equation (11)) with a hyperparameter β. The idea is that we do not take just the mean value but we balance it with respect to the variance of the distribution. In this way, if there are regions of the parameter space not explored, then in those regions there will be

high variance and so the new parameter $\boldsymbol{\theta}_k$ will be taken in that region. The algorithm then evaluates $f_{\overline{\varphi}}$ for the new parameter $\boldsymbol{\theta}_k$, adds the new observation to the training set, and performs a Bayesian update to obtain the new $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$. GP-UCB stops when there are no more improvements; if this does not happen, it has anyway a fixed number n_{local} of maximum iterations. We omit it for brevity and refer the reader to Srinivas et al. (2012) for more details. We used the implementation of Nenzi et al. (2018). After preliminary experiments, we set $n_{\text{local}} = 15$ and all other parameters (including prior distributions and β) to be the same as Nenzi et al. (2018).

5.3 Fitness Function

We designed BUSTLE for the two kinds of problem identified in Section 4: two-class and one-class. We define the fitness function f accordingly.

For the two-class problem, where the learning data is $X_{\text{learn}} = (X^+_{\text{learn}}, X^-_{\text{learn}})$, we set:

$$f(\varphi; X^+_{\text{learn}}, X^-_{\text{learn}}) = -\frac{\mu_{\varphi, X^+_{\text{learn}}} - \mu_{\varphi, X^-_{\text{learn}}}}{\sigma_{\varphi, X^+_{\text{learn}}} + \sigma_{\varphi, X^-_{\text{learn}}}} \tag{12}$$

where:

$$\mu_{\varphi, X} = \frac{1}{|X|} \sum_{\boldsymbol{x} \in X} \rho(\varphi, \boldsymbol{x}) \tag{13}$$

$$\sigma_{\varphi, X} = \sqrt{\frac{1}{|X|} \sum_{\boldsymbol{x} \in X} \big(\rho(\varphi, \boldsymbol{x}) - \mu_{\varphi, X}\big)^2} \tag{14}$$

This fitness function measures the (negative) difference between the average robustness $\rho(\varphi, \boldsymbol{x})$ of φ on the trajectories of X^+_{learn} and X^-_{learn}. We divide the difference by the standard deviation of the $\rho(\varphi, \boldsymbol{x})$ for taking into account the scale of the robustness. By minimizing this f, we promote formulae that "greatly satisfy" trajectories in X^+_{learn} and "greatly dissatisfy" trajectories in X^-_{learn}.

For the one-class problem, where the learning data is $X_{\text{learn}} = X^+_{\text{learn}}$, we set:

$$f(\varphi; X^+_{\text{learn}}) = \alpha \frac{1}{\big|X^+_{\text{learn}}\big|} \big|\{\boldsymbol{x} \in X^+_{\text{learn}} : \boldsymbol{x} \not\models \varphi\}\big| + \frac{1}{\sigma'_{\varphi, X^+_{\text{learn}}} \big|X^+_{\text{learn}}\big|} \sum_{\boldsymbol{x} \in X^+_{\text{learn}}} \big|\rho(\varphi, \boldsymbol{x})\big| \tag{15}$$

where $\sigma'_{\varphi, X}$ is the standard deviation of $|\rho(\varphi, \boldsymbol{x})|$ across all $\boldsymbol{x} \in X^+_{\text{learn}}$. This fitness function is composed of two parts: the former measures the portion of trajectories in X^+_{learn} that is not satisfied by φ; the latter measures the average absolute robustness of all trajectories in X^+_{learn}. We sum them up with a weight factor α, being a hyperparameter. After preliminary experiments, we set $\alpha = 0.1$. Since both components are positive, minimizing f means minimizing both components. By minimizing the first part, we leverage the knowledge, available in the one-class problem, of the trajectories being regular. By minimizing the second part, we build a φ that is tight with respect to the trajectories in X^+_{learn}; that is, we want φ to be as sensitive as possible to small variations in the trajectories.

6 Experimental Evaluation

We performed several experiments aimed at answering the following research questions:

RQ1 Is BUSTLE capable of finding good solutions?

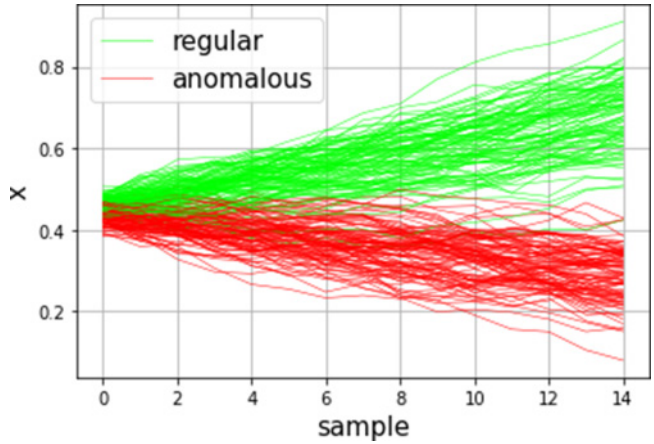

Figure 5: Linear system trajectories. Green trajectories are regular; red trajectories are anomalous.

