

Learning Probabilistic Automata from Single Continuous-Valued System Logs

Simon Dierl¹, Nastaran Kianersi², Falk Howar^{1,3}, and Sean Kauffman²

¹ TU Dortmund University, Dortmund, Germany
{simon.dierl, falk.howar}@tu-dortmund.de

² Queen’s University, Kingston, Ontario, Canada
{sean.k, nastaran.kianersi}@queensu.ca

³ Fraunhofer ISST, Dortmund, Germany

Abstract. We introduce a method for the automatic learning of probabilistic automata from continuous-valued logs of long-running cyber-physical systems. Our method supports learning from sparse logs where no ground truth exists and succeeds without the need for manual hyperparameter tuning. We propose improvements to the Timed IOalergia algorithm and integrate it with a learning pipeline to tune discretization hyperparameters through non-convex optimization of a novel metaheuristic we call revision score. We demonstrate the effectiveness of our method through experiments on real-world and simulated datasets.

Keywords: Discretization · Event Traces · Markov Chains · Passive Learning

1 Introduction

Automata learning is a useful tool to bridge the gap between real-world systems and formal analysis methods. By creating a representative model of a system, analyses that cannot be directly performed on the real system can instead be run on a learned automaton that models the system’s behavior. Automata-like structures offer a good compromise between representing stateful systems’ behavior and the possibility of algorithmic analysis.

Learning methods for automata can be divided into *active* and *passive* techniques. The former build a model by repeatedly querying the system under test, while the latter work from past observations only. Active algorithms usually yield more accurate models but are not suitable for learning black-box systems that can – or should – not be manipulated from the outside. For example, a control system for the signal lights of a rail network has observable outputs, but one cannot query its logic by forcing it into different scenarios without either endangering the safety of the network or taking it out of service for some time. In such scenarios, only passive algorithms can be used.

Passive automata learning algorithms usually build upon the pattern introduced in Biermann and Feldman’s pioneering work [4]: first, all observations are combined into a prefix tree, then, tree states are repeatedly merged according

to a similarity criterion. However, existing algorithms in this paradigm usually have three requirements on the observation data: a) they must be discretized, b) observations from multiple runs must be provided, and c) both positive and negative examples, i.e., traces that can and cannot be produced by the system, respectively, are needed.

We were presented with a case that met none of these constraints when an industrial partner approached us to automatically infer the behavior of a control system for a domestic heat pump used for heating and cooling a home. This was motivated by the need to predict the system’s behavior to schedule, e.g., defrost cycles. They could provide us with a historical time series of sensor values representing external information such as the temperature of intake water and the outdoor temperature. However, they could not supply a working model or a trace of the controller logic’s discrete state transitions. The heat pump was installed in a home and restarting the heat pump to observe initial conditions would have disrupted the occupants. As a result, they could only supply one long time series. For the same reason, they could only provide positive examples.

Automata learning algorithms operate on discrete, symbolic sequences rather than raw continuous signals. To bridge this gap, we identify points where the system’s behaviour changes significantly, called *events*. Each event is labeled as a system output, and the interval between two consecutive events represents a single state. This segmentation converts an unstructured time series into a symbolic trace that can be fed into an automata learning algorithm.

We present a novel learning approach that has none of the constraints mentioned earlier, operating on a single, unstructured observation trace with continuous-valued data. Our approach is based on a recurrent process: We first discretize the unstructured data, labeling the detected discrete events as system outputs. We then remove irrelevant events and derive a probabilistic automaton model. Since each of these steps is governed by hyperparameters, the process is initialized from a random configuration. We then evaluate the model’s performance using a quality metric, which measures how well the automaton conforms to the observed trace. This metric is maximized using a nested non-convex optimization algorithm. Until convergence or for a maximum number of iterations (whichever comes first), we loop back to the discretization step, adjusting the hyperparameters accordingly.

Figure 1 shows the major elements of the learning pipeline we propose. In the figure, system logs in the form of a multivariate time series are the only inputs to the process. In the case of the heat pump data from our partner, these logs represent sensor readings like “flow temperature” as well as system state like “defrost” that is set to 1 when the pump is defrosting and zero otherwise. The discretization part of the learning transforms these logs into event traces of system outputs that we use for probabilistic automata learning. The learned automata are then evaluated using a novel metric called *revision score*. While the revision score is below a chosen threshold, we repeat the learning starting from the discretization step but choosing new hyperparameters, using Particle Swarm Optimization (PSO) to optimize for the revision score.

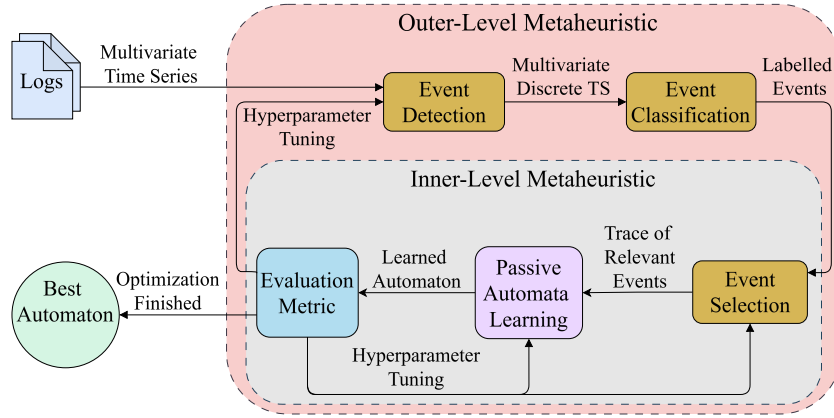


Fig. 1. Structure of the optimization pipeline.

Our contributions are several-fold. First, we introduce a learning pipeline that operates on an unlabeled, multivariate time series without manual hyperparameter tuning. The components of the pipeline are largely existing algorithms, but the novelty is in how they are combined to form a method for generalized automata learning. Our second contribution is the automata learning algorithm. While the Timed IOALERGIA algorithm [19] can infer automata from discretized data, it requires a large number of traces that have a substantial amount of overlap to yield good results. We introduce several configurable relaxations to the algorithm’s state merge heuristics that enable it to operate on the sparse input traces typical for our use case. Finally, our third contribution is the *revision score* metaheuristic for evaluating our learned automata. Evaluation without a ground truth automaton is extremely tricky, and we make the case here that our metric does a good job of capturing the correctness of the learned model.

We evaluate our approach on two datasets: the heat pump logs from our industrial partner and logs from a simulated buck-boost converter, a well-known benchmark from prior work. Our evaluation shows that our method is able to effectively learn probabilistic models from long duration traces of continuous behaviors without manual intervention or ground truths.

Related Work. Several approaches for discretizing multivariate time series have been proposed in the literature. For example, [21] introduces an event classification approach for power grid control centers, assuming that the number of event types is known in advance. Similarly, [28] proposes a wavelet-based method for event detection and classification under the assumption of a fixed number of event types.

For a broad overview on automata learning techniques, we refer to the surveys by de la Higuera [12], Howar and Steffen [14], and Wang et al. [30]. An overview of Markov chain comparison techniques using ground truths is given by García et al. [11].

