TADAM: Learning Timed Automata from Noisy Observations

Lénaïg Cornanguer[1], Pierre-François Gimenez[2]

Abstract

Timed Automata (TA) are formal models capable of representing regular languages with timing constraints, making them well-suited for modeling systems where behavior is driven by events occurring over time. Most existing work on TA learning relies on active learning, where access to a teacher is assumed to answer membership queries and provide counterexamples. While this framework offers strong theoretical guarantees, it is impractical for many real-world applications where such a teacher is unavailable. In contrast, passive learning approaches aim to infer TA solely from sequences accepted by the target automaton. However, current methods struggle to handle noise in the data, such as symbol omissions, insertions, or permutations, often resulting in excessively large and inaccurate automata. In this paper, we introduce TADAM, a novel approach that leverages the Minimum Description Length (MDL) principle to balance model complexity and data fit, allowing it to distinguish between meaningful patterns and noise. We show that TADAM is significantly more robust to noisy data than existing techniques, less prone to overfitting, and produces concise models that can be manually audited. We further demonstrate its practical utility through experiments on real-world tasks, such as network flow classification and anomaly detection.

1 Introduction

The inference of system behavior models from observational data allows gaining insight into complex systems without requiring expert knowledge and a time consuming process. Such models can then be used to automatize various tasks such as monitoring, diagnostics, or scheduling [15, 1]. Timing of events is particularly crucial in many domains, including real-time systems and communication protocols. To capture this aspect, these systems can be modeled using timed automata (TA) [2], a well-established formalism for event sequence data that includes temporal information. For instance, Fig. 1 illustrates an automaton modeling in a simplified way the establishment of a TCP network connection between two devices.

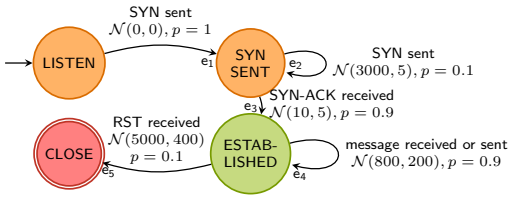

Figure 1: A timed automaton extended with probabilistic parameters as model of a simplified TCP connection.

The current state of the system can be mapped to a *location* in the automaton (LISTEN, SYN SENT, etc.) at a given time. The state transitions triggered by an event are modeled by *transitions* between locations that are labeled with a corresponding *symbol* (SYN sent, SYN-ACK received, etc.). A *guard* on each transition describes the temporal delay associated with an event. For example, SYN messages can be regularly sent from the sender until SYN-ACK message is received, triggering the transition from the location SYN SENT to ESTABLISHED. Such sequences of events occurring during an execution of the system are called *words*.

Several methods have been proposed to learn timed automata from observational data. However, the issue of noise in the data remains unaddressed, despite the fact that available data are often incomplete and noisy, making it harder to infer accurate models. Noise can arise from various sources: limited measurement accuracy, probe configuration errors [8], incorrect labelling, etc. The current methods are therefore unsuited for a large proportion of real-world applications.

A second challenge in inferring timed automata is to find the good level of generalization between two extremes: an automaton with a single location and only self-loop transitions which accepts every possible words, and an automaton with one branch per event sequence which only accepts the learning data.

To address both these challenges, we present an approach based on the Minimum Description Length (MDL) principle [7], a model selection criterion that favors simple models that most efficiently compress the data. By balancing complexity and goodness of fit, the

[1]CISPA Helmholtz Center for Information Security
lenaig.cornanguer@cispa.de
[2]CentraleSupélec, Inria, Univ. Rennes, IRISA
pierre-francois.gimenez@centralesupelec.fr
Both authors contributed equally.

Accepted at SIAM SDM 2025

MDL principle is particularly relevant for noisy data as it discourages overly complex models that might capture random noise rather than true underlying patterns.

This paper presents the following contributions:

(1) An MDL encoding for timed automata (Sec. 4.2);

(2) A word correction strategy for data encoding (Sec. 4.3);

(3) TADAM, the first algorithm to learn TA from noisy data (Sec. 5).

2 Related Work

The vast majority of timed automata learning literature relies on active learning, with access to a teacher that can provide counterexample to an equivalence query or the answer to the membership query of one word [3, 17].

Learning TA solely from observational data (passive learning) has received much less attention due to the complexity of the absence of counter-examples to guide the process. Cornanguer et al. [4] have proposed the TAG algorithm that controls the model generalization level by factorizing the TA on sub-sequence of events of fixed length. Verwer et al. [16] have proposed the RTI+ algorithm which transforms the learned automaton by performing operations that improve the likelihood of the data with the model. Yet, these algorithms are not designed to be robust to noise in the data.

In the broader literature on model inference without temporal consideration, various strategies have been proposed to tackle the challenge of noise. Wiegand, Klakow, and Dietrich [19] proposed a greedy MDL-based approach to learn an event pattern graph. To handle noise, they compute *covers* to indicate how to read the data with a given pattern graph. There is no notion of time and the graph edges are not weighted, making the search problem simpler. Wallner, Aichernig, and Burghard [18] learn Moore machines using SAT solving by looking for the deterministic model that is consistent with the largest proportion of the input sample. In addition to have no consideration of time, their algorithm requires to specify the number of state of the model, and there are no clear model selection criterion, only guidelines.

3 Preliminaries

We first present our modeling formalism based on timed automata, and then introduce the MDL principle.

3.1 Timed Automata Timed automata (TA) [2] are finite-state automata extended with temporal parameters that can model systems governed not only by the occurrence of events but also by time. In this paper, we consider a sub-class of TA, called Real-Time Automata (RTA) [5], with a single type of temporal parameter: the delay between two events.

Definition 1. (Real-Time Automaton) *A real-time automaton is a tuple $\mathcal{A} = (\mathcal{Q}, \Sigma, \mathcal{E}, q_0, \mathcal{F})$ where $\mathcal{Q}$ is a finite set of locations, Σ is a finite set of symbols (or events) called alphabet, $\mathcal{E}$ is a finite set of transitions, $q_0 \in \mathcal{Q}$ is the initial location, and $\mathcal{F}$ is a finite set of accepting locations.*
$\mathcal{E} \subseteq \mathcal{Q} \times \Sigma \times \mathbb{N}^2 \times \mathcal{Q}$ is a finite set of transitions of the form (q, a, g, q') where q and q' are respectively the source location and destination location, $a \in \Sigma$ is a symbol, g is a guard in the form of an interval constraining the delay.

Motivated by the noisy learning environment setup, we introduce a probabilistic variation of the RTA denoted pRTA. A probabilistic transition has a probability to be triggered from its source location and the interval guard is replaced by a probability distribution. The TA displayed in Fig. 1 is a pRTA.

Definition 2. (Probabilistic transition) *A probabilistic transition $e = (q, a, G, p, q')$ is a transition where G is a normal distribution $\mathcal{N}(\mu, \sigma^2)$ and p is the probability $p(e|q)$ of the transition from q.*

We now present the concepts of timed words (informally referred to as event sequences in the introduction) and timed automata language.

Definition 3. (Timed word) *A timed word $w = (a_0, t_0)(a_1, t_1)... \in (\Sigma \times \mathbb{N})^*$ is a finite sequence of symbols and non-decreasing timestamps.*
In the context of pRTA, we can re-write a timed word as a sequence of symbol and delay $w = (a_0, d_0)(a_1, d_1)...$ where $d_0 = 0$ and $d_i = t_i - t_{i-1}$.

