TAG: Learning Timed Automata from Logs

Lénaïg Cornanguer, Christine Largouët, Laurence Rozé, Alexandre Termier

TAG: Learning Timed Automata from Logs

Lénaïg Cornanguer[1], Christine Largouët[2], Laurence Rozé[3], Alexandre Termier[4]

[1]Inria, Univ Rennes, CNRS, IRISA
[2]Institut Agro, Univ Rennes, Inria, CNRS, IRISA
[3]Univ Rennes, INSA Rennes, CNRS, Inria, IRISA
[4]Univ Rennes, Inria, CNRS, IRISA
lenaig.cornanguer@irisa.fr

Abstract

Event logs are often one of the main sources of information to understand the behavior of a system. While numerous approaches have extracted partial information from event logs, in this work, we aim at inferring a *global* model of a system from its event logs.

We consider real-time systems, which can be modeled with Timed Automata: our approach is thus a Timed Automata learner. There is a handful of related work, however, they might require a lot of parameters or produce Timed Automata that either are undeterministic or lack precision. In contrast, our proposed approach, called TAG, requires only one parameter and learns a deterministic Timed Automaton having a good tradeoff between accuracy and complexity of the automata. This allows getting an interpretable and accurate global model of the real-time system considered. Our experiments compare our approach to the related work and demonstrate its merits.

Introduction

The classic way of understanding a complex physical, or software system is to perform a manual analysis of this system, in order to produce a model, that should be as accurate as possible. This process is tedious and extremely time-consuming for the human(s) performing it, especially if the system is large and with many possible states. Due to this complexity, it is not done in practice for most running systems. Instead, the systems are equipped to produce detailed *logs*, that can help to detect odd behaviors on the fly, or at least allow to perform a post-mortem analysis after a failure happened. Data mining techniques can help analyzing these logs to discover human-understandable *local* regularities in the log, for example in the form of episodes or chronicles (Zou et al. 2010; Tatti and Vreeken 2012; Sahuguède, Le Corronc, and Le Lann 2018), that can directly correspond to parts of the global model. Process mining techniques (van der Aalst 2016) are dedicated to logs exploration, with notably one part dedicated to the construction of a global untimed model in the form of Petri net or BPMN (Business Process Model and Notation) to realize performance analysis or conformance checking. There are recent attempts to include time in the construction of the model and not only in the posterior model analysis (Tax et al. 2019) but time is still considered in a qualitative way. One can also mention the specification mining field which focuses on program logs to learn specifications in the form of untimed state machines (Ammons, Bodík, and Larus 2002) or Timed Regular Expressions (given its untimed template) (Narayan and Fischmeister 2019). However, none of those approaches considers the problem of discovering a *human-understandable global temporal model* of the real-time system having generated the log. This is the problem tackled in this paper.

Including time quantitatively in models is crucial for many applications such as self-driving cars or security protocols. To model real-time systems, the formalism of Timed Automata (TA) has been defined by Alur and Dill (Alur and Dill 1994). TA extends the formalism of automata by adding clocks expressing explicit timing constraints on the model. TA are well studied, both in theory and practice. The success of TA comes from a powerful formalism with high expressiveness associated with efficient algorithms and tool support. TA have been applied for the analysis of many real-time systems for solving problems as diverse as optimal planning, scheduling, or controlled synthesis (Clarke et al. 2018). They have proved to be relevant outside the field of real-time systems, for biological ecosystems, home care plans, agricultural processes, or software systems. In the wide range of applications possible with TA, we can cite optimization of animal waste allocation to crops (Hélias, Guerrin, and Steyer 2008), synthetic data generation from real data for privacy matter (Connes, De La Higuera, and Le Capitaine 2021), or anomaly detection in water treatment plant (Xu, Ali, and Yue 2021). Moreover, the graphical representation and explicit timing constraints between events can offer a human-understandable view of the model, as long as the size remains limited.

Our objective is thus to learn a timed automaton consistent with the logs of the observed system. There is a large body of work on learning (non-timed) automata and state machines stemming from the grammatical inference field (Angluin 1987; de la Higuera 2010) and extended for various classes of models (e.g., DFAs, mealy machines, Moore machines). Several algorithms exist (e.g., EDSM, Alergia, MDI) and efficient implementations can be found in LearnLib (Isberner, Howar, and Steffen 2015).

However, learning a model where time constraints play a key role is much more complicated and only a few works have tackled this problem. Learning timed automata is def-

initely a new and promising field of research. While some of the existing approaches rely on interactions with a user (active learning setting) (An et al. 2020; Henry, Jéron, and Markey 2020; Grinchtein, Jonsson, and Pettersson 2006), we focus our contribution on the classical "passive learning" setting, where the learner has only access to the input logs. In this setting, three main works have been proposed: RTI+ (Verwer, de Weerdt, and Witteveen 2012), GenProgTA (Tappler et al. 2018), and Timed k-Tail (Pastore, Micucci, and Mariani 2017). These works can achieve the discovery of TA from logs, however, they have several drawbacks, either in the number of parameters, the lack of precision of the TA learned, or not outputting a deterministic TA.

In this paper we propose a new algorithm, called Timed Automata Generator (TAG). TAG learns a timed automaton to describe and understand a time-dependent system only from its generated logs, without any a priori knowledge. Compared to other works dedicated to TA learning, our contributions are the following :

- TAG, a new algorithm that easily automates the TA learning process from logs since it only requires one parameter. This parameter controls the level of generalization of the learned TA, and thus the level of interpretability of the generated model. The learned TA is *deterministic* which contributes to the model understanding.
- An original pipeline that allows to apply model-checking techniques to query the most complex learned models that are not visually understandable.
- The first extensive experimental study and comparison with state-of-the-art algorithms for learning TA. These experiments show that TAG provide the best performances both for scalability and for the precision/recall trade-off. We also provide an experimentation on real-world data from logs of TV programs.

