One-Shot Learning of Ensembles of Temporal Logic Formulas for Anomaly Detection in Cyber-Physical Systems

Patrick Indri[1,3], Alberto Bartoli[2][0000−0003−4132−416X], Eric Medvet[2][0000−0001−5652−2113], and Laura Nenzi[1,2,3][0000−0003−2263−9342]

[1] Department of Mathematics and Geosciences, University of Trieste, Italy
[2] Department of Engineering and Architecture, University of Trieste, Italy
[3] TU Wien Informatics, Austria

Abstract. Cyber-Physical Systems (CPS) are prevalent in critical infrastructures and a prime target for cyber-attacks. Multivariate time series data generated by sensors and actuators of a CPS can be monitored for detecting cyber-attacks that introduce anomalies in those data. We use Signal Temporal Logic (STL) formulas to tightly describe the normal behavior of a CPS, identifying data instances that do not satisfy the formulas as anomalies. We learn an ensemble of STL formulas based on observed data, without any specific knowledge of the CPS being monitored. We propose an algorithm based on Grammar-Guided Genetic Programming (G3P) that learns the ensemble automatically in a single evolutionary run. We test the effectiveness of our data-driven proposal on two real-world datasets, finding that the proposed one-shot algorithm provides good detection performance.

Keywords: Ensemble learning · Grammar Guided Genetic Programming · Specification mining

1 Introduction

Cyber-Physical Systems (CPS) consist of large collections of mechanical components controlled by software modules. Interactions between software modules and physical world occur through the numerous sensors and actuators that compose the CPS. Such systems are prevalent in critical infrastructures and, consequently, have to be considered as a prime target for cyber-attacks. A cyber-attack usually introduces anomalies in the multivariate time-series data generated by sensors and actuators, i.e., deviations from the data generated when the CPS operates normally. Defining which data instances have to be considered normal and which ones anomalous is very complex, though, because such a classification requires a significant amount of CPS-specific knowledge.

Signal Temporal Logic (STL) [16,10] is a formal language for describing properties of signals and, as such, can be used for specifying and monitoring the behavior of a CPS. In this work we address the problem of modeling the correct

behavior of a CPS by means of STL formulas learned automatically from data collected while the CPS is working and not annotated in any way—CPS data is only assumed to be attack-free and fault-free. Violations of the formulas can then be used to identify anomalous behavior when it occurs. Our approach requires no specific domain knowledge and is *template-free*, i.e., we learn both structure and parameters of STL formulas.

We frame the learning task as an optimization problem, employing Grammar Guided Genetic Programming (G3P) [28]. We propose a one-shot evolutionary algorithm that learns an ensemble of STL formulas in a single evolution. Each formula tightly models the available data based on a subset of the observed signals—in a sense, each formula defines a CPS property discovered automatically. The voting scheme used for the ensemble defines a data instance as an anomaly when that instance violates an excessive amount of properties. We apply our technique on two real-world case studies—the SWaT water treatment plant [8] and the N-BaIoT set of Internet-of-Things (IoT) devices [20]—and assess the learned formulas on testing data that include real attacks.

2 Related work

Specification mining is the research field dealing with methods for determining and formalizing the requirements of a target system based on its observed behavior. An important line of research in specification mining for CPSs focuses on STL, in particular, on finding STL formulas satisfied by the observed executions as much as possible. *Template-based* methodologies [13,12] rely on a specific, user-defined template formula, and limit the learning process to determining good parameters for the template. Such methodologies usually assume that the available data do not contain any anomalies and introduce a *tightness* metric to favor STL formulas that satisfy the data as tightly as possible [12]. The more challenging *template-free* approaches, on the other hand, learn both the structure and the parameters of the STL formulas. Unlike our proposal, template-free approaches usually require training data with both normal and anomalous examples annotated as such [21,2,6]. Template-free STL mining with normal data only was proposed in [24], that also exploited evolutionary computation (as [21] did): differently than this work, the cited papers do not produce ensembles of STL formulas, and are hence less suitable for CPSs where more properties should be monitored at once for an effective anomaly detection.

Other kinds of artifacts, different from STL specification, have been used for anomaly detection in CPSs. A powerful method on the SWaT testbed has been proposed in [7], based on data-driven mining of invariants (i.e., relations between data instances) expressed with an ad-hoc formalism. Anomaly detection on the same testbed has been developed with Generative Adversarial Networks (GAN) [14,15] and Deep Neural Networks (DNN) [11,9]. The anomaly criteria embedded in the neural networks trained on the observed data, however, are intrinsically much less interpretable than those resulting from an ensemble of STL formulas.

Ensemble learning has been proposed as way for learning models from data that collectively capture several properties of the underlying system, hence improving the models effectiveness. There are several cases in which ensemble learning has been combined with evolutionary computation and, in particular, with GP. The author of [27] proposed a simple variant of standard GP that allows to learn an ensemble of formulas for symbolic regression in a single evolutionary run, similarly to our case. Forms of ensemble learning with GP have been proposed also as the learning of different formulas on different partitions of the training data—*sequential covering* [23] and *separate-and-conquer* [19,3]. However, those approaches learn the formulas constituting the ensemble sequentially and are thus intrinsically unable for a one-shot framework.

3 Background: Signal Temporal Logic

Syntax and semantics. We assume that the system under analysis can be described by a set of n real-valued variables $V = \{x_1, \ldots, x_n\}$. We define a *signal* (or trace or trajectory) $\boldsymbol{w}$ a function $\boldsymbol{w} : \mathbb{T} \to \mathbb{R}^n$, where $\mathbb{T} = \mathbb{R}_{\geq 0}$ is the time domain, and denote by $x_i(t)$ the value of the i-th variable of $\boldsymbol{w}$ at time $t \in \mathbb{T}$. A signal describes the evolution of the system over time.