A (timed) word w is consistent with an automaton $\mathcal{A}$ (or accepted by it) if there exists a *path* from the initial location to an accepting location, i.e., a sequence of transitions $q_0 \xrightarrow{e_0} q_1 \xrightarrow{e_1} ... \xrightarrow{e_p} q_f$. For a timed automaton $\mathcal{A}$, the *(timed) language* denotes the set of timed words accepted by $\mathcal{A}$. In a RTA, all words in its language have the same importance, while in a pRTA, the probability of occurrence of the timed words is no longer uniform thanks to the probabilistic transitions.

On a final note, to ease the computations, we consider a unique ending symbol to identify the end of the words. In the automaton, the transitions labeled by this ending symbol all lead to a unique final location q_f.

3.2 The MDL Principle The Minimum Description Length (MDL) principle [13, 7] is a model selection criterion that states that the best description for

some observed data is the shortest one. This principle is rooted in algorithmic information theory, where the amount of information conveyed by data or its description can be quantified in bits. In this work, we focus on the two-part MDL, where, among a model class $\mathcal{M}$, the best model $M \in \mathcal{M}$ is the one minimizing $L(M) + L(\mathcal{D}|M)$, with $L(M)$ the length of model M in bits and $L(\mathcal{D}|M)$ the length of data $\mathcal{D}$ encoded with this model M. This description in two-part allows us to balance model complexity with the accuracy of data representation. To apply the MDL principle, an encoding for the data and the model must be defined.

4 MDL for Timed Automata Learning

This section presents how to leverage MDL to learn pRTA from noisy timed words.

4.1 Noise Model The noise in a timed word can take different forms depending on the application. We consider four different noise types, illustrated with an initial timed word $w = (a_0, d_0)(a_1, d_1)(a_2, d_2)$: deletion ($w = (a_0, d_0)(a_2, d_2)$), insertion ($w = (a_0, d_0)(a_3, d_3)(a_1, d_1)(a_2, d_2)$), transposition ($w = (a_0, d_0)(a_2, d_2)(a_1, d_1)$), and symbol repetition ($w = (a_0, d_0)(a_1, d_1)(a_1, d_3)(a_2, d_2)$) (note that we allow the repeated symbol to be associated with a different delay). TADAM can also be parameterised with a subset of noise types depending on the domain's specificity.

4.2 Model Encoding We define an MDL encoding for the pRTA model. The more complex the pRTA, the more expensive it should be to encode. The description length of a pRTA $\mathcal{A}$ corresponds to the sum of the cost of encoding its locations, its alphabet, for each transition the source and destination locations, the symbol, the guards' normal distributions parameters, and its initial and accepting locations, and is given by

$$L(\mathcal{A}) = L_{\mathbb{N}}(|\mathcal{Q}|) + L_{\mathbb{N}}(|\Sigma|) + \sum_{e \in \mathcal{E}} \Big(2\log_2(|\mathcal{Q}|) + \log_2(|\Sigma|) + L_{\mathbb{N}}(\lfloor \mu_e \rfloor) + L_{\mathbb{N}}(\lfloor \sigma_e^2 \rfloor) \Big) + 2\log_2(|\mathcal{Q}|) \,,$$

where $L_{\mathbb{N}}$ is the MDL-optimal encoding for integers defined by [14].

4.3 Data Encoding We now define how to encode the data $\mathcal{D}$ given an automaton $\mathcal{A}$. Due to the presence of noise, many words in $\mathcal{D}$ might be not accepted by the automaton even if $\mathcal{A}$ is equivalent to the true generating process of $\mathcal{D}$. For this reason, we do not use the classical data encoding that relies on the word probability [7], because all the noisy words rejected by $\mathcal{A}$ would have a null probability and would have to be encoded explicitly. Instead, we integrate into the encoding the notion that such words could be accepted by $\mathcal{A}$ with some light changes. To do so, we associate the timed words with an *edit operation sequence* that describes how to read and correct them such that they are accepted by the pRTA. We propose several edit operations that mirror the noise types presented in Sec. 4.1:

- Add (ad): add a pair (a, d) to the word;
- Skip (sk): ignore a pair (a, d);
- Transpose (tr): permute two consecutive pairs;
- Deduplicate (de): remove a pair (a, d) following a pair with an identical symbol
- Follow (fo): read the pair (a, d) as it is.

Note that the follow edit operation allows us to handle the already accepted words in the same way as the words needing to be corrected.

More formally, an edit operation sequence R is a sequence of tuple in $\{add, skip, transpose, deduplicate, follow\} \times \mathcal{E} \cup \{\epsilon\} \times \mathbb{N}$, with ϵ a dummy transition that is not in $\mathcal{E}$, used when no transtition is to be associated.

For example, given the automaton in Fig. 1, the timed word (SYN sent, 0),(ACK sent, 500),(SYN sent, 2000),(RST received, 6500) is not recognized. We can associate it with the following edit operation sequence R: (follow, e_1, 0), (skip, ϵ, 500), (follow, e_2, 2000), (add, e_3, 10), (follow, e_5, 6500), resulting in the corrected timed word (SYN sent, 0),(SYN sent, 2000),(SYN-ACK received, 10),(RST received, 6500) that is accepted. Note that there can be multiple edit operation sequences possible for a same word. We propose in Sec. 5.1 an algorithm to find the edit operation sequence associated with a timed word that minimizes its encoding cost.

Let us now define the data encoding based on this notion. We denote $\mathcal{D}_r$ the subset of words in $\mathcal{D}$ that are accepted by a pRTA $\mathcal{A}$ or for which there exists an edit operation sequence R that results in a word accepted by $\mathcal{A}$. Encoding $\mathcal{D}_r$ with $\mathcal{A}$ consists in encoding R, the flattened edit operation sequence over all words. Let R_O with $O \subseteq \{\text{fo}, \text{sk}, \text{de}, \text{tr}, \text{ad}\}$ denotes a subsequence of R consisting of all tuples (o, e, d) where $o \in O$. We define the description length of $\mathcal{D}_r$ given $\mathcal{A}$ as

$$\begin{aligned} L(\mathcal{D}_r|\mathcal{A}) = & \sum_{(o,e,d) \in R} -\log_2(p(o)) \\ & + \sum_{(o,e,d) \in R_{\{\text{tr,fo,ad}\}}} -\log_2 p(e \mid q_s(e)) \\ & + \sum_{(o,e,d) \in R_{\{\text{tr,fo,de}\}}} -\log_2 p(d \mid e) \\ & + \sum_{(o,e,d) \in R_{\{\text{sk}\}}} (L_{\mathbb{N}}(d) + \log_2(|\Sigma|)) \end{aligned}$$

where $q_s(e)$ is the source location of e and $\log_2(p(o))$ is a fixed cost for each edit operation. The second sum term corresponds to the probability of the triggered transition given its source location, the third sum term corresponds to the probability of the delay given the triggered edge, and the fourth sum term corresponds to the explicit encoding of the skipped pairs using an uniform distribution to encode the symbol and a universal encoding for the delay. Remark that we only encode information necessary to retrieve the original word: for example, there is no need to encode the delay of a pair introduced by the "add" operation.