Background

We start by introducing formally the notion of Timed Automata, first through a simple example and then with the definitions and notations used throughout the paper.

Our example considers modeling the behavior of a simple light in a meeting room with a timed automaton, depicted in Figure 1. A single sensor sends a *press* event when the switch is pressed. The bulb can be *off* or *on* with low or high lighting. When the bulb is *off*, a single press on the switch turns the low light on, while a double press (in less than 2 sec.) makes the light bright. In case the delay is too long between the first and second press, the light turns off.

The TA defines the behavior of the light by three states (*off*, *light*, *bright*) and the event *press* tags the transitions. A single clock *c* measures the time between each event. In TA, transitions are instantaneous and allow the reset of clocks. When a *press* event occurs, a transition is triggered between the states *off* and *light*, and the clock *c* is reset. If the next *press* event occurs after 3 time units, the system moves back to *off*.

In this illustrative example, we notice that time is of paramount importance. Without knowledge about the value of the clock, the light state cannot be inferred.

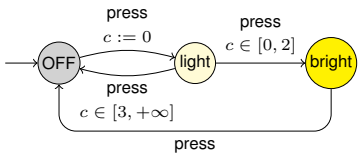


Figure 1: Timed Automaton of a light.

We now define formally the notions of *clock* and *timed automata*, followed by the semantics of the TA and the language it entails.

Clocks. Let T be a set of time domains. Let C be a finite set of variables called clocks. A clock valuation is a mapping $v : C \to T$ which assigns to each clock a time value. Let T^C be the set of all clock valuations over C. The set of clock constraints $Const(C, T)$ is the set of constraints $c \sim t$ with $c \in C$, $\sim \in \{<, \leq, =, \geq, >\}$, and $t \in T$.

Timed Automaton. A non-deterministic timed automaton $A \in NTA$ is a tuple $A = (\mathcal{Q}, \Sigma, C, \mathcal{E}, q_0, \mathcal{F})$ where $\mathcal{Q}$ is a finite set of locations, Σ is a finite set of events or symbols, C is a finite set of clocks, $\mathcal{E}$ is a finite set of transitions, and $q_0 \in \mathcal{Q}$ is the initial location. $\mathcal{E} \subseteq \mathcal{Q} \times \Sigma \times Const(C, T), \times \mathcal{P}(C) \times \mathcal{Q}$ is a finite set of transitions of the form (q, a, g, r, q') where q and q' are respectively the source location and destination location, g is a guard i.e., a constraint on the value of a clock, and r is the set of clocks being reset on the transition.

Semantics. The semantics of $A \in NTA$ is given by the timed transition system (S, s_0, T, Σ, E) where $S = \{(q, v) \in \mathcal{Q} \times T^C\}$ is the set of states with $s_0 = (q_0, 0)$. $E \subseteq S \times (\Sigma \cup T) \times S$ is the transition relation. The set E contains two kinds of transitions: *timed transitions (delays)* when the clocks valuations evolve in a location, and *discrete transitions (jumps)* denoted $(q, v) \to^a (q', v')$ with $a \in \Sigma$, expressing that there exists a transition $(q, a, g, r, q') \in \mathcal{E}$ such that the valuation v satisfies the guard g and $\forall c \in C$ if $c \in r$ then $v'(c) = 0$ otherwise $v'(c) = v(c)$.

Language of Timed Automaton. An untimed word $w \in \Sigma^*$ is a finite sequence of input symbols (elements from Σ, called alphabet). A timed word $tw \in (\Sigma \times T)^*$ is a finite sequence of input symbols and timestamps (non-decreasing). In the following, the set of all possible timed words over Σ and T is called $TS(\Sigma, T)$. A timed word tw induces a set of runs over a timed automaton. A word tw is consistent with an automaton $\mathcal{A}$ if there exists a run ending in an accepting location. For a timed automaton $\mathcal{A}$, the language $\mathcal{L}(\mathcal{A})$ denotes the set of timed words accepted by $\mathcal{A}$.

Sub-classes of Timed Automata. To learn efficiently timed automata from logs, timed-automata formalism has been restricted to sub-classes (Verwer, de Weerdt, and Witteveen 2012). Deterministic timed automaton (DTA) does not allow, from a location, two transitions having the same symbol and overlapping guards towards two different target locations. $1DTA$ is a DTA restricted to one clock. $DRTA$ (Deterministic Real-Time Automata) is a sub-class of $1DTA$ where the clock represents the time delay between two consecutive events. The guard is then defined as a closed interval. Each transition resets the clock, and to be triggered

the time value of the event should satisfy the guard transition. A PDRTA (Probabilistic Deterministic Real-Time Automata) adds probabilities on the DRTA events.

State-of-the-Art

Learning a TA from logs is a complex problem and it has already been proved that the identification of an untimed automaton is NP-complete (Gold 1978). All the algorithms presented in this section focus on TA sub-classes and take as input a set of timestamped event sequences, or timed words. In our context, we don't need to distinguish the notions of source and location, and we will use the term "state" for both.

GenProgTA. GenProgTA (Tappler et al. 2018) is an algorithm to learn TAs which are deterministic. It is based on genetic programming. At each generation, three operations can be performed (randomly chosen): a mutation (state merging, transition splitting, clock reset addition, state addition), a cross-over (exchange of a part of the automata) between two TA of the same population, or a cross-over between TAs from each population. These operations aren't based on the observations and thus can introduce inconsistencies. The evaluation step scores the individuals with several criteria such as the automaton size, the fitness with the observations, or the determinism. This algorithm has numerous parameters such as the probability of each operation, the population size, or the weights on the evaluation criteria. It also requires information about the timed properties of the system (number of clocks, approximate largest constant in clock constraints). Therefore, this algorithm is more suitable for expert users as it requires a lot of prior knowledge to set the parameters and to choose the final model. Due to the stochastic nature of the approach, different local optima may be found.