The logical statements of STL consist of a combination of temporal operators, Boolean connectives, and propositions, according to the following syntax. Formally, an *STL formula* φ is a string defined as:

$$\varphi \coloneqq \top \mid \mu \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \mid \varphi \mathcal{S}_I \varphi \mid \mathcal{O}_I \varphi \mid \mathcal{H}_I \varphi \tag{1}$$

where $\top$ is the true value, $\neg$, $\wedge$, $\vee$, and $\to$ are the negation, conjunction, disjunction, and implication logical operators, $\mu : \mathbb{R}^n \to \{\top, \bot\}$ is an *atomic proposition* (an inequality of the form $y(x_1, \ldots, x_n) \lesseqgtr c$, with $y : \mathbb{R}^n \to \mathbb{R}$ and $c \in \mathbb{R}$), $\mathcal{S}_I$, $\mathcal{O}_I$, and $\mathcal{H}_I$ are the *Since*, *Once*, and *Historically* temporal operators, and $I \subseteq \mathbb{T}$ is an interval of the form $I = [a, b]$, with $0 \leq a < b$ and $a, b \in \mathbb{T}$.

For a given signal $\boldsymbol{w}$, the satisfaction of an STL formula φ with respect to the signal at a time t can be evaluated according to either qualitative (Boolean) or quantitative (real-valued) semantics [16]. The former states if a signal satisfies a formula at time t; the latter outputs the degree of satisfaction defined in a continuous range. Here, we report only quantitative semantics; the reader may refer to [16,5,17] for more details. The *quantitative satisfaction function* ρ returns a value $\rho(\varphi, \boldsymbol{w}, t) \in \mathbb{R} \cup \{-\infty, +\infty\}$ that quantifies the *robustness* (or satisfaction) degree of an STL formula φ with respect to a signal $\boldsymbol{w}$ at time t, and is defined inductively as:

$$\begin{aligned}
\rho(\top, \boldsymbol{w}, t) &= +\infty \\
\rho(\mu, \boldsymbol{w}, t) &= \begin{cases} y(x_1(t), \ldots, x_n(t)) - c & \text{if } \mu \equiv y(x_1(t), \ldots, x_n(t)) \geq c \\ -y(x_1(t), \ldots, x_n(t)) + c & \text{otherwise} \end{cases} \\
\rho(\neg\varphi, \boldsymbol{w}, t) &= -\rho(\varphi, \boldsymbol{w}, t) \\
\rho(\varphi_1 \wedge \varphi_2, \boldsymbol{w}, t) &= \min(\rho(\varphi_1, \boldsymbol{w}, t), \rho(\varphi_2, \boldsymbol{w}, t))
\end{aligned}$$

$$\rho(\varphi_1 \vee \varphi_2, \boldsymbol{w}, t) = \rho(\neg(\neg\varphi_1 \wedge \neg\varphi_2), \boldsymbol{w}, t)$$
$$\rho(\varphi_1 \rightarrow \varphi_2, \boldsymbol{w}, t) = \rho(\neg\varphi_1 \vee (\varphi_1 \wedge \varphi_2), \boldsymbol{w}, t)$$
$$\rho(\varphi_1 \mathcal{S}_{[a,b]} \varphi_2, \boldsymbol{w}, t) = \sup_{t' \in t-[a,b]} \left(\min\left(\rho(\varphi_2, \boldsymbol{w}, t')\right), \inf_{t'' \in [t', t[} \left(\rho(\varphi_1, \boldsymbol{w}, t'')\right) \right)$$
$$\rho(\mathcal{O}_{[a,b]} \varphi, \boldsymbol{w}, t) = \rho(\top \mathcal{S}_{[a,b]} \varphi, \boldsymbol{w}, t)$$
$$\rho(\mathcal{H}_{[a,b]} \varphi, \boldsymbol{w}, t) = \rho(\neg\mathcal{O}_{[a,b]} \neg\varphi, \boldsymbol{w}, t)$$

The sign of the robustness $\rho(\varphi, \boldsymbol{w}, t)$ provides a link to Boolean semantics [4]. If $\rho(\varphi, \boldsymbol{w}, t) \geq 0$, then $\boldsymbol{w}$ satisfies φ at t, denoted by $(\boldsymbol{w}, t) \models \varphi$; otherwise $\boldsymbol{w}$ does not satisfies φ at t, denoted by $(\boldsymbol{w}, t) \not\models \varphi$.

Tightness. Given two signals $\boldsymbol{w}$, $\boldsymbol{w}'$ over the same time domain and a time t in that domain, the *correctness property* holds, stating that if $(\boldsymbol{w}, t) \models \varphi$ and $\|\boldsymbol{w} - \boldsymbol{w}'\|_\infty < \rho(\varphi, \boldsymbol{w}, t)$ then $(\boldsymbol{w}', t) \models \varphi$, where $\|\boldsymbol{v}\|_\infty = \max_i |v_i|$ is the infinity norm. Intuitively, the correctness property suggests that the value of ρ is an upper bound for perturbations in a signal $\boldsymbol{w}$ for ensuring its satisfaction of φ.

Based on the correctness property, we define the *tightness* of a formula φ with respect to a signal $\boldsymbol{w}$ as $\|\rho(\varphi, \boldsymbol{w}, \cdot)\|_\infty = \max_{t \in \mathbb{T}} |\rho(\varphi, \boldsymbol{w}, t)|$, where $\rho(\varphi, \boldsymbol{w}, \cdot) : \mathbb{T} \rightarrow \mathbb{R} \cup \{-\infty, +\infty\}$ is the *robustness signal* that describes how $\rho(\varphi, \boldsymbol{w}, t)$ varies over time. Moreover, we say that a formula φ *tightly models* a signal $\boldsymbol{w}$ if its tightness is $\|\rho(\varphi, \boldsymbol{w}, \cdot)\|_\infty = 0$. Intuitively, this means that φ is satisfied by the signal $\boldsymbol{w}$, but it is not satisfied by any other signal that is slightly different than $\boldsymbol{w}$.