RQ2 How do single-level and bi-level optimization differ in the efficiency-efficacy trade-off?

RQ3 Is BUSTLE effective when compared to other state-of-the-art approaches?

We experimented with four datasets, detailed in the next subsection, to verify how BUSTLE performs across different data distributions.

For all experiments, we normalized data in $[0, 1]$ before the evolutionary search. For the two-class case, and following the common practices for binary classification assessment, we computed the Acc, FPR, and FNR using a five-fold cross-validation strategy. For the one-class case, we used 20% (randomized) of the regular trajectories as a hold-out validation set and computed the FPR with it (i.e., X^+_{test} of Equation (3) consists of the hold-out validation set), while $X^-_{\text{test}} = X^-$.

6.1 Data

6.1.1 Linear System

The first data set consists of the simulation of a simple dynamical linear system and serves as a basic classification task. The data set collects 200 (partitioned into 100 regular and 100 anomalous) trajectories of 1 dimension ($A = \{\mathrm{x}\}$) we simulated from a dynamical linear system of the form:

$$\frac{dx}{dt} = \begin{cases} 0.03x + w & \text{regular} \\ -0.03x + w & \text{anomalous} \end{cases} \quad t \in \{0, 1, \ldots, 14\} \tag{16}$$

where $w \sim \mathcal{N}(0, 0.2)$.

Each trajectory has 15 sample points. See Figure 5 for a visualization of the trajectories (for the sake of visualization, we plotted at a higher sample resolution than the one effectively used in our experiments). This system is the same used by Mohammadinejad et al. (2020b) in their experiments.

6.1.2 Train Cruise Control

The second data set consists of the cruise control of a train and serves as a first real-world task, with a clear-cut condition to classify between regular and anomalous trajectories

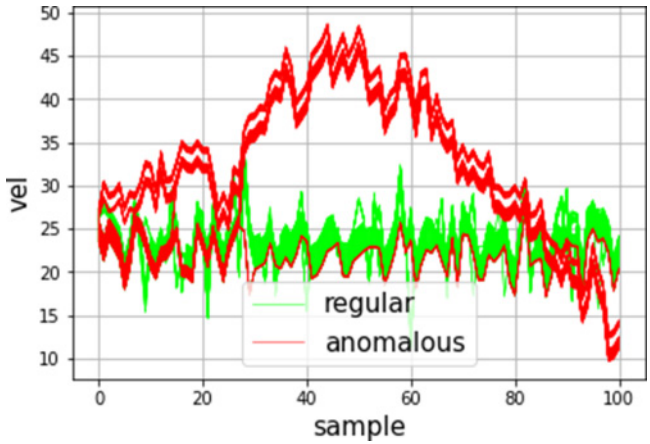

Figure 6: Train cruise control trajectories. Green trajectories are regular; red trajectories are anomalous.

(with the exception of one outlier). The data set collects 200 trajectories of 1 dimension ($A = \{\texttt{vel}\}$), partitioned into 100 regular and 100 anomalous. Each trajectory has 101 sample points. Trajectories consist of the velocity of a train whose controller must keep the velocity at 25 m s^{-1}, oscillating within a window of ± 2.5 m s^{-1}. Thus, anomalous trajectories correspond to the failure of the controller to do so, and regular trajectories to success. See Figure 6 for a visualization of the trajectories. This data set is the same used by Mohammadinejad et al. (2020b) in their experiments.

6.1.3 Maritime Surveillance

The third data set consists of a maritime surveillance case and serves as a more challenging task, with no clear condition to classify between regular and anomalous trajectories. The data set collects 2,000 trajectories of 2 dimensions (thus $A = \{\texttt{x1}, \texttt{x2}\}$) of vessels, partitioned into 1,000 regular and 1,000 anomalous trajectories. Each trajectory has 61 sample points. Trajectories consist of the longitudinal and latitudinal coordinates of vessels transiting over a harbor, with anomalous trajectories corresponding to vessels involved in illegal activities, and regular trajectories corresponding to vessels not involved in illegal activities. See Figure 7 for a visualization of the trajectories. This data set is the same used by Nenzi et al. (2018) and Bombara et al. (2016) in their experiments.

6.1.4 Urban Driving

The fourth data set consists of an urban driving case and serves as a complex classification task, involving twice the number of dimensions of the maritime data set. The data set collects 300 trajectories of 4 dimensions (thus $A = \{\texttt{x1}, \texttt{x2}, \texttt{x3}, \texttt{x4}\}$) of cars, partitioned into 150 regular and 150 anomalous trajectories. Each trajectory has 500 sample points. Trajectories consist of the relative position and velocity (along the y and z axes) of cars with respect to other cars in a simulation of a road crossing. For each car, the other cars can drive adopting an aggressive (anomalous label) or safe (normal label) behavior, while taking different turns at the crossing and braking to allow for a pedestrian to cross, hence the complexity of the data set. The 4-dimensional nature of the data set does not allow us to visualize it, but it is the same used by Aasi et al. (2023) in their experiments.

