Timed automata learning from time series

Lénaïg Cornanguer

COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Informatique*

Par
Lénaïg CORNANGUER

Timed automata learning from time series

Thèse présentée et soutenue à Rennes, le 02/11/2023
Unité de recherche : IRISA

Rapporteurs avant soutenance :

Antoine CORNUÉJOLS Professeur, AgroParis Tech
Anthony BAGNALL Full Professor, Université d'East Anglia, Angleterre

Composition du Jury :

Président :	Nicolas MARKEY	Directeur de recherche, CNRS
Examinateurs :	Florent MASSEGLIA	Directeur de recherche, INRIA Montpellier
	Sergio YOVINE	Professeur, Université ORT, Uruguay
	Laurence ROZÉ	Maître de conférences, INSA Rennes
Dir. de thèse :	Christine Largouët	Maître de conférences, Institut Agro
Co-dir. de thèse :	Alexandre TERMIER	Professeur, Université Rennes 1

ACKNOWLEDGEMENTS

I would like to thank Anthony Bagnall and Antoine Cornuéjols for reviewing this thesis, as well as Nicolas Markey, Florent Masseglia, and Sergio Yovine for agreeing to be part of my jury and for our discussion during the defence. I would also like to thank François Coste and Yannick Pencole for following me as part of my thesis committee.

I would like to express my sincere gratitude to my supervisors, Christine Largouët, Laurence Rozé, and Alexandre Termier, for introducing me to the world of research, which was completely unknown to me at the beginning. I appreciated their trust and the freedom they gave me to explore the topics that interested me. I will truly miss our weekly meetings, both for their scientific value and for their conviviality, and I hope that we will have many opportunities to meet again in the future.

I would like to thank the all the members of the LACODAM team for creating a welcoming environment, as well as for the engaging reading groups and seminars.

Lastly, I would like to express my special gratitude to my husband, family and friends for their unwavering support during the most challenging times.

TABLE OF CONTENTS

Introduction 1

I Timed Automata learning from event sequences 9

Introduction of the Part 10

1 State of the art around Timed Automata 11
1.1 Modeling the behavior of Discrete Event System . . . 11
1.1.1 Discrete-Event System . . . 11
1.1.2 Untimed models . . . 17
1.1.3 Timed models . . . 23
1.1.4 Conclusion on Discrete Event Systems modeling formalisms . . 28
1.2 A focus on Timed Automata . . . 29
1.2.1 Decision problems for Timed Automata . . . 29
1.2.2 Timed Automata subclasses . . . 31
1.3 Timed Automata learning . . . 34
1.3.1 Active learning . . . 35
1.3.2 Passive learning . . . 36
1.3.3 Conclusion on Timed Automata learning . . . 43

2 TAG: Passive learning of TA from logs 45
2.1 Motivation . . . 45
2.2 General presentation . . . 46
2.2.1 Language subclass learned . . . 46
2.2.2 TAG's algorithm outline . . . 46
2.3 Algorithm . . . 47
2.3.1 Automaton initialization . . . 48
2.3.2 State merging . . . 51
2.3.3 Transition splitting . . . 54

2.3.4 Additional considerations on the temporal refinement 57
2.4 Consistency proof 58
2.4.1 Lemma 1: If $s \in S$ then $s \in \mathcal{L}(Init(S))$ 59
2.4.2 Lemma 2: If $s \in \mathcal{L}(\mathcal{A})$ then $s \in \mathcal{L}(merge(\mathcal{A}))$ 59
2.4.3 Lemma 3: If $s \in \mathcal{L}(\mathcal{A})$ then $s \in \mathcal{L}(split(\mathcal{A}))$ 60
2.5 Experiments on synthetic data 61
2.5.1 Method 61
2.5.2 TAG's parameter and operations impact 66
2.5.3 Scalability experiment 70
2.5.4 Comparison with the State-of-the-Art algorithms 74
2.5.5 Conclusion on the experiment on synthetic data 75
2.6 Experiment on real data: TV logs 75
2.6.1 Method 76
2.6.2 Friday mornings subset 76
2.6.3 July and August subset 78
2.7 Conclusion 79

II Timed Automata learning from numerical data 81

Introduction 82

3 State of the art around time series 83
3.1 Time series discretization 83
3.1.1 Univariate discretization 85
3.1.2 Multivariate discretization 88
3.2 Rule mining in time series 91
3.2.1 Sequential rule mining 92
3.2.2 Allen's temporal relation-based rule mining 93

4 Univariate discretization for TA Learning 95
4.1 Persist with Wasserstein distance 96
4.1.1 Persistence score 96
4.1.2 Improving Persist for Timed Automata learning 99
4.1.3 Experiment on a classification task 101

4.2 Multi-objective optimization-based discretization for Discrete-Event Systems . . . 106
4.2.1 Multi-Objective Optimization Problem . . . 107
4.2.2 Multi-Objective Optimization Problem solving . . . 108
4.2.3 Solution selection . . . 111
4.2.4 Evaluation of the solution selection techniques . . . 112
4.2.5 Conclusion on MOODES . . . 114
4.3 Timed Automata-based anomaly detection on time series . . . 114
4.3.1 Motivation . . . 114
4.3.2 Anomaly detection with ensemble of Timed Automata . . . 115
4.3.3 Experiment on synthetic data . . . 121
4.3.4 Experiment on BATADAL challenge . . . 124
4.3.5 Conclusion on the anomaly detection approach . . . 132

5 Synchronization-preserving discretization 133
5.1 Problematic . . . 133
5.2 Synchronization discovery in time series . . . 137
5.2.1 Discretization via agglomerative hierarchical clustering . . . 137
5.2.2 Bivariate synchronization discovery . . . 138
5.2.3 Combining discretization and synchronization discovery . . . 142
5.2.4 Identifying synchronizations in the data structure . . . 144
5.3 Algorithm . . . 147
5.3.1 Enumeration problem . . . 147
5.3.2 Handling noise . . . 149
5.3.3 Synchronization-preserving discretization . . . 150
5.3.4 Timed I/O Automata learning . . . 152
5.4 Experiment on synthetic data . . . 152
5.4.1 Data generation . . . 152
5.4.2 Discretization . . . 153
5.4.3 Results . . . 153
5.5 Conclusion . . . 156

Conclusion and perspectives 157

Bibliography 161

List of abbreviations **173**

A TAG implementation in Python **175**
A.1 Input sample syntax 175
A.2 Usage 175
A.2.1 Example 175
A.2.2 Arguments 175
A.3 Output 176
A.4 Requirements 176

B Comparison of the TAs learning algorithms on the scalability experiment **177**
B.1 Alphabet size 177
B.2 Outdegree 178
B.3 State number 179
B.4 Twinned transitions proportion 180
B.5 Timed event sequences number 181

Summary in French (Résumé en français) **183**

INTRODUCTION

Motivation

As sensors and data storage capacities become increasingly affordable, the amount of data generated each year continues to grow at an unprecedented rate. This profusion of data presents a unique opportunity to gain a deeper understanding of our world. In particular, the repeated collection of data over a same object or system offers invaluable insights into its dynamics. From industrial systems to biological ecosystems, these systems often exhibit intricate behavior and remain only partially understood. Observational data over time is a valuable resource to extract meaningful patterns and derive global behavior models of these systems.

To illustrate this motivation, let's consider a production line in a factory. Industrial processes often involve both human operators and machines, each having their own dynamics while interacting with each other. Despite having documentation for the machines and operator task procedures, obtaining a precise, global view of the complex system's functioning is often challenging. The automatic inference of behavior models from observational data offers the possibility to get a non-biased and comprehensive understanding of such system. These models should precisely capture timing constraints that impact the dynamical behavior and enable to depict the interactions between different entities. Beyond their informative purpose, these models should enable system control and monitoring. Moreover, such models and their inference process should be interpretable by a human to ensure the user's trust.

The research problem addressed in this thesis is the automatic inference of system behavior models from times series data. The requirements for precise timing consideration, interpretability, versatility, and interactions support led us to select the formalism of Timed Automata (TAs) introduced by Alur and Dill (1994). TAs are state-based machines that incorporate discrete semantics along with a quantitative consideration of time.