RTI+. RTI+ (Verwer, de Weerdt, and Witteveen 2010) learns the PDRTA sub-class of TA where transitions are labeled with their probability of occurrence. RTI+ is based on the EDSM (Evidence Driven State Merging) algorithm (Lang, Pearlmutter, and Price 1998) used in grammatical inference to learn untimed automata. The first step is to create an automaton where there exists only one path leading to each state, and consistent with the input sample. Every transition is assigned to the same interval guard bounded by the minimal and maximal observed time value in the whole sample. Thereafter, this tree-shaped automaton is modified in a red-blue framework, i.e. progressively from the initial state to the extremities. Two modification operations can be realized: merging two states and splitting a transition. To decide whether two states should be merged or a transition should be split, a likelihood ratio test is computed. The operation among all the possible operations increasing the most the likelihood of the data with the model is selected. Due to the red-blue framework, the split operation is only applied to transitions which are followed by parts of the automaton not modified (factorized by the merges) yet. RTI+ algorithm has been used to model driving behaviors in the field of autonomous vehicles (Lin et al. 2019), however, its management of temporal constraints leads to TAs that lack sensitivity.

Timed k-Tail. Timed k-Tail (TkT) (Pastore, Micucci, and Mariani 2017) is based on the k-Tail algorithm (Biermann and Feldman 1972) used to learn untimed automata. In these two algorithms, the states from which the same future symbol sequences of length k are possible are merged. The sequences are compared on the transition symbols and the set of sequences is called the k-future of the state. Timed k-Tail learns a sub-class of TA that has clock resets, local clocks, and a global clock that measures the time since the beginning of a run. It is designed for systems with nested operations, i.e., an operation (B) can occur while another operation (A) is in progress, as long as B ends before the end of operation A. In the absence of nested operations, the local clocks can be resumed to a unique clock that measures the delay between two operations, as in a RTA. In the automaton learned by Timed k-Tail, the choice of transition at each state is determinist only if we consider the k next symbols.

TAG Algorithm

State-of-the-art algorithms have drawbacks in terms of number of parameters and the determinism or precision of their output. Based on this observation, our objective was to develop a new algorithm to overcome these limitations and combine their strengths. The outcome is TAG, which stands for Timed Automata Generator, a novel algorithm to learn DRTA from positive timestamped event sequences. In addition to the previously presented DRTA characteristics, the transitions of DRTA learned by TAG are enriched with an indicative probability of occurrence, and a guard on a global clock never reinitialized which measures the time since the beginning of a run. The idea of the algorithm is to first produce an automaton which is basically a graphical representation of the input sample with all its redundancies. The automaton will then be factorized on these structurally redundant parts to obtain a more compact TA. After the size reduction, the temporal values are recomputed, and it may be necessary to refine the automaton in function of the time. The three main operations are the automaton initialization, the states merging, and the transition splitting. Algorithm 1 summarizes the learning process. After a tree-shaped automaton initialization (`initTA` function), the first step of the TAG Algorithm consists in reducing the automaton size by merging all the states that can be merged together (`all_possible_merge`), disregarding the temporal values. Algorithm 2 describes the `merge` operation. Two states can be merged if, from both states, the same events sequences can happen to the system within the k next transitions (the k-future). When no more states can be merged, the algorithm attempts to capture the temporal logic of the system with the `split` operation (described in algorithm 3). TAG's splits, unlike in RTI+, are applied after the merging step because the factorization has gathered the paths considering the events, so a split that should have been done multiple times only needs to be done once, and only the event sequences that remain specific to a time window are isolated. The split of transitions creates a temporal determinism where time influences the system's evolution. During this step, merges can also be realized but only if no more transition split is needed and if the merge won't be cancelled by

a split. Splits and merges are realized in a breadth-first order fashion. TAG ceases when no more split or merge can be done. Each transition is associated with a probability corresponding to the proportion of time the transition has been taken in the input sample. These probabilities are initialized in the automaton initialization and updated after each merge or split with respect to the new distribution. All the subfunctions of the algorithm as well as the consistency proof are given in the supplementary material and an implementation of the algorithm is available[1].

Algorithm 1: TAG : $\mathcal{P}(TS(\Sigma, T)) \rightarrow \mathcal{A}$

Require: an input $S = \{S_+\}$
Return: a DRTA consistent with S meaning that $\forall s \in S, s \in \mathcal{L}(A)$
$A = \text{initTA}(S)$
A=all_possible_merge(A)
repeat
  A=all_possible_split(A)
  SAVE=A
  A=merge(A)
until SAVE=A
return A

Algorithm 2: merge : $\mathcal{A} \rightarrow \mathcal{A}$

Require: a DRTA A
Return: a DRTA $merge(A)$ such that $\mathcal{L}(A) \subseteq \mathcal{L}(merge(A))$ and $|merge(A)| < |A|$.
$(q_1, q_2) = choice\_locations\_to\_merge(A)$
for all $t \in \mathcal{E}_{in}(q_2)$ **do**
  replace $t = (q, a, g, q_2)$ with (q, a, g, q_1)
end for
for all $t \in \mathcal{E}_{out}(q_2)$ **do**
  replace $t = (q_2, a, g, q)$ with (q_1, a, g, q).
end for
$Q = Q - \{q_2\}$
if $q_2 = q_0$ then $q_0 = q_1$
$transform(A, q_1)$
return A

We now illustrate the three major steps of the algorithm through simple examples.