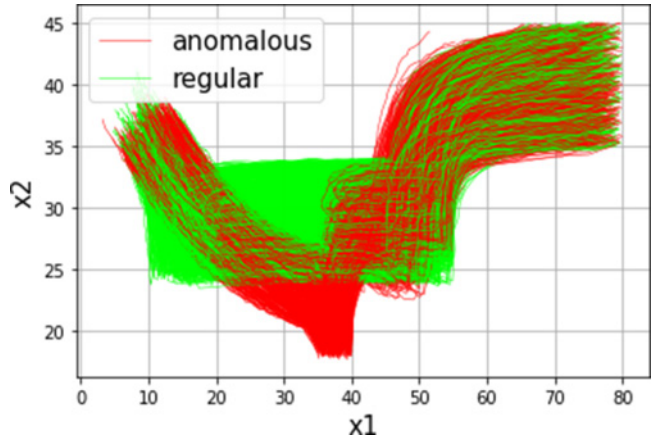

Figure 7: Maritime surveillance trajectories. Green trajectories are regular; red trajectories are anomalous.

6.2 Settings

We implemented the code of BUSTLE and the one used for doing the experiments in the Java programming language and made it publicly available.[1] We built on top of the evolutionary framework JGEA (Medvet et al., 2022) and the monitoring tool MoonLight (Bartocci et al., 2020; Nenzi et al., 2023). We performed 30 independent runs varying the random seed for the EA. We performed all statistical tests with the Mann-Whitney U test for independent samples, using 0.05 as confidence level. We ran all the experiments on an Apple M1 MacBook Pro with 8 cores at 3.2 GHz and 8 GB RAM.

6.3 Results

6.3.1 RQ1: Is BUSTLE Capable of Finding Good Solutions?

To assess whether BUSTLE finds good solutions, we visualize the fitness f and use FPR, FNR, Acc, and n_{nodes} as performance indices. Moreover, we also inspect and report the evolved formulae to verify whether they are interpretable to a human.

We visualize the fitness f in Figure 8 in terms of median $\pm$ standard deviation across the best evolved individual in each run. In addition, we report in Table 1 the performance indices of BUSTLE (namely, FPR, FNR, Acc, running time, and n_{nodes}).

We report the results for all the experiments and compare BUSTLE with a random search baseline (Luke, 2009), which is a basic stochastic search strategy. At every iteration, our random search strategy samples a new individual to evaluate using the ramped half-and-half procedure (see Section 5.1.1) and returns the best found individual after 25,000 fitness evaluations (same as BUSTLE) have elapsed. Before evaluation, every individual undergoes the same search for parameters with GP-UCB of Section 5.2.

From the figure, we see that the number of fitness evaluations is sufficient for BUSTLE to converge. Moreover, BUSTLE significantly outperforms random search (p-values are significant for every comparison), meaning that evolution is indeed performing an intelligent search.

[1] https://github.com/pigozzif/BUSTLESTLLearningFromData

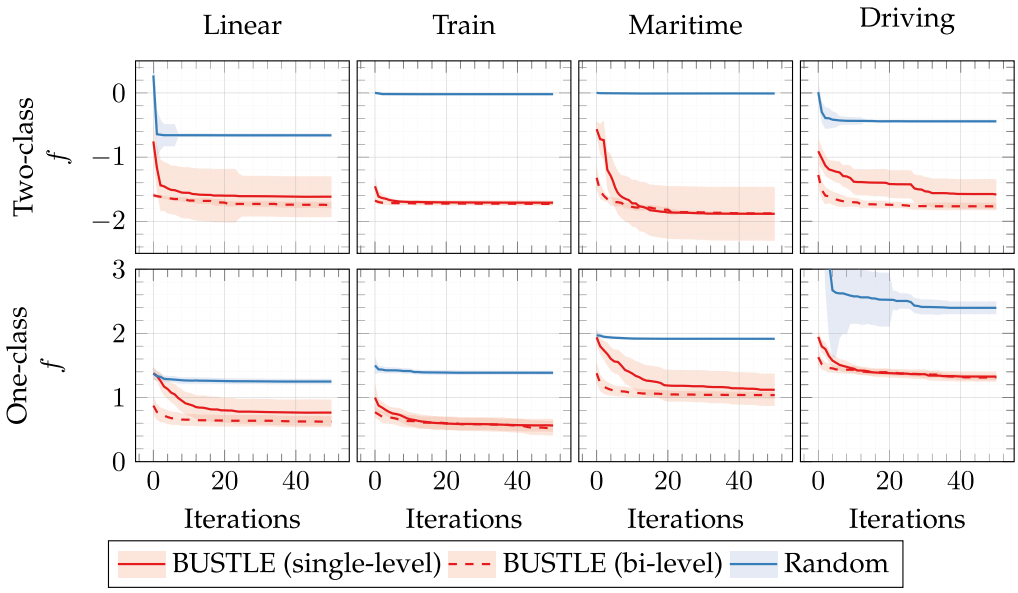

Figure 8: Median ± standard deviation (solid line and shaded area) of the fitness f for the best individuals found during each evolutionary run. BUSTLE evolves solutions that are more effective than random search.