To introduce TAs, let's consider a lamp that has two light intensity levels, controlled via a switch. This lamp has three states: off, on with normal intensity, and

on with bright intensity. Its state changes with the occurrence of an event: a switch press. If the lamp is off, a press will switch it on with a normal intensity. To obtain a bright light, the switch must be pressed again quickly (within 2 seconds) after the initial press. Otherwise, the lamp will simply switch off. Figure 1 displays a TA modeling

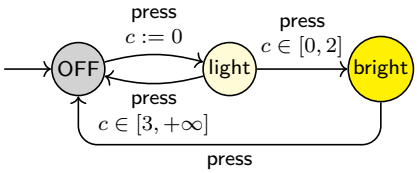
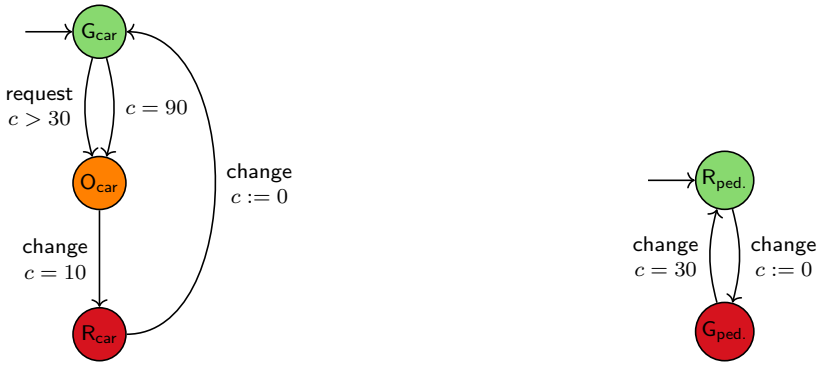


Figure 1 – TA modeling a lamp with variable light intensity.

the functioning of this lamp. The TA states correspond to the states of the lamp (OFF, light, bright), and the possible state changes are represented by transitions labeled with the event that triggers them (press). Additionally, these transitions are labeled with timing constraints using temporal variables called clocks (c), constraining the state changes based on time. For example, the two switch presses must occur within two seconds to get a bright light, indicated by the value of clock c being set to 0 at the first press ($c := 0$) and determining the next transition ($c \in [0, 2]$ or $c \in [3, +\infty]$).

In the last decade, learning TAs from event sequences have been an active field of research, leading to several algorithms capable of producing TAs given discrete observational data (Verwer, Weerdt, and Witteveen 2010; Verwer, Weerdt, and Witteveen 2012; Pastore, Micucci, and Mariani 2017; Tappler et al. 2019). Furthermore, the high expressiveness of TA formalism has led to the development of various tools for monitoring and verifying their properties.

Another advantage of TAs is the possibility of considering multiple TAs simultaneously, said to be in interaction, and synchronizing their transitions via shared events. Let's imagine a road junction two traffic lights, one for the vehicles and the second for the pedestrians. The vehicle traffic light can be green, orange, or red, indicating that the road users can cross the intersection, that they should proceed with caution, or that they must stop, respectively. The pedestrian traffic light can be green or red, indicating whether it is safe or not for the pedestrians to cross the road. The traffic lights are synchronized, ensuring that both lights are not green simultaneously. Pedestrians can also request a green light by pressing a button (if at least some time have elapsed since the last color change). Figure 2 displays two interacting automata, each one modeling one of the traffic lights. The transition from green to red for the vehicle



(a) Vehicles. (b) Pedestrian.

Figure 2 – Two interacting TAs modeling the functioning of the car and pedestrian traffic lights at a junction.

traffic light can either be triggered by the "request" event, or because the maximal delay is reached. Then, the shared event "change" will synchronize the color changes of the traffic lights, ensuring both are never green at the same time.

Challenges

The automatic inference of TAs from observational data faces several challenges.

Learning Timed Automaton from discrete observational data

Automatically inferring a system's behavior model in the form of a TA solely from observational data is not a straightforward task. If the resulting model adheres too closely to the data, the training data must be exhaustive, otherwise any new — yet similar — data will be inconsistent with the model. In practice, it is rare to have exhaustive training data, and it is impossible to assess anyway. On the other hand, if the model generalizes too much from the observations, it may lack power to predict future behavior or to check new observations. Thus, achieving a good balance between generalization and precision poses a significant challenge in TA learning. Additionally, identifying complex timing constraints based on multiple clocks remains an open question.

Time series discretization for Discrete Event System learning

TAs are well-suited for modeling Discrete Event System (DES), where state changes are caused by event occurrences, such as the lamp with variable light intensity. However, observational data may take the form of numerical time series, especially when events are not explicitly known, and systems are observed via sensors. In such cases, an additional step is required to identify events in the numerical data, a task referred to as *time series discretization* (Figure 3). The choice of the discretization method, for

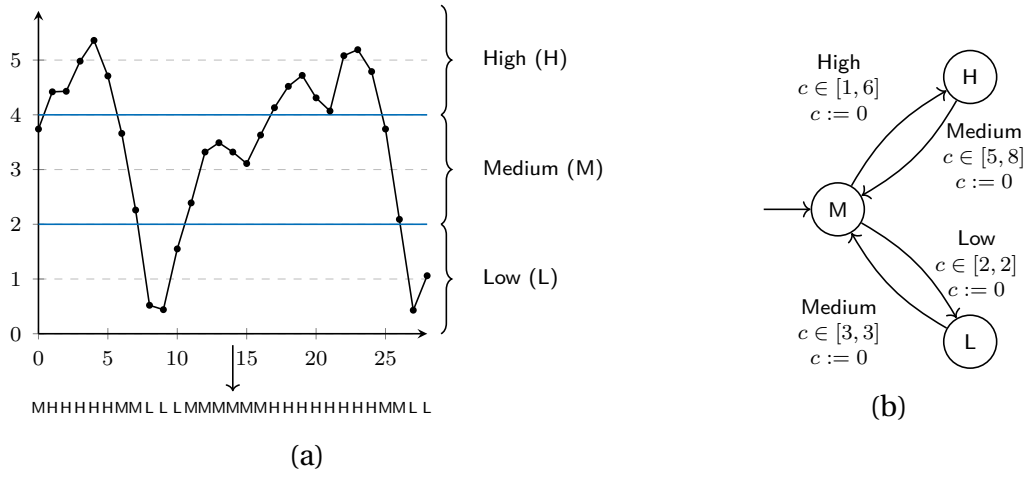


Figure 3 – From a time series to a Timed Automaton. Here, the time series is discretized by binning its value range. The resulting discrete values (Low (L), Medium (M) and High (H)) are interpreted as events for the TA learning.

instance SAX (Lin et al. 2003) or Persist (Mörchen and Ultsch 2005), can strongly impact the resulting discrete event sequences and, subsequently, the learned TA. Hence, accurately discretizing time series is a crucial challenge to enable successful TA learning.

Interacting Timed Automata learning

While an algorithm has been proposed to simultaneously learn interacting TAs given interaction sequences (Vain, Kanter, and Anier 2019), learning interacting TAs from event sequences without knowing the interactions remains an underexplored area. When there are no shared events in the event sequences, it has been proposed to independently learn TAs from event sequences recorded at the same time and to synchronize the TAs using shared clocks (Vain, Miyawaki, et al. 2009). Learning

interacting TAs from time series is even less explored. One work has been done to model biological regulatory networks (Ben Abdallah et al. 2017). Starting from pre-discretized multivariate time series and a priori knowledge of which components interact, they were able to learn small TAs whose transitions are conditioned by the others. The time series were pre-processed because an inadequate discretization can hide the synchronizations. Thus, the final challenge is to learn interacting TAs from time series without further knowledge.

Contributions and publications

To address these challenges, this thesis makes four contributions to the fields of Discrete Event System (DES) modeling, TA learning, time series discretization, and anomaly detection:

1. A novel algorithm for learning TAs from timed event sequences;
2. The first time series discretization method designed for Discrete Event System (DES) models learning;
3. A time series anomaly detection approach based on ensembles of TAs;
4. The first synchronization-preserving discretization algorithm for multivariate time series, allowing learning interacting TAs from time series.

We now describe each contribution in more detail.