This formula can be partially rewritten using the notion of entropy, which will be useful in a following section. To this end, we introduce E, the random variable denoting which edge is triggered. We obtain

$$\sum_{\substack{(o,e,d)\in \\ R_{\{\mathrm{tr,fo,ad}\}}}} -\log(p(e \mid q_s(e))) = |R_{\{\mathrm{tr,fo,ad}\}}| \cdot H(E \mid q_s(E)) \ .$$

Similarly, we can show that

$$\sum_{(o,e,d)\in R_{\{\mathrm{tr,fo,de}\}}} -\log(p(d \mid e)) = |R_{\{\mathrm{tr,fo,de}\}}| \cdot H(D \mid E) \ ,$$

where D denotes the random variable of the delay, and that encoding the edit operation corresponds to

$$\sum_{(o,e,d)\in R} -\log_2(p(o)) = |R| \cdot H(O) \ .$$

As there may also exist timed words that cannot be associated with any edit operation sequence, we must also define an encoding for the uncorrectable words $\mathcal{D}_u \in \mathcal{D}$. This situation can happen when the noise model is narrowed and some of the edit operations are forbidden. Similarly as in the case of the skip operation, we define the description length of $\mathcal{D}_u$ given $\mathcal{A}$ as

$$L(\mathcal{D}_u|\mathcal{A}) = \sum_{(s,d)\in\mathcal{D}_u} (L_{\mathbb{N}}(d) + log_2(|\Sigma|)) \ .$$

Besides, we need to indicate whether a timed word is encoded as corrected or uncorrectable. We use a random variable X_r denoting to which set, $\mathcal{D}_r$ or $\mathcal{D}_u$, a timed word belongs to. The resulting description length of $\mathcal{D}$ given $\mathcal{A}$ is given by

$$L(\mathcal{D}|\mathcal{A}) = L(\mathcal{D}_u|\mathcal{A}) + L(\mathcal{D}_r|\mathcal{A}) + |\mathcal{D}| \cdot H(X_r) \ .$$

5 Algorithm

This section first describes the word correction algorithm and then the actual TA learning algorithm.

5.1 Word Correction Algorithm As explained in Sec. 4, our data encoding relies in some part on a word correction strategy that we present here. Our goal is to find the sequence of edit operations that leads to the minimal description length of the words corrected to be accepted by $\mathcal{A}$. To this end, we adapt the algorithm proposed by Oommen and Loke [11] to compute the optimal string alignment distance between two words to the problem of aligning a timed word with a pRTA (i.e., to the words from its language). Since the main novelty lies in the cost function, we focus on its description here.

Let $\mathcal{A}$ be a pRTA and $w = (w_0, w_1, \ldots, w_k)$ a timed word with $w_l = (a_l, d_l)$. Let $p(w, q, q')$ be the probability of a (sub-)timed word from location q to q', or 0 if no such path exist. We initialize the cost function as $cost(0, q) = \min_{w\in(\Sigma\times\mathbb{N})^*} -\log_2(p(w, q_0, q)) - |w| \cdot \log_2(p(\mathrm{ad}))\ \forall q \in \mathcal{Q}$. It indicates, for empty timed words, the most probable (i.e., with minimal cost) sequence of pairs (a, d) to add to reach q from q_0. Let us define an intermediate cost $cost'(l+1, q)$ to consider the different edit operation, given by the minimum of

$$\begin{cases} cost(l, q') - \log_2(p(\mathrm{fo})) - \log_2(p((w_{l+1}), q', q) \text{ if} \\ \quad \text{transition } q' \to q \text{ is labelled with } s_{l+1} \textit{ (follow)} \\ cost(l, q) - \log_2(p(\mathrm{de})) - \log_2(p(d_{l+1} \mid e_l)) \text{ if} \\ \quad s_{l+1} = s_l, \text{ where } e_l \text{ is the edge triggered by } w_l \\ \quad \textit{(deduplicate)} \\ cost(l-1, q') - \log_2(p(\mathrm{tr})) \\ \quad + \min_{q',q''\in\mathcal{Q}} \big(-\log_2(p((w_{l+1}), q', q'') \\ \quad - \log_2(p((w_l), q'', q))\big) \textit{ (transpose)} \\ cost(l, q) + L_{\mathbb{N}}(d_{l+1}) + \log_2(|\Sigma|) \textit{ (skip)} \end{cases}$$

The cost of reaching a location q from position $l+1$ in the word is given by $cost(l+1, q) = \min_{q'\in\mathcal{Q}}(cost'(l+1, q) + \min_{w\in(\Sigma\times\mathbb{N})^*} -\log_2(p(w, q, q')) - |w| \cdot \log_2(p(\mathrm{ad})))$, i.e., the minimal cost of reaching some intermediate location q' and then to reach location q through additions.

5.2 Transformation Operations Given $\mathcal{D}$, the learning process aims to find the automaton $\mathcal{A}^* \in \mathrm{pRTA}$ such that $\mathcal{A}^* = \arg\min_{\mathcal{A}\in\mathrm{pRTA}} L(\mathcal{A}) + L(\mathcal{D}|\mathcal{A})$. Enumerating all possible automata where each transition is used by at least one word $w \in \mathcal{D}$ is impractical, and allowing some parts of $\mathcal{D}$ to remain unrecognized by the automaton further expands the search space. Therefore, we opt for a greedy algorithm that iteratively transforms the automaton to reduce the MDL cost. We define multiple transformations that can be applied to a given pRTA $\mathcal{A}$. To lighten the notations, we write $\mathcal{E}_{l,out}$ to refer to the outgoing transitions of location l, i.e., the set of transitions $\{(q, a, G, p, q') \in \mathcal{E} | q = l\}$, and similarly $\mathcal{E}_{l,in}$ to refer to the incoming transitions of l.

5.2.1 Location Merging The location merge simplifies $\mathcal{A}$ by merging two locations. The merge of q and q' in $\mathcal{A}$ results in their replacement in $\mathcal{Q}$ by q'' with $\mathcal{E}_{q'',out} = \mathcal{E}_{q,out} \cup \mathcal{E}_{q',out}$ and $\mathcal{E}_{q'',in} = \mathcal{E}_{q,in} \cup \mathcal{E}_{q',in}$.

5.2.2 Subpart Deletion The subpart deletion reduces the size of $\mathcal{A}$ and the timed language it accepts. Let $e = (q, a, G, p, q')$ be a probabilistic transition in $\mathcal{A}$. Deleting e results in $\mathcal{E} \leftarrow \mathcal{E} \setminus \{e\}$ and in the deletion of the subpart of A that is no more reachable from q_0, or from which q_f is no more reachable.

5.2.3 Location Split The location split replaces a location $q \neq q_0$ by a set of new locations Q_{new}, changes the destination of the transitions in $\mathcal{E}_{q,in}$ to locations in Q_{new}, and recreates the transitions in $\mathcal{E}_{q,out}$ with a location in Q_{new} as source whenever it is needed for a timed word, as shown in Fig. 2. This transformation is especially interesting with our MDL-based score because it affects this score very locally, meaning that despite the high number of new possible partitions P (exponential in $\mathcal{E}_{q,in}$), it is straightforward to estimate the gain or loss induced by it. We present in the following how to estimate the gain given a new partition P, and then how to find a good partition P for a given location q.

We start by showing how we can estimate the MDL data gain brought from a split. Let $\mathcal{A}_1$ be a pRTA and $\mathcal{A}_2$ the pRTA resulting from the split of q in $\mathcal{A}_1$. Let $f : \mathcal{E} \to \mathcal{Q}$ be the destination assignment function that takes a transition from $\mathcal{A}_1$ and return its destination location in $\mathcal{A}_2$. This function allows us to map the transitions in $\mathcal{E}_{q,in}$ to their new destination in Q_{new} after the split. To make our notation clearer in the following, we provide an example in Fig. 2, but the explanation holds for any $e_i \in \mathcal{E}_{q,in}$ (resp. $e_j \in \mathcal{E}_{q,out}$) with source location q' (resp. destination location q''). By construction, the transitions $q' \xrightarrow{e_i} q \xrightarrow{e_j} q''$ are

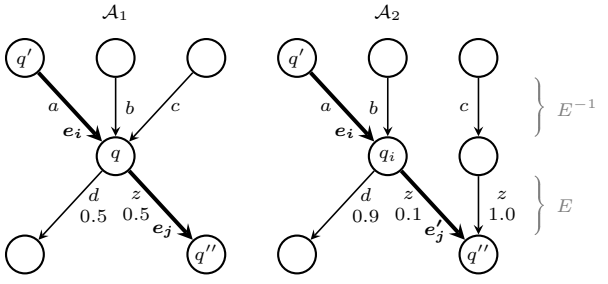

Figure 2: An example of the split of a location q ($\mathcal{A}_1$ is before and $\mathcal{A}_2$ after). Guards are omitted.