Finite length signals. Signals related to real systems are defined on a limited time domain, i.e., $\mathbb{T} = [0, t_{\max}]$ instead of $\mathbb{T} = \mathbb{R}_{\geq 0}$. It follows that it is not possible to compute the robustness for certain STL formulas on signals that are defined over a too short time domain. For instance, the operator $\mathcal{H}_{[0,b]}$ cannot be evaluated on signals defined on $[0, t_{\max}]$ with $t_{\max} < b$.

For formalizing this requirement, we introduce the *necessary length* [16] of a formula. The necessary length $\|\varphi\| \in \mathbb{R}_{\geq 0}$ of a formula φ, is defined inductively as:

$$\|\top\| = 0 \qquad \|\varphi_1 \rightarrow \varphi_2\| = \max\left(\|\varphi_1\|, \|\varphi_2\|\right)$$
$$\|\mu\| = 0 \qquad \|\varphi_1 \mathcal{S}_{[a,b]} \varphi_2\| = \max\left(\|\varphi_1\|, \|\varphi_2\|\right) + b$$
$$\|\neg\varphi\| = \|\varphi\| \qquad \|\mathcal{O}_{[a,b]} \varphi\| = \|\varphi\| + b$$
$$\|\varphi_1 \vee \varphi_2\| = \max\left(\|\varphi_1\|, \|\varphi_2\|\right) \qquad \|\mathcal{H}_{[a,b]} \varphi\| = \|\varphi\| + b$$
$$\|\varphi_1 \wedge \varphi_2\| = \max\left(\|\varphi_1\|, \|\varphi_2\|\right)$$

Given a signal $\boldsymbol{w}$, we denote by $|\boldsymbol{w}| = t_{\max}$ the *signal length*. The robustness $\rho(\varphi, \boldsymbol{w}, t)$ of a formula φ can be evaluated only for $\|\varphi\| \leq t \leq |\boldsymbol{w}|$. As a consequence, if $|\boldsymbol{w}| < \|\varphi\|$, the robustness cannot be computed for any t and the degree of satisfaction is undecidable.

In an ideal experimental setting, $|\boldsymbol{w}| \gg \|\varphi\|$ holds and no indecision arises; this condition is satisfied by all our experimental settings.

On discrete signals. In many practical cases, as, e.g., for CPSs, the signal describing the evolution of the system is a multivariate time series generated by querying, at regular intervals, the sensors and actuators that constitute the system.

Particularly, we assume that the system is observed every Δt seconds, and that interval bounds are expressed in units of Δt. It follows that the time domain is $\mathbb{T} \subseteq \mathbb{N}$ and that the length of a time interval corresponds to its cardinality—e.g., for $I = [1, 4]$, $|I| = 4$. Accordingly, the *length* of a signal is redefined as the cardinality of its time domain.

The necessary length for formulas remains as above, with the exceptions of $\|\top\|$ and $\|\mu\|$, that are defined as $\|\top\| = \|\mu\| = 1$.

4 Problem statement

Let $\boldsymbol{w}_{\text{train}}$ be a signal describing the evolution of a discrete-time system that operates normally—i.e., as intended, in absence of anomalies—in the time interval $I_{\text{train}} = [t_0, t_1]$. Let $\boldsymbol{w}_{\text{test}}$ be a signal describing the behavior of the same system in a later time interval $I_{\text{test}} = [t_2, t_3]$, with $t_1 < t_2$, for which it is not known whether the system is operating normally or not normally. We are interested in finding a way for generating, based on $\boldsymbol{w}_{\text{train}}$, a collection Φ of one or more STL formulas that can be used for finding anomalies in $\boldsymbol{w}_{\text{test}}$, i.e., determining the (possibly empty) subset I' of I_{test} containing all and only the time instants in which the system is not operating normally.

We assume that, given a collection $\Phi = \{\varphi_1, \varphi_2, \dots\}$ of STL formulas and a threshold $\tau \in [0, 1]$, the subset of time instants in which the system is not operating normally is obtained as:

$$I_{\text{anomalous}} = \left\{ t \in I_{\text{test}} : \frac{1}{|\Phi|} \left| \{\varphi \in \Phi : (\boldsymbol{w}_{\text{test}}, t) \not\models \varphi\} \right| > \tau \right\}$$

Intuitively, at each time instant t, we consider the proportion of formulas not satisfied by $\boldsymbol{w}_{\text{test}}$ at t and compare it against τ: if the proportion is greater than the threshold, we say that the system is not operating normally at t. The normality of the behavior at t is hence based on a voting scheme based on Φ formulas. It should be noted that, due to the constraints related to the necessary length of formulas, an initial part of $\boldsymbol{w}_{\text{test}}$ cannot be evaluated.

We remark that we are stating the problem as an anomaly detection problem, no observations are present in $\boldsymbol{w}_{\text{train}}$ that describe how the system operates not normally; examples of anomalies are hence not available for generating Φ. Moreover, we remark that we implicitly assume that, when anomalies occur, the system output, captured by $\boldsymbol{w}_{\text{test}}$, differs from the output of the system under normal operation. If this is not the case, detection of anomalous behavior is not possible.

5 Methodology

To address the challenging task of detecting anomalies, learning from normal behavior only, we propose an evolutionary optimization approach based on two key ideas. First, we learn ensembles of STL formulas, instead of single formulas: the aggregation of the predictions of many low-bias, high variance models can favor generalization [27]. Second, we look for STL formulas that tightly model the training signal $\boldsymbol{w}_{\text{train}}$, instead of just modeling it: since we have only observations of the system operating normally, we are hence assuming that small deviations from the observed behavior are anomalous. The combination of these two key ideas and the voting scheme employed when looking for anomalies corresponds to learning an ensemble of STL formulas, each one tightly describing a specific property of the system, and to saying that an anomaly occurs when the behavior of the system is not consistent with at least a given proportion of these properties.