First contribution. The first contribution of this thesis extends the state of the art in TA learning. We propose a new algorithm, called TAG, to learn TAs from timed event sequences corresponding to executions of the system. The learned automaton has a single clock and the state change transitions are deterministic given an event sequence. Experiments shows that TAG achieves a better balance between generalization and precision than the state-of-the-art algorithms. This work has been the subject of two publications in international conferences:

— Lénaïg Cornanguer (May 2021), « Passive learning of Timed Automata from logs (Student Abstract) », *in*: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.*18*, Section: AAAI Student Abstract and Poster Program, pp. 15773–15774

— Lénaïg Cornanguer, Christine Largouët, Laurence Rozé, and Alexandre Termier (June 2022), « TAG: Learning Timed Automata from Logs », *in*: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.*4*, pp. 3949–3958, ISSN: 2374-3468, 2159-5399, DOI: 10.1609/aaai.v36i4.20311

It has also been presented during two invited talks in the following workshops and seminars:

— SUPSEC Workshop Fall 2022: AI for supervision (September 2022, Rennes).

— IRIF Verification Seminar (March 2023, Paris).

Second and third contributions. The second contribution is in two parts, it is dedicated to the identifications of what makes a good discretization to learn TAs. First, we studied an existing time series discretization method called Persist (Mörchen and Ultsch 2005), which had a property that interested us. We modified it to make it more suited to produce discrete event sequences relevant for TA learning. This work was presented in a workshop:

— Lénaïg Cornanguer, Christine Largouët, Laurence Rozé, and Alexandre Termier (Feb. 2023), « Persistence-Based Discretization for Learning Discrete Event Systems from Time Series », *in*: *MLmDS 2023 - AAAI Workshop When Machine Learning meets Dynamical Systems: Theory and Applications*, Washington (DC), United States, pp. 1–6

Then, we identified other criteria and combined them all in a new discretization method, specifically designed for TA learning called MOODES. This method, which is the third contribution, has the particularity to produce multiple discrete event sequences for a same time series. We exploited this particularity to learn ensembles of TAs and to perform explainable and online anomaly detection in time series. A paper gathering MOODES and the anomaly detection approach is in submission for an international publication.

Fourth contribution. The last contribution is an ongoing work. In this work, we address the problem of learning interacting TAs from time series. The difficulty lies not only in identifying events in time series, but also in identifying synchronization events across multiple variables. We propose a new discretization algorithm for multivariate time series that aims to identify and preserve the synchronizations across

variables, if any. The resulting event sequences can be used to learn interacting TAs with a classical TA learner. A paper is in preparation.

Thesis outline

This thesis is divided into two parts. Part I lays the foundation of TA learning for DES modeling. Observational data of such systems typically consist of discrete event sequences.

Chapter 1. This chapter details the background on the representation of a system using the Discrete Event System subclass. Then, we focus on TA, the formalism chosen in this thesis. Finally, we present the state of the art in TA learning, especially in passive setting, i.e., from historical data and without interaction with the system.

Chapter 2. In this chapter, we introduce Timed Automata Generator (TAG), a novel algorithm for passively learning TAs from timed event sequences. It is extensively evaluated on synthetic data, along with a comparison to state-of-the-art algorithms. An additional experiment on television program logs concludes the chapter and illustrates the interpretability of the learned models.

In Part II, we move from discrete event sequences to time series. In order to bridge the gap between the numerical data and the discrete semantics of DESs, we study in particular the discretization task.

Chapter 3. We begin this part by presenting several existing techniques for discretizing univariate or multivariate time series. We then turn our attention to another challenge in time series analysis, the discovery of rules in time series.

Chapter 4. This chapter is dedicated to the development of the discretization method specifically designed for learning DES models. We also present an application in anomaly detection in time series based on ensembles of TAs learned from historical data and modeling the normal behavior of a system.

Chapter 5. The last chapter addresses the problem of learning the model of behavior of systems with multiple components in interaction. We propose to discretize multivariate time series taking into account the synchronizations between the variables. This allows the learning of TAs in interaction.

PART I

Timed Automata learning from event sequences

INTRODUCTION OF THE PART

The classic way to understand a complex physical, or software system is to perform a manual analysis of that 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 when the system is large and has many possible states. Because of this complexity, it is not done in practice for most running systems. Instead, the systems are designed to produce detailed *logs* (timed event sequences), that can help to detect odd behaviors on the fly, or at least allow performing a post-mortem analysis after a failure happened.

In this part, we tackle the problem of discovering a *human-understandable global temporal model* of the system that has generated the event sequences.

We first present a comprehensive review of the state of the art in modeling the behavior of Discrete Event System (DES) by exploring various modeling formalisms, before focusing on Timed Automaton (TA) and on the problem of inferring a TA from event sequences. We then present Timed Automata Generator (TAG), a novel TA learning algorithm with one unique parameter, which produces models that achieve a better balance between precision and generalization than the state-of-the-art algorithms. After a thorough evaluation of TAG, we present an application in data mining on real data by exploiting TA expressivity and tool support.

CHAPTER 1

STATE OF THE ART AROUND TIMED AUTOMATA

This chapter is dedicated to the presentation of the state of the art around TA learning from timed event sequences. First, we introduce DESs and the formalisms for modeling them. After a focus on the formalism of TA, we present the existing algorithm to automatically learn TAs.

1.1 Modeling the behavior of Discrete Event System

A system is an ensemble of interrelated entities interacting with each other and with the environment. The origin of the term system is the ancient Greek word σύστημα, which can be translated as a whole composed of several parts [1]. Modeling a system is creating an abstract representation of this whole and/or its parts and their interactions.

1.1.1 Discrete-Event System

DESs is a subclass of dynamic systems in systems theory. DES are widely used in control, monitoring, and diagnosis of dynamical systems, because they are much more convenient to conceptualize than more complex classes of systems based on ordinary or partial differential equations.

Generalities about Discrete Event Systems

Let's start with some general definitions about systems theory, an interdisciplinary study that explore complex systems as interconnected components interacting with

1. Liddell-Scott-Jones, Greek-English Lexicon (LSJ). Accessible online: https://stephanus.tlg.uci.edu/lsj/eid=104344

each other, forming a whole that is more than the sum of its parts. It provides a framework to study how multiple objects or concepts, associated together, fulfill a function. Systems theory can be applied to fields as diverse as sociology and mechanics. Here, we will focus on systems theory from a computer science perspective.

The upper part of Figure 1.1 schematically presents what is a system. A system is

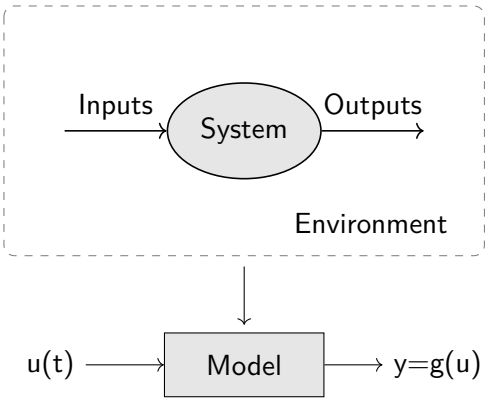


Figure 1.1 – System modeling process.

first defined by its boundaries, which delineate what belongs to the system and what belongs to its environment. In addition to its components, a system is characterized by data that provide information about its current state. This data, often collected from sensors, includes variables that are directly related to the system, and variables that come from the environment but affect the system's behavior. The variables directly affected by the system itself are called *output variables*:

$$y = [y_1, ..., y_m]^T, \text{ with } m \text{ the number of output variables.}$$

while the variables that come from the environment are called *input variables*:

$$u = [u_1, ..., u_p]^T, \text{ with } p \text{ the number of input variables,}$$

Behind the system, there is some mathematical relation between those input and output variables:

$$y = g(u).$$

The nature of this relation is one of the main way to classify systems. In the most basic systems, called *static* systems, the output of a system at a given time $y(t)$ is independent of the past values of inputs $u(\tau)$ with $\tau < t$. As soon as some memory of

the former inputs applied to the system is required to know what will be its reaction to a new input, the system is said to be *dynamic*, which corresponds to the majority of studied systems. It is then possible to determine what will be the dynamic system's behavior (i.e., its reaction) by knowing its *state*.

A system's state refers to its current condition at a particular moment, as a snapshot of it, and defines what its behavior will be. Cassandras and Lafortune (2008) give the following definition of a state:

Definition: State

The state of a system at time t_0 is the information required at t_0 such that the output $y(t)$, for all $t \geq t_0$, is uniquely determined from this information and from $u(t)$, $t \geq t_0$.