For the requirements listed above, methods exist that do not demand *one* of the constraints. E.g., POSEHAD [26] can learn models from continuous data, Dierl et al. [9] demonstrate an approach to single-trace mining based on predefined segmentation rules, MDL-based learning [1] does not require negative-labeled data, and Markov chain learners such as ALERGIA [5] do not require a shared initial state.

Structure. The remainder of this paper is organized as follows. After preliminaries in Section 2, we begin by describing the discretization process in Section 3. We then explain the automaton inference process in Section 4. Section 5 describes the revision score metaheuristic that we use for evaluating automata and Section 6 explains the hyperparameter optimization we use. Section 7 describes our evaluation and Section 8 concludes the paper.

2 Preliminaries

We denote the set of all natural numbers including zero $\mathbb{N}$ and the set of all real numbers $\mathbb{R}$. We assume a set of timestamps $\mathbb{T} \subseteq \mathbb{R}$ with a zero value $\emptyset$. We write $\langle x_1 x_2 \dots x_n \rangle$ to denote a sequence of n values where x_i denotes the i -th element of the sequence.

A multivariate time series with d dimensions is a finite sequence of vectors of the form $\mathbf{x} = \langle x_1 x_2 \dots x_n \rangle$ where $x_i = \langle v_1^i v_2^i v_3^i \dots v_d^i \rangle$ and each element $v_j^i \in \mathbb{R}$. Each dimension j of $\mathbf{x}$, written $\mathbf{x}^j$, can be viewed as a univariate time series and a multivariate time series consists of d univariate time series of length n .

To apply automata learning methods to multivariate time series, we first need to discretize them, i.e., convert from real-valued data to *outputs* (sometimes called actions, or letters) from a finite set. Given a set of *outputs* Ω , a *timed event* is a pair (ω, t) consisting of a *discrete output* $\omega \in \Omega$, and a *time* $t \in \mathbb{T}$. A sequence of timed events such that all times are strictly ascending is an *(event) trace*. We refer to the set of all traces on the finite set of outputs Ω as $\mathcal{T}^\Omega$.

Our target model is based on the *deterministic labeled continuous-time Markov chains* proposed as target models for Timed IOALERGIA [19]. Since our model is based on passive learning from observations, it additionally records empirical observations and derived transition probabilities and exit times from these observations. We exploit this in our evaluation metric (see Section 5).

Definition 1 (Continuous-Time Markov Chain). *Given an output alphabet Ω , a Continuous-Time Markov chain (CTMC) is a tuple $(Q, q_0, \lambda, \delta, f, T)$, comprising*

- a set of states Q ,
- an initial state $q_0 \in Q$,
- an output function $\lambda : Q \rightarrow \Omega$,
- a transition function $\delta : Q \rightarrow 2^Q$,
- a transition frequency function $f : Q \times Q \rightarrow \mathbb{N}$, and

- an exit times function $T : Q \rightarrow \mathbb{T}^*$,

such that the following properties hold:

- for every $\omega \in \Omega$, $\delta(q) \cap \{q' \in Q \mid \lambda(q') = \omega\}$ contains one or zero elements (i.e., δ is deterministic w.r.t. the next output),
- $f(q, q') = 0 \iff q' \notin \delta(q)$ (i.e., missing transitions have frequency zero), and
- $\sum_{q' \in Q} f(q, q') = |T(q)|$ (i.e., the number of observed exit times matches observed transitions).

We additionally define

- the deterministic transition function $\delta' : Q \times \Omega \rightarrow Q \cup \{\perp\}$ by

$$\delta'(q, o) \mapsto \begin{cases} q'' & \text{if } \delta(q) \cap \{q' \in Q \mid \lambda(q') = o\} = \{q''\} \\ \perp & \text{otherwise,} \end{cases}$$

- the global frequency function $F : Q \rightarrow \mathbb{N}$ by

$$F(q) \mapsto \sum_{q' \in Q} f(q, q'),$$

- the transition probability function $p : Q \times Q \rightarrow \mathbb{R} \cap [0, 1]$ by

$$p(q, q') \mapsto \frac{f(q, q')}{F(q)}, \text{ and}$$

- the average exit time function $t : Q \rightarrow \mathbb{T}$ by

$$t(q) \mapsto \begin{cases} \text{average}(T(q)) & \text{if } T(q) \neq \emptyset \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Transition semantics for traces are difficult to define, since the timing behavior is continuous. The probability of observing any particular trace is equal to zero; only probability density functions for traces can be defined. We will revisit the issue in [Section 5](#). We can, however, define expected hitting times to generate answers to our motivating question:

Definition 2 (Expected Hitting Time). *Given a CTMC $(Q, q_0, \lambda, \delta, f, T)$ and a set of target states $\hat{Q} \subseteq Q$, the expected hitting time $h^{\hat{Q}} : Q \rightarrow \mathbb{T} \cup \{\infty\}$ is defined by the solution to the following system of equations:*

$$h^{\hat{Q}}(q) = \begin{cases} 0 & \text{if } q \in \hat{Q} \\ \sum_{q' \in Q} p(q, q') \cdot (t(q) + h^{\hat{Q}}(q')) & \text{otherwise.} \end{cases}$$

For each state q that has a path to a target state, $h^{\hat{Q}}(q) \neq \infty$.

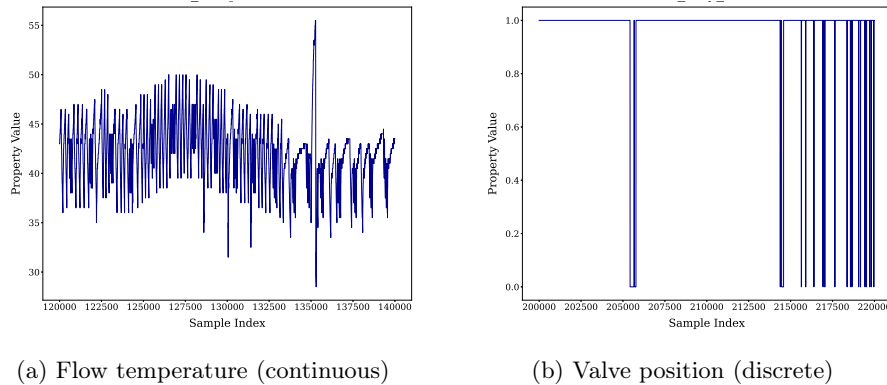


Fig. 2. Examples of system variable types from the heat-pump dataset.

3 Discretizing Continuous Data

Sequences generated by systems of our interest are usually records of certain quantities or parameters during their execution. For instance, the output sequence of a cyber-physical system consists of sensor values sampling multiple physical quantities and logs of memory values indicating system status. Although sampling applies a form of discretization, quantities can be either discrete or continuous in nature. [Figure 2](#) shows examples of discrete and continuous properties, which we refer to as *system variables* or simply *variables* throughout this paper, as observed in our heat pump evaluation dataset. The flow temperature sensor captures continuous data, whereas the position of the two-way valve has only two possible states and is therefore considered a discrete system variable.

The original output sequences of such systems typically contain redundant data due to their high sampling frequency, meaning that some recorded values may not be useful as events for our CTMC learning algorithm. Therefore, we need to identify and keep only the data points that signify considerable changes in the measured quantities. Identifying events in discrete-valued system variables is straightforward; we simply keep the data points where a change occurs. In this section, we propose a method for discretizing continuous-valued data and obtaining an event trace from raw sampled data. The discretization process involves three major steps: a) identifying events, b) categorizing those events into event types, and c) selecting a set of events relevant to our learning problem.