Table 1: Performance indices (median across the runs) for every experimental combination of BUSTLE and, as a baseline, random search. We report running time in s.

	Two-class					One-class				
Variant	FNR	FPR	Acc	Time	n_{nodes}	FNR	FPR	Acc	Time	n_{nodes}
Lin.										
Random	0.20	0.20	0.80	11	8.0	0.98	0.20	0.41	10	8.0
BUSTLE (single-l.)	0.00	0.00	1.00	15	9.5	0.45	0.00	0.77	11	11.0
BUSTLE (bi-l.)	0.00	0.00	1.00	112	12.5	0.40	0.00	0.80	145	11.0
Train										
Random	0.55	0.53	0.46	31	8.0	0.81	0.15	0.52	18	8.0
BUSTLE (single-l.)	0.03	0.05	0.96	26	12.0	0.30	0.12	0.79	25	11.0
BUSTLE (bi-l.)	0.00	0.03	0.98	523	13.0	0.18	0.08	0.87	438	13.5
Marit.										
Random	0.52	0.50	0.49	84	8.0	0.77	0.21	0.51	73	8.0
BUSTLE (single-l.)	0.00	0.00	1.00	109	9.5	0.15	0.49	0.68	72	9.5
BUSTLE (bi-l.)	0.00	0.00	1.00	1,477	9.0	0.38	0.52	0.55	2,008	12.0
Driv.										
Random	0.45	0.23	0.66	47	8.0	0.95	0.27	0.39	44	8.0
BUSTLE (single-l.)	0.00	0.00	1.00	55	6.0	0.94	0.05	0.51	37	9.5
BUSTLE (bi-l.)	0.00	0.00	1.00	598	13.0	0.90	0.03	0.54	1,186	11.0

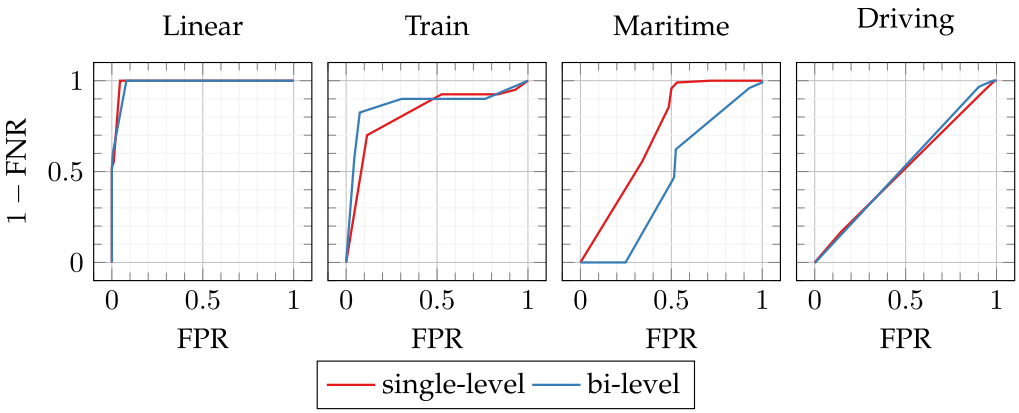

Figure 9: ROC curves of BUSTLE in the one-class case by varying the classification threshold ϵ.

Considering the performance indices of Table 1, for the two-class case, our approach always performs well, reaching FPR and FNR very close to 0.00. For the one-class case, in order to provide fairer and deeper insights, we also visualize how the two indices vary according to the threshold ϵ (see Section 4). We do so in Figure 9 using Receiver Operating Characteristic (ROC) curves, plotting FPR against the true positive rate (i.e., $1 - \text{FNR}$). We recall that in a ROC plot, we ideally want our curves to grab the upper-left corner (corresponding to no false positives and no false negatives). After preliminary experiments, we sampled 21 values of ϵ in $[-0.50, 0.50]$ at regular intervals, as they adequately covered the range of possible behaviors. We remark that the data are normalized, thus ϵ is agnostic with respect to the data set.