triggered in $\mathcal{A}_1$ if and only if the transitions $q' \xrightarrow{e_i} f(e_i) \xrightarrow{e'_j} q''$ ($f(e_i)$ corresponding to q_i) are triggered in $\mathcal{A}_2$. This allow us to express the probability of e_j in $\mathcal{A}_1$ using the knowledge in $\mathcal{A}_2$, which results in $p_{\mathcal{A}_1}(e_j \mid f(e^{-1})) = p_{\mathcal{A}_2}(e'_j \mid q_i)$, where e^{-1} denotes the transition triggered before e_j in $\mathcal{A}_2$. To express this in terms of entropy, we introduce the random variable E (denoting the triggered transition outgoing from a location in Q_{new}) and E^{-1} (the transition triggered before E). Then, $H_{\mathcal{A}_1}(E \mid q_s(E), f(E^{-1})) = H_{\mathcal{A}_2}(E \mid q_s(E))$ because there exists a trivial bijection between $q_s(E), f(E^{-1})$ in $\mathcal{A}_1$ and $q_s(E)$ in $\mathcal{A}_2$. This way, we can estimate the impact of the split operation on the cost of encoding the edges as

$$\begin{aligned}
&|R_{\{\text{tr,fo}\}}|(H_{\mathcal{A}_2}(E \mid q_s(E)) - H_{\mathcal{A}_1}(E \mid q_s(E))) \\
&= |R^q_{\{\text{tr,fo}\}}|(H_{\mathcal{A}_1}(E \mid q_s(E) = q, f(E^{-1})) \\
&\qquad - H_{\mathcal{A}_1}(E \mid q_s(E) = q)) \\
&= -|R^q_{\{\text{tr,fo}\}}| I_{\mathcal{A}_1}(E; f(E^{-1}) \mid q_s(E) = q) \leq 0
\end{aligned}$$

where $|R^q_{\{\text{tr,fo}\}}|$ is the number of transpose and follow operations using an edge whose source is q, and $I_{\mathcal{A}_1}(E; f(E^{-1}) \mid q_s(E))$ is the mutual information between E and $f(E^{-1})$ given $q_s(E)$ in $\mathcal{A}_1$.

The intuition behind this formula can be explained with Fig. 2. Assume that q in $\mathcal{A}_1$ can be entered with either a, b, or c, with equal proportion, and that the next symbol is either z or d in equal proportion, which makes the encoding cost of the next symbol high. Consider now the split of this location, revealing that c is always followed by z. Now, the cost of z following c is zero because it must happen with probability equal to 1. From the location reached by a and b, the encoding cost of d is much lower in $\mathcal{A}_2$ than it was in $\mathcal{A}_1$, leading to an overall lower encoding cost. This is captured in the formula by the mutual information that is high between two consecutive symbols. The same reasoning can show that $|R_{\{\text{tr,fo,de}\}}| \cdot (H_{\mathcal{A}_2}(D \mid E) - H_{\mathcal{A}_1}(D \mid E)) = -|R^q_{\{\text{tr,fo,de}\}}| \cdot I_{\mathcal{A}_1}(D; f(E^{-1}) \mid E, q_s(E) = q)$.

Location split comes however at a cost in model description because it adds new locations and transitions. Therefore, there is an optimal trade-off in the number of new locations. We propose to find a good partition with a greedy method presented in Algorithm 1. In practice, we first start by splitting edges in $\mathcal{E}_{q,in}$ by using a Gaussian mixture model on the delays associated with them, creating one edge per Gaussian mixture model, to allow for more refined partitioning.

5.3 Learning Algorithm We now present TADAM (*Timed Automata Discovery Applying MDL*), an algorithm that integrates the transformation operations, the word correction strategy, and the MDL cost computa-

Algorithm 1 Split search

Require: A location q, a timed automaton $\mathcal{A}$
Ensure: A partition of $\mathcal{E}_{q,in}$

split incoming edges of q using Gaussian mixtures
$P \leftarrow [\mathcal{E}_{q,in}]$; $mean\_edge\_cost \leftarrow L(\mathcal{A})/|\mathcal{E}|$
$best\_partition \leftarrow P$
while $|P[0]| > 1$ **do**
  for all edge e of $P[0]$ **do**
    for $i = 0$ **to** $|P|$ **do**
      $P' \leftarrow P$
      remove e from $P'[0]$
      if i = 0 **then**
        append new set $\{e\}$ to P'
      else
        add e to $P'[i]$
      $score \leftarrow |R^q_{\{tr,fo\}}| \cdot I(E; f_{P'}(E^{-1}) \mid q_s(E)) - (|P'| - 1) \times |\mathcal{E}_{q,out}| \times mean\_edge\_cost$
      if $score > score(best\_partition)$ **then**
        $best\_partition \leftarrow P'$
  if $P = best\_partition$ **then**
    break
  $P \leftarrow best\_partition$
return $best\_partition$

Algorithm 2 TADAM

Require: Input sample of timed sequences $\mathcal{D}$
Ensure: A pRTA $\widehat{\mathcal{A}}$ partially consistent with $\mathcal{D}$

$\widehat{\mathcal{A}} \leftarrow$ initTA($\mathcal{D}$)
repeat
  $candidates \leftarrow \emptyset$
  for all operation **do**
    for all target in $\widehat{\mathcal{A}}$ **do**
      $\mathcal{A} \leftarrow$ transform($\widehat{\mathcal{A}}$, operation, target)
      $candidates \leftarrow candidates \cup \{\mathcal{A}\}$
  $\mathcal{A}' \leftarrow \arg\min_{\mathcal{A} \in \text{candidates}} L(\mathcal{A}) + L(\mathcal{D}|\mathcal{A})$
  $gain \leftarrow L(\widehat{\mathcal{A}}) + L(\mathcal{D}|\widehat{\mathcal{A}})$ - $L(\mathcal{A}') - L(\mathcal{D}|\mathcal{A}')$
  if $gain > 0$ **then**
    $\widehat{\mathcal{A}} \leftarrow \mathcal{A}'$
until $gain \leq 0$
return $\widehat{\mathcal{A}}$

tion. We provide its pseudo-code in Alg. 2. An open-source implementation is available[1].

The first step is the creation of an initial automaton accepting all words $w \in \mathcal{D}$. Most passive learners of TA start by a tree-shaped automaton, called prefix tree, where each word correspond to a branch [16, 4, 12]. We propose a *Markov initialization*, where each symbol in Σ is associated with a location in the initial automaton, and there is a transition between two locations if their corresponding symbols ever appear consecutively in $\mathcal{D}$.

From this initial automaton, we iteratively perform transformation operations to improve the MDL score. At each iteration, we test all possible transformation operations (merge, deletion, and split) at every position in the automaton. The words are corrected given the new model, and then the MDL score is computed. The transformed automaton leading to the greatest gain in MDL is kept as new base model for the next iteration. If no operation improves the MDL score, the algorithm stops and outputs the automaton with maximal score.

6 Experiments

Throughout the experiments, we are interested in the following questions: to which extend is TADAM robust to noise in the data; how well does it compare to state-of-the-art TA learners; how competitive is it to other methods on real-world problems. We address the two first questions with an experiment on synthetic data. We then present applications on real data with classification and anomaly detection tasks.

6.1 Experimental Setup

We first describe how we generated synthetic data, and then present the setup for the real-world experiments.