3.1 Event Detection

The system under learning generates a multivariate time series where each dimension represents some continuous- or discrete-valued variable. We separately process each univariate time series to extract an event trace for the corresponding variable. For a univariate time series (i.e., a single-dimensional projection of the input) $\mathbf{x}^j$ containing discrete values such as the *valve position* example, this

translation is straightforward. We simply assign an output $\omega \in \Omega$ to each discrete value in $\mathbf{x}^j$ and add an event with that value at the first timestamp at which it is seen in $\mathbf{x}^j$. For example, if $m : \mathbb{R} \rightarrow \Omega$ is a function that maps time series values to Ω , given a univariate time series $\mathbf{x}^j$ sampled at 10 Hz, the first elements resulting from conversion to an event series would be $\langle (m(x_1^j), 0)(m(x_{i''}^j), (i'' - 1)/10), \dots \rangle$ where $x_{i'}^j = x_1^j \forall 1 < i' < i''$ and $x_{i''}^j \neq x_1^j$. For variables with sampled continuous values, however, we must reduce the number of distinct outputs to only those that represent a more drastic state change. To this end, we employ Change Point Detection (CPD).

Change Point Detection Method. Change point detection aims to discover points in time where a significant shift occurs in the behavior or structure of the observed data. A comprehensive review of a wide range of CPD techniques is proposed in [2], and a review of offline CPD methods is presented in [27]. The choice of a suitable CPD method depends on the application requirements and constraints [27]. Since our objective is to learn a model from historical system logs, we adopt an offline CPD approach, meaning all data must be available in advance. To improve computational efficiency while achieving close-to-optimal solutions, we incorporate batching, resulting in a semi-online method. Moreover, to minimize the need for prior knowledge about the data, the chosen method should be non-parametric; that is, it should not rely on assumptions about the underlying distribution from which the data is drawn. Given these requirements, we employ the Pruned Exact Linear Time (PELT) algorithm [16], a dynamic programming technique that efficiently finds the optimal segmentation of a signal when the number of change points is unknown.

To measure similarity within segments, PELT uses a cost function that specifies the type of change to be detected in the signal. Different cost functions allow the detection of changes in statistics such as the mean, variance, median, or the overall distribution [6, 27]. In our implementation of the CPD method, we support several commonly used cost functions to remain general and minimize prior assumptions about the data, enabling the modeling of a wide range of systems. The PELT parameters related to event detection labeled ED are listed in Table 1 along with their corresponding domains.

3.2 Event Classification

Although detected change points correspond to events, CPD methods do not reveal whether these events are connected or belong to the same category. For example, if 100 change points are identified within a system sequence, it is unlikely (depending on the context and the length of the sequence) that there are 100 distinct events. Rather, the data likely contains recurring patterns, or some events may represent repeated instances of the same event type. Thus, we must classify events into distinct types based on a defined similarity measure. Each distinct type of event is then labeled with the same output; the output alphabet is the set of equivalence classes induced by the measure.

To classify events in this work, we use the k -Means clustering algorithm [18] to group the events where the similarity measure is the Euclidean distance. This algorithm is employed due to simplicity and reasonable performance, but can be replaced with other unsupervised learning methods of choice. For each system variable, we apply the clustering algorithm over a range of values for the number of clusters k and choose the optimum k based on the highest silhouette score [25] achieved. The obtained k then indicates the number of event types for that system variable, and the event trace can be generated by clustering the change points using the obtained clustering model.

3.3 Event Selection

After discretization and event classification, we have a single trace of discrete, timed events for each system variable (i.e., dimension in the multivariate series). Ideally, every such trace could be split into multiple non-overlapping, adjacent parts of the trace which we will refer to as *sub-traces*. Each resulting sub-trace should start at a suspected reset point, i.e., begin in the automaton’s initial state. We approximate this by splitting the trace based on timing by, e.g., splitting at midnight, and designate each sub-trace’s first event as a *key point*.

To use the variable-specific traces in learning, however, we need to merge them and filter out irrelevant or redundant events such as system-specific diagnostic information or readings from multiple similar sensors, respectively. Filtering is necessary since retaining all events may obscure cause-effect relationships in the data. For example, if event **b** immediately follows event **a** with a delay of one second, a learning algorithm can easily discover their relationship. If, however, they are separated by an arbitrary number of keep-alive messages occurring in between, this task is far more challenging.

We therefore select a subset of dimensions $\hat{D} \subseteq \{1, \dots, d\}$ for which we merge the corresponding discretized traces. Additionally, we designate a *key dimension* $\bar{d} \in \hat{D}$ which we use as the canonical source of key points. We begin by combining all traces created from dimensions in $\hat{D}$ and retain the chronological order of events, resulting in a single, large event trace. Next, we divide the trace at the key points identified for $\bar{d}$ into sub-traces and discard all events without a preceding key point. E.g., if k key points were identified in the univariate series $\mathbf{x}^{\bar{d}}$, we obtain k event traces in $\mathcal{T}^\Omega$. An overview of the parameters is shown in Table 1 marked *ES*.

After this step, we have condensed the input data into a set of discrete event traces. Each of these traces contains the discretized events from *all* selected variables. The traces themselves are sub-traces of the entire input, split using the key event.

4 Automaton Inference

Having transformed our unstructured input into a set of traces, we can now use these to derive an automaton modeling the system under learning’s discrete behavior. Note that the system’s full model therefore requires both the discretization

Input: merge target $q_r \in Q$ and merge source $q_b \in Q$
Input: similarity bounds $\varepsilon_t \in [0, 1]$ and $\varepsilon_p \in [0, 1]$
Input: similarity decays $\zeta_t \in [0, 1]$ and $\zeta_p \in [0, 1]$
Input: remaining test depth $k \in \mathbb{N} \cup \{\infty\}$
Output: if q_b should be merged into q_r

```

1 Function CANMERGE( $q_r, q_b, \varepsilon_t, \varepsilon_p, \zeta_t, \zeta_p, k$ ) is
2   if  $\lambda(q_r) \neq \lambda(q_b)$  then
3     return false                                /* outputs incompatible */
4   if  $k > 0$  then
5     if  $FScore(t(q_r), |T(q_r)|, t(q_b), |T(q_b)|) < \varepsilon_t$  then
6       return false                                /* timings dissimilar */
7     foreach  $\omega \in \Omega$  do
8       if  $HOEFFDINGScore(f(q_r, \omega), F(q_r), f(q_b, \omega), F(q_b)) < 2\varepsilon_p$  then
9         return false  /* transition probabilities dissimilar */
10    foreach  $\omega \in \Omega$  with  $q'_r = \delta(q, \omega) \neq \perp$  and  $q'_b = \delta(q, \omega) \neq \perp$  do
11      if CANMERGE( $q'_r, q'_b, \zeta_p \cdot \varepsilon_p, \zeta_t \cdot \varepsilon_t, \max(k-1, 0)$ ) is false then
12        return false                                /* recursive test failed */
13  return true                                /* states are sufficiently similar */

```

Alg. 1. Relaxed merge test function for timed IOALERGIA.

rule and the automaton model and the automaton is not capable of generating multivariate time series, but only event traces. We build our learning algorithm on the established Timed IOALERGIA passive learning algorithm [19]. This algorithm infers CTMCs from positive examples using the approach pioneered by k -Tails [4]: first, all examples are aggregated into a prefix tree acceptor (i.e., each output corresponds to a state), then, sufficiently similar states are iteratively merged. The algorithm uses the Blue Fringe method [17] to select merge candidates.