Automaton Initialization. The first step is to create an automaton consistent with the input sequences. In such preliminary automaton, there exists one unique path leading to each state. Each sequence is represented by one of these paths and two sequences share a portion of path only if they have the same symbolic prefix. As an example, let's consider a set of two sequences: $\{\langle r{:}2\ p{:}6\ t{:}5\rangle, \langle r{:}3\ s{:}1\rangle\}$. Each element of a sequence is a pair of symbol and delay. Figure 2 presents the resulting preliminary automaton for these input sequences after the initialization step. First, an initial state

[1]TAG source code can be found here: https://gitlab.inria.fr/lcornang/tag/. The remainder of the supplementary materials can be found here: https://gitlab.inria.fr/lcornang/aaai22_tag_supplementary_materials.

Algorithm 3: split : $\mathcal{A} \times \mathcal{P}(TS(\Sigma, T)) \rightarrow \mathcal{A}$

Require: a DRTA $A = (\mathcal{Q}, \Sigma, \mathcal{E}, q_0)$ and $S \in \mathcal{P}(TS(\Sigma, T))$
Return: a DRTA consistent with S
$(t, g_1, Q_1) = choice\_transition\_to\_split(A)$ with $t = (q_1, a, g, q_2)$
add a new location $q_{2,split}$ in $\mathcal{Q}$
for $(q_2, e, g, q_3) \in \mathcal{E}$ *with* $q_3 \in Q_1$ **do**
  replace in the transition q_2 by $q_{2,split}$
end for
In the transition $q_1 \rightarrow^a_g q_2$ replace the guard g by g_2 where $g_2 \cup g_1 = g$ and $g_2 \cap g_1 = \emptyset$.
Add the transition $q_1 \rightarrow^a_{g_1} q_{2,split}$ in $\mathcal{E}$.
return A

$S0$ is created as the starting state for each sequence. The first pair of the first sequence is r:2. Starting from $S0$, there is no transition labeled with the symbol r, therefore, a transition is created towards a new state, $S1$, and labeled with the symbol r and the guard $[2, 2]$. Now considering the second pair p:6, a transition must be created from $S1$ to a new state $S2$, and this transition is labeled with the symbol p and the guard [6,6]. This process is repeated until the end of the sequence. The evaluation of the next sequence restarts from the initial state. Since the first pair r:3 displays a symbol already carried by an outgoing transition of $S0$, there is no need to create a new transition, and the guard of the transition labeled with r is enlarged to [2, 3] to accept this new temporal value. Then, a new transition is created from $S1$ for the last pair s:1. Unlike the RTI+ approach, each transition is associated with a specific guard characterizing the observed temporal constraints in the sequences. Finally, the transitions are given a probability corresponding to the learning sample sequences distribution, and an interval guard corresponding to the time elapsed since the beginning of the sequences when they use the given transition. The guards on the global clock aren't shown in the section's figures since they are not used for the model construction.

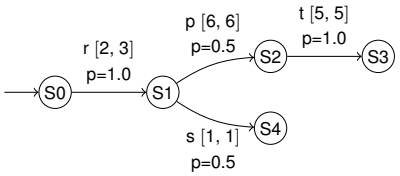
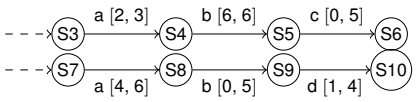
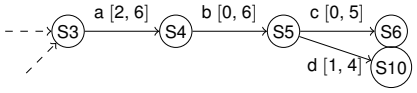


Figure 2: A TA initialized with a set of events sequences.

States Merging. As in k-Tail and Timed k-Tail, two states are considered for merging if their k-futures are identical. Let's consider the automaton of Figure 2, and $k = 2$, the number of future events to consider per sequence. The k-future of the state $S0$ is a set of two sequences: $\{\langle r, p\rangle, \langle r, s\rangle\}$, with $\langle r, p\rangle$ corresponding to the path going through $S0$, $S1$ and $S2$, and $\langle r, s\rangle$ corresponding to the path going through $S0$, $S1$, and $S4$. In this automaton, there is no other state having the same k-future. Otherwise, the two



(a) Unmerged parts



(b) Merged parts

Figure 3: Two parts of the automaton that would have been successively merged and then split in the absence of the guard overlap requirement.

states would have been merged following the procedure presented in Algorithm 2. Merging two states consists in accumulating their outgoing and incoming transitions on the first and to delete the second. Due to our determinism requirement, if this operation induces a situation where two transitions have the same symbol and the same source, these transitions and their destination states will also be merged. Merging two transitions leads to the guard enlargement to include both former guards. This is called the determinization process. After a merge, the probabilities are updated with the new word's distribution. In TAG's last step (splits and merges succession), an overlapping-guards requirement is also added to ensure convergence so a merge doesn't cancel the result of splits (Figure 3). k is the only parameter of TAG. By controlling the length of the event sequences to compare, it allows tuning the trade-off between generalization and over-fitting of the model. If the input sample is exhaustive or if detecting wrong behavior is more important than having a small and easily interpretable model, k should be increased. Its default value is set to 2.

Transitions Splitting. The split procedure relies on both automaton analysis and input sequences. Let's consider a transition ϵ and guard $[\tau_{min}, \tau_{max}]$ from state q to q', q' leading to different parts of the automaton: $part1$ and $part2$. Given the analysis of the input sequences, if we observe that some timed words moving through ϵ from q to q' are always going to $part1$ with a delay in $[\tau_{min}, \tau]$, with $\tau \in [\tau_{min}, \tau_{max}]$, while others lead to $part2$ with a delay in $[\tau + 1, \tau_{max}]$, the transition labeled ϵ can be split. A new state q" reached from q is created, such that q' leads to $part1$ and q" to $part2$. Both transitions outgoing from q are labeled by the symbol of ϵ but associated to different guards $[\tau_{min}, \tau]$ and $[\tau + 1, \tau_{max}]$. Given the automaton displayed in Figure 4 (left), the traces (not shown here) inform us that some events b at the beginning of the sequences, associated with a delay lower than 5, are followed by an event a. Some other events b at the beginning, associated with a greater delay, are followed by an event c. These events b correspond to the transition between $S0$ and $S3$, which will be split as presented in Figure 4 (right).