6.1.1 Synthetic Data

We generated 35 random target timed automata as data generating process. To ensure a diversity in the target automata, we considered different values of number of locations (from 5 to 50), of alphabet size (5 to 20), and of outdegree (from 1.25 to 2). For each target automaton, we generated a set of timed words. We then injected noise in the timed words, affecting up to 50% of the pairs (a, d) according to the noise model defined in Sec. 4.1 to constitute the training sample $\mathcal{D}_{train}$ for TADAM and its competitors. We considered the following levels of noise n: 0, 0.001, 0.005, 0.01, 0.025, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4, and 0.5. We set the default training sample size to 500, nevertheless, we also considered different training sample sizes (from 100 to 1000) for 5 of the target automata (with the following default characteristics: 14 locations, 10 symbols, and an outdegree of 1.25). For the evaluation, we also generated test sets of timed words $\mathcal{D}^+_{test}$ and $\mathcal{D}^-_{test}$ with, respectively, words accepted by the target automaton, and words from the target but with added noise and thus not accepted anymore. Note that words from $\mathcal{D}^-_{test}$ may also belong to $\mathcal{D}_{train}$ where $n \geq 0$.

To evaluate the 600 timed automata learned by the TADAM (and its competitors), we computed the recall, precision, and F1-score using $\mathcal{D}_{test}$. We also computed

[1] https://github.com/Fos-R/TADAM

the Jaccard distance between the untimed language of the target and the learned automata, which measures their similarity without taking into account the word probabilities. We also computed the Jensen-Shannon divergence between the untimed word distribution in the languages of the target and learned automata.

We also conducted an ablation study on TADAM using the automata generated with default values. We tested different initialization strategies, and removed the transformation operations one by one.

6.1.2 Classification We performed a classification task on network traffic, based on the ISCXVPN2016 dataset [6]. In the raw network captures, there are four classes of network traffic: chat, email, streaming, and VoIP. A usual method to classify such data relies on network flow descriptive features, such as number of packets, data rate, etc. We relied on the NTLFlowLyzer software to extract these features. Due to issues with it, we had to leave out some of the original dataset classes. We split the traffic into a training set (with 80% of data) and testing set (with 20% of data). The symbols are formed from the TCP flags, the message directions, and an indicator of the payload, leading to an alphabet with 23 symbols. We created one word per communication flow between two IP addresses, each packet corresponding to a pair (symbol, payload size).

We used a Random Forest on the network flow features as a domain-specific baseline. For the TA learning methods, TADAM, TAG and RTI+, we learned one automaton per class. To predict a class with an automaton, we consider the most likely automaton given a timed word.

6.1.3 Anomaly Detection We conducted an anomaly detection experiment in a collection of HDFS message logs [20]. The HDFS (Hadoop Distributed File System) is a storage system designed to distribute and manage large datasets across multiple machines in a Hadoop cluster. The dataset contains 29 different kind of messages (received data, failed to transfer, etc.). To obtain timed words, we used the message type as symbol and the delay between two messages as time value. We randomly selected 1,000 traces labelled as normal among the 575,061 available traces to constitute the training sample $\mathcal{D}$. Using the learned timed automaton, we then computed the probability of the remaining traces (557,223 normal and 16,838 abnormal) to perform the anomaly detection.

6.2 Validation on Synthetic Data We report in Table 1 the average F1-score, precision, recall and execution time over all data. TADAM significantly outperforms TAG and RTI+ by reaching a high recall (the TA accept most true words from the target model) while keeping a high precision (they reject most words where noise was injected). These performances come at a price of a longer execution time (TADAM and TAG are implemented in Python while RTI+ in C++).

Learner	F1	Precision	Recall	Time (s)
TADAM	**0.884**	**0.849**	**0.938**	1649
TAG	0.803	0.756	0.878	267
RTI+	0.783	0.837	0.763	**0.4**

Table 1: Global learning performance of our method and its competitors.

6.2.1 Noise Robustness We show the impact of the noise ratio on the learned models in Fig. 3. We provide the full results for the other metrics in the appendix. While TAG and RTI+ are immediately impacted by the presence of noise even in a small proportion, TADAM shows more robustness. The Jaccard distance computed between the untimed languages of the learned model and the target increases more progressively for TADAM than its competitors, showing that it has successfully rejected noisy words. The number of edges, which is an indicator of the size of the learned models, remains small. The recall drop of TAG and RTI+ is a symptom of their inability to generalize over the available learning data due to the noise. For TADAM, there could also be a risk to recognize less true words due to inopportune deletions. The empirical

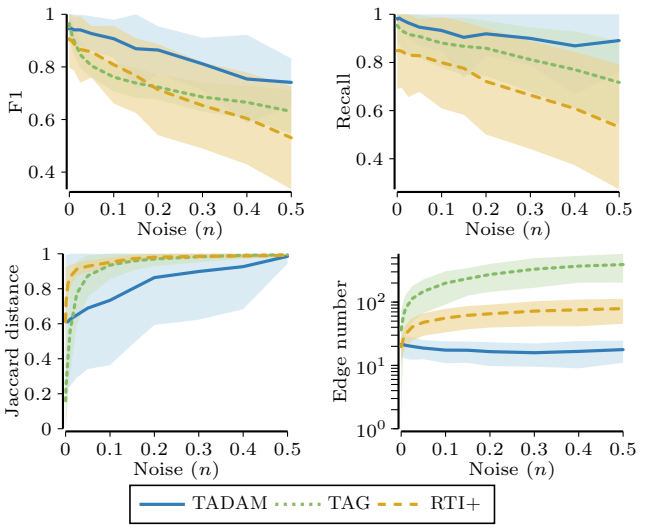

Figure 3: Impact of the noise proportion in the training data on the learned models.

results suggests that this phenomenon is limited.

6.2.2 Ablation Study We tested different initialization strategies: the tree-shaped prefix automaton, the universal automaton, or our Markov initialization. The results confirm the latter is the most appropriate for TADAM. When initialized with the prefix automaton, the runtime quickly skyrockets due to the number of operations to test and of merges needed. The universal automaton is not ideal either, because splitting this single location becomes increasingly complicated as n grows, due to the lack of clear structure in the data.

Running TADAM by removing the transformation operations one by one shows that the most important is the deletion. It is not surprising as this operation brings robustness to noise and induces the word correction process. The other operations appear less critical as our initialization ensures a compact automaton with high mutual information between successive symbols.

6.3 Real-world Applications We now present the results of using the learned timed automata to classify network data and detect anomalies.

6.3.1 Classification We report the classification results in Table 2. TADAM performs slightly better than TAG, though both fall short compared to the domain-specific baseline that achieved a F1-score of 0.720. As in the previous experiments, RTI+ shows significantly lower performance, close to random guess (that would have about 20% accuracy). The feature importance analysis from the Random Forest reveals that high-level metrics, such as average rate of data or the number of certain events, are important for classification. Automata lack the capacity to express this kind of metrics. Finally, TADAM exhibits very little overfitting, achieving similar results on train set and test set, unlike TAG.

	Train set	**Test set**	
Learner	**Accuracy**	**Accuracy**	**F1**
TADAM	0.607	**0.617**	**0.494**
TAG	**0.672**	0.585	0.486
RTI+	0.260	0.255	0.197

Table 2: *Classification performance of TA learners on ISCXVPN2016.* We report the accuracy on the train and test sets, and the mean F1-score on test set.