Sufficient similarity of states is established based on a recursive, fourfold test with two parameters, ε_t and $\varepsilon_p \in [0, 1]$, the similarity thresholds for exit times and transition probabilities, respectively. A state can be merged into a second state if a) their outputs are identical, b) the F-test [7] score for the observed exit times is below ε_t , c) for each successor output, the Hoeffding test [13] score for the observed transition probabilities is below $2\varepsilon_t$ ⁴, and d) for any input that has a matching successor in both states, the similarity test also holds for the successors. The subsequent merge operation will recursively merge the successors. Both tests take into account not only the observed averages, but also the number of observations, with more observations requiring “more similar” averages for the tests to succeed. The required threshold defines the tests’ strictness, with higher values only accepting “more similar” observations. For $\varepsilon = 1$, we define the checks to always fail; for $\varepsilon = 0$, we define automatic success.

⁴ The test is customarily defined with thresholds in $[0, 2]$. We halve the value for simplicity.

However, our intended use case violates two assumptions informing IOALERGIA’s design. First, the algorithm is intended to operate on *independent and identically distributed* inputs. We cannot guarantee this property for our split input traces. First, incorrect splits may introduce dependencies between the traces and second, some behavioral changes (e.g., a summer vacation by the users) may not be reflected in the trace variables, resulting in mismatched distributions. We attempt to mitigate this by varying IOALERGIA’s parameters to find a configuration yielding satisfactory results, but cannot guarantee that the algorithm’s performance does not degrade and must accept “best effort” results.

To ensure stochastic soundness, the observations used for IOALERGIA’s stochastic tests are not subject to the merge procedure in the algorithm definition by Mao et al. [19]; instead, the original information in the prefix tree acceptor is used. As a result, when a state is merged, its observations are effectively discarded for the purpose of the merge eligibility check. On sparse prefix trees, like those generated in our use case, the tests therefore become irrelevant, since they require a minimum number of observations to falsify state equivalences. Here, we are forced to violate IOALERGIA’s second assumption. As a possible remedy, Mao et al. suggest that the tests can be performed on merged observations as well. This, too, breaks assumptions necessary for their proofs of IOALERGIA’s convergence behavior. We therefore define a *classic* mode that retains the stochastically sound behavior and a *merged* mode that uses merged observations in its tests.

In our case, an additional challenge is imposed by the length of the output traces. A merge test may well fail due to similarity checks thousands of states deep. To address this issue, we build upon the *relaxed* IOALERGIA variant proposed by Demessance et al. [8], which only performs the F- and Hoeffding tests on the initial states of a merge. We extend this idea by introducing two extensions. First, we define a k -bounded mode in which we stop performing the stochastic tests in recursion depth $k \in \mathbb{N} \cup \{\infty\}$. Second, we allow for a more gradual relaxation of tests in deep merges by requiring weaker evidence at higher recursion depth. Mathematically, we model this using exponential decay by introducing decay parameters ζ_t and $\zeta_p \in [0, 1]$. In each recursion step, ε_t (ε_p) is multiplied by ζ_t (ζ_p). The original algorithm then corresponds to $k = \infty$ and $\zeta_t = \zeta_p = 1$, while the relaxed version has $k = 1$.

An overview of our extended merge test function is shown in [Alg. 1](#). [Lines 2 and 3](#) define the output compatibility check, [line 4](#) disables statistical tests above the k bound, [lines 5 and 6](#) perform the F-test, [lines 7, 8, and 9](#) run the Hoeffding test on every output, and [lines 10, 11, and 12](#) recurse into successors present on both states, reducing the k bound and weakening the test thresholds according to the decay parameters. Only if all tests succeed, we accept the merge in [line 13](#).

A summary of all parameters is shown in [Table 1](#), marked *AL*. Since all of these parameters must be finely tuned to the input at hand, no general defaults exist. This is exacerbated by the fact that suitable values likely also depend on the output of the previous steps and, therefore, the change point detection and event selection parameters. It is therefore desirable to optimize these values at runtime using the metaheuristic approach described in [Section 6](#). Optimizing

Input: CTMC $A = (Q, q_0, \lambda, \delta, f, T)$
Input: start state q^0
Input: event traces $s_0, \dots, s_i \in \mathcal{T}^\Omega$
Output: similarity measure $\in [0, 1]$

```

1 Function  $REVISIONSCORE(A, s_0, \dots, s_i)$  is
2    $\hat{f}(q, q') \leftarrow 0$  with global frequencies  $\hat{F}$            /* observed frequencies */
3    $\hat{T}(q) \leftarrow \emptyset$  with averages  $\hat{t}$                  /* observed timings */
4   foreach  $s \in s_0, \dots, s_i$  do
5      $q \leftarrow q^0$ 
6     foreach  $\langle \omega, t \rangle \in s$  do
7       add  $t$  to  $\hat{T}(q)$ 
8        $q' \leftarrow \delta'(q, \omega)$ 
9       if  $q' = \perp$  then
10        return 0                                           /* series mismatch */
11      increment  $\hat{f}(q, q')$ 
12     $q \leftarrow q'$ 
13   $s_t \leftarrow \text{average}_{q \in Q} \text{FScore}(t(q), |T(q)|, \hat{t}(q), |\hat{T}(q)|)$ 
14   $s_f \leftarrow \text{average}_{q \in Q, \omega \in \Omega} \text{HoeffdingScore}(f(q, \omega), F(q), \hat{f}(q, \omega), \hat{F}(q))$ 
15  return  $\text{average}(s_t, s_f)$ 

```

Alg. 2. Revision score computation function.

only the parameters of the discretization step nonetheless seems to overcome the lack of automata-learning tuning. We suspect this is due to the interaction between them: it is possible to choose discretization parameters that work for the chosen automata learning parameters.

5 Automaton Validation

To evaluate the quality of a learned CTMC against a validation set of event traces, we require an evaluation metric. Most comparison measures for CTMCs described in the literature, however, are designed to analyze automata evolution scenarios and thus require “ground truth” automata for both sides of the comparison. These are not applicable in our scenario. Other typical approaches for *discrete*-timed Markov chains measure the likelihood of generating the validation trace, with a high value indicating a good learning result. However, this approach has two shortcomings.

First, in a continuous-timed chain, the likelihood of any given trace is zero, since exit times are defined over a continuous domain. This can, in theory, be addressed by substituting Perkins’ measure [23] that approximates how “late” an observed timing is with

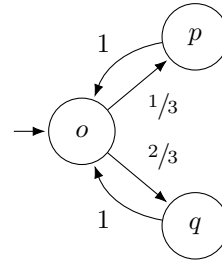


Fig. 3. Simple probabilistic automaton.

respect to the average. Second, the likelihood of traces may not be a suitable measure for learning performance at all, even in discrete settings. Consider, e.g., the automaton sketched in Fig. 3 and the trace (o, p, o, q, o, q) . Although this trace *exactly* reproduces the probabilities in the automaton, it will only have probability $4/27$. The trace (o, q, o, q, o, q) , which should cast serious doubt on the correctness of the probabilities, however, has probability $8/27$.