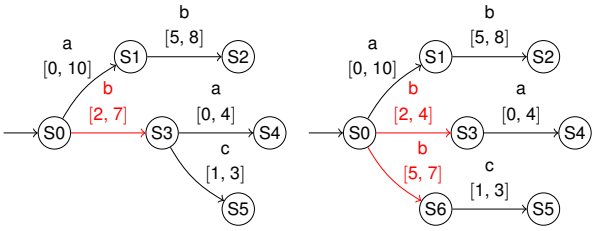


Figure 4: A transition division (left: before, right: after).

Experiments

Following the presentation of the state-of-the-art algorithms and our contribution, this section will tackle the evaluation and comparison of the algorithms on synthetic and real-world data. Full data, scripts and results can be found in the supplementary materials.

Method

To evaluate TAG, we addressed four questions:

Q1. How does TAG scale with the complexity of the data?
Q2. How does its parameter k impact the result?
Q3. What is the contribution of each operation to the result?
Q4. How does TAG cope with real-world data?

We studied the first concern by challenging RTI+, Timed k-Tail, and our algorithm TAG on the quality of their models and their runtime in various situations using synthetic data (we couldn't test GenProgTA because its interface doesn't allow the addition of new data and the source code isn't available). These same data were used to investigate Q2 and Q3. We also carried out a real-data experiment to demonstrate TAG's ability to deal with real-world data and to produce interpretable models, using the logs of the programs of a Canadian TV channel and querying TAG's output with model-checking.

Experiment on synthetic data. To study how TAG and the state-of-the-art algorithms scale with the complexity of the input data, we identified multiple factors possibly inferring the runtime and model quality. There are two types of factors. The first factors are inherent to the expected model:

- The alphabet size (number of different events or symbols),
- The number of states of the system,
- The outdegree (average number of outgoing transitions per state),
- The proportion of twinned-transitions (transitions from the same state labeled with the same symbol but having non-overlapping guards), these twinned-transitions create an untimed underminism.

In real life, these factors are not controllable as the expected model depends on the studied system. The second type of factor is related to the learning process and these can be, to some extent, adjusted by the user:

- The size of the input sample i.e., the number of timed words the algorithms will have to process,

- TAG's parameter k, only studied for our algorithm.

We defined a set of values to test for each factor (Table 1). To evaluate the impact of the factors independently from each other, we varied the factors one after the other. When one factor varies, the others have a fixed value (in bold in Table 1). We then ranked these factors according to the estimated impact they would have on the model quality and the runtime. We tested the factors in order of increasing impact (average impact on model quality and runtime), which corresponds to the order of the factors in Table 1. To control the factors related to the expected model and to have a ground truth, we started from synthetic model TAs, generated to have a given value for each factor. We generated 200 TAs per combination to gain confidence in the results. The datasets are composed of runs of these model TAs, each run being a timed word. For the evaluation of the learned TA, we also generated 100 other timed words consistent with the model and 100 words inconsistent with the model. We used the same protocol for the ablation study. 200 TA were generated with the default value for every factor.

Factor	Tested values
Alphabet size	2, 4, **5**, 6, 8, 10, 15
Outdegree	1, 1.25, **1.5**, 1.75, 2, 2.25, 2.5
State number	4, 6, 8, **10**, 15, 25, 50, 75, 100, 500
Twinned transitions proportion	0, 0.10, **0.25**, 0.50, 0.7
Number of timed words	50, 100, **500**, 1000, 2500
TAG parameter k	**2**, 3, 4, 5, 6

Table 1: Factors order and values (default value in bold).

The runtime was measured on a MacBook Pro with an Intel Core i9 processor clocked at 2,4 GHz and a memory of 16 Go 2667 MHz DDR4. TAG is implemented in Python, Timed k-Tail's author implementation (where k is hard coded to 2) is in Java and RTI+ in C++. RTI+ was executed with a significance value of 0.05 for the likelihood ratio test (default value). TkT was executed with no nested events considerations (absent in our data) and no enlargement of the guards (default value). To evaluate the learned TAs, two accuracy scores were selected. The True Positive Rate (TPR), also called recall, is the probability for a word consistent with the model to be recognized by the learned automaton. The Positive Predictive Value (PPV), also called precision, is the probability for a word recognized by the learned automaton to be consistent with the model automaton. A high recall and precision are both desired but are in practice rather impossible to have simultaneously. As we haven't a specific field of application, we consider them equally important. The F1-score is the harmonic mean of these two measures. Aiming for a good F1-score leads us to a good trade-off between recall and precision.

Experiment on real-world data. To assess TAG's ability to learn an interpretable and exploitable model from real data, we used the logs of the programs of the Canadian TV channel CBC Windsor (Canadian Radio-television and Telecommunications Commission 2015). We first used the data of the Friday mornings of August 2020 (from 6:00 AM to 12:00 AM, one word per day). Then, we used the data of every day of July and August 2020 (Canadian summer school vacations months). The entries of the logs were summarized by their class (commercial message, promotion for a program ...) or category (program for children, news...) in case of a program. A word consists of the sequence of entries class/category and their duration for a day.

To query the automaton learned by TAG with the summer data, we can take advantage of both the classical expressiveness of TA and the probabilities associated with the TAG's TA transitions. We used UPPAAL SMC (Bulychev et al. 2012), a model-checking tool for Timed Automata with stochastic properties. UPPAAL SMC extends the basic query language of UPPAAL (Bengtsson et al. 1996), which is a subset of Timed Computation Tree Logic (TCTL), with queries related to the stochastic behavior of systems. The result of UPPAAL SCM queries is obtained by monitoring simulations of the system and by statistical hypothesis testing. The answer is an interval of probability with a confidence of 0.95 of being within. We used both classical UPPAAL and UPPAAL SMC queries and for the latter, the number of simulations was fixed at 10000.