The state $x(t)$ of a system at a given time t can be determined with the initial condition of the system, its initial state, and all the input variables values since then. The set of possible values that the system's state can take, called *state space* of the system, is generally continuous, and the evolution of the system's state can be visualized as a *state trajectory*.

Example: Car system

Let's take the example of a car system. All the mechanical components of the car constitute the system, while the driver and everything around the car is its environment. We can collect data related to the car and its functioning, such as its velocity, the orientation of its wheel, the wind force, the shape of the road, and the position of the accelerator pedal controlled by the driver. It is important to note that by selecting different variables or setting the system's boundary at a different location, for example by including the driver in the system, will lead to a different definition of the system, which may be just as valid as this one.

The wind force, the shape of the road, and the position of the accelerator pedal are input variables that affect the system but come from the environment. The car's velocity and the orientation of its wheels are output variables, they are the reaction of the system to the input variables.

This system is dynamic because the behavior of the car is not constant over time given the same input. For example, the car's velocity depends not only on the current position of the accelerator pedal, but also on its position in the past, as well as the current and past slope of the road. Thus, the history of input variables deter-

mines the state of the car system, which in turn determines the value of its output variables. The state space of this system is continuous, and it can be visualized as a trajectory of the different variables, such as velocity and orientation, over time.

We now introduce a subclass of systems called Discrete Event System (DES). In a DES, the state space isn't continuous, but defined in a set of discrete states. The system's state evolution can now be visualized as a piecewise constant function with jumps from one discrete state value to another. Describing the dynamical behavior of a DES is easier because there are distinct and well-defined individual state changes.

The state change of a DES, called *transition*, is triggered by an *event*.

Definition: Event

An event is the instantaneous occurrence of something noteworthy.

An event can be a specific action, or the result of some conditions on variable values, and can occur at any time. The system is said to be *event-driven*, the dynamic is determined by the occurrence of events that trigger the state changes. Event-driven is opposed to *time-driven* where the system's state changes at each discrete time point with potentially the occurrence of an event synchronously. Cassandras and Lafortune (Cassandras and Lafortune 2008) give the following definition of a Discrete Event System (DES):

Definition: Discrete Event System (DES)

A Discrete Event System is a discrete-state, event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time.

We can define *input* and *output events*. Input events are external to the system and generated by the environment, while output variables are internally generated by the system, generally in reaction to the inputs. The state changes of a DES are triggered by the input events. Yet, an output event can indirectly cause a state change by causing a subsequent input to occur.

Example: Elevator system

Let's now consider an elevator system. This DES consists of several elevators and their controls, while the users of the elevators are part of the environment. The state of the system is the configuration of the elevators floor and doors (e.g., a first

elevator is on the second floor with the doors open, and a second is on the second floor with the doors closed). This state representation is discrete. Transitions between states are triggered by input events from the environment: a user pressing a floor or door button. In response to the input events, the output events are the elevator movements.

A DES can be stochastic or deterministic. It is important to note that the meaning of the term *deterministic* is not the same in automata theory and in system theory. A system is stochastic if at least one of its output variables is a random variable. Consequently, the behavior of such system can be described only probabilistically. In a stochastic system, the system state cannot be determined only based on the inputs history and initial state, and the initial state itself may not be precisely defined. Therefore, the evolution of a stochastic system can only be characterized by probability distributions over the state space, rather than a defined trajectory. In this thesis, we focus on deterministic DES known to be a subclass sufficient to describe a wide range of real-world systems. They are computationally less demanding to analyze than stochastic systems, and they are much more easier to conceptualize by end users, making them more practical for many applications.

Generalities about Discrete Event System modeling

Sommerville (2016) defines system modeling as "the process of developing abstract models of a system, with each model presenting a different view or perspective of that system". The first important aspect of this definition is that the model is an abstraction, a simplified representation of the system that highlights important characteristics of the complex system. Second, there is not a single model for a given system. There are a multitude of models, each designed to highlight specific aspects. The general process of system modeling is illustrated in Figure 1.1. It consists of defining a mathematical relationship between the inputs $u(t)$, the outputs $y(t)$, and the states $x(t)$, that fits the observed behavior of the system, and this mathematical relation is called the *dynamics* of the system. Modeling a system is almost inevitable when studying a system. It gives a representation of it that can be manipulated, simulated, controlled, or analyzed, and the modeling process itself helps to understand how the system of interest works. Many different representations can correspond to the same system and will provide complementary information and usage possibilities.

First, the models can be designed from different perspectives, corresponding to different aspects of the system we are interested in. There is no universal perspective classification, it depends on the kind of systems being modeled. In this thesis, we will focus on the behavioral perspective, which corresponds to the way a system reacts to environmental inputs given its state.

Example: Elevator modeling perspectives

Let's take the example of the elevator system. Some modeling perspectives could be:

— Structural perspective: How are the internal physical components of the elevators interconnected.

— Functional perspective: Which mechanisms are involved in each operation.

— Behavioral perspective: How are coordinated the movements of the different elevators to optimize efficiency and minimize user's waiting times.

Once the perspective is chosen, different levels of abstraction can be used for the representation. Cassandras and Lafortune (Cassandras and Lafortune 2008) describe three levels of abstraction to model DES:

— Untimed (or logical) abstraction level,

— Timed abstraction level,

— Stochastic abstraction level.

Again, the choice of abstraction level depends on the final task and on the system being modeled. Having more information sometimes only add unnecessary complexity. Because DES are event-driven, time may be superfluous to model the dynamic of a system. When modeling the car, a logical model may be sufficient to describe the mechanical operations. However, to model the behavior of the car while interacting with other road users, time becomes critical. In that case, a timed representation is much more useful than an untimed representation. The third stochastic abstraction level is beyond the scope of this thesis, interested readers can read the dedicated chapters of Cassandras and Lafortune (2008) for an overview.

Modeling the behavior of a DES is identifying the possible transitions between the discrete states based on the event occurrences. The behavior of a DES can be described as a sequence of events that occur during the system's execution. In this context, the set of possible events e (also referred to as *symbol*) is called the *alphabet*

and is denoted Σ. An execution of the system can be written as an *event sequence* (also referred to as *word* or *string*) where $e_i \in \Sigma$ are the successive events:

$$\langle e_1, e_2, ..., e_n \rangle$$

It is possible to include temporal information, giving a *timed event sequence* (also referred to *timed word* or *timed string*) where e_i are the successive events $e_i \in \Sigma$ and $t_i \in T$ their timestamp of occurrence:

$$\langle (e_1, t_1), (e_2, t_2), ..., (e_n, t_n) \rangle$$

The set of all (timed) event sequences corresponding to possible executions of the system is called the *(timed) language* model of the system, noted $\mathcal{L}$. The (timed) language consists of all the behaviors of the system. However, manipulating directly a (timed) language model of a DES is not convenient for many applications, and this raw information is hard to grasp/comprehend. Fortunately, many modeling formalisms can be used to represent those languages and manipulate them more easily. We describe in the next sections multiple common untimed and timed formalisms for DES modeling.

1.1.2 Untimed models

The behavior of a DES can be described using an untimed formalism. In such models, time is considered qualitatively with the ordering of the events. Modeling a DES using an untimed formalism is less complex than using a higher level of abstraction, the resulting model is easier to comprehend, and its use is less computationally demanding. We introduce the main untimed mathematical formalisms suitable for DES modeling, namely automata and Petri nets.

Finite State Automata

The most basic way to model a DES is to use the formalism of Finite State Automata (FSAs). Finite State Automata (FSAs) is a mathematical model to represent languages composed of words (or event sequences). A FSA is a state machine, it has a finite set of states and transitions. The events are labeling transitions between states, and the set of possible events at each step depends on the current state.

Definition: Finite State Automaton (FSA)

A non-deterministic FSA A is a tuple $\langle Q, Q_0, \Sigma, \delta, F \rangle$ where:

- Q is a finite set of states,
- $Q_0 \subseteq Q$ is a finite set of initial states,
- Σ is a finite set of events or symbols called alphabet,
- $\delta : Q \times \Sigma \to Q$ is a transition function,
- $F \subseteq Q$ is a finite set of accepting (or final) states.

The language of a FSA is the set of event sequences that are *accepted* by the automaton, i.e. for which there exists a sequence of transitions, or *path*, that starts from an initial state $q \in Q_0$ and ends in an accepting state $q \in F$.