We address this issue by introducing the novel *revision score* metric. In essence, we “replay” the validation traces in the learned automaton and collect the observed timed and transition frequencies. This enables us to apply the Hoeffding and F-tests also used in the timed IOALERGIA algorithm to compare the observations backing the learned automaton with those gathered from validation traces. Since these tests yield quantifiable measures of similarity, we can average them to obtain a single numeric result between zero and one. In the above example, assuming two observations on the q - and one on the p -transitions, the first trace would yield a score of 1, while the second would yield ≈ 0.46 .

Replaying a series can either be done from the initial location or by trying each state in the automaton as a starting state and using the best-scoring one, similar to Viterbi’s algorithm [29]. Since the automaton is deterministic w.r.t. outputs, each starting state yields one or zero possible sequences of states and transitions. If no sequence exists, we assign a final score of zero. To handle more than one validation series, we can either average the per-trace scores or replay every trace⁵ and compute a single score for all collected observations.

The algorithm is shown in Alg. 2. Lines 5, 8, and 12 defines the matching of a trace to a sequence of states. Line 10 handles mismatches, while lines 7 and 11 aggregate observations. The score averaging is done in lines 13, 14, and 15. The double averaging is necessary since the number of Hoeffding tests exceeds that of F-tests. For a per-trace computation, this function would be called for each trace (with averaged results). Here, we can also try every starting state and select the best resulting score. For a global analysis, the function is called only once, using all traces as inputs.

6 Hyperparameter Optimization

When prior knowledge about the input execution traces is available, hyperparameter values can be manually selected to yield an accurate automaton. Lacking such knowledge (e.g., in our motivating example), we have to select hyperparameters without guidance, significantly degrading the accuracy of the learned automaton. To remedy this, we employ a stochastic optimization algorithm to automatically search for the optimal hyperparameter values that maximize the revision score of the resulting automaton.

The CPD method used in the discretization step requires three parameters to be specified: the minimum distance between change points, the choice of cost function for the PELT algorithm, and a penalty term for fine-tuning PELT. The

⁵ In this case, we only consider starting in the initial state to avoid combinatory explosion.

Table 1. Parameters for all steps in the learning pipeline.

Param.	Dom.	Description
ED	δ	$\mathbb{N}$ minimum distance between change points
ED	β	$\mathbb{R}^+$ penalty value in PELT algorithm
ED	$\mathcal{C}$	$\{l1, l2, n, c, r\}$ cost function in PELT algorithm (<i>l1</i> , <i>l2</i> , <i>normal</i> , <i>cosine</i> , or <i>rbf</i>)
ES	$\hat{D}$	$2^{\{1, \dots, d\}}$ dimensions to retain
ES	$\bar{d}$	$\hat{D}$ key dimension for changepoints
AL	ε_t	$[0, 1]$ similarity threshold for exit times
AL	ε_p	$[0, 1]$ similarity threshold for transition probabilities
AL	$\mathfrak{M}$	$\{c, m\}$ observation data for the tests (<i>classic</i> or <i>merged</i>)
AL	ζ_t	$[0, 1]$ similarity threshold decay for exit times
AL	ζ_p	$[0, 1]$ similarity threshold decay for transition probabilities
AL	k	$\mathbb{N} \cup \{\infty\}$ stochastic test depth bound

ED: Event Detection ES: Event Selection AL: Automata Learning

cost function is selected from a predefined set of options, while the minimum segment length and the penalty term correspond to integer-valued and real-valued parameters, respectively. In addition, the event selection step requires determining the subset of variables used for merging. In the absence of prior knowledge of the data, it is not possible to identify the relevant system variables in advance. Finally, the learning algorithm requires several configuration parameters, as described in Section 4. A complete list of all hyperparameters and their corresponding domains is provided in Table 1. All of these parameters are jointly optimized during the optimization process.

We employ Particle Swarm Optimization (PSO) [15] to search for the global optimum of these hyperparameters within the proposed pipeline. The high-level structure of this pipeline is shown in Fig. 1, which also illustrates how the metaheuristic is applied. PSO maintains a swarm of particles, where each particle represents a candidate solution in the search space. Particles move through this space according to two update equations called flight equations [15] that adjust their positions and velocities based on their personal best solution and the swarm’s global best. We selected this metaheuristic because it is well suited to non-convex optimization problems and is relatively simple to implement. The PSO algorithm is well established in the literature [8, 31], and a detailed theoretical discussion is beyond the scope of this paper. We therefore focus on the practical considerations most relevant to achieving strong performance in our setting.

The optimization problem involves heterogeneous hyperparameters, including continuous, discrete, set-based, and binary parameters (e.g., ε_t , δ , $\bar{d}$, and $\hat{D}$, respectively, in Table 1). To handle this heterogeneity, we employ different PSO variants tailored to each parameter type [24, 33]. Since the hyperparameters naturally separate according to pipeline stages, we adopt a nested PSO structure in which a level-1 swarm optimizes event detection parameters, and a level-2 swarm optimizes event selection and learning parameters during the evaluation of

Input: Multivariate Time Series μ
Input: Swarm sizes N_1 and N_2 , and number of iterations $i_1^{\max}$ and $i_2^{\max}$
Input: Event detection hyperparameters X_1
Input: Event selection and learner hyperparameters X_2
Input: Objective function: combination of discretization, learning, and revision score calculation
Output: Best parameter vectors X_1^* and X_2^* that maximize the revision score
Output: Optimal revision score Y^*

- 1 Initialize swarm $S_1 = \{X_1^{(k)}\}_{k=1}^{N_1}$ and level-1 velocities
- 2 Initialize personal best positions $px_1^{(k)} \leftarrow X_1^{(k)}$ and global best position $gx_1 \leftarrow \text{mean}(S_1)$
- 3 Initialize personal best fitness $py_1^{(k)} \leftarrow \infty$ and global best fitness $gy_1 \leftarrow \infty$
- 4 **for** $i \leftarrow 1$ to $i_1^{\max}$ **do**
- 5 Update flight equations for S_1
- 6 **foreach** level-1 particle $X_1^{(k)} \in S_1$ **do**
- 7 Perform event detection and classification on μ with $X_1^{(k)}$
- 8 Initialize swarm $S_2 = \{X_2^{(k)}\}_{k=1}^{N_2}$ and level-2 velocities
- 9 Initialize $px_2^{(k)} \leftarrow X_2^{(k)}$ and $gx_2 \leftarrow \text{mean}(S_2)$
- 10 Initialize $py_2^{(k)} \leftarrow \infty$ and $gy_2 \leftarrow \infty$
- 11 **for** $j \leftarrow 1$ to $i_2^{\max}$ **do**
- 12 Update flight equations for S_2
- 13 **foreach** level-2 particle $X_2^{(k)} \in S_2$ **do**
- 14 Perform event selection and learning with $X_2^{(k)}$
- 15 Evaluate revision score induced by $(X_1^{(k)}, X_2^{(k)})$
- 16 Update level-2 personal and global bests
- 17 (Optional) Check for early convergence based on gy_2
- 18 $py_1^{(k)} \leftarrow gy_2^k$
- 19 Update gx_1 and gy_1
- 20 (Optional) Check for early convergence based on gy_1
- 21 **return** (X_1^*, X_2^*, Y^*) , $X_1^* \leftarrow gx_1$, $X_2^* \leftarrow gx_2$

Alg. 3. Nested Particle Swarm Optimization. All parameters with subscripts 1 and 2 refer to the level-1 (outer search) and level-2 (inner search) parameters, respectively.

each level-1 particle. This structure reflects the hierarchical dependency between pipeline stages, as event detection decisions constrain subsequent event selection and learning configurations. An overview of the nested optimization approach is provided in [Alg. 3](#); details of the underlying PSO algorithm are omitted for brevity.