Results

Experiment on synthetic data. The whole results of the comparison of the algorithms for every factor are in the supplementary materials. We only present here the most significant results. For all the statistical tests, the significance level is of $\alpha = 0.05$. On the figures, one asterisk indicates a significant difference with a p-value of at least α.

Q1. How does TAG scale with the complexity of the data?

In the first place, we focus on the runtime. Globally, the runtime of the three algorithms is comparable, with RTI+ (median $\tilde{x} = 0.22s$, interquartile range $IQR = 0.25s$) followed by TAG ($\tilde{x} = 0.46s$, $IQR = 0.21s$) and Timed k-Tail ($\tilde{x} = 0.64s$, $IQR = 0.06s$). The factors positively correlated with TAG's runtime are the following: the outdegree as more merges are necessary; the number of states since TAG process the pair of states in a breadth-first order to find split candidates; the parameter k as the states k-future is constructed recursively; and obviously the number of timed words (Figure 6b). The twinned transition proportion is slightly negatively correlated to the runtime.

Turning now on the quality of the obtained models, evaluated with the recall, the precision, and the F1-score, Figure 5 presents these scores for the three algorithms and for all the tested combinations. Globally, TAG's precision always stays good (0.92 on average) and significantly better than the other algorithms (rate of increase of 0.48 with RTI+ and 0.34 with Timed k-Tail). Due to the trade-off, it leads to a slight loss of recall in comparison to Timed k-Tails (rate loss of 0.02, giving a recall of 0.97 on average) but the gain in accuracy is significant with an F1-score above the others on average. Timed k-Tail has the best recall overall but tends to produce models too generalized because of the absence of splits. RTI+ produces models not precise enough because

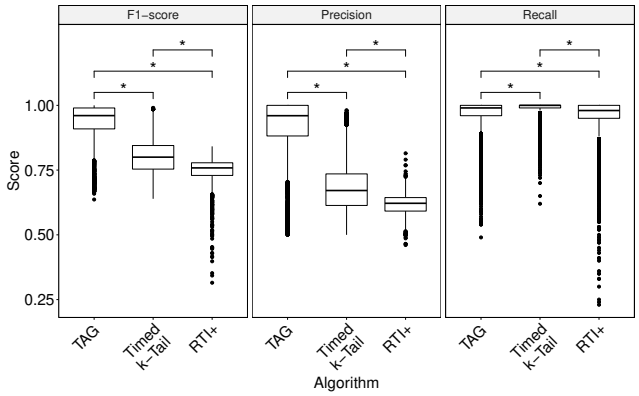


Figure 5: Scores per algorithm.

of its bad management of the time constraints that leads to guards too wide. Let's now look at the effect of the different factors. As shown in Figure 7, TAG is less sensitive to the alphabet size than Timed k-Tail, which produces models too compact and thus not much precise when there are few different symbols, and RTI+, which doesn't generalize enough in presence of many symbols and therefore doesn't recognize well new words of the language. TAG is sensitive to the complexity of the model automaton (outdegree, number of states, and twinned transitions proportion). As it becomes more complex, the recall decreases since the represented part of the language in the sample data becomes smaller, and thus the new words will be less probably recognized. However, its precision stays good (0.79 on average for the worst case corresponding to an outdegree of 2.5) and systematically above the other algorithms, as well as the F1-score. This means that TAG automata are accurate and not too generalized. Finally, the recall (and F1-score) of the TA of the three algorithms unsurprisingly increases with the number of timed words in the input data sample (Figure 6a), with a stabilization of the score at 500 timed words for our parameter's combination.

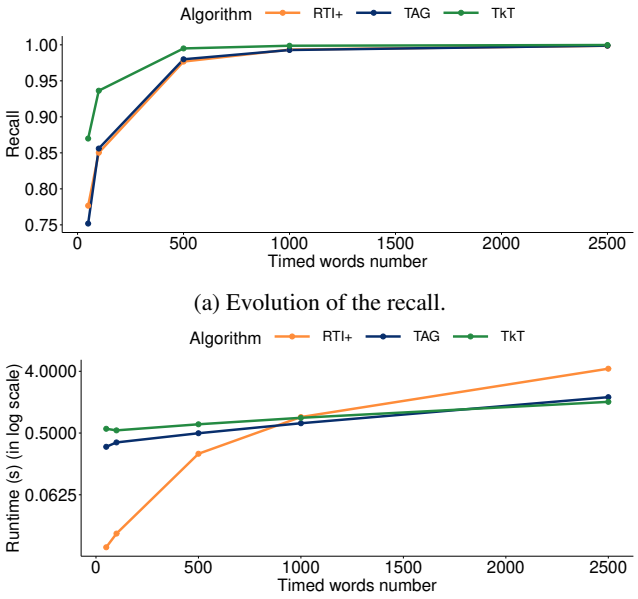
In summary, these results show that TAG offers a better trade-off between recall and precision than the state-of-the-art algorithms with a comparable runtime. Globally, TAG automata are accurate and not too generalized. The factor impacting the most TAG is the number of timed words in the input sample, which increases the runtime but also the quality of the obtained TA. The complexity factors penalizing TAG's quality are the number of states and the outdegree.

Q2. How does TAG's parameter k impact the result?

As k increases, the precision tends to increase while the recall decreases (Figure 8). However, the gain in precision is inferior to the loss of recall. Therefore, the default value is set to $k = 2$, letting the user freedom to tune it in function to its application and its needs. Particularly, there can be a visual impact and thus an interpretability impact if the model is meant to be human-understandable.

Q3. What is the contribution of each operation to the result?