6.3.2 Anomaly Detection We present the results on the anomaly detection task in Table 3. Although we do not achieve the performance levels of state-of-the-art techniques for log-based anomaly detection, in particular deep learning techniques such as CNN or LogRobust for which F1-score over 0.98 are reported [9], our results are promising. Indeed, the rate of false alarms (FPR) of our learned model is low and its rate of detection (TPR) very high. The learned automaton still demonstrates a strong ability to detect anomalies, despite that learning data was both noisy and limited, and that no adaptation was made for this task. The advantage of the noise robustness becomes evident when we compare our performances with the other TA learners. We also learned an Hidden Markov Model (HMM) as this family of probabilistic models is also used for log structure learning and anomaly detection [10]. However, HMMs clearly lack the expressiveness needed for effective discrimination in this task.

Learner	**AU-ROC**	**TPR**	**FPR**	**F1**
TADAM	**0.982**	0.998	**0.025**	**0.705**
TAG	0.891	**1**	0.142	0.298
RTI+	0.790	**1**	0.292	0.171
HMM	0.608	0.640	0.085	0.288

Table 3: *Anomaly detection performance on HDFS_v1 dataset.* We report the TPR, FPR and F1-score for the threshold maximizing TPR-FPR.

7 Conclusion

We present TADAM, a TA learning algorithm from noisy observational data. Traditional passive TA learners struggle with noise, as they attempt to construct models that strictly align with all observations, resulting in overly complex models with poor generalization. We address this issue by introducing a data correction strategy that selects the most appropriate edit operations to modify a timed word such that it is accepted by a given timed automaton. Based on this strategy, we propose a two-part MDL encoding and a greedy algorithm to learn a TA by balancing the complexity of the model with the fit to noisy data. Experiments on synthetic data demonstrate that TADAM is significantly more robust to noise than existing methods, producing compact and interpretable models that remain faithful to the true data generation process. In addition, real-world experiments highlight the practical utility of the learned automata for tasks such as classification and anomaly detection. This work open new application perspectives for timed automata that can now be more easily inferred from real data. This learning approach could also be extended to other formalisms, such as pushdown automata that can model different phenomena.

8 Acknowledgment

This work has been partially supported by Inria under the SecGen collaboration between Inria, CentraleSupélec and CISPA Helmholtz Center for Information Security.

References

[1] Y. Abdeddaïm, E. Asarin, and O. Maler, *Scheduling with timed automata*, Theoretical Computer Science, 354 (2006), pp. 272–300. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003).

[2] R. Alur and D. L. Dill, *A theory of timed automata*, Theoretical Computer Science, 126 (1994), pp. 183–235.

[3] J. An, M. Chen, B. Zhan, N. Zhan, and M. Zhang, *Learning one-clock timed automata*, in International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2020, pp. 444–462.

[4] L. Cornanguer, C. Largouët, L. Rozé, and A. Termier, *TAG: Learning timed automata from logs*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 3949–3958. Issue: 4.

[5] C. Dima, *Real-Time Automata*, Journal of Automata, Languages and Combinatorics, 6 (2001), pp. 3–24.

[6] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, *Characterization of encrypted and vpn traffic using time-related*, in Proceedings of the 2nd international conference on information systems security and privacy (ICISSP), 2016, pp. 407–414.

[7] P. Grünwald, *The Minimum Description Length Principle*, MIT Press, 2007.

[8] M. Lanvin, P.-F. Gimenez, Y. Han, F. Majorczyk, L. Mé, and E. Totel, *Errors in the cicids2017 dataset and the significant differences in detection performances it makes*, in International Conference on Risks and Security of Internet and Systems, Springer, 2022, pp. 18–33.

[9] V.-H. Le and H. Zhang, *Log-based anomaly detection with deep learning: how far are we?*, in Proceedings of the 44th International Conference on Software Engineering, ICSE '22, New York, NY, USA, 2022, Association for Computing Machinery, p. 1356–1367.

[10] B. Mor, S. Garhwal, and A. Kumar, *A systematic review of hidden markov models and their applications*, Archives of computational methods in engineering, 28 (2021), pp. 1429–1448.

[11] B. J. Oommen and R. K. Loke, *Pattern recognition of strings with substitutions, insertions, deletions and generalized transpositions*, Pattern Recognition, 30 (1997), pp. 789–800.

[12] F. Pastore, D. Micucci, and L. Mariani, *Timed k-tail: Automatic inference of timed automata*, in IEEE International conference on software testing, verification and validation (ICST), 2017, pp. 401–411.

[13] J. Rissanen, *Modeling by shortest data description*, 14 (1978), pp. 465–471.

[14] ——, *A universal prior for integers and estimation by minimum description length*, 11 (1983), pp. 416–431.

[15] S. Tripakis, *Fault diagnosis for timed automata*, in Formal Techniques in Real-Time and Fault-Tolerant Systems, W. Damm and E. R. Olderog, eds., Berlin, Heidelberg, 2002, Springer Berlin Heidelberg, pp. 205–221.

[16] S. Verwer, M. de Weerdt, and C. Witteveen, *A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data*, in Grammatical Inference: Theoretical Results and Applications: 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings 10, Springer, 2010, pp. 203–216.

[17] M. Waga, *Active learning of deterministic timed automata with myhill-nerode style characterization*, in International Conference on Computer Aided Verification, Springer, 2023, pp. 3–26.

[18] F. Wallner, B. K. Aichernig, and C. Burghard, *It's not a feature, it's a bug: Fault-tolerant model mining from noisy data*, in Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ICSE '24, New York, NY, USA, 2024, Association for Computing Machinery.

[19] B. Wiegand, D. Klakow, and J. Vreeken, *Mining Easily Understandable Models from Complex Event Logs*, in Proceedings of the 2021 SIAM International Conference on Data Mining (SDM), 2021, pp. 244 – 252.

[20] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, *Detecting large-scale system problems by mining console logs*, in Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, SOSP '09, New York, NY, USA, 2009, Association for Computing Machinery, p. 117–132.

A Proofs

In Section 4.3, we state that:

$$\sum_{\substack{(o,e,d)\in \\ R_{\{\text{tr,fo,ad}\}}}} -\log(p(e \mid q_s(e))) = |R_{\{\text{tr,fo,ad}\}}| \cdot H(E \mid q_s(E))$$

Here is the proof of that claim:

$$\begin{aligned}
\sum_{(o,e,d)\in R_{\{\text{tr,fo,ad}\}}} -\log(p(e \mid q_s(e))) &= -\sum_{e'\in\mathcal{E}} \sum_{(o,e,d)\in R_{\{\text{tr,fo,ad}\}}} \mathbb{1}[e'=e]\log(p(e' \mid q_s(e'))) \\
&= -\sum_{e'\in\mathcal{E}} \log(p(e' \mid q_s(e'))) \sum_{(o,e,d)\in R_{\{\text{tr,fo,ad}\}}} \mathbb{1}[e'=e] \\
&= -\sum_{e'\in\mathcal{E}} \log(p(e' \mid q_s(e')))|R_{\{\text{tr,fo,ad}\}}|p(e') \\
&= -\sum_{e'\in\mathcal{E}} \log(p(e' \mid q_s(e')))|R_{\{\text{tr,fo,ad}\}}|\frac{p(q_s(e'),e')}{p(q_s(e') \mid e')} \\
&= -\sum_{e'\in\mathcal{E}} \log(p(e' \mid q_s(e')))|R_{\{\text{tr,fo,ad}\}}|p(q_s(e'),e') \\
&= |R_{\{\text{tr,fo,ad}\}}|H(E \mid q_s(E))
\end{aligned}$$

In Section 5.2.3, we state that:

$$|R_{\{\text{tr,fo,de}\}}| \cdot (H_{\mathcal{A}_2}(D \mid E) - H_{\mathcal{A}_1}(D \mid E)) = -|R^q_{\{\text{tr,fo,de}\}}| \cdot I_{\mathcal{A}_1}(D; f(E^{-1}) \mid E)$$