Definition: Path

A path in a FSA $A = \langle Q, Q_0, \delta, \Sigma, F \rangle$ is a sequence of transitions

$$\langle (q_i, a_i, q_{i+1}), (q_{i+1}, a_{i+1}, q_{i+2}), ..., (q_{n-1}, a_{n-1}, q_n) \rangle$$

with $\forall i \in [1...n], (q_i, a_i, q_{i+1}) \in \delta$.

A FSA can be represented graphically by a directed labeled graph as in Figure 1.2. The language of the automaton represented in the figure is any string starting and

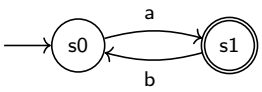


Figure 1.2 – Graphical representation of a FSA. The states are represented by circles. The transitions are represented by arrows from a source state to a destination state and are labeled by the triggering event. The initial state has an incoming arrow without source state. The final states are identified by a double circle.

ending with an a, with a symbol b between every two a (e.g. $\langle a \rangle$, $\langle aba \rangle$, $\langle ababa \rangle$), which corresponds to the following regular expression: $a(ba)^*$.

It is easy to draw a parallel between FSAs and DESs. The finite set of states of the FSA represents the states of the system, the event-triggered transitions correspond to the labeled automaton's transitions. To an execution of the system corresponds a path in the automaton, which starts from an initial state and ends in a final state, and

the language of the automaton describes the behavior of the system.

In practice, FSAs are often deterministic, meaning that if an event occurs in a current state, there is only one possible next state. Consequently, there is no uncertainty about the current state given an event sequence because there is a unique corresponding path. In a Deterministic Finite Automaton (DFA), there is only one initial state, and there is only one possible outgoing transition from a state for a given symbol.

Definition: Deterministic Finite Automaton (DFA)

A DFA $A = \langle Q, Q_0, \delta, \Sigma, F \rangle$ is a FSA where:

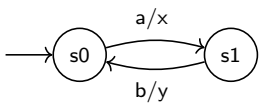
— $q_0 \in Q$ is the only initial state,

— $\forall q \in Q, \forall a \in \Sigma, |\delta(q,a)| \leq 1$.

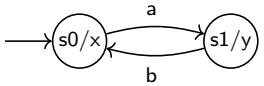
The FSA in Figure 1.2 is deterministic.

Mealy and Moore machines

Mealy and Moore machines are a special kind of DFA with output events in addition to the classical inputs events that form the alphabet Σ of a DFA, and no final state. Both machines respond to environmental inputs by producing outputs. In a Mealy machine (Figure 1.3a), the output is determined by both the current state and the input event. The output labels the transition along the corresponding input. In contrast, in a Moore machine (Figure 1.3b), the output is determined by the current state only. The output label the corresponding state.



(a) Mealy machine.



(b) Moore machine.

Figure 1.3 – Graphical representations of a Mealy machine and a Moore machine. In a Mealy machine, the transitions are labeled with an input (a and b) and an output (x and y). In a Moore machine, the transitions are labeled with an input and the states with their name and output.

Definition: Mealy machine

A Mealy machine M is a tuple $\langle Q, q_0, \Sigma, O, \lambda \rangle$ where:

— Q is a finite set of states,

— q_0 is the initial state,

— Σ is the input alphabet,

— O is the output alphabet,

— $\delta : Q \times \Sigma \to Q$ is a transition function,

— $\lambda : Q \times \Sigma \to O$ is a output function.

Definition: Moore machine

A Moore machine M is a tuple $\langle Q, q_0, \Sigma, O, \lambda \rangle$ where:

— Q is a finite set of states,

— q_0 is the initial state,

— Σ is the input alphabet,

— O is the output alphabet,

— $\delta : Q \times \Sigma \to Q$ is a transition function,

— $\lambda : Q \to O$ is a output function.

Generally, fewer states are needed for a Mealy machine than for a Moore machine because multiple transitions labeled with different outputs can have the same source state. The main difference lies in their reactivity. In practice, the current state of these machines is updated at regular intervals (on clock ticks). When an input is received, the output of the Mealy machine is updated immediately while the output of the Moore machine is changed on the next tick when the state has changed. Depending on the application, one or the other characteristic may be more desirable.

Common applications for Mealy and Moore machines include models for cipher machines in cryptography, digital circuit design (e.g., for timing control and data transmission), control of reactive systems that require inputs and outputs (e.g., for sensors and actuators), and design and implementation of communication protocols.

Petri nets

One of the alternative to automata for DESs modeling is Petri nets, proposed by Petri (1962). In contrast to FSAs, they are not specifically designed to represent languages. Petri nets can represent concurrent processes with shared information or resources, while FSA are designed to represent sequential processes with a single event at a time. Consequently, the structure and the interpretation differ. Let's take an example to introduce the Petri net structure:

> **Example: Petri net of a computer system**
>
> Let's consider a computer system where a resource (e.g. a memory area) must be allocated to some processes. The resource allocation is a system event. This event corresponds to a *transition* in the Petri net. Before performing this resource allocation, the resource must be available at the right place. The resource corresponds to *tokens* in the Petri net, and the *pre condition* about the resource availability is checked in *places* where the tokens must be. When the pre condition holds, the transition is *enabled*, i.e., that it can be fired, but the firing is not mandatory. The firing of the transition corresponds to the event occurrence: the resources are consumed so the tokens are removed from the pre condition place, and a resource appears somewhere else, in the *post condition* place.

A Petri net has *places*, *transitions*, *arcs*, and *tokens*. An arc goes from a place to a transition or from a transition to a place. The places at the source (resp. destination) of the arc are called input or preset (resp. output or postset) places. The places hold a number of *tokens* and the distribution of tokens over the places is called *marking*. The marking will constrain which transitions are enabled, i.e., which could occur. Arcs have a weight which corresponds to a tokens number. A transition is enabled if all its input places have at least the token number associated with the connecting arc. When a transition is fired, the input places lose as many tokens as the weight of their connecting arc, and the output places gain as many tokens as the weight of their connecting arc. The sum of lost tokens by the input places can be different from the sum of tokens gained by the output places, meaning that there might be an infinite set of possible markings for a finite Petri net.

Definition: Petri net

A Petri net N is a tuple $\langle P, T, F, W, m_0 \rangle$ where:

— P is a finite set of places,

— T is a finite set of transitions,

— $F \subseteq (P \times T) \cup (T \times P)$ is a set of (directed) arcs, or flow relation,

— $W : ((P \times T) \cup (T \times P)) \to \mathbb{N}$ is the arc weight mapping,

— $m_0 : P \to \mathbb{N}$ is the initial marking representing the initial distribution of tokens.

A Petri net can be represented graphically as in Figure 1.4, and this graphical representation is sometimes called a Petri net graph to distinguish it from the mathematical model Petri net.

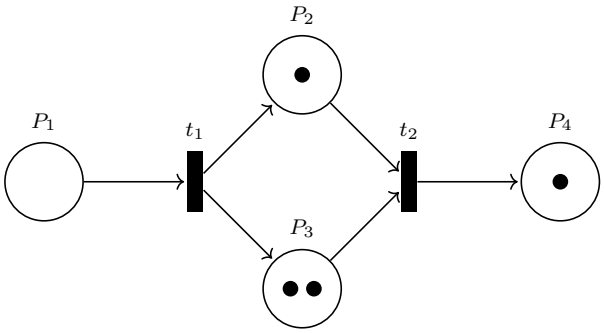


Figure 1.4 – Graphical representation of a Petri net. The places are represented by circles. The transitions are represented by rectangles. Tokens represent resources held by a place, and are symbolized by a black dot. The absence of label on the arcs means that their weight is 1. With this marking (token distribution), only transition t_2 is enabled.

Petri Nets are mainly used to model systems with concurrent events with constraints on the concurrence, precedence, frequency of these occurrences (Peterson 1977). For example, Petri nets are well suited to model manufacturing systems, transportation systems, or biological systems.

Petri nets and FSAs expressiveness are not equivalent.

The language of a Petri net represents the set of all possible sequences of transition occurrences that it can generate, while the language of a FSA is the set of all possible sequences of events it can generate. While it is in some cases possible to translate a FSA into a Petri net without changing the formalism (Badouel, Bernardinello,

and Darondeau 2015), it is sometimes necessary to label the transitions of the Petri net. Instead of recognizing sequences of transitions, a labeled Petri net will recognize sequences of labels along the runs of the Petri net. This enables a labeled Petri net to recognize any language that can be recognized by a Finite State Automaton (Valk and Vidal-Naquet 1981). Conversely, not all languages of Petri nets are regular languages, consequently they cannot be represented by a regular expression or a FSA.