7 Evaluation

In this section, we will first establish the correlation between our revision score and established similarity metrics for Markov chains and traces. Next, we will

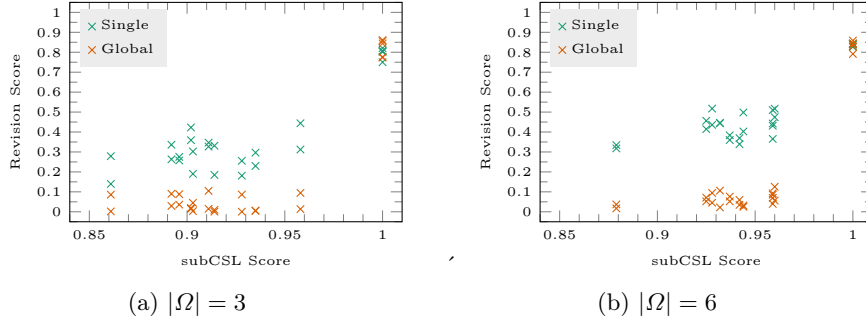


Fig. 4. subCSL vs. revision scores on pairs of randomly-generated automata.

evaluate our proposed learning pipeline using revision scores and show that it can successfully learn highly-scoring CTMCs from limited data.

Event detection and classification were implemented in [Python](#) using the [scikit-learn](#) [22] and [ruptures](#) [27] libraries. The trace merger, learner, and the revision score computation are implemented in [Kotlin](#) based on the [LearnLib framework](#) [10]. The complete pipeline, including all components and the optimization process, was implemented in Python and executed in a distributed manner on an HPC cluster to exploit the parallel nature of the PSO algorithm.

7.1 Revision Score

First, we need to establish that the revision score is a suitable metric to measure the correspondence between observed traces and CTMCs. To this end, we compare revision scores to the metric used by Mao et al. to measure the convergence of Timed IOALERGIA. They measure the satisfaction of subCSL formulas (a linear fragment of continuous stochastic logic [3]) by generating traces from a given automaton and computing the share of traces on which the subCSL property holds. We therefore assume that two automata are similar if, for a given subCSL formula, the share of traces satisfying the formula is similar for both automata.

Working from this assumption, we define the subCSL score of two sets of traces S and S' and a set of subCSL formulas Φ as

$$s_s = \text{average}_{\varphi \in \Phi} \left\| \frac{|s \in S \mid s \models \varphi|}{|S|}, \frac{|s' \in S' \mid s' \models \varphi|}{|S'|} \right\|.$$

We will show that the revision score of a set of traces S and a continuous-timed Markov chain A is proportional to the subCSL score of S and a second set generated from A .

We conducted two evaluation campaigns. In each, we randomly generated 5 CTMCs with output alphabets of size 3 and 6, respectively. For the smaller alphabets, we generated between 4 and 8 states per automaton, for the larger, 8 to 16. For each automaton, we generated 100 traces of length 500. Next, we randomly created 100 subCSL formulas for the respective alphabet. For each

pair of automata, we can then compare the subCSL score of their traces and the revision score of the first automaton’s trace and the second automaton.

The results for both averaged per-trace scores as well as global scores are shown in Fig. 4. For per-trace scores, no difference between best and initial starting states was observed; the figure shows the results of using the initial state. While the subCSL scores never drop below 0.85 and are even higher for larger alphabets, the revision scores are distributed over a larger range. Note that self-comparisons are included; these have subCSL scores of one. In general, an acceptable correlation between subCSL results and revision scores can be observed. Per-trace scoring seems to be superior to global scoring, which tends to result in relatively low scores for imperfect matches.

7.2 Learning Pipeline

To evaluate the overall method, we conducted several experiments using the implemented pipeline on two datasets. The first dataset consists of experiments on a buck-boost converter, which has been used as a benchmark in the automata learning literature [20, 32]. The second dataset is the domestic heat pump dataset, which was the motivating example for our proposed learning pipeline.

A buck-boost converter is a DC-DC power conversion circuit that produces an output voltage that is either higher or lower than the input voltage. Its behavior is governed by switching between two operating modes, while the output voltage is regulated to track a given reference voltage within a specified tolerance. The buck-boost dataset includes measurements of the inductor current *plant_il* and the capacitor voltage *plant_vc*, which are the continuous system variables, as well as the active operating mode *ctrl_mode*, which represents the discrete part of the system’s behavior. Fig. 5 illustrates the recorded system variables from a buck-boost converter experiment. The evaluation dataset comprises multiple experiments, each stored in a separate file and corresponding to a distinct trace. These experiments are conducted with different, but closely spaced, reference voltages, and the boundaries between files serve as system reset points.

The second evaluation dataset comprises heat pump system logs spanning four months (December 2023–March 2024) with approximately 100 variables. Without prior knowledge of the system’s internal dynamics, variable relationships, or even identification of some variables by name, we performed visual inspection to categorize variables as discrete or continuous (see Fig. 2). For this analysis, we selected 10 continuous variables from the initial set of 100 variables for one heat pump experiment. For the remaining heat pump experiments, we used the same 10 continuous variables together with 8 discrete variables as inputs to the learning pipeline.

Table 2 summarizes the experimental results for different pipeline configurations on the buck-boost and heat pump datasets. The reported number of samples corresponds to the total multivariate time-series data points, split into training and test sets using a 70–30 ratio for the buck-boost dataset and an 80–20 ratio for the heat pump dataset. Although the pipeline is optimized with respect to the test revision score, results for both training and test sets are reported. For

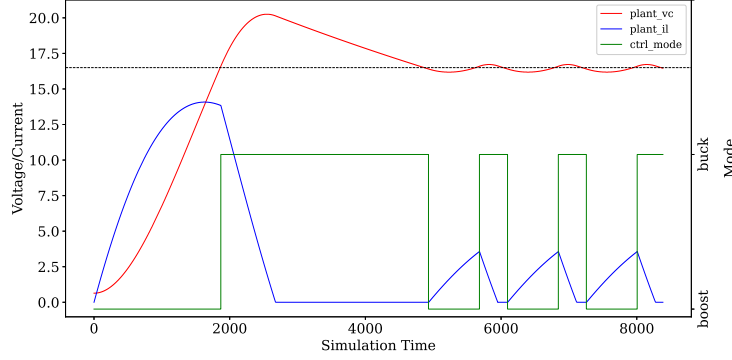


Fig. 5. Recorded system variables for a buck-boost converter experiments.

each dataset, we vary the key dimension $\bar{d}$, scoring type, and the use of variable subset optimization (through the selection of $\hat{D}$). The number of traces after event merging is also reported.