We use a grammar-based version of Genetic Programming (Grammar-Guided GP, G3P) for performing the search in the space of (ensemble of) STL formulas. G3P is naturally suited to our scenario, since the language of formulas is defined by means of a context-free grammar (see Eq. (1)). Moreover, GP has been recently shown to be naturally suited for learning ensembles of models in an efficient and effective way [27]. In fact, we propose a *Complex Simultaneous Ensemble Learning Algorithm* (CESL-Alg) [27], that is, an algorithm that obtains an ensemble of estimators in a single GP evolution, where we exploit the fact that GP itself is a population-based technique and naturally deals with ensembles of individuals. When doing our evolutionary search each individual is a single STL formula, but the overall outcome is an ensemble and the ensemble is learned in a single evolution—i.e., we do *one-shot learning* of ensembles. In the next sections, we describe the key components of our approach.

Solution representation. During the evolutionary search, each individual is a string of the language defined by the context-free grammar of Figure 1. The grammar encodes numbers with a precision of two decimals in the $[0.00, 0.99]$ range and interval bounds with a single digit precision in the $[0, 9]$ range. Intervals are interpreted as $I = [d_1, d_1 + \max(1, d_2)]$. The grammar also defines the temporal operators $\mathcal{S}_{[a,b]}$, $\mathcal{O}_{[a,b]}$, and $\mathcal{H}_{[a,b]}$, and the logical operators $\wedge$, $\vee$, $\neg$, and $\rightarrow$. For simplicity, and for the kind of problems we deal with in this study, the grammar specify propositions that are in the form $x_i \lessgtr c$.

The grammar of Figure 1 poses no explicit limit on the complexity of a formula, allowing for formulas with very large necessary length resulting from the nesting of many temporal operators. However, during the evolution, we enforce a maximum depth to the derivation trees of the formulas, which limits the nesting. Moreover, the range of temporal operators is limited to $[0, 9]$. Other means could be used to impact on the complexity of evolved formulas as, e.g., using a different grammar for the same language with repeated production rules [22]—as shown in [18], this could result in better evolvability.

$$
\begin{aligned}
\langle \text{formula} \rangle &\coloneqq \langle \text{proposition} \rangle \mid \langle \text{operator} \rangle \\
\langle \text{operator} \rangle &\coloneqq \neg \langle \text{formula} \rangle \mid \langle \text{formula} \rangle \wedge \langle \text{formula} \rangle \mid \langle \text{formula} \rangle \vee \langle \text{formula} \rangle \mid \\
&\quad \langle \text{formula} \rangle \rightarrow \langle \text{formula} \rangle \mid \langle \text{formula} \rangle \mathcal{S}_{[\langle \text{ibound} \rangle, \langle \text{ibound} \rangle]} \langle \text{formula} \rangle \mid \\
&\quad \mathcal{O}_{[\langle \text{ibound} \rangle, \langle \text{ibound} \rangle]} \langle \text{formula} \rangle \mid \mathcal{H}_{[\langle \text{ibound} \rangle, \langle \text{ibound} \rangle]} \langle \text{formula} \rangle \\
\langle \text{proposition} \rangle &\coloneqq \langle \text{variable} \rangle \langle \text{comparison} \rangle \langle \text{number} \rangle \\
\langle \text{ibound} \rangle &\coloneqq \langle \text{digit} \rangle \\
\langle \text{variable} \rangle &\coloneqq x_1 \mid \ldots \mid x_n \\
\langle \text{comparison} \rangle &\coloneqq \geq \mid < \\
\langle \text{number} \rangle &\coloneqq 0.\langle \text{digit} \rangle \langle \text{digit} \rangle \\
\langle \text{digit} \rangle &\coloneqq 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

Fig. 1: Our context-free grammar for a system with variables $x_1, \ldots, x_n$.

Fitness function. We aim at defining a fitness function that measures, for a given signal, (i) the tightness of a formula and (ii) the overall length of time intervals when the formula is not satisfied. For both, the lower, the better. Thus, we measure the fitness $f(\varphi, \boldsymbol{w})$ of a candidate STL formula on a signal $\boldsymbol{w}$ as:

$$
f(\varphi, \boldsymbol{w}) = \frac{1}{|\boldsymbol{w}| - \|\varphi\|} \sum_{\|\varphi\| \leq t \leq |\boldsymbol{w}|} \begin{cases} \rho(\varphi, \boldsymbol{w}, t) & \text{if} \quad \rho(\varphi, \boldsymbol{w}, t) \geq 0 \\ k & \text{if} \quad \rho(\varphi, \boldsymbol{w}, t) < 0 \end{cases} \tag{2}
$$

where $k \in \mathbb{R}_{>0}$ is a parameter corresponding to a penalty for instants when the signal does not satisfy the formula.

The proposed function reaches its minimum (zero) for formulas that tightly model the signal $\boldsymbol{w}$, i.e., those having a robustness always equals to 0. Additionally, the fitness function favors formulas with positive robustness: by means of the parameter k, formulas with robustness signal that assumes many negative values will be penalized. A higher value of k will penalize formulas with negative robustness more. This, in turn, favors solutions that better agree with the training data, leading to fewer false positives: since we learn STL formulas from normal data only and we consider instants with negative robustness as anomalous, the parameter k can be interpreted as a penalty for false positives on the training data, on the assumption that a *positive* instant is an anomaly.

One-shot evolutionary algorithm. Inspired by [27], we propose a variant of G3P that produces an ensemble of STL formulas in a single evolutionary run.

A key requirement for ensemble learning to be effective is that models of the ensemble should be independent. While G3P, being a population-based optimization algorithm, can very efficiently learn many models at the same time, it might fall short in ensuring their independency, due to the lack of diversity and premature convergence that frequently afflict GP [25]. In our proposal, we attempt to minimize the risk of premature convergence as detailed below.

Our proposal employs iteratively a form of *extinction* to remove individuals from the population once it has converged to a solution, substituting them with *random immigrants*, randomly generated individuals that introduce fresh genetic material and favor diversity. Particularly, at each iteration we perform three phases.