1.1.3 Timed models

When timing becomes critical for a DES, the formalisms presented in the previous section may not be expressive enough to accurately represent the system's behavior Time has to be considered quantitatively, and with ways to constrain events accordingly. First, we give various definitions related to time representation, and then various timed extensions of automata and Petri nets.

Modeling time

Furia et al. (2010) have realized a survey on time modeling in different domains of computing. We only present here the notions of *clock*, *delay*, *timer*, *age*, and *lifetime* that will be used in the timed formalisms presented later.

Let's first define clocks and the related notions.

Definition: Clock

A clock c is a non-negative real-valued variable, whose value increases continuously at a constant rate as time elapses.

Definition: Clock valuation

A clock valuation v on a set of clocks $c \in C$ over a domain T is a function $v : c \to T$ which assigns a time value to each clock.

The domain T can be the ensemble of non-negative real numbers $\mathbb{R}_{\geq 0}$, the ensemble of non-negative integer numbers $\mathbb{Z}_{\geq 0}$, or a given set of timestamps. If we consider time as a discrete domain, we can define a *clock tick*.

Definition: Clock tick

A clock tick is a discrete event that simulates the passage of time by incrementing the value of a clock variable.

In addition to the natural evolution of the clocks, two operations can be performed on them: they can be reset, and their value can be checked.

Definition: Clock reset

A clock reset $c_r : c \to 0$ is an operation assigning the value 0 to the clock c.

A clock reset allows to change the reference time point to measure the elapsed time since a specific moment.

The value of the clocks can be checked to constraint a behavior according to time.

Definition: Clock constraint

For a set C of clocks defined on T, with $x, y \in C$ and $t \in T$, a clock constraint α is of the form $\alpha = x \sim t \mid x - y \sim t \mid \neg\alpha \mid (\alpha \wedge \alpha)$, where$\{<, \leq, =, \geq, >\}$.

Other concepts are specifically designed to address the relative representation of time. We first introduce *delays*, that are usually used to represent waiting time between processes. Delays are a way to measure time relatively to a specific event.

Definition: Delay

A delay δ is the positive difference between two clock valuations.

A delay can be measured using a clock by resetting it at the time we want as reference, and then checking its value. Otherwise, it is possible to constraint a behavior by using a delay interval.

Definition: Delay constraint

A delay constraint for a delay δ is an expression of the form $t_1 \leq \delta \leq t_2$. It is usually specified using an interval $[t_1, t_2]$ with $t_1, t_2 \in T$ and $t_2 \geq t_1$.

Timers are an alternative way to consider time in a relative manner. They are mainly used when an action needs to be triggered after a specific amount of time. In contrast to clocks variables whose value increases, timers are set to a specific value and then decrease until reaching zero.

Definition: Timer

A timer is a positive real-valued variable, whose value decreases as time elapses.

The advantage of timers over clocks is that there is no need to keep in memory clock constraints, but the time constraint is pre-defined and precise.

Finally, *age* and *lifetime* are two terms to express the value of a clock that is (generally) set to zero at a specific event, and increases continuously without restriction nor reset.

Timed versions of Moore and Mealy machines

Before describing the classical timed version of FSAs, we describe the timed version of Moore and Mealy machines which are less complex. Multiple timed versions of Moore and Mealy machines were proposed in the literature. Indeed, the simplicity of Moore and Mealy machines and their inputs and outputs make of them popular formalisms for some applications.

Timed Moore Machines (TMM) were proposed as a less expressive alternative to the classical timed version of FSAs for applications such as manufacturing control systems design and asynchronous circuit design. The states have a lifetime (finite or infinite) and the transition function δ is decomposed into an external transition function δ_{ext} that specifies state changes due to external inputs and an internal transition function δ_{int} that specifies state changes due to the state lifetime.

Time delay Mealy machines were introduced to model PLC software (Caldwell, Cardell-Oliver, and French 2016). They accept any input at any time, have a unique clock that is reset at each input and that measures the time elapsed before the next output, and have outputs associated with guards on these delays.

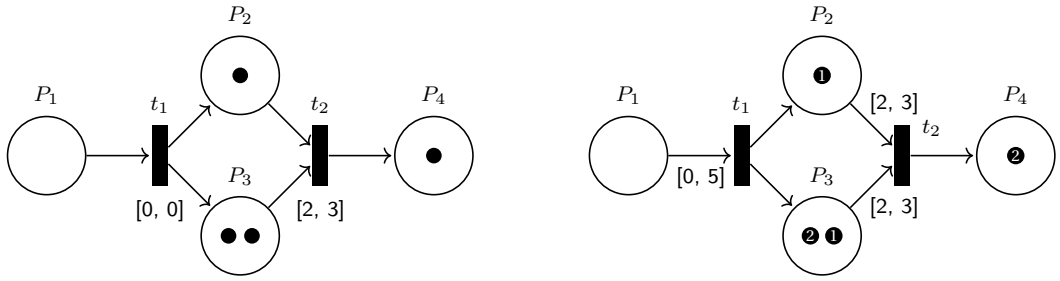
Mealy machine with timers (MMT) were introduced by Vaandrager and Jonsson (2018) for communication protocol modeling. They use timers instead of clocks, and timer's value can be set to a specific value or paused on a transition. There are no guards, when a timer reach zero, a timeout internal input occurs.

Timed versions of Petri net

Several ways to include timing variables in Petri nets have been proposed, often named by identical or similar names, which can create confusion. The timing variables are either associated to the places, the transitions, the arcs, the token, or a combination of these. Their signification also varies, from delay, age, to duration. The following list is not exhaustive.

The first attempt was proposed by Ramchandani (1973) with the aim of representing the time to complete an action, by adding firing duration on transition: transition firing is not instantaneous. The input tokens are removed when the transition starts to be fired, and the output tokens are created only at the end of the firing. This formalism is often referred to as Timed Transition Petri Net.

Around the same time, Merlin (1974) proposed another version referred to as Time Petri Nets, where transition firing is instantaneous, but transitions cannot fire before a certain delay once enabled (Figure 1.5a). The delay is associated to the transition. If



(a) Time Petri net with delay on transitions. (b) Timed (arc) Petri net with token with age.

Figure 1.5 – Graphical representations of Petri nets with timing variables.

the interval upper bound is reached the transition is fired (urgency).

Finally, Walter (1983) has proposed Timed Petri nets, where the time constraints are associated with the arcs from a place to a transition instead of the transition directly. In this model, tokens have an age, which is the time elapsed since their creation. A token in place p can be used by a transition t if its age satisfies the constraint on the flow relation from p to t. One often uses the term Timed arcs Petri nets to refer to this model.

Merlin's Time Petri Nets seem to be the most popular timed version of Petri nets. For a survey on the role of time in Petri nets, we refer readers to Bowden (2000).

Timed Automata

Timed Automata (TAs) are FSA extended with of set of clocks, timing constraints, and clock resets on transitions. They were introduced by Alur and Dill (1994) and have since been extensively studied in the literature. Figure 1.6 is a graphical representation of a TA.

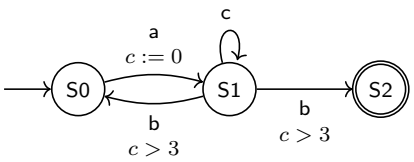


Figure 1.6 – Graphical representations of a TA. The clocks (here only c) can be reset ($c := 0$) and their values constraint the transitions.

In a TA, the concepts of *state* and *transition* differ from those in an untimed automaton. The state of a TA is a couple composed of a location (equivalent of a FSA state) and a clock valuation: (q, v). Thus, a transition between two states can either be time elapsing, or an effective transition between locations:

— Delay transition: $(q, v) \xrightarrow{\tau} (q, v + \tau)$,

— Discrete transition: $(q, v) \xrightarrow{a} (q', v')$.

In the original definition, the clock constraints ($Const(C)$) are associated with the (discrete) transitions and referred to as *guards*, but many definitions may also include clock constraints on locations referred to as *invariants* (not discussed here) (Alur 1999).

This leads to the following definitions.