From the figure, we see that, in general, BUSTLE performs well, as ROC curves approach the upper-left corner for two out of four datasets. The results of the Maritime and Driving datasets deserve more discussion. We believe the reason to be the difficulty of the datasets, whose regular and anomalous trajectories are probably less separable. In particular, ROC curves for the Driving data set stretch along the diagonal, meaning that performance is not different from that of a random classifier. These facts imply that BUSTLE opens up venues for future research on this topic, which are yet to be explored by the literature.

As any good anomaly detection system, BUSTLE allows the engineer to tune the trade-off between sensitivity (proportion of regular trajectories correctly identified as such, i.e., $1 - \text{FNR}$) and specificity (proportion of anomalous trajectories correctly identified as such, i.e., $1 - \text{FPR}$). As a result, an engineer can tweak ϵ to reflect their preferences between specificity and sensitivity, and this is indeed a great advantage since ϵ is fixed offline and does not impact the evolution itself. These facts altogether point to the flexibility of BUSTLE.

Concerning interpretability, we analyzed all the formulae and found them to be, in general, understandable to a human. We showcase here some examples for the two-class case, which we hand-picked by considering how easily we were able to interpret them. For instance, a formula for the Maritime data set is as follows:

$$\mathrm{G}_{[11,60]}((22.04 < \texttt{x2} < 40.74) \wedge (4.01 < \texttt{x1} < 56.26))$$

which is telling us that, always on the interval [11, 60], x2 must lie in [22.04, 40.74], while x1 must lie in [4.01, 56.26]. It turns out that this formula rules out those trajectories stretching towards the south (see Figure 7), effectively classifying them as anomalous. For the Linear data set, an example of a formula is:

$$(0.35 < \mathtt{x} < 0.58) \wedge (\mathrm{G}_{[12,14]}(\mathtt{x} > 0.48))$$

meaning that x must lie in [0.35, 0.58], and, for the last two samples of the trajectory, it must be greater than 0.48. The formula captures the upward trend of the regular trajectories and rules out the downward trend of the anomalous trajectories. For the Train case, we evolved:

$$(\mathrm{F}_{[22,40]}(\mathtt{vel} > 24.48)) \wedge (\mathrm{F}_{[46,49]}(19.00 < \mathtt{vel} < 26.44))$$

meaning to stay above 24.48 at least once in the interval [22, 40], and between 19.00 and 26.44 at least once in [46, 49]. The formula captures the oscillatory nature of the regular trajectories; at the same time, it excludes deviations from the interval regular trajectories oscillate within, and it does so in an interval (from 22 to 49) where anomalous trajectories have already deviated quite a lot from the regular ones.

We also showcase some examples of the one-class case. For an example on the Linear data set (with $\alpha = 0.1$), we showcase:

$$(\mathtt{x} > 0.27) \wedge ((\mathtt{x} < 0.49)\mathrm{U}_{[3,8]}(\mathtt{x} > 0.49))$$

meaning that, while always being above 0.27, the trajectory must be below 0.49 until a time in the interval [3, 8], at which it must become and stay above 0.49. The formula models the upward behavior of regular trajectories posing stricter constraints than the two-class cases. For the Train data set (with $\alpha = 0.1$), an example is:

$$((\mathtt{vel} > 23.70)\mathrm{U}_{[28,32]}(\mathtt{vel} < 23.70)) \wedge$$
$$((\mathtt{vel} > 25.27)\mathrm{U}_{[31,42]}(\mathtt{vel} < 25.27))$$

meaning to stay above 23.70 in [28, 32], at which the system must be above 23.70; the same pattern appears in the second operand of the $\wedge$ operator with different threshold value and interval. This formula models the oscillating behavior of regular trajectories in [28, 42], an interval that is devoid of anomalous trajectories.

If we take formula complexity into account, Table 1 lists the values of n_{nodes} for all the experimental variants. As we can see, n_{nodes} is in line with the number of nodes in the syntax tree for the formulae reported above, pointing to the human-readability of our formulae in general. We aimed for concise formulae to enhance readability and ease of comprehension for human readers, but BUSTLE can in general discover more complex formulae by, for example, relaxing the temporal depth limit requirements.

To conclude, our results point to the effectiveness of BUSTLE for both variants of our approach.

6.3.2 RQ2: How Do Single-Level and Bi-Level Optimization Differ in the Efficiency-Efficacy Trade-Off?

We aim to assess to what measure single-level and bi-level optimization differ. To this end, we compare the usual performance indices (FPR, FNR, Acc, and running time).

Table 2 reports the p-values obtained from testing, with the Mann-Whitney U test, whether single-level and bi-level optimization have the same median FPR, FNR, and Acc for the two cases. While in the majority of cases single-level optimization performs comparably with bi-level, for the Maritime data set in the one-class case the difference

Table 2: p-values for the two-sided Mann-Whitney U test against the null hypothesis that, for the same data set and case, single-level and bi-level optimization have the same median FNR, FPR, and Acc. Single-level optimization performs comparably with bi-level.