- *Population update.* For each variable defined by the grammar, a *group* including all the individuals (STL formulas) that contain said variable is built. Individuals with multiple distinct variables will thus belong to multiple groups. The best individual of each group is copied into the next generation, as a form of *elitism*. The offspring for the next generation is completed by reproducing individuals from the current generation—including elites and with no consideration for grouping—using tournament selection with enforced-diversity (i.e., genetic operators are applied on parents until the child is not present in the population) and a non-overlapping generational model.
  Since all groups propagate their best individual, the population update promotes the presence of all variables at each iteration, avoiding the utter predominance of a small percentage of variables that may appear in fit individuals.
- *Solutions update.* If some individuals *solve the problem* (e.g., they have $f = 0$), consider the groups these solutions belong to. All the individuals belonging to these groups are removed from the population (extinction) and added to the solutions ensemble. The population is then refilled with newly generated individuals (random immigrants).
  The solution update exploits individuals homogenisation to find near-optimal solutions: once a solution is found, individuals belonging to its groups are expected to have good—although sub-optimal—performance whilst retaining some diversity. They can thus be added to the solution ensemble and be replaced with new, randomly generated individuals, to encourage exploration of other areas of the search space.
  Usually, not every individual of the population has variables in common with individuals that currently solve the problem, and the population is not entirely replaced with new individuals (which would essentially be equivalent to restarting the evolutionary process). This is especially true if many variables are involved. A computational advantage over repeated standard G3P evolutions is therefore expected.
- *Stop condition.* When n_{target} distinct variables have been solved (i.e., n_{target} distinct variables appear in formulas that *solve the problem*), the stop condition is met.
  The stop condition can be used to explore a greater amount of the search space and act against premature convergence. It controls the trade-off between performance and efficiency.

We remark that in the context of anomaly detection for CPSs, it is not "hard" to find a formula with a perfect fitness: a trivial case is the one of a proposition $x_i \geq c$ for a signal whose $x_i(t) = c$ for any t. On one hand, this makes the solution

update phase actually triggerable. On the other hand, it makes the ability of the ensemble to discover anomalies dependent on the number of variables occurring in the formulas: for this reason, we use n_{target} as stopping criterion. Moreover, since no pruning is performed on the ensemble, the ensemble size is (indirectly) controlled by n_{target}.

Algorithm 1 presents our one-shot G3P algorithm in detail. The *population update* step (lines 6–14) builds the variable *groups*, propagates the best individual for each group and fills the population of n_{pop} individuals. In line 12, a single child individual is generated by the `selectAndReproduceWithEnfDiv` procedure, using tournament selection with enforced-diversity. In the *solutions update* step (lines 15–25), the variables of the individuals that satisfies the `isSolution` condition are considered and added to the set of solved variables V_{solved}; all individuals that contain at least one of these variables are extracted from the population and added to the solutions ensemble S. If necessary, the population is refilled with newly generated individuals (lines 26–28). The stop condition (line 5) counts the number of distinct variables in V_{solved} and, stops the iterative algorithm if $V_{\text{solved}} \geq n_{\text{target}}$. The algorithm returns the ensemble of solutions S.

6 Experimental evaluation

6.1 Datasets and preprocessing

We considered two real-world case studies to evaluate our proposal, to investigate its performance in the anomaly detection task and its efficiency in learning the ensemble of STL formulas.

The Secure Water Treatment (SWaT) [8] testbed is a scaled down water treatment plant for research in the area of cyber security. Data log is collected every $\Delta t = 1\,\text{s}$ for 495 000 s under normal operation ($|\boldsymbol{w}_{\text{train}}| = 495\,000$) and for 449 920 s with attack scenarios ($|\boldsymbol{w}_{\text{test}}| = 449\,920$). The dataset consists of 24 sensors and 26 actuators readings, for a total of 50 attributes. Sensor readings are numerical variables, whilst actuator readings are ternary non-ordinal variables. In $\boldsymbol{w}_{\text{test}}$ the dataset contains 36 attacks: attacks can affect a single component of the testbed, or span across different components and different stages of the water treatment procedure. A detailed description of the attacks can be found in [8]. Actuators assume the binary values on/off, and a third value corresponding to a short-lasting transition state [26]; we convert actuator variables to binary variables, replacing the transition state with the state towards which the transition is headed.

N-BaIoT is a suite of nine datasets originally proposed in [20] obtained by monitoring nine commercial IoT devices, operating both under normal and anomalous conditions. Benign data and attack data under several attack conditions is collected; particularly, the devices are infected with Mirai and BASHLITE, two well known IoT malware that can be used to perform botnet attacks. Botnet attacks aim at the creation of a network of infected devices, to perform distributed attacks. The attacks are described in greater detail in [20]. Separately for each device, the datasets collect 115 traffic statistics every, extracted

```
1  function evolve():
2      P ← initialise(n_pop)
3      S ← ∅
4      V_solved = ∅
5      while countDistinct(V_solved) < n_target do
6          P' ← ∅
7          {P_1, ..., P_n} ← buildGroups(P)
8          foreach i ∈ {1, ..., n} do
9              P' ← P' ∪ best(P_i)
10         end
11         while |P'| < n_pop do
12             P' ← P' ∪ selectAndReproduceWithEnfDiv(P)
13         end
14         P ← P'
15         foreach p ∈ P do
16             if isSolution(p) then
17                 {v_1, ..., v_k} ← getVariables(p)
18                 V_solved = V_solved ∪ {v_1, ..., v_k}
19                 foreach v ∈ {v_1, ..., v_k} do
20                     S' ← getIndividualsWithVariable(P, v)
21                     S ← S ∪ S'
22                     P ← P \ S'
23                 end
24             end
25         end
26         if |P| < n_pop then
27             P ← P ∪ initialise(n_pop − |P|)
28         end
29     end
30     return S;
31 end
```

Algorithm 1: One-shot algorithm.

from raw network traffic, $\Delta t = 1\,\mathrm{s}$; all attributes are numerical. Considering all datasets, a total of 555 937 benign and 7 329 517 malign observations is collected. Similarly to [20], we used 2/3 of the benign observations as the training set, and concatenated the remaining benign observations and the attack observations to build the test set. Considering median values across the nine datasets, $|\boldsymbol{w}_{\text{train}}| = 33\,032$ and $|\boldsymbol{w}_{\text{test}}| = 844\,327$.

