



Adapting the Gini Impurity Criterion for Splitting and Merging Strategies in Extended Automata Learning

Tobiasz Zalewski

Responsible Professor: Dr.ir. S. E. Sicco Verwer

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Tobiasz Zalewski
Final project course: CSE3000 Research Project
Thesis committee: Sicco Verwer, Kubilay Atasu

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

Abstract

Learning Extended Deterministic Finite Automata (EDFAs) from traces provides an interpretable way to model sequential behaviour, but the quality of the learned automaton strongly depends on the heuristic evaluation function used during state refinement. This paper investigates a new heuristic evaluation function for EDSM-based EDFA learning inspired by the Gini impurity criterion used in CART decision trees. Three Gini-based variants are introduced: a local variant that evaluates impurity at the immediate split, a future-state variant that evaluates impurity over the affected future states, and a weighted variant that uses a parameter λ to balance the influence of final and intermediate tails. The variants are implemented in the FlexFringe framework and evaluated on several time-series classification datasets and the CTU-13 network traffic dataset. Hyperparameters are selected using validation data, and final performance is measured on test sets using accuracy, specificity, EDFA size, and final-state consistency. The results show that Gini-based refinement can improve the structural quality of learned EDFAs compared to RTI by producing smaller automata and more class-consistent final states while maintaining competitive accuracy. The weighted variant obtains the best average EDFA accuracy among the proposed variants, although among the Gini variants, higher consistency often comes at the cost of larger automata. These findings suggest that Gini impurity is a promising basis for heuristic evaluation in EDSM-based EDFA learning, especially when interpretable and class-consistent automata are desired.

1 Introduction

Software systems are often intended to follow an abstract behavioural model, but maintaining such models is costly in fast-paced development. When the system under test (SUT) lacks an explicit model, one can infer an approximate finite-state model from observed behaviour. For complex systems, finite-state machines can be extended with guarded multivariate transitions, yielding Extended Deterministic Finite Automata (EDFAs).

When the SUT is available to interact with, active learning can be used to build the underlying DFA by directly querying the subject software. This setting has been studied extensively, with Angluin’s L^* algorithm being a classical example for learning regular languages from membership and equivalence queries [2].

However, in many cases, the underlying system is not directly available or is unknown - examples include reverse engineering external software or analysing network traffic behaviour between unknown parties. Here, only pre-generated logs or traces are available, so the underlying model can only be approximated from a finite and possibly unrepresentative sample.

For simpler systems that can be modelled by a small DFA, efficient state-merging algorithms such as EDSM have been developed [7]. Even though complex systems with memory can also be represented as a DFA, they become exponentially larger in terms of number of states when compared to automata with guards [12]. Therefore, efficient Extended DFA learning is important for compact and interpretable model representations of complex systems.

Verwer et al. [12] developed the EDSM-based RTI algorithm to learn DRTAs - a specific case of an Extended DFA with only one variable. The algorithm has been augmented to also learn Extended DFAs, whose implementation is available in the FlexFringe framework [13]. This paper focuses on exploring a new RTI-based algorithm for EDFAs inspired by Decision Trees.

Research question This paper addresses the following main research question:

Can a Gini-based refinement evaluation heuristic improve EDFA learning compared to RTI, while maintaining interpretable automaton structure?

This question is divided into three subquestions:

1. How do Gini-based EDFA variants compare to RTI and CART in classification accuracy?
2. How do the variants affect EDFA size and final-state consistency?
3. Does weighting final and intermediate future states improve over simpler Gini-based variants?

2 Related Work

2.1 Passive DFA Inference and State Merging

Lang, Pearlmuter and Price introduced Evidence-Driven State Merging (EDSM) in the context of the Abbadingo DFA learning competition [7]. For each pair of candidate states in the APTA (see Section 4), the learner temporarily performs the proposed merge. If this creates nondeterminism, the target states of nondeterministic transitions must also be merged; this determinization process is described by Lang et al. [7]. Candidate merges that would combine incompatible states are rejected. All remaining candidate merges receive an evidence score equal to the number of accepting-accepting and rejecting-rejecting state pairs merged by the proposed merge and its forced determinization merges. The valid candidate with the highest score is then performed.

The Blue-Fringe framework reduces the search space by restricting candidate merges to pairs between a connected red core and its blue fringe [7]. This pattern is reused in later passive automata-learning work, including RTI for DRTAs [12].

2.2 Real-Time Automata and RTI

Verwer, de Weerd and Witteveen extended the state-merging approach from DFAs to Deterministic Real-Time Automata (DRTAs) [12], a specific subtype of Timed Automata [1]. A DRTA extends a regular DFA with a single delay value measuring the time since the previously received symbol. Transitions can then impose bounds on this delay.

RTI is based on red-blue EDSM, but constructs a timed APTA that initially ignores delay values [12]. It resolves timing inconsistencies by splitting transition guards; for example, a guard interval $[n, n']$ can be split at t into $[n, t]$ and $[t + 1, n']$.

This split operation is the main conceptual addition over EDSM. RTI must decide between three kinds of actions: merging two candidate states from the red core and blue fringe, splitting a transition into two new future states, or performing an **extend** operation that promotes a blue state to red. The algorithm scores these actions with evidence functions inspired by EDSM. Their experiments show that RTI substantially outperforms a sampling approach that first removes timing by encoding delays as ordinary symbols and then runs EDSM on a regular DFA. This result is important for the present work: it suggests that guarded transition structure should be learned directly, rather than indirectly through a blown-up DFA with attribute values encoded within the alphabet.

2.3 Position of This Work

This work is rooted in the FlexFringe framework [13], which provides the core framework for EDFA learning - the Blue Fringe state expansion, as in the EDSM algorithm, as well as evidence-based splitting and merging, as in the RTI algorithm. The focus of this paper is to propose a novel evidence scoring heuristic for splitting and merging refinements in the EDFA setting.

3 Preliminaries

3.1 Background: CART and Gini

Classification and Regression Trees (CART) are supervised decision-tree models that recursively partition a data set by selecting tests on input features [4]. For classification, a training set is usually written as

$$D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\},$$

where $\mathbf{x}_i$ is a feature vector and y_i is a class label from a finite class set C . A binary split divides the input space into two subregions, for example by testing whether the j th feature satisfies $x_j < \theta$ for some threshold θ [3, p. 633].

CART chooses splits by measuring how mixed the class labels are before and after the split. One common measure is the Gini impurity. For a subset $A \subseteq D$, let

$$p_c(A) = \frac{|\{(\mathbf{x}, y) \in A \mid y = c\}|}{|A|}$$

be the empirical fraction of examples in A with class label $c \in C$. The Gini impurity is the probability of misclassification if an example from A were assigned a label randomly according to the class distribution in A :

$$\begin{aligned} G(A) &= \sum_{c \in C} p_c \sum_{c' \in C \setminus \{c\}} p_{c'} = \sum_{c \in C} p_c (1 - p_c) \\ &= \sum_{c \in C} p_c - \sum_{c \in C} p_c^2 = 1 - \sum_{c \in C} p_c^2 \end{aligned}$$

The value is zero when all examples in A have the same class label, and is larger when several classes are mixed. For a

candidate split that partitions A into A_L and A_R using some threshold value θ , the weighted post-split impurity is

$$G_{\text{split}}(A_L, A_R) = \frac{|A_L|}{|A|} G(A_L) + \frac{|A_R|}{|A|} G(A_R).$$

The preferred split is the one that minimizes this value, or equivalently the one that maximally reduces impurity compared with $G(A)$. That is,

$$(j^*, \theta^*) = \arg \max_{j, \theta} \left[G(A) - \left(\frac{|A_L(j, \theta)|}{|A|} G(A_L(j, \theta)) + \frac{|A_R(j, \theta)|}{|A|} G(A_R(j, \theta)) \right) \right],$$

where $A_L(j, \theta)$ and $A_R(j, \theta)$ are the subsets induced by the candidate feature-threshold pair (j, θ) .

3.2 Definition of an EDFA

EDFAs or EFSMs are used to refer to any formal structure based on a well-established DFA (for the definition see e.g. Sipser [11]) which augments its capabilities in any way. In literature on Software Models or Software Verification, EDFAs can be seen to be based on regular symbol transitions and predicates over parameters and context variables [8]. They might also treat the attached parameters as memory contents that can be updated at every state [14]. In order to avoid confusion, this paper includes its separate definition, which fits more precisely into the FlexFringe learning framework.