Definition: Timed Automata (TAs)

A non-deterministic Timed Automaton $A \in NTA$ is a tuple $A = (\mathcal{Q}, \Sigma, C, \mathcal{E}, Q_0, F)$ 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,

— $\mathcal{E} \subseteq \mathcal{Q} \times \Sigma \times Const(C) \times 2^C \times \mathcal{Q}$ is a finite set of transitions,

— $Q_0 \subseteq \mathcal{Q}$ is a finite set of initial locations,

— $F \subseteq Q$ is a finite set of final locations.

The tuple $(q, a, g, r, q') \in \mathcal{E}$ is a transition where:

— q and q' are the source location and destination location, respectively,

— a is the symbol that triggers the transition,

— g is a guard i.e., clock constraints that must be satisfied for the transition to be enabled,

— r is the set of clocks being reset on the transition.

A *path* in a TA is a sequence of transition that can be followed to reach a given state q from an initial state q_0:

$$q_0 \xrightarrow{g_1,a_1} q_1 \xrightarrow{g_2,a_2} \dots \xrightarrow{g_p,a_p} q$$

The *timed language* of a TA is the set of timed event sequences for which there exists a path from an initial location to an accepting location q_p in the automaton. There is a *run* of the automaton along this path, i.e., a sequence of state transitions:

$$(q_0, v_0) \xrightarrow{\tau_1} (q_0, v_0 + \tau_1) \xrightarrow{a_1} (q_1, v_1) \xrightarrow{\tau_2} (q_1, v_1 + \tau_2) \dots \xrightarrow{a_k} (q_k, v_k)$$

It is important to note that here, we defined a TA with final state acceptance, however there exists many more acceptance conditions such as the Büchi and Muller acceptance conditions (Alur and Dill 1994). In function of this acceptance condition, the definition of the timed language will be different.

TAs are mainly used in model-checking, controller synthesis, and planning.

1.1.4 Conclusion on Discrete Event Systems modeling formalisms

DESs are event-driven systems whose behavior can be defined in terms of sequences of events over time. This behavior can be modeled on different levels of abstraction, from a logical to a timed and probabilistic representation. The main formalisms for such models are finite automata and Petri nets, and their various extensions.

In this thesis, we are interested in modeling DESs from observational data, where time is critical for the system's behavior, and with the hypothesis that the observations contain all the information needed to know the system's state. In this context, the timed level of abstraction is the most appropriate. Within the timed formalisms presented, we choose TAs over a timed version of Petri Nets for the following reasons:

— Language representation: TAs are specifically designed to represent timed languages, and their dedicated structure is more convenient for a precise analysis of the temporal behavior;

— Expressivity: TAs have been proven to be more expressive than (bounded) Time

Petri Nets (Bérard et al. 2005);

— Extensive literature: TAs and their properties have been extensively studied and used, forming a larger available literature;

— Composability: In order to represent complex systems, TAs can be easily composed to form a complex model, in contrast to timed versions of Petri nets;

— Tool support: Many tools have been developed to use TAs, in particular for simulation, model-checking, and controller synthesis.

1.2 A focus on Timed Automata

The formalism of TA being in the center of this work, we make a focus on what are the usual practical uses of these models, and we define some common subclasses of TAs that can be found in the literature.

1.2.1 Decision problems for Timed Automata

Besides being useful by design by allowing the modeling of systems, TAs are specifically adapted for system's properties verification and system's control.

Model checking

Model checking is a formal verification technique to determine if the model of a system satisfies some given properties or requirements. The typical requirements are liveness (something good is ensured), safety (something bad will never happen) and reachability (something can happen). Given a state-graph model of the system, and a property, generally expressed in temporal logic, the model-checker gives a binary answer, whether the property holds or not, and provides a counter-example in the latter case.

There are two main kinds of model-checkers. The first one explores all possible systems states and check whether they are consistent with the properties. The second one, called symbolic methods (Burch et al. 1992), were introduced to confine the state explosion problem of the first ones. They manipulate states by sets and use a boolean encoding.

Symbolic methods for TA model-checking have been implemented in several tools such as KRONOS (Bozga et al. 1998) and UPPAAL (Bengtsson et al. 1996). Nowadays, UPPAAL is the main model checker for TA, with many extensions such as UPPAAL SMC (Bulychev et al. 2012) to support probabilistic models. UPPAAL queries are based on Temporal Computation Tree Logic (TCTL) (Alur, Courcoubetis, and Dill 1993), a temporal logic where time can be considered quantitatively and using bounded intervals. A basic UPPAAL query, here translated as "Are we sure to reach location $loc1$ with the value of clock c under 10?", will look like the following:

$$A <> Automaton.loc1 \text{ and } c < 10$$

where A means "for all path", $<>$ means "some state in a path". UPPAAL SMC extends the basic query language of UPPAAL with queries related to the stochastic behavior of systems. For the latter, simulations of the system are performed and some statistical tests are realized on the results to obtain the query's answer with a degree of confidence. A UPPAAL SMC query using statistical model checking, here translating to "What is the probability to reach location loc1 with the value of clock c under 10 within t time units?", will look like the following:

$$Pr[<= t](<> Automaton.loc1 \text{ and } c < 10)$$

where $Pr[<= t]$ means the "probability that the property is satisfied within t time units".
For a tutorial on how to use UPPAAL SMC, readers are referred to David et al. (2015).

Model checking on TAs using UPPAAL have been applied to the verification of communication protocols (Ravn, Srba, and Vighio 2010), model-driven software development (Yang et al. 2016), and at industrial level (Larsen, Lorber, and Nielsen 2018).

Controller synthesis

Let's consider a system's model and a set of specifications that the system should satisfy. Controller synthesis addresses how to control the model's behavior to ensure that the specifications are met. The controller can usually only partially control the model's behavior. It can therefore be seen as a game to find the best input sequences to adhere to the specifications. It can also be seen as an optimization problem.

The system is typically modeled by a TA. The controller can send inputs that will trigger TA transitions, or let time pass (Asarin et al. 1998). Controller synthesis using TA is interesting for many fields such as robotics (Zhou, Maity, and Baras 2016) or cyber-physical systems (Canadas et al. 2018), as it allows controlling the behavior of real-time systems that can have complex timing specificities.

1.2.2 Timed Automata subclasses

TAs are powerful models to describe the behavior of DESs. However, their high expressiveness comes with a computational cost, and in some cases, the full formalism may be overly complex for the task. Consequently, several subclasses of TAs have been defined for various applications, making their use and analysis more efficient. Additionally, other subclasses have been defined to match more closely to the properties of the modeled system. We present some of them that will be used in the remainder of this thesis.

Deterministic Timed Automata

The term *deterministic* carries different meanings in systems theory versus automata theory. A TA is said to be *determinist* if the run of a timed event sequence always leads to the same state sequence, and if there is one unique initial state. In such automaton, there can't be two transitions outgoing from the same location labeled with the same event and overlapping timing constraints.

Definition: Deterministic Timed Automaton
A Timed Automaton A is *deterministic* if $|Q_0| = 1$ and for every $q \in Q$ and every $a \in \Sigma$, if there exists $(q, a, g_1, r, q') \in T$ and $(q, a, g_2, r, q'') \in T$, $q' = q''$ or $g_1 \wedge g_2$ is unsatisfiable.

Determinism is a common property, most TAs subclasses can have a deterministic equivalent subclass.

One-clock Automata

A one-clock TA is a TA with only one clock. This subclass is especially convenient to model systems in order to check its properties (model checking). Notably, while

classical property checking on TAs induces a computational blow-up due to the timing constraints in comparison to property checking on untimed graphs, it is largely limited for this particular subclass and properties expressed in a specific temporal logic (TCTL$_{\leq,\geq}$, subclass of the TCTL logic) (Laroussinie, Markey, and Schnoebelen 2004).

Real-Time Automata

A Real-Time Automaton (RTA) has only one clock reset on each transition that measures the delay between two events. The time constraints are generally represented in the form of an interval of acceptable delay. It can be used to model the behavior of systems where the time dependencies are local (between two events).

In the original formulation by (Dima 2001), the temporal constraints and events are carried by the locations. For convenience, Dima (2001) also defined *transition-labeled RTA*, which is now commonly referred to as RTA and correspond to the subclass defined below.