In accordance with the grammar of Figure 1, we rescaled numerical features to $[0.00, 0.99]$, using min-max normalization, and converted binary states $\{\text{off}, \text{on}\}$ to numerical variables $\{0.00, 0.99\}$. It should be noted that we perform rescaling on the training set only. Consequently, test observations can assume values outside $[0.00, 0.99]$ on numerical variables. This is consistent with the proposal of modeling normal behavior, where values outside the normal ranges may suggest anomalous behavior. Additionally, this choice makes online anomaly detection

feasible, since the rescaling of a test observation does not require the entirety of the test set.

6.2 Procedure and evaluation metrics

We investigated both the the efficiency of our one-shot G3P and the effectiveness in detecting anomalies of the evolved STL. In particular, we were interested in verifying that (i) our one-shot G3P learns STL ensembles faster than a set of executions of plain G3P and (ii) the evolved ensembles are better in detecting anomalies than single STL formulas.

For putting the results of one-shot G3P in perspective, we considered a baseline consisting in a serial execution of 30 runs, with different random seeds, of a plain version of G3P with the same representation, genetic operators, and fitness function (along with other key parameters) of our one-shot G3P. By taking the ensemble composed of the best individuals (all those with perfect fitness) at the last generation of n of the 30 runs, we were able to compare our one-shot G3P against a baseline that evolves few STL formulas (with $n = 1$) or with a G3P-based ensemble learning technique that is not one-shot (with $n > 1$). In other words, in this baseline n allows to control the efficiency-effectiveness trade-off, on the assumption that the larger the ensemble, the better the detection effectiveness and the longer the learning. We call this baseline *multi-run G3P*.

In our one-shot G3P, we used $n_{\text{target}} = 20$, Equation (2) with $c = 1$ as the fitness function, and $f = 0$ as the `isSolution()` condition. In both our proposal and the baseline we used the ramped half-and-half initialization with derivation trees depth in $[3, 20]$, a population size of 200 individuals, a maximum tree depth of 20 when applying genetic operators, a tournament size of 5. We used standard G3P mutation and crossover for producing, respectively, 20 % and 80 % of the offspring. When enforcing diversity, we did a maximum of 100 applications of the genetic operators.

Concerning the thresholds τ, we set it in such a way that 20 and 1 not satisfied formulas, respectively for SWaT and N-BaIoT, suffice for raising an anomaly. We set these values after exploratory analysis.

We implemented[4] our proposal in the Java programming language, building on the tool of [24] which in turns employ the STL monitoring tool Moonlight [1].

For both one-shot G3P and the baseline, we used the trailing 20 % of $\boldsymbol{w}_{\text{train}}$ as a validation signal. We computed the fitness on the leading 80 % and, at the end of the evolution, we discarded STL formulas that resulted in FPR > 0 on the validation set.

Effectiveness and efficiency metrics. We evaluated the anomaly detection effectiveness by means of the True Positive Rate (TPR), the False Positive Rate

[4] The code is publicly available at `https://github.com/pindri/OneShot-ensemble-learning-anomaly-detection-MTS`.

(FPR), and the Area Under the Curve (AUC), obtained by varying τ at prediction time. We adopt the convention that *positive* denotes anomalous instants and *negative* denotes normal (i.e., not anomalous) instants.

Concerning learning efficiency, since in G3P the largest proportion of the computational effort lies in determining the fitness of a solution, we use the number of fitness evaluations f_{evals} to measure efficiency.

6.3 Results

Table 1 presents a comparison of the multi-run G3P and our one-shot G3P over the 10 problems. For the multi-run approach, a single ensemble was produced out of 30 runs for each dataset. For the one-shot algorithm, instead, median values across 10 runs are reported.

The one-shot approach compares favorably with the multi-run one, reaching higher AUC in 6 out of 10 datasets. The one-shot algorithm performs markedly better in N-BaIoT-4 and N-BaIoT-8, where the multi-run baseline ensemble detects no anomaly. Moreover, the one-shot G3P requires significantly fewer f_{evals}, resulting in a substantial efficiency improvement. It should however be noted that the one-shot G3P results in larger ensembles: on SWaT, the median ensemble size is 934 with one-shot and 82 with multi-run—for both, no ensemble pruning was performed.

Table 1: Comparison of multi-run and one-shot G3P. For each dataset, the highest AUC and the lowest f_{evals} between the two approaches are highlighted.

	Multi-run G3P (30 runs)				One-shot G3P ($n_{\text{target}} = 20$)			
Dataset	TPR	FPR	AUC	f_{evals}	TPR	FPR	AUC	f_{evals}
SWaT	0.6648	0.0005	0.8321	43 243	0.6571	0.0007	**0.8401**	**11 767**
N-BaIoT-1	0.9981	0.0000	**0.9990**	47 152	0.8952	0.0011	0.9475	**3297**
N-BaIoT-2	0.9996	0.0016	0.9989	355 696	1.0000	0.0422	**0.9998**	**5732**
N-BaIoT-3	0.9949	0.0000	**0.9974**	51 979	0.9596	0.0076	0.9739	**5965**
N-BaIoT-4	0.0000	0.0002	0.4998	298 158	0.9272	0.0025	**0.9632**	**35 811**
N-BaIoT-5	0.6152	0.0012	0.7681	156 033	0.7492	0.0010	**0.8742**	**7898**
N-BaIoT-6	0.7192	0.0011	**0.8594**	371 358	0.6807	0.0023	0.8387	**12 235**
N-BaIoT-7	0.7070	0.0000	0.8534	269 708	0.6896	0.0009	**0.9072**	**16 736**
N-BaIoT-8	0.0000	0.0000	0.5000	1 015 286	0.4166	0.0027	**0.7050**	**88 921**
N-BaIoT-9	0.7812	0.0005	**0.8905**	260 259	0.7440	0.0011	0.8702	**13 696**

If we consider an attack scenario as detected when at least one instant during anomalous behavior is labeled as anomalous, then the one-shot approach detects a median of 9 on 36 attacks on SWaT and all attacks on N-BaIoT.