An Extended Deterministic Finite Automaton *EDFA* is a 7-tuple $(Q, \Pi, \Gamma, \Delta, q_0, \mathcal{C}, \mathcal{T})$, where

- Q is a finite set called the *states*,
- $\Pi = \Sigma_1 \times \Sigma_2 \times \dots \times \Sigma_N$ is a *joint alphabet*, where each Σ_i is the alphabet of the corresponding symbol component,
- $\Gamma = \mathcal{D}_1 \times \mathcal{D}_2 \times \dots \times \mathcal{D}_\eta$, where each $\mathcal{D}_i$ is a totally ordered *attribute domain* of the corresponding *attribute*,
- $\Delta \subseteq Q \times \Pi \times (\Gamma \times \Gamma) \times Q$ is a finite set of *guarded transitions*. A transition $(q, \pi, (\mathbf{l}, \mathbf{u}), q') \in \Delta$ can be taken only if the attribute vector $\mathbf{a} \in \Gamma$ satisfies the guard $\forall j \in \{1, \dots, \eta\} : l_j \leq a_j < u_j$,
- $q_0 \in Q$ is the *starting state*,
- $\mathcal{C}$ is the *set of types of final states*,¹
- $\mathcal{T} : Q \rightarrow \mathcal{C}$ is a partial mapping of a *state* to its *type*.

For $\mathbf{l}, \mathbf{u} \in \Gamma$, the notation

$$[\mathbf{l}, \mathbf{u}) = \{\mathbf{a} \in \Gamma \mid \forall j \in \{1, \dots, \eta\} : l_j \leq a_j < u_j\}$$

denotes the corresponding half-open hyperrectangle in the attribute space.

Let $M = (Q, \Pi, \Gamma, \Delta, q_0, \mathcal{C}, \mathcal{T})$ be an EDFA and let $t = (\pi_1, \mathbf{a}_1)(\pi_2, \mathbf{a}_2) \dots (\pi_n, \mathbf{a}_n)$ be a *trace* where each tuple $(\pi_i, \mathbf{a}_i) \in \Pi \times \Gamma$, π_i is a member of the *joint alphabet* and $\mathbf{a}_i = (a_{i1}, \dots, a_{i\eta})^T$ is a multivariate *attribute vector*. A *valid* finite computation of M over a trace t is a sequence of states $r_0, r_1, \dots, r_n \in Q$ with three conditions:

¹Often $\mathcal{C} = \{\text{Accept}, \text{Reject}\}$, but arbitrary types are also allowed

- $r_0 = q_0$,
- for every $i \in \{1, \dots, n\}$, there exists a guarded transition $(r_{i-1}, \pi_i, (\mathbf{l}_i, \mathbf{u}_i), r_i) \in \Delta$ such that
$$\mathbf{a}_i \in [\mathbf{l}_i, \mathbf{u}_i],$$
- $\mathcal{T}(r_n)$ is defined.

A trace receives its predicted class from the final state of its computation in M . We write

$$\text{Type}_M(t) = \mathcal{T}(r_n)$$

when the computation is valid. Otherwise, $\text{Type}_M(t)$ is invalid.

For each class $c \in \mathcal{C}$, the language of EDFA M of type c , denoted $\mathcal{L}_c(M)$, is the set of traces whose derived type is c :

$$\mathcal{L}_c(M) = \{t \in (\Pi \times \Gamma)^* \mid \text{Type}_M(t) = c\}.$$

The multiclass language recognized by M is therefore the family

$$\mathcal{L}(M) = \{\mathcal{L}_c(M) \mid c \in \mathcal{C}\}.$$

Two traces $t = (\pi_1, \mathbf{a}_1)(\pi_2, \mathbf{a}_2) \dots (\pi_n, \mathbf{a}_n) \in (\Pi \times \Gamma)^*$ and $t' = (\pi'_1, \mathbf{a}'_1)(\pi'_2, \mathbf{a}'_2) \dots (\pi'_m, \mathbf{a}'_m) \in (\Pi \times \Gamma)^*$ are said to be *symbolically identical* if and only if they have the same length and the same sequence of joint symbols:

$$n = m \quad \text{and} \quad \forall i \in \{1, 2, \dots, n\} \pi_i = \pi'_i.$$

3.3 Determinism

Automaton M is deterministic if any two guarded transitions with the same source state and joint symbol but different target states have disjoint guards. That is,

$$\begin{aligned} & \forall (q, \pi) \in Q \times \Pi \\ & \forall \substack{(q, \pi, (\mathbf{l}, \mathbf{u}), p) \in \Delta \\ (q, \pi, (\mathbf{l}', \mathbf{u}'), p') \in \Delta} \quad p \neq p' \implies [\mathbf{l}, \mathbf{u}] \cap [\mathbf{l}', \mathbf{u}'] = \emptyset \end{aligned}$$

For deterministic EDFAs, the guarded transitions induce a partial transition function

$$\delta : Q \times \Pi \times \Gamma \rightarrow Q,$$

where $\delta(q, \pi, \mathbf{a}) = q'$ exactly when there exists a guarded transition $(q, \pi, (\mathbf{l}, \mathbf{u}), q') \in \Delta$ such that

$$\mathbf{a} \in [\mathbf{l}, \mathbf{u}].$$

Note that δ being a partial function expresses the determinism condition above: for each state $q \in Q$, joint symbol $\pi \in \Pi$, and attribute vector $\mathbf{a} \in \Gamma$, there is at most one successor state. If $\delta(q, \pi, \mathbf{a})$ is undefined, the trace is invalid.

3.4 Nerode Equivalence for EDFAs

For a single language $L \subseteq (\Pi \times \Gamma)^*$, the usual Nerode equivalence relates two prefixes when no possible continuation can distinguish them with respect to membership in L [10; 5]. In the multiclass setting, an EDFA M recognizes a family of languages

$$\mathcal{L} = \{\mathcal{L}_c \subseteq (\Pi \times \Gamma)^* \mid c \in \mathcal{C}\}.$$

For two traces $x, y \in (\Pi \times \Gamma)^*$, define the multiclass Nerode relation $\equiv_{\mathcal{L}}$ by

$$\begin{aligned} x \equiv_{\mathcal{L}} y & \iff \forall z \in (\Pi \times \Gamma)^* \quad \forall c \in \mathcal{C} \\ & \quad xz \in \mathcal{L}_c \iff yz \in \mathcal{L}_c. \end{aligned}$$

Let $M = (Q, \Pi, \Gamma, \Delta, q_0, \mathcal{C}, \mathcal{T})$ be a deterministic EDFA and let $\mathcal{L}(M) = \{\mathcal{L}_c(M) \mid c \in \mathcal{C}\}$ be its family of class languages. The partial transition function δ extends to traces as

$$\delta^* : Q \times (\Pi \times \Gamma)^* \rightarrow Q,$$

where $\delta^*(q, \epsilon) = q$ and

$$\delta^*(q, w(\pi, \mathbf{a})) = \delta(\delta^*(q, w), \pi, \mathbf{a}),$$

whenever the intermediate state and next transition are defined. The EDFA M therefore induces an equivalence relation $\equiv_M$ on traces whose runs are defined:

$$x \equiv_M y \iff \delta^*(q_0, x) = \delta^*(q_0, y).$$

Two traces are equivalent with respect to M when they reach the same state. For deterministic EDFAs, traces that reach the same state have the same future behaviour and therefore the same type after every continuation. Hence $\equiv_M$ refines the multiclass Nerode congruence of $\mathcal{L}(M)$:

$$x \equiv_M y \implies x \equiv_{\mathcal{L}(M)} y.$$

In learning, the target language $\mathcal{L}(M)$ is unknown. A finite labelled sample can only reveal a partial approximation of the future behaviour of observed prefixes. State merging can be understood as hypothesizing that two observed prefixes belong to the same Nerode equivalence class. Conversely, transition splitting can be understood as hypothesizing that two symbolically identical prefixes with different attribute values have distinguishable futures and should therefore reach different hypothesis states.

4 Problem Description

We assume that the target system is behaving deterministically and can be modelled by an EDFA $M = (Q, \Pi, \Gamma, \Delta, q_0, \mathcal{C}, \mathcal{T})$. Given is a finite labelled sample of traces over the same joint alphabet and attribute domains. We write this sample as a family

$$S = \{S_c \subseteq (\Pi \times \Gamma)^* \mid c \in \mathcal{C}\},$$

where S_c contains the observed traces labelled with class c . Equivalently, the sample may be viewed as a set of labelled pairs (t, c) with $t \in S_c$. In the binary case this reduces to the usual positive/negative setting by taking $\mathcal{C} = \{\text{Accept}, \text{Reject}\}$.