Data set	Case	p (FNR)	p (FPR)	p (Acc)
Linear	two-class	0.3681	0.5173	0.1041
	one-class	0.1567	0.5524	0.7331
Train	two-class	0.8089	1.0000	0.1041
	one-class	$<$0.0001	0.3210	0.7334
Maritime	two-class	0.1117	0.4499	0.1041
	one-class	0.8596	0.0179	0.0140
Driving	two-class	0.3168	0.0110	$<$0.01
	one-class	0.4358	0.2240	0.0457

in FNR is significant, as well as the difference in FPR for the Driving data set in the two-class case.

Focusing on the running time, Table 1 includes the running times for all the experiments. Single-level optimization runs at an order of magnitude faster than bi-level optimization, which tunes the numerical parameters with a GP-UCB local search procedure at every fitness evaluation. On average, single-level optimization runs 1,783% faster than bi-level optimization, a massive gain in speed. We do not report p-values, as they are trivially significant.

6.3.3 RQ3: Is BUSTLE Effective When Compared to Other State-of-the-Art Approaches?

To assess the relevance of our work, we tested BUSTLE against state-of-the-art approaches for optimizing the structure and the parameters of STL formulae. For the comparison, we use FPR, FNR, Acc, and running time as performance indices.

For the two-class case, we summarize the comparison in Table 3; we recall that no other approaches exist for the one-class case. For Nenzi et al. (2018), we re-implemented the approach and executed it on the same machine and with the same number of fitness evaluations we used for the experiments with BUSTLE. For the other works, we report the figures from the corresponding papers (when available).

As far as FPR and FNR are concerned, we appreciate how BUSTLE (both single-level and bi-level) is always competitive with other state-of-the-art approaches, sometimes even more effective. Whenever medians differ, p-values are significant. As far as the running time is concerned, BUSTLE (single-level) runs faster than all of the others (p-values significant), with the exception of Mohammadinejad et al. (2020a) on the Maritime data set. Conversely, the GP-UCB local search procedure makes BUSTLE (bi-level) run slower than the others. It is worth mentioning that the time metric might not necessarily reflect the true performance unless the algorithms were implemented using the same language, the same libraries, and the same machine. Still, we believe our comparisons to be fair, as we tested Nenzi et al. (2018) on the same machine of BUSTLE and, coincidentally, all of the other works carried out their experiments on a MacBook Pro with similar specifics to ours, making the results actually rather comparable.

For the one-class case, no benchmarks are present in the literature. It is worth mentioning that Jha et al. (2019) tackled the problem of learning STL formulae from regular

Table 3: Median FPR, FNR, Acc, and running time (in s) for BUSTLE, other state-of-the-art approaches, and the random search baseline, two-class case. BUSTLE (both single-level and bi-level) performs comparably with (or even better than) random search and other state-of-the-art approaches.

Data set	Algorithm	FNR	FPR	Acc	Time
Linear	Random	0.20	0.20	0.80	11
	BUSTLE (single-level)	0.00	0.00	1.00	15
	BUSTLE (bi-level)	0.00	0.00	1.00	112
	Nenzi et al. (2018)	0.00	0.00	1.00	113
	Mohammadinejad et al. (2020b)	N/A	N/A	0.98	39
Train	Random	0.55	0.53	0.46	31
	BUSTLE (single-level)	0.03	0.05	0.96	26
	BUSTLE (bi-level)	0.00	0.03	0.98	523
	Nenzi et al. (2018)	0.10	0.00	0.95	576
	Mohammadinejad et al. (2020b)	N/A	N/A	0.98	32
Maritime	Random	0.52	0.50	0.49	84
	BUSTLE (single-level)	0.00	0.00	1.00	109
	BUSTLE (bi-level)	0.00	0.00	1.00	1477
	Nenzi et al. (2018)	0.00	0.00	1.00	1599
	Mohammadinejad et al. (2020b)	0.05	0.02	0.96	73
	Bombara and Belta (2021)	N/A	N/A	0.98	140
Driving	Random	0.45	0.23	0.66	47
	BUSTLE (single-level)	0.00	0.00	1.00	55
	BUSTLE (bi-level)	0.00	0.00	1.00	1186
	Nenzi et al. (2018)	0.03	0.00	0.99	1048

trajectories only. While their work is relevant, they do not test the learned STL formulae on unseen anomalous trajectories. To our knowledge, BUSTLE is the first such framework addressing the one-class case.

7 Concluding Remarks

In this paper, we consider the problem of learning specification from data, considering both a two-class and one-class case. In particular, we present BUSTLE, a framework for learning Signal Temporal Logic (STL) formulae. The approach also affords the possibility to choose between a single-level and a bi-level optimization. We evaluate the methodology on three different case studies, comparing, when it was possible, with state-of-the-art approaches.