Figure 2 presents an alternative comparison between the baseline and the one-shot approach. The results are displayed in terms of f_{evals} vs. AUC, with

the optimum being located in the top left corner (i.e., few fitness evaluations—denoting high efficiency—and high AUC). The results show that reducing the number of runs for building ensembles in multi-run G3P monotonically increases efficiency, but reduces effectiveness as well. Single run ensembles learned with G3P have a tendency to produce solutions with AUC ≈ 0.5 which, in this case, usually denotes STL solutions that are always satisfied by the test set and that identify no anomalies. This, in retrospective, motivates the ensemble learning approach, since the results show that a single standard G3P run does not reliably produce useful formulas.

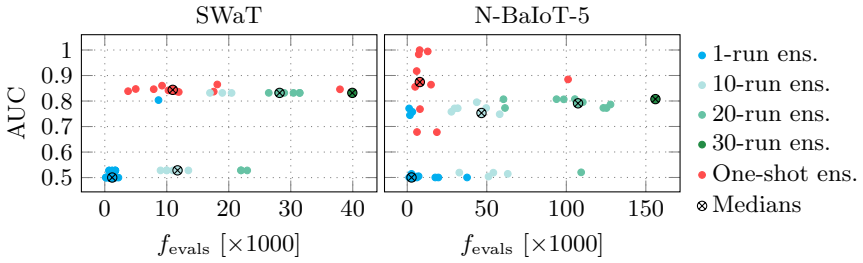

Fig. 2: Comparison of multi-run and one-shot ensembles, efficiency (f_{evals}) vs. performance (AUC).

Formulas complexity. Figure 3 considers all the STL formulas generated with either the standard G3P approach or our one-shot algorithm. Limiting the analysis to SWaT, we investigate the complexity of the formulas—in isolation, regardless of the ensemble they belong to—in terms of the number of distinct variables, and the necessary length. The latter can be used as a measure of temporal complexity.

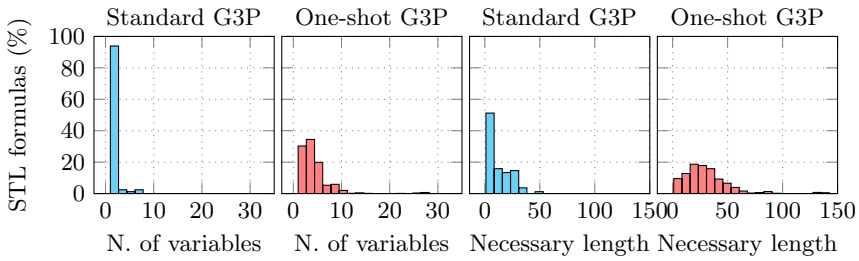

Fig. 3: Complexity of STL formulas obtained with standard G3P and one-shot algorithm on SWaT, in terms of number of variables and necessary length.

With regards to the number of variables, the standard G3P runs produce significantly simpler formulas, with more than 60 % of the formulas containing a

single variable. The one-shot algorithm, instead, produces a larger percentage of solutions with more variables, with some STL formulas containing more than 20 variables. With regards to the necessary length, the one-shot algorithm produces, once more, formulas that are significantly more complex. In fact, approximately 35 % of standard G3P formulas have unitary necessary length: this indicates that they do not contain temporal operators, but rather consist of (combinations of) atomic propositions.

For the standard G3P approach, the large percentage of formulas with small number of variables and small necessary length suggests that premature convergence may indeed be a problem: the evolutions typically converge to *temporally simple* formulas that consider only few variables. Each of these solutions perfectly model the training set in terms of fitness, but is not able to capture actual anomalous behaviors, since it considers only few aspects of the CPS.

Comparison with literature We can perform a qualitative[5] comparison of our results and the existing literature. [14] and [15] perform anomaly detection on the SWaT testbed using Generative Adversarial Networks (GAN), whilst [11] uses Deep Neural Networks (DNN). These approaches reach comparable TPR but, when specified, suffer from higher FPR: TPR = 0.6464 and FPR = 0.0046 for [14], TPR = 0.6734 and precision = 0.9897 for [15], and TPR = 0.6785 and precision = 0.9830 for [11]. [7], instead, performs significantly better than our proposal, reaching TPR = 0.7087 with comparable FPR, using invariant-based anomaly detection. The N-BaIoT dataset is used in [20] employing Deep Autoencoders, where all anomalies are detected with low FPR.

Thus, our proposal is competitive on SWaT, whilst it compares unfavourably on N-BaIoT, where it reaches a perfect detection rate only on N-BaIoT-2. However, as mentioned, on N-BaIoT at least one anomalous instant for each attack is correctly identified, and all attacks might thus be considered as identified.

With regards to interpretability and explainability, however, our proposal is potentially better than GANs, DNNs, and Deep Autoencoders. Crucially, approaches based Neural Networks result in black-box models, where an in-depth investigation on the detected anomalies is usually impossible. Our approach is, to a degree, both interpretable and explainable, since the STL expressions that cause the detection of an anomaly could be singled out and investigated, and are human-readable. These considerations suggest that, in cases where our proposal is bested by more performing approaches, it could be used as a complementary tool, offering insights on the detection process.