Now, similar to DFA and DRTA learning [12], the goal of the learning procedure is to find a small EDFA $\widehat{M} = (\widehat{Q}, \Pi, \Gamma, \widehat{\Delta}, \widehat{q}_0, \mathcal{C}, \widehat{\mathcal{T}})$ such that every labelled sample trace is assigned its observed class:

$$\forall c \in \mathcal{C} : S_c \subseteq \mathcal{L}_c(\widehat{M}).$$

The initial hypothesis in passive automata learning is an Augmented Prefix Tree Acceptor (APTA). An APTA exactly represents the observed sample: each observed prefix corresponds to a state, the empty prefix corresponds to the root, and two traces share the same path for as long as they share the same symbolic prefix. In the EDFA setting, the initial APTA ignores the attribute values altogether, same as the

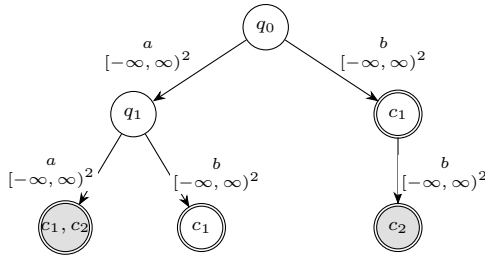


Figure 1: Example EDFA APTA over two attributes. Double circles denote final states. The trace $(a, (1, 4))(a, (2, 3))$ labelled with class c_1 and the trace $(a, (7, 1))(a, (8, 2))$ labelled with class c_2 end in the same inconsistent state because the initial APTA ignores attribute values. Every initial transition guard covers the full attribute domain $[-\infty, \infty)^2$.

timed APTA in RTI [12]. Consequently, if traces with different class labels have the same symbolic prefix but different attribute values, they can end in the same state. Such a state is temporarily inconsistent because it carries evidence for multiple classes. The initial guards cover the whole attribute domain, and later transition splits specialize these guards to remove inconsistencies. An example of an initial APTA is shown in Figure 1.

Starting from the APTA, the learner tries to generalize the sample and move the hypothesis toward consistency. State merging identifies two states that appear to have the same future behaviour. This reduces the size of the hypothesis, but can also force additional merges when determinism would otherwise be violated. For EDFAs, the learner can also refine the hypothesis by splitting transitions. This operation is analogous to the transition-splitting routine used by RTI for DRTAs [12], but it must work over multiple attributes instead of a single delay value.

Let

$$\delta = (q, \pi, (\mathbf{l}, \mathbf{u}), q')$$

be a guarded transition in the current hypothesis. Consider a candidate split of δ on attribute j at threshold θ , where $l_j < \theta < u_j$. Let $\mathbf{u}_\theta$ be equal to $\mathbf{u}$ except $(\mathbf{u}_\theta)_j = \theta$, and let $\mathbf{l}_\theta$ be equal to $\mathbf{l}$ except $(\mathbf{l}_\theta)_j = \theta$. The split removes the old transition δ from Δ and adds two new guarded transitions with non-overlapping bounds:

$$\delta^L = (q, \pi, (\mathbf{l}, \mathbf{u}_\theta), q'') \quad \delta^U = (q, \pi, (\mathbf{l}_\theta, \mathbf{u}), q')$$

$$\Delta' = (\Delta \setminus \{\delta\}) \cup \{\delta^L, \delta^U\},$$

where q'' is the newly created successor state. As in RTI, the traces that fire δ are then separated into two sets of labelled suffixes. For a trace $t = x(\pi, \mathbf{a})z$, where x reaches q and $(\pi, \mathbf{a})$ fires δ , define

$$S_{<\delta}(j, \theta) = \{(z, c) \mid x(\pi, \mathbf{a})z \in S_c, c \in \mathcal{C}, a_j < \theta\},$$

and

$$S_{>\delta}(j, \theta) = \{(z, c) \mid x(\pi, \mathbf{a})z \in S_c, c \in \mathcal{C}, a_j \geq \theta\}.$$

After the split, the suffixes in $S_{>\delta}(j, \theta)$ remain assigned to the old successor state q' , while the suffixes in $S_{<\delta}(j, \theta)$ are

assigned to a newly created successor state q'' . The hypothesis behind such a split is that the two successor states do not represent the same Nerode class: the attribute values separated by the threshold should lead to distinguishable future behaviour.

The central problem is to decide what should be counted as evidence for or against this hypothesis. The proposed approach uses a modified Gini criterion to measure how homogeneous the separated suffix sets are with respect to their class labels and future behaviour, and therefore to score whether the candidate split on (δ, j, θ) is useful.

5 Proposed Gini Scoring for EDFAs

5.1 Split refinements

Consider a guarded transition

$$\delta = (q, \pi, (\mathbf{l}, \mathbf{u}), v)$$

from a source state q to a successor state v on symbol π . A candidate split chooses an attribute j and a threshold θ . The guard is stored on the outgoing transition of q , but the data being separated is the subtree rooted at v . The split partitions the tails into two groups, $S_{<\delta}(j, \theta)$ and $S_{>\delta}(j, \theta)$, where

$$S_{<\delta}(j, \theta) \text{ contains tails with } a_j < \theta,$$

$$S_{>\delta}(j, \theta) \text{ contains tails with } a_j \geq \theta.$$

$S_{>\delta}(j, \theta)$ is routed into the old state v , while $S_{<\delta}(j, \theta)$ is routed to a new state v' . Every trace with a suffix in $S_{<\delta}(j, \theta)$ that would follow the future states $v \rightarrow w \rightarrow p \rightarrow \dots$ before the split now has a duplicated future routed through v' : $v' \rightarrow w' \rightarrow p' \rightarrow \dots$. This duplicated structure is shown in Figure 2.

After the split, δ is removed, while δ^L and δ^U are created:

$$\delta^L = (q, \pi, (\mathbf{l}, \mathbf{u}_\theta), v') \quad \delta^U = (q, \pi, (\mathbf{l}_\theta, \mathbf{u}), v)$$

Every state stores information about traces being routed through that state, and which of them finish at that state or continue further. Let $T^\delta(u)$ be the set of tails stored in state u that passed through the transition δ . Let $\text{Tr}(t)$ denote the complete trace from which a tail t originates. In the classification setting considered here, the class set $\mathcal{C}$ is the set of observed final trace labels, and every tail inherits the label of its complete trace. Thus $y(t) = y(\text{Tr}(t))$, and

$$T_c^\delta(u) = \{t \in T^\delta(u) \mid y(t) = c\}.$$

Three variants of the Gini impurity evaluation for a node u in an EDFA M are proposed:

Variant 1: Local The simplest possible approach is to treat the split exactly like a decision-tree node split. Before the split, only the class distribution at the state v is considered. After the split, only v and its duplicate v' are considered. The Gini impurity score $G(v)$ of the state v is

$$G^\delta(v) = 1 - \sum_{c \in \mathcal{C}} p_c^2 = 1 - \sum_{c \in \mathcal{C}} \left(\frac{|T_c^\delta(v)|}{|T^\delta(v)|} \right)^2$$

Before performing the split (δ, j, θ) , record

$$G_{pre}^\delta(v) = G^\delta(v)$$

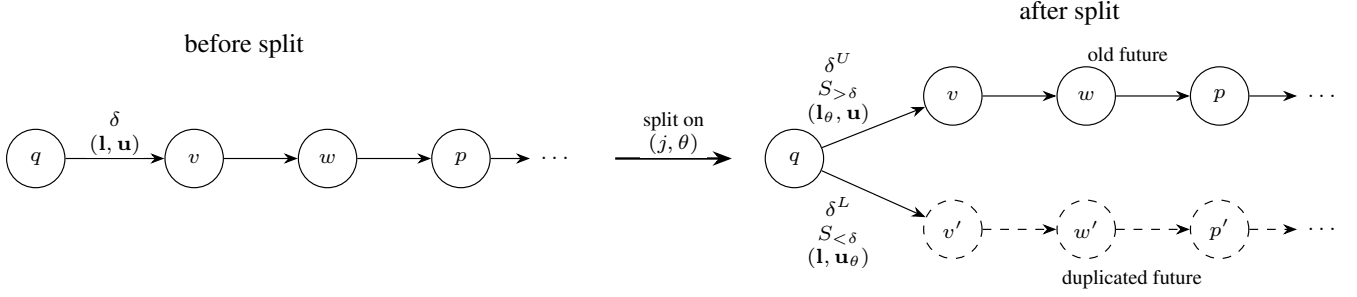


Figure 2: Horizontal view of a transition split. The old transition into v is replaced by two guarded transitions: observations in $S_{>\delta}(j, \theta)$ continue through the old future rooted at v , while observations in $S_{<\delta}(j, \theta)$ are routed through the new successor v' and a duplicated future subtree.