Here is the proof of that claim:

$$\begin{aligned}
|R_{\{\text{tr,fo,de}\}}| \cdot (H_{\mathcal{A}_2}(D \mid E) - H_{\mathcal{A}_1}(D \mid E)) &= |R_{\{\text{tr,fo,de}\}}| \cdot (H_{\mathcal{A}_2}(D) - I_{\mathcal{A}_2}(D;E) - H_{\mathcal{A}_1}(D) + I_{\mathcal{A}_2}(D;E)) \\
&= |R_{\{\text{tr,fo,de}\}}| \cdot (I_{\mathcal{A}_1}(D;E) - I_{\mathcal{A}_2}(D;E)) \\
&= |R^q_{\{\text{tr,fo,de}\}}| \cdot (I_{\mathcal{A}_1}(D;E|q_s(E)=q) - I_{\mathcal{A}_2}(D;E|q_s(E)=q)) \\
&= |R^q_{\{\text{tr,fo,de}\}}| \cdot (I_{\mathcal{A}_1}(D;E|q_s(E)=q) - I_{\mathcal{A}_1}(D;E,f(E^{-1})|q_s(E)=q)) \\
&= -|R^q_{\{\text{tr,fo,de}\}}| \cdot I_{\mathcal{A}_1}(D;f(E^{-1})|E,q_s(E)=q)
\end{aligned}$$

We exploit the fact that $H_{\mathcal{A}_1}(D) = H_{\mathcal{A}_2}(D)$ (because the definitions of $\mathcal{A}_1$ and $\mathcal{A}_2$ do not change the observation of D). The passage from line 2 to 3 is motivated by the fact that $I_{\mathcal{A}_1}(D;E) = I_{\mathcal{A}_2}(D;E)$ for all sequences of symbols and delays except those whose source locations of the triggered edges are q, because $\mathcal{A}_1$ and $\mathcal{A}_2$ only differ for these edges.

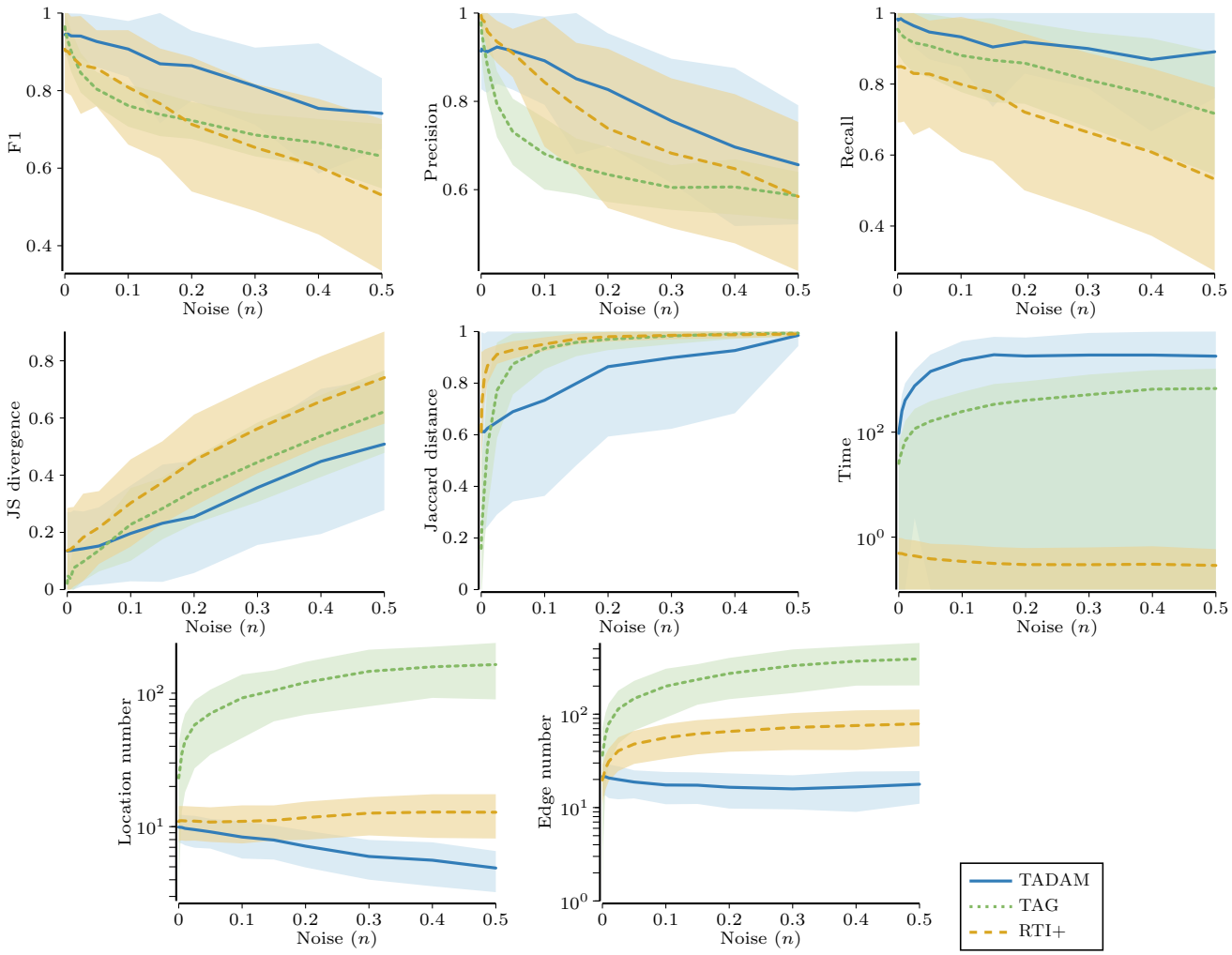

Figure 4: Full results of the synthetic data experiment showing the impact of the noise proportion in the training data on TADAM and its competitors.

B Synthetic Data Experiment

B.1 Robustness to Noise We present in Fig. 4 the full results of the synthetic data experiment.

B.2 Ablation study We first present in Fig. 5 the experimental results for TADAM with different initialization strategies. We either used the classical tree-shaped prefix tree strategy, the universal automaton (a single location with one self-transition per symbol in Σ), or our Markov initialization (one location per symbol). When initialized with the prefix automaton, the runtime quickly skyrockets. From a noise proportion n of 0.025, it starts to exceed 6 hours (we stopped the learning process after these 21600 seconds) due to the number of operations to test and of merges needed. There is also an increased risk of inopportune deletions because deleting a whole subtree will greatly reduce the model cost and it might exceed the increase in data cost on small datasets. In one of the cases, it happened from $n = 0.05$. When initialized with the universal automaton, the only possible operations are the deletion of a transition (improbable at this stage as it would affect all the words have the corresponding symbol), or a split. As the noise proportion rises, less splits appear to be interesting, and no transformation is performed. From $n = 0.2$, the performances dropped in comparison to our proposed initialization. In conclusion, the automaton with a single location per symbol seems to be the best initialization strategy for TADAM.