We performed an ablation study to confirm the importance of each operation of TAG. Figure 9 compares the



(a) Evolution of the recall.

(b) Evolution of the runtime.

Figure 6: Evolution of the recall and the runtime w.r.t.the number of timed words.

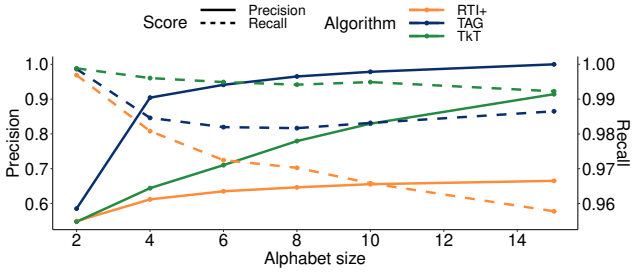


Figure 7: Precision and recall when the alphabet size varies.

precision, the recall, and the F1-score obtained for the TA produced with only the tree-shaped automaton initialization, then with the initialization and the merges, and finally with the initialization, the merges, and the splits. With only the initialization, the precision is maximal since there is no generalization, the automaton language corresponds to the input traces (plus the values inside the guard intervals). For the same reason, the recall is lower than with the merges and the splits. Merges induce a significant augmentation of the recall by generalizing the model. Lastly, TAG recall and precision are improved by the splits leading to a better F1-score. Beyond these positive results on synthetic and random data, the importance of the splits becomes more evident in the case of systems where some events would happen after a first event only within a limited time window and never otherwise. In an TA learned with splits, the timed condition would be necessary to access this part of the TA.

Experiment on real-world data.

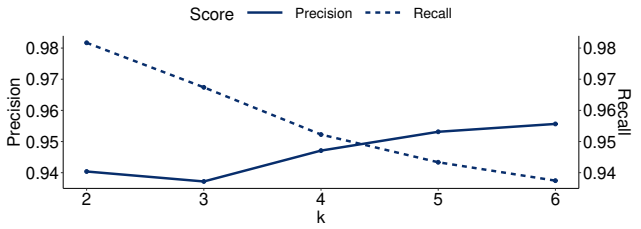


Figure 8: TAG's Precision and recall when k varies.

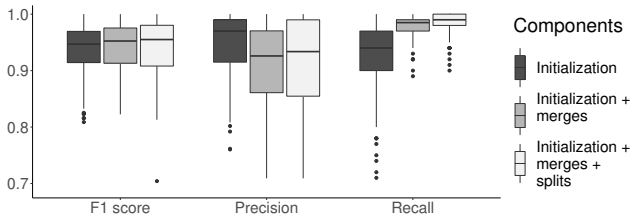


Figure 9: Ablation study on recall, precision and F1-score.

Q4. How does TAG cope with real-world data?

The TA learned by TAG with the TV logs of the Friday mornings (329 events) is shown in Figure 10. The guards are originally in seconds and have been formatted to make the figure more apprenhensible. The first guard in bracket limits the delay of occurrence after the last event. The second guard in bracket preceded by the letter "t" corresponds to the value of a global clock started at 6:00 AM in the initial state and never reset.

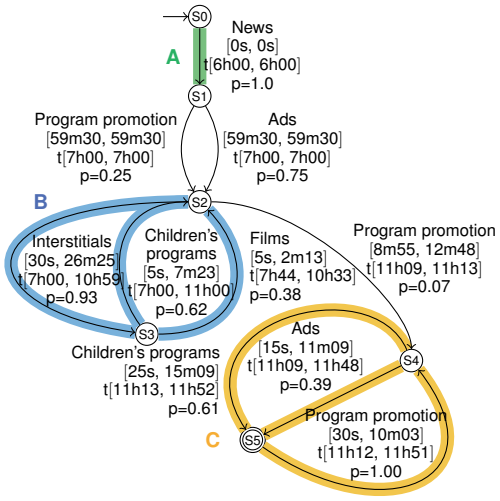


Figure 10: Learned TA from the Friday morning logs of a Canadian TV channel.

This TA has 6 states, 10 transitions, and 6 different symbols. This TA shows that after the morning news, which lasts about one hour (A on the figure), and a session of ads or program promotions, films and/or children's programs follow one another until 11:00 AM (B on the figure). Children's programs are more probable and these programs are frequently cut by interstitials. Then, the ads and program's teasers are back and cut children's programs until the 12:00 AM news (C on the figure). The interest of the probabilities and the guards on the global clock is demonstrated here since they provide substantial information for the system comprehension.

Query	Result	Time (s)
What is the probability to watch one hour of children's program without interruption?	[0.07,0.08]	0.722
If we watch this channel all the day long, are we sure to have, at some point, a children's program?	No	0.003
Between 10h and 11h, what is the probability to watch a children's program?	[0.46,0.56]	0.049
Globally, is it more probable to have a film than a children's program?	No	0.011

Table 2: Queries, result, and execution time.

The TA learned with the whole data of July and August (54065 events) is naturally much bigger since the time slot is wider and the day types differ. 15240 merges and 12 splits were necessary. It has 65 states, 125 transitions, and 14 different symbols. Table 2 presents the queries submitted to the model-checker, the results, and the execution time. In less time than a human would take to analyze even the first TV programs automaton, the responses to useful questions can be obtained with model-checking queries.

Conclusion

This paper introduces a new algorithm, TAG, to learn Timed Automata from logs of real-time systems. This learned automaton can be used to model a time-dependent system to understand its behavior without any *a-priori* knowledge about it. The unique parameter k offers a trade-off between precision and recall of the learned model, depending on the application domain or the desired level of interpretability. On this relatively new subject, this study is the only one that compares the existing approaches. Experiments have shown that TAG is fully capable of inferring a TA describing the system including realistic time constraints while remaining interpretable, visually if the model is small, and thanks to model-checking otherwise. The model can then be used to perform anomaly detection, data generation, prediction, or else verification. An interesting perspective would be to learn a model composed of various interacting subsystems to limit the global model size. As more and more physical systems are being equipped with sensors producing time series, it could also be useful to develop a method to obtain a TA from this kind of data.