After performing the split, trace counts at v and v' are updated. Two new guards δ^L and δ^U take place of δ . Let

$$N_v = |T^{\delta^U}(v)| \quad \text{and} \quad N_{v'} = |T^{\delta^L}(v')|.$$

The fractions of tails routed through v and v' are then

$$r_v = \frac{N_v}{N_v + N_{v'}},$$

$$r_{v'} = \frac{N_{v'}}{N_v + N_{v'}}.$$

Now, the local impurity reduction $h(\delta, j, \theta)$ can be calculated as

$$h(\delta, j, \theta) = G_{pre}^{\delta}(v) - r_v G^{\delta^U}(v) - r_{v'} G^{\delta^L}(v')$$

Every split score $h(\delta, j, \theta)$, is tested against the threshold α , so that the final evaluation is

$$h^*(\delta, j, \theta) = \begin{cases} h(\delta, j, \theta), & \text{if } h(\delta, j, \theta) \geq \alpha, \\ -1, & \text{otherwise.} \end{cases}$$

Here α corresponds to the implementation parameter `min_split_score`.

This approach ignores the future subtree completely. It only asks whether the immediate successor state becomes more label-homogeneous after separating the tails.

Variant 2: Future aware Let $F^\delta(u)$ be the set of future states reachable from u by the tails $T^\delta(u)$. Note that $u \in F^\delta(u)$. After the split, the class distributions of all states in $F^\delta(u)$ could possibly be changed. Therefore, we propose including all states $F^\delta(u)$ in the Gini impurity calculation, so that it represents the expected label disagreement at a randomly chosen future tail occurrence, weighted by tail mass. We will consider a general case where each tail can contribute a count based on some weight w , which from now on will be referred to as *tail mass*. Consequently,

$$M[T] = \sum_{t \in T} w(t)$$

is the total mass of a set of tails T .

For the future subtree rooted at v , define the total mass

$$M_\delta(v) = \sum_{k \in F^\delta(v)} M[T^\delta(k)].$$

Let $p_{n,\delta}$ be the fraction of the combined tail mass from transition δ that is passing through n . That is,

$$p_{n,\delta} = \frac{M[T^\delta(n)]}{M_\delta(v)}.$$

Also define within-state class probabilities

$$p_{n,\delta}^c = \frac{M[T_c^\delta(n)]}{M[T^\delta(n)]}.$$

Then we define the future-aware Gini impurity at state v as (see Appendix for full derivation)

$$G^\delta(v) = \sum_{n \in F^\delta(v)} p_{n,\delta} \sum_{c \in \mathcal{C}} p_{n,\delta}^c \sum_{c' \in \mathcal{C} \setminus \{c\}} p_{n,\delta}^{c'}$$

$$= 1 - \frac{1}{M_\delta(v)} \sum_{n \in F^\delta(v)} \sum_{c \in \mathcal{C}} \frac{M[T_c^\delta(n)]^2}{M[T^\delta(n)]}.$$

Let

$$m_v = \sum_{k \in F^{\delta^U}(v)} M[T^{\delta^U}(k)],$$

$$m_{v'} = \sum_{k \in F^{\delta^L}(v')} M[T^{\delta^L}(k)],$$

$$r_v = \frac{m_v}{m_v + m_{v'}} \quad \text{and} \quad r_{v'} = \frac{m_{v'}}{m_v + m_{v'}}.$$

Before evaluating a candidate split, the pre-split impurity is recorded as

$$G_{pre}^\delta(v) = G^\delta(v),$$

computed on the subtree rooted at v . Similarly to Variant 1, the Gini impurity score is

$$h(\delta, j, \theta) = G_{pre}^\delta(v) - r_v G^{\delta^U}(v) - r_{v'} G^{\delta^L}(v').$$

For Variant 2, the tail weighing function $w(t)$ is simple:

$$w(t) = 1$$

Variant 3: λ -weighted future-aware Variant 3 uses the future-aware formulation of Variant 2, but assigns tail weights according to a parameter $\lambda \in [0, 1]$. For a split (δ, j, θ) on a transition $\delta = (q, \pi, (\mathbf{l}, \mathbf{u}), v)$, let $L_\delta(\text{Tr}(t))$ be the number of future tail occurrences contributed by the complete trace $\text{Tr}(t)$ after firing δ , including the final tail occurrence. Define

$$w_\lambda(t) = \lambda \mathbf{1}[\text{Final}(t)] + \frac{1 - \lambda}{L_\delta(\text{Tr}(t))}.$$

Two main issues motivate this choice. First, if continuing and final tails were weighted the same, continuing tails would dominate the contributions to the final score. For a tail of remaining length l starting from δ , it is counted $l - 1$ times as a continuing tail, and only once as a final tail. For longer traces, final tails would become negligible, possibly leading to states with lower consistency. Second, if the trace sample contains both short and long traces, then the longer traces are counted more frequently since they pass through more states. That leads to a higher contribution of longer traces to the Gini impurity, which skews the true impurity. The first issue is addressed by introducing the parameter λ , which allows the learner to assign more weight to final tails. Setting λ closer to 1 favours consistency in final states. The second issue is resolved by dividing every tail's non-final share by the remaining length of the corresponding trace, counted starting from the transition δ . It effectively limits a single trace's total contribution to the impurity to 1 when added throughout all the states it traverses after δ .

5.2 Merge refinements

The FlexFringe framework selects two candidate states $q, q' \in Q$ for a merge refinement. The Gini merge scoring requires the separate impurities of these states before the merge, and then the impurity of the new state q^* after the merge. The scoring routines for the variants follow from their split counterparts.

Variant 1: Local For a state u , let $T(u)$ be the set of tails stored in u , and let

$$T_c(u) = \{t \in T(u) \mid y(t) = c\}$$

be the tails in u whose complete trace has class label c . The local Gini impurity of u is

$$G(u) = 1 - \sum_{c \in \mathcal{C}} \left(\frac{|T_c(u)|}{|T(u)|} \right)^2.$$

Let

$$N_q = |T(q)| \quad \text{and} \quad N_{q'} = |T(q')|,$$

and define

$$r_q = \frac{N_q}{N_q + N_{q'}} \quad \text{and} \quad r_{q'} = \frac{N_{q'}}{N_q + N_{q'}}.$$

The pre-merge impurity is

$$G_{pre}(q, q') = r_q G(q) + r_{q'} G(q').$$

The post-merge impurity is

$$G_{post}(q, q') = G(q^*).$$

The local merge score is therefore

$$\begin{aligned} h_{merge}(q, q') &= G_{pre}(q, q') - G_{post}(q, q') \\ &= r_q G(q) + r_{q'} G(q') - G(q^*). \end{aligned}$$

Variant 2: Future aware Merging is a reversed splitting routine, but there is no transition δ whose routed traces must be isolated. Weight function $w(t)$ is the same as in splitting.

Let

$$T_{u,c}(n) = \{t \in T_u(n) \mid y(t) = c\}$$

be tails in n originating from u whose complete trace has class label c . As in the split routine, let

$$M[T] = \sum_{t \in T} w(t)$$

be the total mass of a set of tail occurrences T .

For the future region rooted at u , define the total mass

$$m_u = \sum_{k \in F(u)} M[T_u(k)].$$

The future-aware impurity rooted at u is then

$$G(u) = 1 - \frac{1}{m_u} \sum_{n \in F(u)} \sum_{c \in \mathcal{C}} \frac{M[T_{u,c}(n)]^2}{M[T_u(n)]}.$$

States with zero mass are ignored in the summation.

Before the merge, the two candidate states are scored separately. In particular,

$$m_q = \sum_{n \in F(q)} M[T_q(n)] \quad \text{and} \quad m_{q'} = \sum_{n \in F(q')} M[T_{q'}(n)].$$

Define

$$r_q = \frac{m_q}{m_q + m_{q'}} \quad \text{and} \quad r_{q'} = \frac{m_{q'}}{m_q + m_{q'}}.$$

The pre-merge impurity is

$$G_{pre}(q, q') = r_q G(q) + r_{q'} G(q').$$

The post-merge impurity is

$$G_{post}(q, q') = G(q^*).$$

The Variant 2 merge score is therefore

$$h_{merge}(q, q') = G_{pre}(q, q') - G_{post}(q, q')$$

Variant 3: λ -weighted future aware Variant 3 uses the same merge routine as Variant 2, but replaces the unit weight $w(t) = 1$ by the λ -weighted future mass used in the split routine. Since a merge has no transition δ , the normalising quantity is not the remaining length after a transition. Instead, before each candidate merge, the algorithm makes a single pass over the future occurrences rooted at q and q' and counts how many occurrences from the same trace appear in each candidate future region.