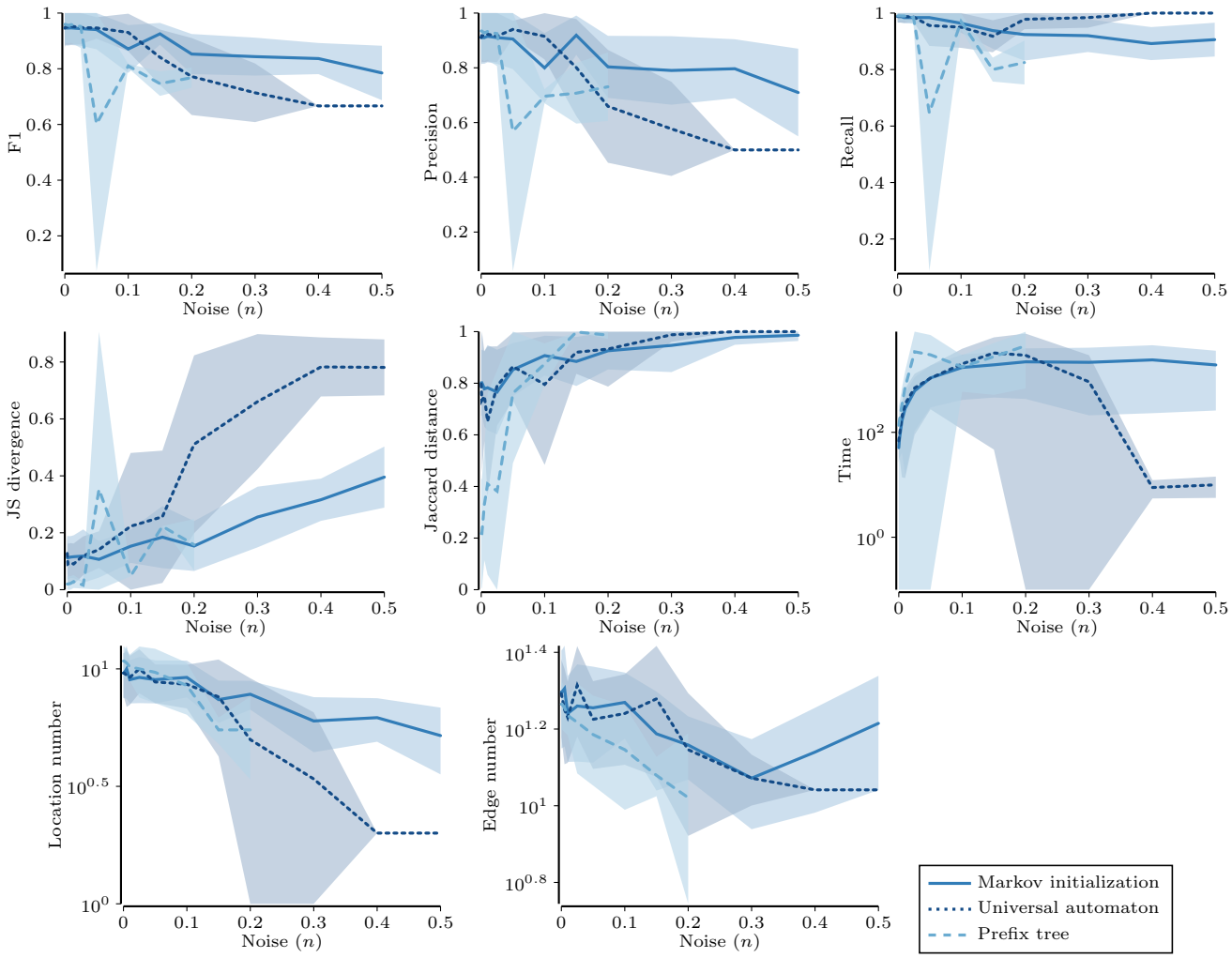

Figure 5: Results of the ablation study on TADAM's transformation operation. The prefix tree initialization caused the learning process to fail due to a learning time exceeding 6 hours (failed cases: 0/20 for $n \in [0, 0.01]$, 1/5 for $n = 0.025$, 2/5 for $n = 0.05$, 9/15 for $n \in [0.1, 0.2]$, 15/15 for $n \geq 0.3$).

We also run TADAM without the possibility to perform either merges, splits, or deletions and present the results in Fig 6. Without the merges, the performances in terms of word acceptance are not significantly different. However, the learned automata are bigger (up to twice as much edges), resulting in a higher runtime (more transformation targets). Without deletions, TADAM unsurprisingly looses its robustness to noise. The average F1-score falls below 0.7 from $n = 0.05$. Without splits, we get slightly worst results. The low difference might be explained by the moderate complexity of the target automata.

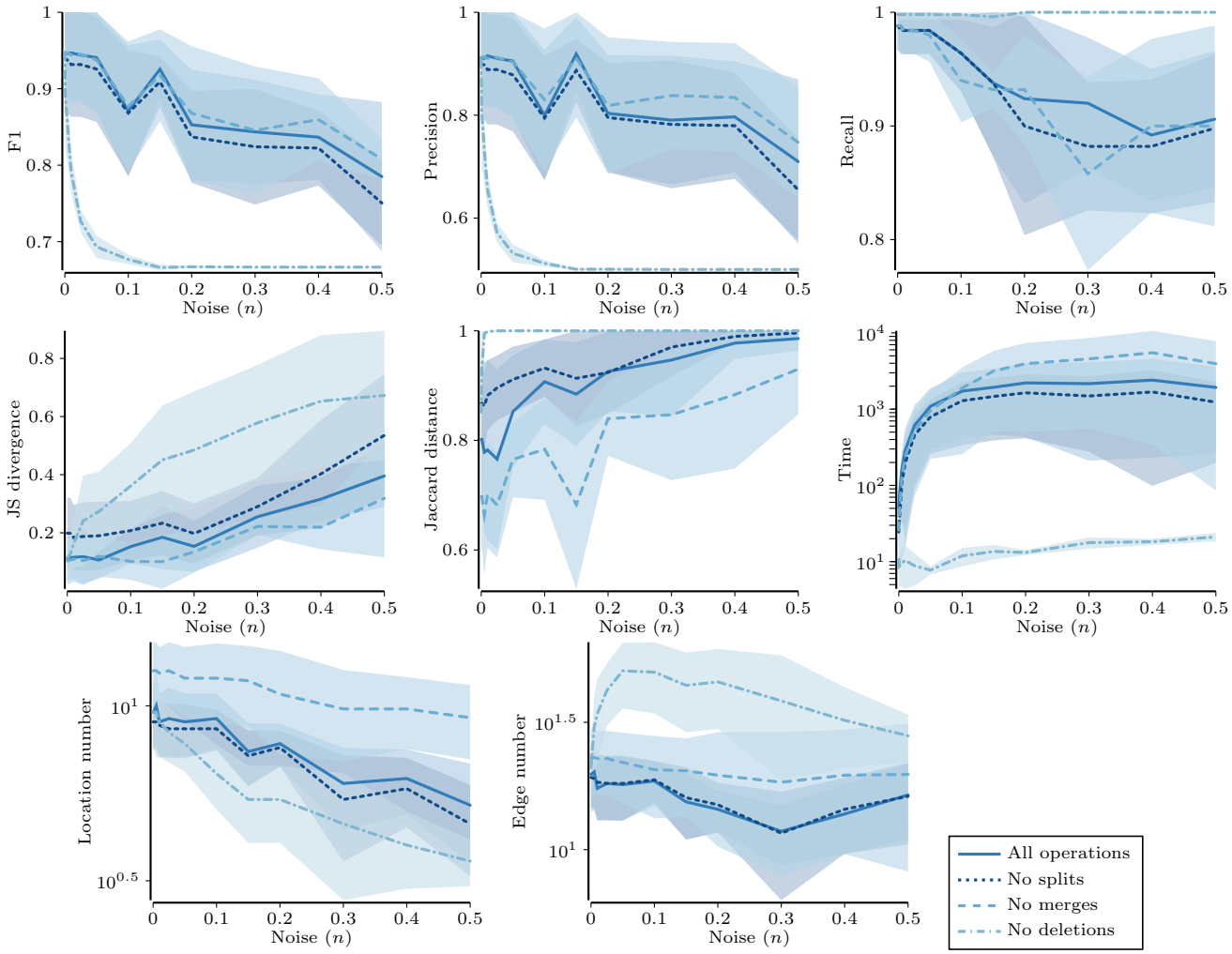

Figure 6: Results of the ablation study on TADAM's transformation operation.