References

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

Ammons, G.; Bodík, R.; and Larus, J. R. 2002. Mining specifications. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '02*, 4–16. Portland, Oregon: ACM Press. ISBN 978-1-58113-450-6.

An, J.; Chen, M.; Zhan, B.; Zhan, N.; and Zhang, M. 2020. Learning One-Clock Timed Automata. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 444–462.

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2): 87–106.

Bengtsson, J.; Larsen, K.; Larsson, F.; Pettersson, P.; and Yi, W. 1996. UPPAAL — a tool suite for automatic verification of real-time systems. In Goos, G.; Hartmanis, J.; van Leeuwen, J.; Alur, R.; Henzinger, T. A.; and Sontag, E. D., eds., *Hybrid Systems III*, volume 1066, 232–243. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-61155-4 978-3-540-68334-6. Series Title: Lecture Notes in Computer Science.

Biermann, A. W.; and Feldman, J. A. 1972. On the Synthesis of Finite-State Machines from Samples of Their Behavior. *IEEE Transactions on Computers*, C-21(6): 592–597.

Bulychev, P.; David, A.; Larsen, K. G.; Mikučionis, M.; Bøgsted Poulsen, D.; Legay, A.; and Wang, Z. 2012. UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata. *Electronic Proceedings in Theoretical Computer Science*, 85: 1–16.

Canadian Radio-television and Telecommunications Commission. 2015. Television program logs. https://open.canada.ca/data/en/dataset/800106c1-0b08-401e-8be2-ac45d62e662e. Accessed Aug. 23, 2021.

Clarke, E. M.; Henzinger, T. A.; Veith, H.; and Bloem, R. 2018. *Handbook of Model Checking*. Springer.

Connes, V.; De La Higuera, C.; and Le Capitaine, H. 2021. Using Grammatical Inference to Build Privacy Preserving Data-sets of User Logs. In *International Conference on Grammatical Inference*. Nantes, France.

de la Higuera, C. 2010. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press.

Gold, E. M. 1978. Complexity of automaton identification from given data. *Information and Control*, 37(3): 302–320.

Grinchtein, O.; Jonsson, B.; and Pettersson, P. 2006. Inference of Event-Recording Automata Using Timed Decision Trees. In *CONCUR 2006 – Concurrency Theory*, 435–449.

Henry, L.; Jéron, T.; and Markey, N. 2020. Active Learning of Timed Automata with Unobservable Resets. In *Formal Modeling and Analysis of Timed Systems FORMATS*, 144–160.

Hélias, A.; Guerrin, F.; and Steyer, J.-P. 2008. Using timed automata and model-checking to simulate material flow in agricultural production systems—Application to animal waste management. *Computers and Electronics in Agriculture*, 63(2): 183–192.

Isberner, M.; Howar, F.; and Steffen, B. 2015. The Open-Source LearnLib - A Framework for Active Automata Learning. In *Computer Aided Verification (CAV)*, 487–495.

Lang, K. J.; Pearlmutter, B. A.; and Price, R. A. 1998. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference LNCS*, volume 1433, 1–12.

Lin, Q.; Zhang, Y.; Verwer, S.; and Wang, J. 2019. MOHA: A Multi-Mode Hybrid Automaton Model for Learning Car-Following Behaviors. *IEEE Transactions on Intelligent Transportation Systems*, 20(2): 790–796.

Narayan, A.; and Fischmeister, S. 2019. Mining Time for Timed Regular Specifications. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 63–69. Bari, Italy: IEEE. ISBN 978-1-72814-569-3.

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

Sahuguède, A.; Le Corronc, E.; and Le Lann, M.-V. V. 2018. Chronicle Discovery for Diagnosis from Raw Data: A Clustering Approach. In *10th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes, SAFEPROCESS 2018*, 8p. Varsaw, Poland.

Tappler, M.; Aichernig, B. K.; Larsen, K. G.; and Lorber, F. 2018. Learning Timed Automata via Genetic Programming. *arXiv:1808.07744 [cs]*.

Tatti, N.; and Vreeken, J. 2012. The long and the short of it: summarising event sequences with serial episodes. In Yang, Q.; Agarwal, D.; and Pei, J., eds., *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, 462–470.

Tax, N.; Alasgarov, E.; Sidorova, N.; Haakma, R.; and van der Aalst, W. M. P. 2019. Generating time-based label refinements to discover more precise process models. *J. Ambient Intell. Smart Environ.*, 11(2): 165–182.

van der Aalst, W. M. P. 2016. *Process Mining: Data Science in Action*. Berlin, Heidelberg: Springer Berlin Heidelberg : Imprint: Springer, 2nd ed. 2016 edition. ISBN 978-3-662-49851-4.

Verwer, S.; de Weerdt, M.; and Witteveen, C. 2010. A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data. In *International Colloquium on Grammatical Inference*, 203–216. Springer.

Verwer, S.; de Weerdt, M.; and Witteveen, C. 2012. Efficiently identifying deterministic real-time automata from labeled data. *Machine Learning*, 86(3): 295–333.

Xu, Q.; Ali, S.; and Yue, T. 2021. Digital Twin-based Anomaly Detection in Cyber-physical Systems. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, 205–216.

Zou, J.; Xiao, J.; Hou, R.; and Wang, Y. 2010. Frequent Instruction Sequential Pattern Mining in Hardware Sample Data. In Webb, G. I.; Liu, B.; Zhang, C.; Gunopulos, D.; and Wu, X., eds., *IEEE International Conference on Data Mining (ICDM)*, 1205–1210.