Let $O_u(\text{Tr}(t))$ be the number of future tail occurrences contributed by the complete trace $\text{Tr}(t)$ in the future region rooted at u . Then the merge weight is

$$w_{\lambda,u}(t) = \lambda \mathbf{1}[\text{Final}(t)] + \frac{1 - \lambda}{O_u(\text{Tr}(t))}.$$

The counts O_q , $O_{q'}$, and O_{q^*} are computed separately: first for the future region of q , then for the future region of q' , and finally for the future region of the temporarily merged state q^* .

The Gini formulas are otherwise identical to Variant 2, but use the corresponding $w_{\lambda,u}$ when computing each mass term $M[T]$.

5.3 Final evaluation

After calculating the scores of all merges and splits, the one with the highest score is performed. The score of an **extend** operation is 0. Ties are broken by choosing a refinement resulting in smallest size of the automaton.

6 Experimental Setup and Results

The accuracy of the Gini heuristic was assessed on Time-Series classification problems. They can be encoded as a trace of a constant symbol with attached values from the time series, e.g. time series $(1, 2, 3, \dots)$ can be encoded as $(a, 1)(a, 2)(a, 3) \dots$ with an arbitrary symbol a .

The Gini variants were compared to RTI and CART. For CART, the data were encoded as a feature vector of size $\text{length} \times \text{dimensions}$.

Datasets used in the experiment were sourced from the *aeon* [9] Python package, which provides a unified interface to the UCR time series repository. Table 4 (see Appendix) summarizes the selected datasets. They were then converted to the Abbadingo csv format. The UCR datasets are pre-split into train and test partitions. The train partition was subsequently split into train and validation sets in a 75/25 ratio. The validation set was used to tune λ , α , and RTI’s confidence_bound; the random seed was 190.

The Gini evaluation function is also tested on a network flow dataset CTU-13 [6]. The dataset contains logs of network traffic at the Czech Technical University in Prague, where regular traffic is intertwined with botnet traffic. The logs from all 13 botnet scenarios were collected into coherent sessions between hosts, sampled uniformly in lengths from 5 to 30 and converted to the Abbadingo format for training and prediction. Any trace containing at least one botnet-labelled flow was labelled *botnet*, and the rest — *benign*. The number of traces was stratified to 1500 per class, and the train/validation/test split is 15/15/70. For CART, every part of the trace was encoded numerically into a feature vector of size $1 + \text{max.length} \times (\text{num.attributes} + 1)$, where the first feature stores the trace length and the extra per-step feature stores the symbol. Shorter traces were padded on the right, with zeros for missing attributes and -1 for symbols.

Gini Variant 3 relies on two parameters: λ and the split score threshold α . The experiments were run with different combinations of these parameters on an 11×11 grid, chosen from the range $\lambda \in [0, 1]$, $\alpha \in [0.01, 0.2]$. Parameter α is never set to 0, since then it accepts every possible split, which leads to an exponential growth of the number of states and long runtime. It is also capped at 0.2, since higher values allow little to no splits to occur, which often causes the APTA to collapse to a single state. Similarly, the other variants and RTI were run with different values of α and confidence_bound respectively. Best parameters were chosen based on the validation set.

Alongside accuracy and specificity, final EDFA size and state consistency are reported. For multiclass datasets, specificity is computed as the macro-average of one-vs-rest specificity over all classes. Consistency is the fraction of final states that are pure, where a final state has at least one training tail ending in it and is pure if all such tails have the same

Table 1: Summary of the experimental results. See the full table in the Appendix.

Dataset	Gini accuracy on test set [%]			RTI accuracy[%] (conf. bound)	CART baseline [%]
	V1(α)	V2(α)	V3(α, λ)		
Average	60.27	62.95	67.04	63.86	80.22

Table 2: Summary of the structure metrics for learned EDFAs. See the full table in the Appendix.

Dataset	V1		V2		V3		RTI	
	Size	Consistency	Size	Consistency	Size	Consistency	Size	Consistency
Average	4.00	0.73	12.30	0.78	41.20	0.94	303.10	0.53

class label. It is interpreted together with EDFA size, since consistency is easier to achieve with larger automata.

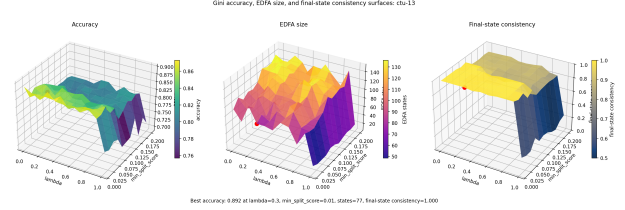


Figure 3: Accuracy, EDFA size, and final-state consistency surfaces for CTU-13.

7 Discussion

Variant 1 & 2 Since Variant 1 completely disregards the deeper states after the split, it was expected to perform the worst. This is true on average; however, for 3 datasets it scored better than Variant 2, and for the CTU-13 dataset it scored the highest out of all methods used, which was unexpected. Running the experiment with other random seeds (2, 180) confirms this result. Figure 4 shows this EDFA.

Variant 2 scored on average 2.68pp more than Variant 1, which suggests that extending the local implementation to consider all the future states is a reasonable choice. Even though Variant 2 yields better accuracy in general, Variant 1 cannot be fully discarded as worse, since it performed best on the CTU-13 dataset.

Variant 3 The performance of Variant 3 shows promising results. It matched or outperformed Variant 2 on 9 of the 10 datasets, with CTU-13 being 1.5pp worse. In terms of accuracy, the results therefore give little reason to choose Variant 2 over Variant 3 whenever a validation set is available, although Variant 2 may still be preferable when compactness is the main concern. Comparing the EDFAs of two variants in Figure 6 and Figure 7 (see Appendix), Variant 2 is less entangled and is more easily interpretable than Variant 3. Table 6 (see Appendix) shows that in 3 cases Variant 3 settles on $\lambda = 0$, which for datasets with traces of equal length, as in the case of UCR datasets, behaves like Variant 2. Then, in both cases each tail has constant weight, hence constant contribution to the total impurity. In the other 7 cases, a dataset-specific λ is chosen, that puts more significance on final tails.

Lambda All runs except for *FreezerRegularTrain* show a sharp decline in accuracy, size and consistency of the final EDFA when $\lambda = 1$, that is, when only final states are considered in the impurity calculation and intermediate states are irrelevant. An example of this pattern can be seen in Figure 3.

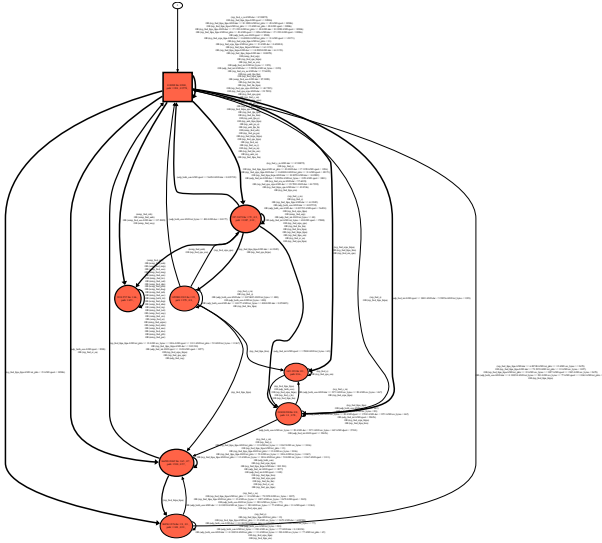


Figure 4: EDFA learned with Gini Variant 1 on CTU-13.

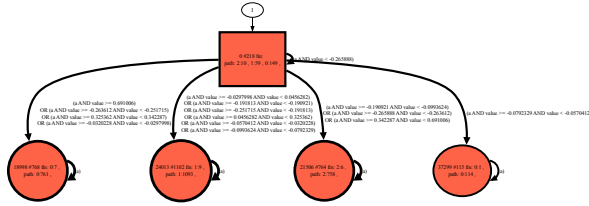


Figure 5: EDFA learned by Variant 3 on CBF.