Overall, the results indicate strong performance across most configurations. Tests 1 and 2 yield identical results despite different scoring types, while Test 3 achieves a revision score of 1.0 on both training and test sets for the global scoring type, outperforming earlier tests. In Test 4, variable subset optimization is disabled to include all three variables; this results in a lower test revision score, suggesting that including all variables is not optimal for maximizing the revision score in this dataset.

When using global scoring, we consistently obtain perfect training scores of 1.0. On the buck-boost data, this is replicated in the test scores; on the heat pump, it is not. Since the final scores are obtained using the same scoring method and are therefore not comparable with single-mode scores, we can not state with confidence if this indicates that global scoring is better at guiding the optimization process or not.

Tests 5–8 present results for the heat pump dataset. Test 5 considers only continuous variables, with the key dimension set to *flow_temp*. Tests 6–8 focus on modeling defrost behavior under different scoring types, where single best performs best on the test set, followed by global and then single initial.

Finally, it is important to note that in Tests 1, 2, and 6, the training revision score is slightly lower than the test revision score, indicating mild underfitting. This may be due to the test data capturing only a subset of the system behavior and thus being less representative of the full dynamics present in the training set.

7.3 Threats to Validity

Revision Score. Considering the internal validity of the revision score evaluation, we observed a similar correspondence between subCSL and revision scores to the one described above when using different parameterizations of

Table 2. Experimental results showing revision scores obtained with different configurations of buck-boost and heat pump datasets.

Test Idx. Dataset	Num. of		Num. of Traces		Revision		Scoring Type	Key Dim.
	Variables	Processed Samples	after Merging		Score			
			Train	Test	Train	Test		
1 Buck-boost	3	186,727	7	3	0.5714	0.6667	single best	<i>ctrl_mode</i>
2 Buck-boost	3	186,727	7	3	0.5714	0.6667	single init.	<i>ctrl_mode</i>
3 Buck-boost	3	186,727	7	3	1.0	1.0	global	<i>ctrl_mode</i>
4 Buck-boost	3	186,727	7	3	0.5714	0.4781	single best	<i>ctrl_mode</i>
5 Heat Pump	10	332,412	4	1	0.8501	0.7983	single best	<i>flow_temp</i>
6 Heat Pump	18	332,412	22	11	0.7936	0.8324	single best	<i>defrost</i>
7 Heat Pump	18	332,412	22	11	0.7202	0.5987	single init.	<i>defrost</i>
8 Heat Pump	18	332,412	22	11	1.0	0.7717	global	<i>defrost</i>
Inner Search Configurations: Variable selection is optimized for all tests except for Test 4.								

the random generations. However, our setup never yielded low subCSL scores, showing a possible limitation of the setup. In general, we are confident in the experiment’s internal validity. Externally, however, our experiment focuses on *accidental* similarities of *random* automata (i.e., two synthetic automata happen to behave similarly). However, in the intended use case, we repeatedly compare automata learned from traces similar to those used for validation. This setting is fundamentally different, since we compare non-random automata to traces with non-accidental similarities. Here, the score correlation may be weaker than that in our experimental setup, posing a threat to external validity.

Learning Pipeline. In the second experiment, the results are again consistent for multiple configurations and random seeds. Based on the first evaluation step, we assumed that the revision score is a meaningful measure of automaton quality and can therefore be used as the optimization metric. Consequently, optimizing for the revision score makes the validity of the results contingent on the validity of the first evaluation step. Additionally, the revision score does not fully reflect the automaton’s usability as a model of the heat pump, but it rewards models that are not violated by the test data. This may result in generic models with weak predictions, a phenomenon we also observed in some runs. We will discuss possible remedies as future work (cf. [Section 8](#)). Both phenomena threaten the results’ internal validity. Similar to the revision score experiments, external validity is also limited by our benchmark set. To the best of our knowledge, one promising source of benchmarks for evaluating the full pipeline is the field of process mining, which we plan to explore in future work. While we were able to use realistic models in the evaluation, the use of two models is insufficient to generalize from the results with confidence.

8 Conclusion

In this work, we introduced a novel method to automatically learn probabilistic automata from cyber-physical system logs. Our method applies PSO in a multi-stage process to optimize for revision score, a new metaheuristic for evaluating probabilistic automata without a ground truth. We introduced improvements to the Timed IOALERGIA algorithm that enable learning on sparse traces, and we showed how to obtain events from multivariate time series such as those produced by many real-world systems. Our work is supported by an evaluation over logs from a real domestic heat pump and a simulation of a buck-boost converter.

Future Work. Several aspects of the pipeline can be improved, including the discretization step and the optimization process. One possible extension is to incorporate anomaly detection before discretization, as outliers can negatively affect event classification. Additionally, the clustering algorithm we used for event classification relies solely on change-point magnitudes. While this works for certain continuous dynamics, it is not sufficiently general and fails to capture meaningful events in some time series patterns. Including additional features, such as change direction, rate of change, confidence levels, and pre- and post-change statistics, would make the method applicable to a broader range of system dynamics.

Considering the optimization process, two research directions could lead to further improvements. Although our implementation of PSO includes early convergence checks, coefficient tuning, and a nested structure, it still relies on the standard algorithm. Exploring more advanced PSO variants could further improve robustness and reduce the risk of convergence to local optima. In addition to that, the use of more complex goal functions could be investigated. While our proposed revision score can be used to measure compliance between automata and traces, it tends to reward very generic models. By adding additional optimization criteria to reward complex models (e.g., a high number of included variables or many automaton states), this effect may be counteracted and the optimization would hopefully yield complex, but accurate models with better explanatory power.

Acknowledgments. S. Dierl and F. Howar would like to thank the Federal Government and the Heads of Government of the Länder, as well as the Joint Science Conference (GWK), for their funding and support within the framework of the NFDI4Ing consortium. Funded by the German Research Foundation (DFG) – project numbers [442146713 \(NFDI4Ing\)](#); [495857894 \(STING\)](#). N. Kianersi and S. Kauffman’s work has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Digital Research Alliance of Canada.

References

1. Adriaans, P., Jacobs, C.: Using MDL for grammar induction. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) Grammatical Inference:

- Algorithms and Applications. Lecture Notes in Computer Science, vol. 4201, pp. 293–306. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11872436_24
2. Aminikhanghahi, S., Cook, D.J.: A survey of methods for time series change point detection. *Knowledge and Information Systems* **51**(2), 339–367 (May 2017). <https://doi.org/10.1007/s10115-016-0987-z>
 3. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering* **29**(6), 524–541 (Jun 2003). <https://doi.org/10.1109/TSE.2003.1205180>
 4. Biermann, A.W., Feldman, J.A.: On the synthesis of finite-state machines from samples of their behavior. *IEEE Transactions on Computers* **C-21**(6), 592–597 (1972). <https://doi.org/10.1109/TC.1972.5009015>
 5. Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: Carrasco, R.C., Oncina, J. (eds.) *Grammatical Inference and Applications*. Lecture Notes in Computer Science, vol. 862, pp. 139–152. Springer, Berlin, Heidelberg (1994). https://doi.org/10.1007/3-540-58473-0_144
 6. Celisse, A., Marot, G., Pierre-Jean, M., Rigai, G.: New efficient algorithms for multiple change-point detection with reproducing kernels. *Computational Statistics and Data Analysis* **128**, 200–220 (2018). <https://doi.org/10.1016/j.csda.2018.07.002>
 7. Cox, D.R.: Some applications of exponential ordered scores. *Journal of the Royal Statistical Society: Series B (Methodological)* **26**(1), 103–110 (Jan 1964). <https://doi.org/10.1111/j.2517-6161.1964.tb00544.x>
 8. Demessance, T., Bi, C., Djebali, S., Guérard, G.: Hidden Markov model to predict tourists visited places. In: 2021 22nd IEEE International Conference on Mobile Data Management (MDM). pp. 209–216. IEEE (Jun 2021). <https://doi.org/10.1109/MDM52706.2021.00041>
 9. Dierl, S., Howar, F.M., Kauffman, S., Kristjansen, M., Larsen, K.G., Lorber, F., Mauritz, M.: Learning symbolic timed models from concrete timed data. In: Rozier, K.Y., Chaudhuri, S. (eds.) *NASA Formal Methods*. Lecture Notes in Computer Science, vol. 13903, pp. 104–121. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-33170-1_7
 10. Frohme, M., Howar, F., Steffen, B.: LearnLib: 10 years later. In: Piskac, R., Rakamarić, Z. (eds.) *Computer Aided Verification*. Lecture Notes in Computer Science, vol. 15934, pp. 141–160. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-98685-7_7
 11. García, J., Visús, A., Fernández, F.: A taxonomy for similarity metrics between Markov decision processes. *Machine Learning* **111**(11), 4217–4247 (Nov 2022). <https://doi.org/10.1007/s10994-022-06242-4>
 12. de la Higuera, C.: A bibliographical study of grammatical inference. *Pattern Recognition* **38**(9), 1332–1348 (Sep 2005). <https://doi.org/10.1016/j.patcog.2005.01.003>
 13. Hoeffding, W.: A non-parametric test of independence. *The Annals of Mathematical Statistics* **19**(4), 546–557 (Dec 1948). <https://doi.org/10.1214/aoms/1177730150>
 14. Howar, F., Steffen, B.: Active automata learning in practice. In: Bennaceur, A., Hähnle, R., Meinke, K. (eds.) *Machine Learning for Dynamic Software Analysis: Potentials and Limits*, Lecture Notes in Computer Science, vol. 11026, pp. 123–148. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-96562-8_5
 15. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: *Proceedings of ICNN’95 - International Conference on Neural Networks*. pp. 1942–1948 (1995). <https://doi.org/10.1109/ICNN.1995.488968>

16. Killick, R., Fearnhead, P., Eckley, I.A.: Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association* **107**(500), 1590–1598 (Oct 2012). <https://doi.org/10.1080/01621459.2012.737745>
17. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the Abbadingo One DFA Learning Competition and a new evidence-driven state merging algorithm. In: Honavar, V., Slutzki, G. (eds.) *Grammatical Inference. Lecture Notes in Computer Science*, vol. 1433, pp. 1–12. Springer, Berlin, Heidelberg (1998). <https://doi.org/10.1007/BFb0054059>
18. Lloyd, S.P.: Least squares quantization in PCM. *IEEE Transactions on Information Theory* **28**(2), 129–137 (Mar 1982). <https://doi.org/10.1109/TIT.1982.1056489>
19. Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning deterministic probabilistic automata from a model checking perspective. *Machine Learning* **105**(2), 255–299 (Nov 2016). <https://doi.org/10.1007/s10994-016-5565-9>
20. Nguyen, L.V., Johnson, T.T.: Benchmark: DC-to-DC switched-mode power converters (buck converters, boost converters, and buck-boost converters). In: Frehse, G., Althoff, M. (eds.) *ARCH14-15. 1st and 2nd International Workshop on Applied verification for Continuous and Hybrid Systems. EPIc Series in Computing*, vol. 34, pp. 19–24. EasyChair (2015). <https://doi.org/10.29007/23pm>
21. Pandey, S., Srivastava, A.K., Amidan, B.G.: A real time event detection, classification and localization using synchrophasor data. *IEEE Transactions on Power Systems* **35**(6), 4421–4431 (Nov 2020). <https://doi.org/10.1109/TPWRS.2020.2986019>
22. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Müller, A., Nothman, J., Louppe, G., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. *arXiv preprint arXiv:1201.0490*, arXiv (Jun 2018). <https://doi.org/10.48550/arXiv.1201.0490>
23. Perkins, T.J.: Maximum likelihood trajectories for continuous-time markov chains. In: Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., Culotta, A. (eds.) *Advances in Neural Information Processing Systems*. pp. 1437–1445. Curran Associates, Inc. (2009), <https://proceedings.neurips.cc/paper/2009/hash/afda332245e2af431fb7b672a68b659d-Abstract.html>
24. Rezaee Jordehi, A., Jasni, J.: Particle swarm optimisation for discrete optimisation problems: a review. *Artificial Intelligence Review* **43**, 243–258 (Feb 2014). <https://doi.org/10.1007/s10462-012-9373-8>
25. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics* **20**, 53–65 (Nov 1987). [https://doi.org/10.1016/0377-0427\(87\)90125-7](https://doi.org/10.1016/0377-0427(87)90125-7)
26. Saberi, I., Faghih, F., Babil, F.S.: A passive online technique for learning hybrid automata from input/output traces. *ACM Transactions on Embedded Computing Systems* **22**(1), 9 (Oct 2022). <https://doi.org/10.1145/3556543>
27. Truong, C., Oudre, L., Vayatis, N.: Selective review of offline change point detection methods. *Signal Processing* **167**, 107299 (2020). <https://doi.org/10.1016/j.sigpro.2019.107299>
28. Vaz, R., Moraes, G.R., Arruda, E.H.Z., Terceiro, J.C.B.S., Aquino, A.F.C., Decker, I.C., Issicaba, D.: Event detection and classification through wavelet-based method in low voltage wide-area monitoring systems. *International Journal of Electrical Power and Energy Systems* **130**, 106919 (2021). <https://doi.org/10.1016/j.ijepes.2021.106919>
29. Viterbi, A.J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory* **13**(2), 260–269 (Apr 1967). <https://doi.org/10.1109/TIT.1967.1054010>

30. Wang, F., Cao, Z., Tan, L., Zong, H.: Survey on learning-based formal methods: taxonomy, applications and possible future directions. *IEEE Access* **8**, 108561–108578 (2020). <https://doi.org/10.1109/ACCESS.2020.3000907>
31. Xu, G., Yu, G.: On convergence analysis of particle swarm optimization algorithm. *Journal of Computational and Applied Mathematics* **333**, 65–73 (2018). <https://doi.org/10.1016/j.cam.2017.10.026>
32. Yang, X., Beg, O.A., Kenigsberg, M., Johnson, T.T.: A framework for identification and validation of affine hybrid automata from input-output traces. *ACM Transactions on Cyber-Physical Systems* **6**(2), 13 (Apr 2022). <https://doi.org/10.1145/3470455>
33. van Zyl, J.P., Engelbrecht, A.P.: Set-based particle swarm optimisation: a review. *Mathematics* **11**(13), 2980 (2023). <https://doi.org/10.3390/math11132980>