7 Conclusions

We proposed a one-shot GP algorithm for ensemble learning, evaluating its performance on an anomaly detection task. We compared our proposal with ensembles obtained by repeated evolutions of a standard GP implementation. We

[5] For the SWaT testbed, different versions of the dataset exist. Thus, no direct quantitative comparison can be made.

deem our results satisfactory and we can summarize the merits of our proposal as follows: (i) it obtains an ensemble of STL formulas more efficiently than repeated independent GP runs, whilst reaching comparable detection performance, (ii) it competes with some of the results available in literature and, when surpassed by more performing but less interpretable methods, can still be useful to gain insights on the detection procedure.

In the future, this work could possibly be extended by an analysis of the role of n_{target} in the trade-off between performance and efficiency, and, additionally, by the implementation of techniques to reduce the size of the one-shot ensembles, to improve explainability and interpretability.

References

1. Bartocci, E., Bortolussi, L., Loreti, M., Nenzi, L., Silvetti, S.: Moonlight: A lightweight tool for monitoring spatio-temporal properties. In: International Conference on Runtime Verification. pp. 417–428. Springer (2020)
2. Bartocci, E., Bortolussi, L., Sanguinetti, G.: Data-driven statistical learning of temporal logic properties. In: International conference on formal modeling and analysis of timed systems. pp. 23–37. Springer (2014)
3. Bartoli, A., De Lorenzo, A., Medvet, E., Tarlao, F.: Learning text patterns using separate-and-conquer genetic programming. In: European Conference on Genetic Programming. pp. 16–27. Springer (2015)
4. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Formal Methods in System Design **51**(1), 5–30 (2017)
5. Donzé, A., Ferrere, T., Maler, O.: Efficient robust monitoring for STL. In: International Conference on Computer Aided Verification. pp. 264–279. Springer (2013)
6. Ergurtuna, M., Gol, E.A.: An efficient formula synthesis method with past signal temporal logic. IFAC-PapersOnLine **52**(11), 43–48 (2019)
7. Feng, C., Palleti, V.R., Mathur, A., Chana, D.: A Systematic Framework to Generate Invariants for Anomaly Detection in Industrial Control Systems. In: NDSS (2019)
8. Goh, J., Adepu, S., Junejo, K.N., Mathur, A.: A dataset to support research in the design of secure water treatment systems. In: International conference on critical information infrastructures security. pp. 88–99. Springer (2016)
9. Goh, J., Adepu, S., Tan, M., Lee, Z.S.: Anomaly detection in cyber physical systems using recurrent neural networks. In: 2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE). pp. 140–145. IEEE (2017)
10. Goranko, V., Rumberg, A.: Temporal Logic. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, summer 2020 edn. (2020)
11. Inoue, J., Yamagata, Y., Chen, Y., Poskitt, C.M., Sun, J.: Anomaly detection for a water treatment system using unsupervised machine learning. In: 2017 IEEE International Conference on Data Mining Workshops (ICDMW). pp. 1058–1065. IEEE (2017)
12. Jha, S., Tiwari, A., Seshia, S.A., Sahai, T., Shankar, N.: TeLEx: learning signal temporal logic from positive examples using tightness metric. Formal Methods in System Design **54**(3), 364–387 (2019)

13. Jin, X., Donzé, A., Deshmukh, J.V., Seshia, S.A.: Mining requirements from closed-loop control models. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **34**(11), 1704–1717 (2015)
14. Li, D., Chen, D., Goh, J., Ng, S.k.: Anomaly detection with generative adversarial networks for multivariate time series. arXiv preprint arXiv:1809.04758 (2018)
15. Li, D., Chen, D., Jin, B., Shi, L., Goh, J., Ng, S.K.: MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks. In: International Conference on Artificial Neural Networks. pp. 703–716. Springer (2019)
16. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, pp. 152–166. Springer (2004)
17. Maler, O., Ničković, D.: Monitoring properties of analog and mixed-signal circuits. International Journal on Software Tools for Technology Transfer **15**(3), 247–268 (2013)
18. Manzoni, L., Bartoli, A., Castelli, M., Gonçalves, I., Medvet, E.: Specializing Context-Free Grammars With a (1+1)-EA. IEEE Transactions on Evolutionary Computation **24**(5), 960–973 (2020)
19. Medvet, E., Bartoli, A., Carminati, B., Ferrari, E.: Evolutionary inference of attribute-based access control policies. In: International Conference on Evolutionary Multi-Criterion Optimization. pp. 351–365. Springer (2015)
20. Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., Elovici, Y.: N-baiot—network-based detection of iot botnet attacks using deep autoencoders. IEEE Pervasive Computing **17**(3), 12–22 (2018)
21. Nenzi, L., Silvetti, S., Bartocci, E., Bortolussi, L.: A robust genetic algorithm for learning temporal specifications from data. In: International Conference on Quantitative Evaluation of Systems. pp. 323–338. Springer (2018)
22. Nicolau, M.: Understanding grammatical evolution: initialisation. Genetic Programming and Evolvable Machines **18**(4), 467–507 (2017)
23. Pappa, G.L., Freitas, A.A.: Evolving rule induction algorithms with multi-objective grammar-based genetic programming. Knowledge and information systems **19**(3), 283–309 (2009)
24. Pigozzi, F., Medvet, E., Nenzi, L.: Mining Road Traffic Rules with Signal Temporal Logic and Grammar-Based Genetic Programming. Applied Sciences **11**(22), 10573 (2021)
25. Squillero, G., Tonda, A.: Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization. Information Sciences **329**, 782–799 (2016)
26. Umer, M.A., Mathur, A., Junejo, K.N., Adepu, S.: Generating Invariants using Design and Data-centric Approaches for Distributed Attack Detection. International Journal of Critical Infrastructure Protection p. 100341 (2020)
27. Virgolin, M.: Genetic programming is naturally suited to evolve bagging ensembles. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 830–839 (2021)
28. Whigham, P.A., et al.: Grammatically-based genetic programming. In: Proceedings of the workshop on genetic programming: from theory to real-world applications. vol. 16, pp. 33–41. Citeseer (1995)