The opposite behaviour of `FreezerRegularTrain` is plotted in Figure 11 (see Appendix). Even though $\lambda = 1$ often results in smaller size, it should not come at a cost of accuracy nor consistency. This result suggests a limitation of purely EDSM-like techniques, which only factor in the changes in final states. It supports a claim that non-final future states contain important evidence regarding the similarity between states.

Greedy search and α In comparison to CART, EDFAs underperform the most for *Coffee*, *SonyAIBORobotSurface2* and *CBF*. Inspecting the corresponding CART models reveals that splitting happens on positions deep into the time series, e.g. *coffee* is only split once on the 159th value (Figure 9 in Appendix). Similarly, Figure 8 (see Appendix) shows how the *CBF* dataset is split on two deeper values and Figure 5 shows the corresponding EDFA. EDFA learning follows a greedy state refinement procedure. When valuing a split, CART considers all positions of the series at once, whereas an EDFA checks the splits sequentially from left to right. As a consequence, when a proper split should happen only late into the series, the greedy search is likely to find locally satisfactory splits above the α threshold before reaching this depth. Consequently, a globally useful late split may never be selected.

Gini vs. RTI RTI remains a strong EDFA baseline. Variant 1 performs worse than RTI on average, although it achieves the best result on CTU-13. Variant 2 reaches almost the same average accuracy as RTI, suggesting that incorporating future-state impurity makes the Gini heuristic competitive with the established RTI method. Variant 3 gives the clearest improvement: it obtains the highest average EDFA accuracy among the EDFA methods and outperforms or matches RTI on 8 of the 10 datasets. This improvement is also reflected in the structural metrics. RTI produces the largest automata on average, with 303.10 states and final-state consistency of 0.53. The Gini variants are substantially smaller, and Variant 3 achieves much higher final-state consistency of 0.94 while still using only 41.20 states on average. Although Variant 3 is larger than the simpler Gini variants, this additional size appears to improve the separation of classes in final states while still remaining far more compact than RTI. This suggests that the additional tail-weighting parameter λ , when selected on validation data, improves the flexibility of the Gini-based heuristic and provides a useful trade-off between accuracy, compactness, and class consistency.

8 Conclusions and Future Work

This paper investigated Gini impurity as a refinement evaluation heuristic for EDSM-based EDFA learning. Three variants were implemented in *FlexFringe*: a local split heuristic, a future-aware heuristic, and a λ -weighted future-aware heuristic. The answer to the main research question is positive, but conditional: Gini-based scoring can improve EDFA learning compared to RTI in average accuracy and structural quality, but the improvement is not uniform across all datasets. The weighted variant achieved the best average EDFA accuracy among the EDFA methods, produced substantially more class-consistent final states than RTI, and remained much smaller on average. Nevertheless, CART still achieved higher average classification accuracy overall.

In terms of accuracy, Variant 3 performed best among the EDFA methods, while Variant 2 was close to RTI and Variant 1 was weaker on average. In terms of structure, the Gini variants generally produced smaller automata than RTI, and Variant 3 achieved the highest final-state consistency, although it was larger than the simpler Gini variants. The results suggest that future-state information and weighting are useful when scoring EDFA refinements: Variant 2 improved over the local Variant 1 on average, while Variant 3 further improved the trade-off between accuracy and structure. When interpretability is the highest priority, however, Variant 1 or Variant 2 may still be preferable because they produce simpler automata, at the cost of lower average accuracy and final-state consistency.

Future work should address the limitations of the greedy refinement strategy. In particular, when the most informative split occurs late in a trace, locally attractive early splits may prevent a better global structure from being learned. Alternatives such as Monte Carlo tree search over candidate refinements could be explored. Further experiments on variable-length traces and additional software or network-behaviour datasets would also help clarify when Gini-based EDFA learning is most useful.

9 Responsible Research

This work does not involve collecting new data from human participants. The time-series datasets are public benchmark datasets obtained through aeon and the UCR archive. The CTU-13 dataset is also public, but it consists of real network data. The experiments only use this data for aggregate model evaluation and do not attempt to identify users, hosts, or organizations whose traffic was recorded in this dataset.

The network-security use case of EDFAs has to be handled carefully. Better malicious-traffic classifiers can support defensive systems, but models trained on such data may also reveal patterns useful for evasion. The results in this paper should be interpreted as an evaluation of a learning heuristic, not as a deployable intrusion-detection system. Any practical deployment would require additional validation, monitoring, and human oversight, especially because false positives or false negatives can have serious consequences.

Research reproducibility is achieved by the use of public datasets, a fixed preprocessing pipeline to convert traces to Abbadingo format, a decision-tree baseline, and a fixed grid over λ and α . The parameter grid, preprocessing scripts, dataset splits, and random seeds are reported together with the results and supplied with the paper. The code from this research is available at a public fork of FlexFringe at <https://github.com/tobizal/flexfringe-tobi.git>.

Development of this research was supported by the use of AI tools. ChatGPT 5.5 helped to fix a memory leak which stalled running EDFA learning on bigger datasets. At the start of this research, FlexFringe did not fully support the EDFA workflow required for these experiments, so the LLM was used to extend saving and loading APTAs with arbitrary data guards on multiple parameters. This was crucial for running the automaton in the predict mode. Scripts for data conversion, experiment execution and results visualization were made with the help of ChatGPT 5.5. Nevertheless, all other components central to this research such as literature review, Gini scoring ideas with derivation and implementation, experiment design, and writing, were made by the author.

References

- [1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [2] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
- [3] C.M. Bishop. *Pattern recognition and machine learning*, volume 4. Springer New York, 2006.
- [4] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.
- [5] Dana Fisman. Inferring regular languages and ω -languages. *Journal of Logical and Algebraic Methods in Programming*, 98:27–49, 2018.
- [6] S. García, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *Computers & Security*, 45:100–123, 2014.
- [7] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1433:1 – 12, 1998.
- [8] Davide Lorenzoli, Leonardo Mariani, and Mauro Pezzè. Automatic generation of software behavioral models. In *2008 ACM/IEEE 30th International Conference on Software Engineering*, pages 501–510, May 2008.
- [9] Matthew Middlehurst, Ali Ismail-Fawaz, Antoine Guillaume, Christopher Holder, David Guijo-Rubio, Guzal Bulatova, Leonidas Tsaprounis, Lukasz Mentel, Martin Walter, Patrick Schäfer, and Anthony Bagnall. aeon: a python toolkit for learning from time series. *Journal of Machine Learning Research*, 25(289):1–10, 2024.
- [10] A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.
- [11] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, Boston, MA, third edition, 2013.
- [12] Sicco Verwer, Mathijs De Weerd, and Cees Witteveen. Efficiently identifying deterministic real-time automata from labeled data. *Machine Learning*, 86(3):295 – 333, 2012. Cited by: 27; All Open Access, Green Open Access, Hybrid Gold Open Access.
- [13] Sicco Verwer and Christian A. Hammerschmidt. flexfringe: A passive automaton learning package. In L. O’Conner, editor, *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017*, pages 638–642, United States, 2017. IEEE. ICSME 2017 : 33rd International Conference on Software Maintenance and Evolution, ICSME ; Conference date: 17-09-2017 Through 24-09-2017.
- [14] Neil Walkinshaw, Ramsay Taylor, and John Derrick. Inferring extended finite state machine models from software executions. *Empirical Software Engineering*, 21(3):811 – 853, 2016. Cited by: 114; All Open Access, Green Open Access.