Results show that BUSTLE performs very well in the two-class case and decently for two out of three datasets in the one-class case. We show, however, how BUSTLE allows the engineer to tune the trade-off between sensitivity and specificity. BUSTLE is able to learn meaningful formulae, understandable to a human. As expected, the single-level optimization runs much faster than the bi-level optimization, but the latter performs slightly better. When compared with other state-of-the-art approaches in the two-class case, BUSTLE (both single-level and bi-level) performs comparatively or even better. No comparison is possible in the one-class case, as BUSTLE is the only tool capable of dealing with this kind of problem.

In the future, we will extend our approach to learning spatio-temporal behaviors. In particular, we will consider the Spatio-Temporal Reach and Escape Logic (STREL) (Nenzi et al., 2022) which is a spatial extension of STL. Considering spatio-temporal data is a much more complicated problem, as the spatial dimension greatly enlarges the expressiveness of the language and, hence, the size of the search space.

Acknowledgments

This research has been partially supported by the Austrian FWF project ZK-35 and by the Italian PRIN project "SEDUCE" n. 2017TWRCNB.

References

Aasi, E., Cai, M., Vasile, C. I., and Belta, C. (2023). Time-incremental learning of temporal logic classifiers using decision trees. In *Learning for Dynamics and Control Conference*, pp. 547–559. PMLR.

Asarin, E., Donzé, A., Maler, O., and Nickovic, D. (2012). Parametric identification of temporal properties. In *Runtime Verification: Second International Conference, Revised Selected Papers 2*, pp. 147–160.

Azad, R., and Ryan, C. (2014). The best things don't always come in small packages: Constant creation in grammatical evolution. In *European Conference on Genetic Programming*, pp. 186–197. Springer.

Bartocci, E., Bloem, R., Maderbacher, B., Manjunath, N., and Ničković, D. (2022). Adaptive testing for specification coverage and refinement in CPS models. *Nonlinear Analysis: Hybrid Systems*, 46:101254. 10.1016/j.nahs.2022.101254

Bartocci, E., Bortolussi, L., Loreti, M., Nenzi, L., and Silvetti, S. (2020). MoonLight: A lightweight tool for monitoring spatio-temporal properties. In *Proceedings of the 20th International Conference*, Runtime Verification, pp. 417–428. Lecture Notes in Computer Science, Vol. 12399.

Bartocci, E., Bortolussi, L., and Sanguinetti, G. (2014). Data-driven statistical learning of temporal logic properties. In *Proceedings of FORMATS*, pp. 23–37.

Bartocci, E., Deshmukh, J., Donzé, A., Fainekos, G., Maler, O., Nickovic, D., and Sankaranarayanan, S. (2018). Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In *Lectures on Runtime Verification*, 135–175. Lecture Notes in Computer Science, Vol. 10457. 10.1007/978-3-319-75632-5_5

Bartocci, E., Mateis, C., Nesterini, E., and Nickovic, D. (2022). Survey on mining signal temporal logic specifications. *Information and Computation*, p. 104957.

Bartoli, A., Carminati, B., Ferrari, E., and Medvet, E. (2016). A language and an inference engine for Twitter filtering rules. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 614–617.

Bartoli, A., De Lorenzo, A., Medvet, E., and Squillero, G. (2019). Multi-level diversity promotion strategies for grammar-guided genetic programming. *Applied Soft Computing*, 83:105599. 10.1016/j.asoc.2019.105599

Bombara, G., and Belta, C. (2021). Offline and online learning of signal temporal logic formulae using decision trees. *ACM Transactions on Cyber-Physical Systems*, 5:1–23. 10.1145/3433994

Bombara, G., Vasile, C.-I., Penedo, F., Yasuoka, H., and Belta, C. (2016). A decision tree approach to data classification using signal temporal logic. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pp. 1–10.

Bruns, R., Dunkel, J., and Offel, N. (2019). Learning of complex event processing rules with genetic programming. *Expert Systems with Applications*, 129:186–199. 10.1016/j.eswa.2019.04.007

Clarke, E. M., Henzinger, T. A., Veith, H., and Bloem, R. (Eds.) (2018). *Handbook of model checking*. Springer.

Dempsey, I., O'Neill, M., and Brabazon, A. (2007). Constant creation in grammatical evolution. *International Journal of Innovative Computing and Applications*, 1(1): 23–38. 10.1504/IJICA.2007.013399

Donzé, A., Ferrer, T., and Maler, O. (2013). Efficient robust monitoring for STL. In *Proceedings of Computer Aided Verification*, pp. 264–279.

Hoxha, B., Dokhanchi, A., and Fainekos, G. E. (2018). Mining parametric temporal logic properties in model-based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer*, 20(1): 79–93. 10.1007/s10009-017-0447-4

Indri, P., Bartoli, A., Medvet, E., and Nenzi, L. (2022). One-shot learning of ensembles of temporal logic formulas for anomaly detection in cyber-physical systems. In *European Conference on Genetic Programming*, pp. 186–197.