A Appendix

Derivation of the Gini formula used by Variants 2 and 3:

$$\begin{aligned}
 G^\delta(v) &= \sum_{n \in F^\delta(v)} p_{n,\delta} \sum_{c \in \mathcal{C}} p_{n,\delta}^c \sum_{c' \in \mathcal{C} \setminus \{c\}} p_{n,\delta}^{c'} \\
 &= \sum_{n \in F^\delta(v)} p_{n,\delta} \sum_{c \in \mathcal{C}} p_{n,\delta}^c (1 - p_{n,\delta}^c) \\
 &= \sum_{n \in F^\delta(v)} p_{n,\delta} \left[\sum_{c \in \mathcal{C}} p_{n,\delta}^c - \sum_{c \in \mathcal{C}} (p_{n,\delta}^c)^2 \right] \\
 &= \sum_{n \in F^\delta(v)} p_{n,\delta} \left[1 - \sum_{c \in \mathcal{C}} (p_{n,\delta}^c)^2 \right] \\
 &= \sum_{n \in F^\delta(v)} p_{n,\delta} - \sum_{n \in F^\delta(v)} p_{n,\delta} \sum_{c \in \mathcal{C}} (p_{n,\delta}^c)^2 \\
 &= 1 - \sum_{n \in F^\delta(v)} \frac{M[T^\delta(n)]}{M_\delta(v)} \sum_{c \in \mathcal{C}} \frac{M[T_c^\delta(n)]^2}{M[T^\delta(n)]^2} \\
 &= 1 - \frac{1}{M_\delta(v)} \sum_{n \in F^\delta(v)} \sum_{c \in \mathcal{C}} \frac{M[T_c^\delta(n)]^2}{M[T^\delta(n)]}.
 \end{aligned}$$

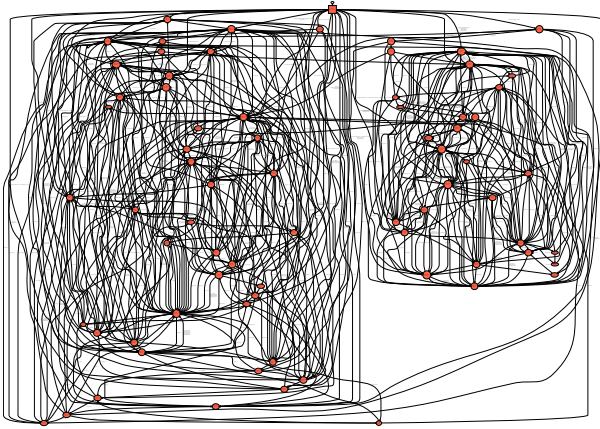


Figure 6: EDFA learned by Variant 3 on the CTU-13 dataset.

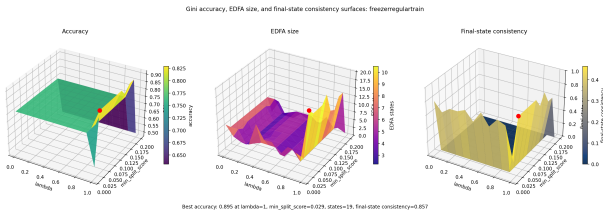


Figure 11: Accuracy, EDFA size, and final-state consistency surfaces for FreezerRegularTrain.

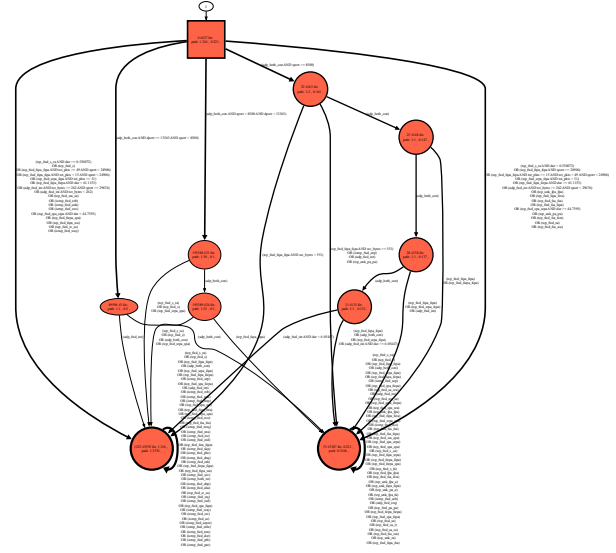
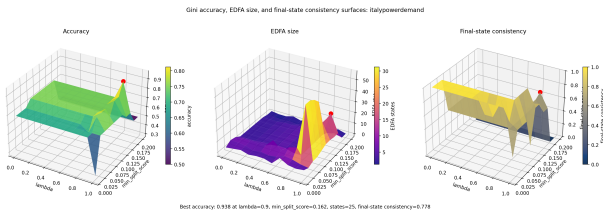


Figure 7: EDFA learned by Variant 2 on the CTU-13 dataset.

Figure 12: Accuracy, EDFA size, and final-state consistency surfaces for italypowerdemand.

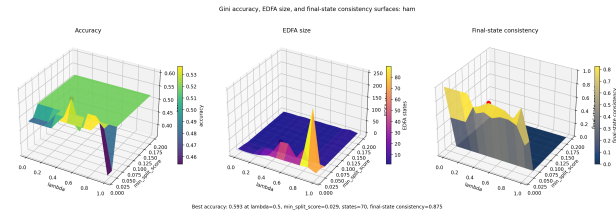


Figure 13: Accuracy, EDFA size, and final-state consistency surfaces for Ham.

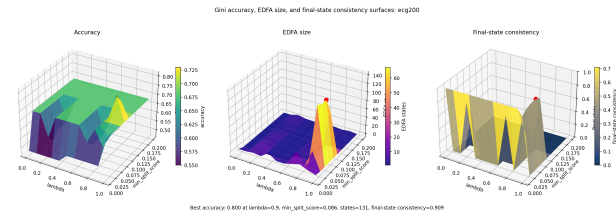


Figure 14: Accuracy, EDFA size, and final-state consistency surfaces for ECG200.

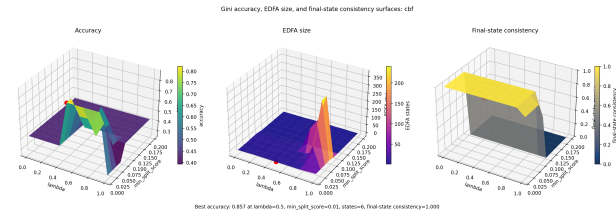


Figure 15: Accuracy, EDFA size, and final-state consistency surfaces for CBF.

Table 3: Detailed EDFA test-set metrics. Accuracy and specificity are reported as percentages. Size is the number of states in the final EDFA, and consistency is the fraction of final states that are class-pure.

Dataset	V1				V2				V3				RTI			
	Acc.	Spec.	Size	Cons.	Acc.	Spec.	Size	Cons.	Acc.	Spec.	Size	Cons.	Acc.	Spec.	Size	Cons.
cbf	36.22	68.17	3.00	1.000	35.33	67.68	4.00	1.000	42.56	71.28	5.00	1.000	33.11	66.67	1.00	0.000
coffee	46.43	45.90	3.00	1.000	46.43	45.90	3.00	1.000	46.43	45.90	7.00	1.000	46.43	45.90	461.00	1.000
ctu-13	92.38	92.36	8.00	0.625	89.32	89.30	10.00	1.000	87.82	87.79	74.00	1.000	82.31	82.28	38.00	0.750
ecg200	64.00	50.00	1.00	0.000	64.00	50.00	1.00	0.000	65.00	55.03	131.00	0.909	69.00	61.81	278.00	0.778
ering	55.56	91.11	8.00	1.000	67.41	93.48	8.00	1.000	67.41	93.48	8.00	1.000	64.81	92.96	279.00	0.818
freezerregulartrain	76.42	76.42	3.00	0.000	80.00	80.00	54.00	0.667	89.61	89.61	19.00	0.857	68.42	68.42	990.00	0.375
ham	49.52	48.97	2.00	1.000	53.33	53.54	26.00	0.500	61.90	61.22	70.00	0.875	61.90	61.71	805.00	0.529
italypowerdemand	66.18	66.14	5.00	0.667	72.79	72.77	11.00	0.667	83.58	83.60	25.00	0.778	83.38	83.38	68.00	0.500
sonyaiborobotsurface2	61.49	60.27	4.00	1.000	59.71	62.72	3.00	1.000	59.71	62.72	4.00	1.000	68.42	69.16	48.00	0.500
twoleadecg	54.52	54.54	3.00	1.000	61.19	61.21	3.00	1.000	66.37	66.38	69.00	1.000	60.84	60.87	63.00	0.000
Average	60.27	65.39	4.00	0.729	62.95	67.66	12.30	0.783	67.04	71.70	41.20	0.942	63.86	69.31	303.10	0.525

Table 4: Time-series datasets selected for the experiments. Dataset statistics follow the format reported by the Time Series Classification website.

Dataset	Train num.	Test num.	Length	No. of dimensions	No. of classes
FreezerRegularTrain	150	2850	301	1	2
cbf	30	900	128	1	3
ham	109	105	431	1	2
sonyaiborobotsurface2	27	953	65	1	2
italypowerdemand	67	1029	24	1	2
ECG200	100	100	96	1	2
twoleadecg	23	1139	82	1	2
ERing	30	270	65	4	6
coffee	28	28	286	1	2

Table 6: Summary of the experimental results. Hyper-parameters were chosen based on validation sets. Accuracy listed was calculated based on the test sets.