Jha, S., Tiwari, A., Seshia, S. A., Sahai, T., and Shankar, N. (2019). TeLEx: Learning signal temporal logic from positive examples using tightness metric. *Formal Methods in System Design*, pp. 1–24.

Jin, X., Donzé, A., Deshmukh, J. V., and Seshia, S. A. (2015). Mining requirements from closed-loop control models. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 34(11): 1704–1717. 10.1109/TCAD.2015.2421907

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*, Vol. 1. MIT Press.

Luke, S. (2009). *Essentials of metaheuristics*. Lulu Raleigh.

Maler, O., and Nickovic, D. (2004). Monitoring temporal properties of continuous signals. In *Proceedings of FORMATS*, pp. 152–166. Lecture Notes in Computer Science, Vol. 3253.

Maler, O., and Nickovic, D. (2013). Monitoring properties of analog and mixed-signal circuits. *International Journal on Software Tools for Technology Transfer*, 15(3): 247–268. 10.1007/s10009-012-0247-9

Marcus, G. (2018). Deep learning: A critical appraisal. arXiv:1801.00631.

Medvet, E., Bartoli, A., and Talamini, J. (2017). Road traffic rules synthesis using grammatical evolution. In *European Conference on the Applications of Evolutionary Computation*, pp. 173–188.

Medvet, E., Nadizar, G., and Manzoni, L. (2022). JGEA: A Modular Java framework for experimenting with evolutionary computation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 2009–2018.

Mohammadinejad, S., Deshmukh, J. V., Puranic, A. G., Vazquez-Chanlatte, M., and Donzé, A. (2020a). Interpretable classification of time-series data using efficient enumerative techniques. In *23rd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 9:1–9:10.

Mohammadinejad, S., Deshmukh, J. V., Puranic, A. G., Vazquez-Chanlatte, M., and Donzé, A. (2020b). Interpretable classification of time-series data using efficient enumerative techniques. *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*.

Nenzi, L., Bartocci, E., Bortolussi, L., and Loreti, M. (2022). A logic for monitoring dynamic networks of spatially-distributed cyber-physical systems. *Logical Methods in Computer Science*, 18(1).

Nenzi, L., Bartocci, E., Bortolussi, L., Silvetti, S., and Loreti, M. (2023). MoonLight: A lightweight tool for monitoring spatio-temporal properties. *International Journal on Software Tools for Technology Transfer*, pp. 1–15.

Nenzi, L., Silvetti, S., Bartocci, E., and Bortolussi, L. (2018). A robust genetic algorithm for learning temporal specifications from data. In *International Conference on Quantitative Evaluation of Systems*, pp. 323–338.

Nguyen, L. V., Kapinski, J., Jin, X., Deshmukh, J. V., Butts, K., and Johnson, T. T. (2017). Abnormal data classification using time-frequency temporal logic. In *Proceedings of Hybrid Systems, Computation, and Control*, pp. 237–242.

Pałka, D., Zachara, M., and Wójcik, K. (2017). Active protocol discoverer based on grammatical evolution. In *International Conference on Information Systems Architecture and Technology*, pp. 95–106.

Pigozzi, F., Medvet, E., and Nenzi, L. (2021). Mining road traffic rules with signal temporal logic and grammar-based genetic programming. *Applied Sciences*, 11(22): 10573. 10.3390/app112210573

Rasmussen, C. E., and Williams, C.K.I. (2005). *Gaussian processes for machine learning (adaptive computation and machine learning)*. MIT Press.

Salamati, A., Soudjani, S., and Zamani, M. (2021). Data-driven verification of stochastic linear systems with signal temporal logic constraints. *Automatica*, 131:109781. 10.1016/j.automatica.2021.109781

Squillero, G., and Tonda, A. (2016). Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization. *Information Sciences*, 329:782–799. 10.1016/j.ins.2015.09.056

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2012). Information-theoretic regret bounds for Gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58:3250–3265. 10.1109/TIT.2011.2182033

Virgolin, M., Alderliesten, T., and Bosman, P. A. (2020). On explaining machine learning models by evolving crucial and compact features. *Swarm and Evolutionary Computation*, 53:100640. 10.1016/j.swevo.2019.100640

Virgolin, M., De Lorenzo, A., Medvet, E., and Randone, F. (2020). Learning a formula of interpretability to learn interpretable formulas. In *Parallel Problem Solving from Nature*, pp. 79–93.

Virgolin, M., Medvet, E., Alderliesten, T., and Bosman, P. A. (2022). Less is more: A call to focus on simpler models in genetic programming for interpretable machine learning. arXiv:2204.02046.

Whigham, P. (1995). Inductive bias and genetic programming. In *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 461–466.

Xu, Z., and Julius, A. A. (2018). Census signal temporal logic inference for multiagent group behavior analysis. *IEEE Transactions on Automation Science and Engineering*, 15(1): 264–277. 10.1109/TASE.2016.2611536