Dataset	Gini accuracy on test set [%]			RTI accuracy[%] (conf. bound)	CART baseline [%]
	V1(α)	V2(α)	V3(α, λ)		
cbf	36.2 (0.01)	35.3 (0.01)	42.6 (0.01, 0.5)	33.1 (0)	71.9
coffee	46.4 (0.01)	46.4 (0.01)	46.4 (0.01, 0)	46.4 (1)	89.3
CTU-13	92.4 (0.2)	89.3 (0.029)	87.8 (0.01, 0.3)	82.3 (0)	91.8
ECG200	64.0 (0.029)	64.0 (0.029)	65.0 (0.086, 0.9)	69.0 (0.5)	73.0
ERing	55.6 (0.143)	67.4 (0.143)	67.4 (0.143, 0)	64.8 (0.3)	70.0
freezerregulartrain	76.4 (0.01)	80.0 (0.029)	89.6 (0.029, 1)	68.4 (0.5)	91.8
Ham	49.5 (0.01)	53.3 (0.029)	61.9 (0.029, 0.5)	61.9 (0.1)	66.7
italypowerdemand	66.2 (0.086)	72.8 (0.067)	83.6 (0.162, 0.9)	83.4 (0.5)	96.8
sonyaiborobotsurface2	61.5 (0.086)	59.7 (0.01)	59.7 (0.01, 0)	68.4 (0.1)	74.6
twoleadecg	54.5 (0.01)	61.2 (0.01)	66.4 (0.067, 0.8)	60.8 (0.1)	76.4
Average	60.27	62.95	67.04	63.86	80.22

Table 5: Structure metrics for learned EDFAs.

Dataset	V1		V2		V3		RTI	
	Size	Consistency	Size	Consistency	Size	Consistency	Size	Consistency
cbf	3.00	1.00	4.00	1.00	5.00	1.00	1.00	0.00
coffee	3.00	1.00	3.00	1.00	7.00	1.00	461.00	1.00
ctu-13	8.00	0.63	10.00	1.00	74.00	1.00	38.00	0.75
ecg200	1.00	0.00	1.00	0.00	131.00	0.91	278.00	0.78
ering	8.00	1.00	8.00	1.00	8.00	1.00	279.00	0.82
freezerregulartrain	3.00	0.00	54.00	0.67	19.00	0.86	990.00	0.38
ham	2.00	1.00	26.00	0.50	70.00	0.88	805.00	0.53
italypowerdemand	5.00	0.67	11.00	0.67	25.00	0.78	68.00	0.50
sonyaiborobotsurface2	4.00	1.00	3.00	1.00	4.00	1.00	48.00	0.50
twoleadecg	3.00	1.00	3.00	1.00	69.00	1.00	63.00	0.00
Average	4.00	0.73	12.30	0.78	41.20	0.94	303.10	0.53

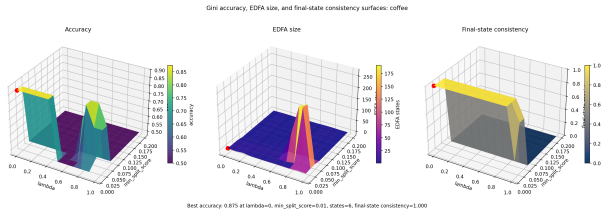


Figure 16: Accuracy, EDFA size, and final-state consistency surfaces for Coffee.

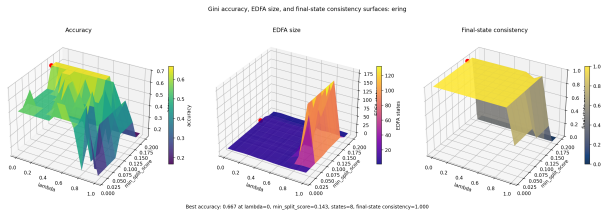


Figure 17: Accuracy, EDFA size, and final-state consistency surfaces for ERing.

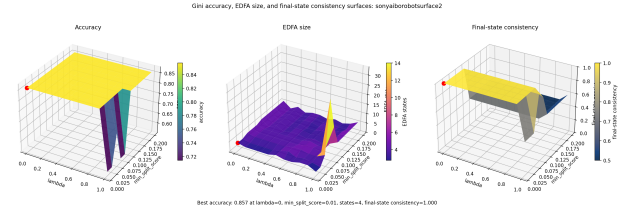


Figure 18: Accuracy, EDFA size, and final-state consistency surfaces for sonyaiborobotsurface2.

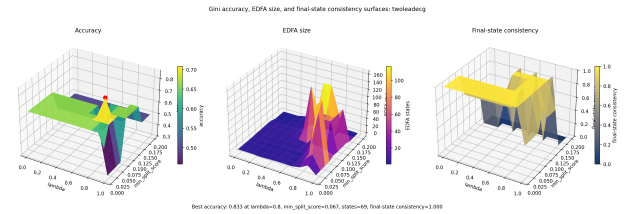


Figure 19: Accuracy, EDFA size, and final-state consistency surfaces for twoleadecg.

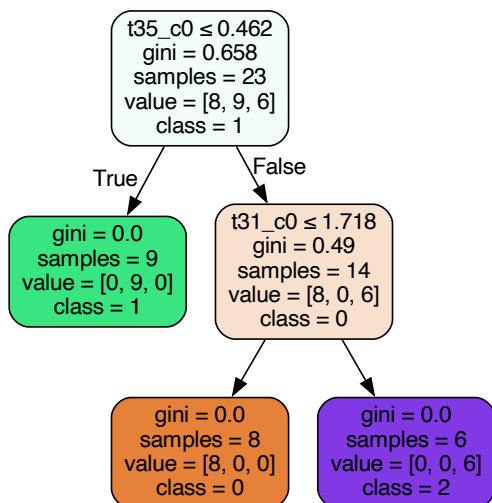


Figure 8: CART decision tree for the CBF dataset.

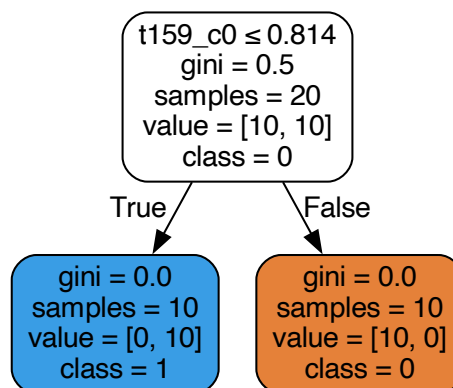


Figure 9: CART decision tree for the Coffee dataset.

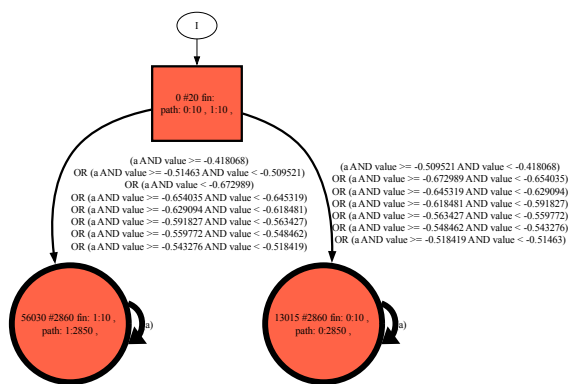


Figure 20: EDFA learned by Variant 1 on the Cofee dataset.

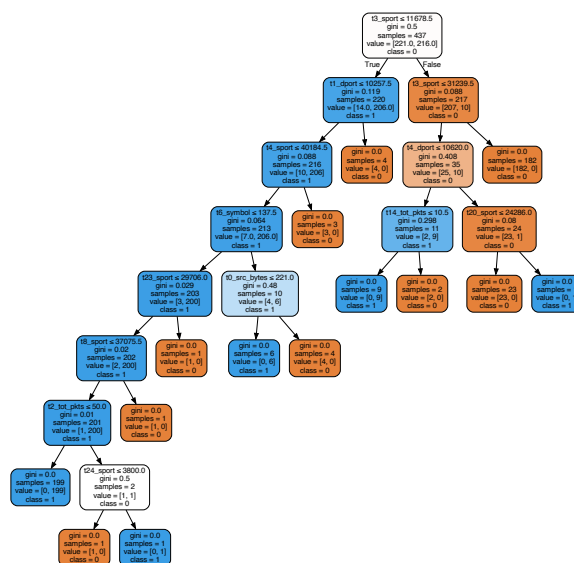


Figure 10: CART decision tree for the CTU-13 dataset.