Definition: Real-Time Automaton (RTA)

A RTA $\mathcal{A}$ is a tuple $\langle Q, \Sigma, \delta, Q_0, F \rangle$ where:

- Q is a finite set of locations,
- Σ is a finite set of symbols called alphabet,
- $\delta \subseteq Q \times \Sigma \times Int \times Q$ is a transition function with Int the set of intervals whose bounds are in $\mathbb{Q}_{\geq 0} \cup \{\infty\}$,
- $Q_0 \subseteq Q$ is a finite set of initial locations,
- $F \subseteq Q$ is a finite set of final locations.

Event-Recording Automata

In an Event-Recording Automaton (ERA) (Alur, Fix, and Henzinger 1999), each distinct event $a \in \Sigma$ is associated to a clock that is reset when the event occurs. The clocks (C_Σ) measure the time elapsed since the last occurrence of the corresponding event. This kind of automata have multiple clocks and clock resets that are easily interpretable. An interesting property of ERAs's is that any ERA can be transformed into an equivalent deterministic one.

Probabilitic Deterministic Real-Time Automata

In addition to a structure of a Deterministic Real-Time Automaton (DRTA) (cf. definition of Deterministic TA), a Probabilitic Deterministic Real-Time Automaton (PDRTA) has probability parameters: an event probability distribution and a temporal probability distribution on each location.

Definition: Probabilitic Deterministic Real-Time Automaton (PDRTA)

A PDRTA $\mathcal{A}$ is a tuple $\langle \mathcal{A}', H, \mathcal{S}, \mathcal{T} \rangle$ where:

— $\mathcal{A}' = \langle Q, \Sigma, \Delta, q_0 \rangle$ is a DRTA without final locations,

— H is a finite set of bins (time intervals) $[v, v']$, $v, v' \in \mathbb{N}$, known as the histogram,

— $\mathcal{S}$ is a finite set of event probability distributions $\mathcal{S}_q = \{Pr(a|q) | a \in \Sigma, q \in Q\}$,

— $\mathcal{T}$ is a finite set of time-bin probability distributions $\mathcal{T}_q = \{Pr(h|q) | h \in H, q \in Q\}$.

This subclass was introduced by Verwer, Weerdt, and Witteveen (2012) to allow TAs learning with positive data only.

Timed I/O Automata

To facilitate the modeling of large and complex systems, TAs can be monitored jointly using synchronizations (Alur, Fix, and Henzinger 1999). The synchronizations are classically events shared by multiple automata as presented in the Introduction. When a shared event occurs, the transitions of the TAs are synchronized.

Kaynar et al. (2006) [2] introduced a more precise definition for the synchronizations with the formalism of Timed I/O Automata (TIOA) (Figure 1.7). TIOAs are TA

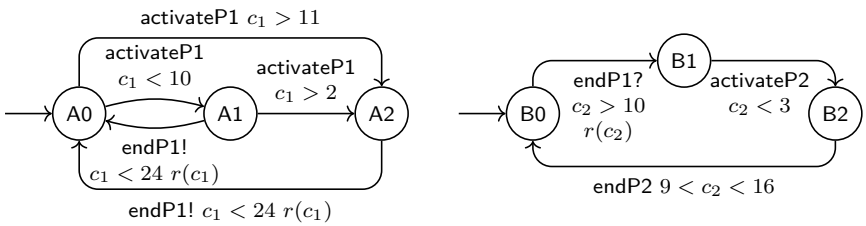


Figure 1.7 – Graphical representation of two parallel TIOAs. Pairs of input and outputs events are followed respectively by ? and !.

2. Reedited in 2011

with input and output (replacing the shared events) to model the interactions between a system and its environment, as well as between components.

Definition: Timed I/O Automata (TIOA)

A Timed I/O Automata $\mathcal{A}$ is a tuple $\langle \mathcal{B}, I, O, H \rangle$ where:

— $\mathcal{B}$ is a Timed Automaton,

— I, O, H partition $\mathcal{B}$'s event set $\mathcal{E}$ into *inputs*, *outputs*, and *internal events* respectively.

This formalism enables the parallel composition of TIOAs, allowing to synchronize their transitions via pairs of input and output. Outputs are locally controlled by at most one automaton, and transitions labeled with these events can be triggered at any time. Input event labeled transitions can be triggered only if a transition with the corresponding output event is triggered in a parallel automaton, in addition to clock constraints satisfaction. The synchronization between the output labeled transition and the input labeled transition is instantaneous.

Events that are not involved in synchronization, the internal events, are also locally controlled by the automata and cannot be shared between automata.

Another way to compose multiple TAs is to use shared clocks. It is not the case with TIOAs, the synchronization is restricted to shared pair of input/output. Therefore, parallel TIOAs can have different clock rates.

1.3 Timed Automata learning

Presently, TAs used for model checking or controller synthesis are most of the time modeled by a system's expert. This process is time- and money- consuming, and demands a complete and unbiased knowledge about the system. To allow a wiser use of TAs, it would be more convenient to have methods to automatically infer such models.

TA learning is the task of inferring a TA H that is equivalent of a target TA $\mathcal{A}$, i.e., that accepts the same language. Many TA learning approaches are based on grammatical inference algorithms which is the process of learning grammar and untimed automata, an active field of research in the 70s (De la Higuera 2010).

Two main directions are being explored: the active learning and the passive learning of TAs. In active learning, the model is learned via interactions with the system to

confirm or infirm hypotheses about it. In passive learning, the model is learned from data related to the system, usually taking the form of event sequences.

The main challenge of TA learning remains the identification of multiple clocks and their resets. Most of the existing approaches focus on the learning of a subclass of TAs to reduce the complexity of the task.

It is important to note that due to the grammatical inference heritage, the term *state* is often used in place of *location* in TA learning by analogy with the untimed automata learning field. From now, *state* will refer to *location* unless specified, whatever the possible value of the clocks.

1.3.1 Active learning

In the active learning setting, a *learner* learns a concept from a *teacher* by asking queries (Grinchtein 2008). In active TAs learning, the concept to learn is the language of the TA. The learning process is based on two kinds of queries:

— Membership query: Does the word w belong to $\mathcal{L}(A)$?

— Equivalence query: Does the language of the hypothesized model $\mathcal{L}(H)$ is the same as $\mathcal{L}(A)$?

In the case of equivalence queries, the teacher supplies a counterexample if the answer is negative, i.e., a word from $\mathcal{L}(A)$ (language of target automaton) or $\mathcal{L}(H)$ (language of hypothesized automaton) that does not belong to the other language.

The most famous active (untimed) automata learning algorithm is the L* algorithm proposed by Angluin in 1987 (Angluin 1987). It introduced the notion of *observation table* to store the queries' answer. Once the observation table satisfies some properties, it can be used to obtain an automaton consistent with it.

Most active learning algorithms for TA are derived from Angluin's algorithm, and differ on how the observation table is filled (Henry, Jéron, and Markey 2020; R. Xu, An, and Zhan 2022). Identifying the temporal constraints is complex and causes a blow-up in the number of queries required to infer the timed language. Consequently, most algorithms learn a subclass instead of directly a TA.

Other approaches explore alternatives for the equivalence queries that are often not possible in practice. Tang et al. (2022) proposed to learn deterministic one-clock TAs with a conformance testing approach combining random testing and mutation-based testing to replace exact equivalence queries. Shen et al. (2020) also learn deter-

ministic one-clock TAs and replace the equivalence query by sampling in the context of Probably Approximately Correct (PAC) learning.

1.3.2 Passive learning

In practice, it may not always be possible to query the system to obtain membership or equivalence information. In the passive setting, no interaction with the system is allowed to guide the learning process and the learner must produce a model that is *consistent* with a given input sample S. This input sample consists of a set of words that belong to the language of the target automaton $\mathcal{A}$ (referred to as positive data, S_+), and potentially a set of words that do not belong to the language of $\mathcal{A}$ (referred to as negative data, S_-). The goal of the learning process is to construct a Timed Automaton H that is consistent with S, i.e., that correctly accepts or rejects the words in S, and ideally that has the same language as $\mathcal{A}$. Three main approaches have been proposed and are presented in the following section.

Real-Time Identification (RTI)

Real-Time Identification (RTI) (Verwer, Weerdt, and Witteveen 2012) is an algorithm for learning TA from positive and negative data. More precisely, it learns a subclass of TA called DRTAs where the timing constraints are delay between consecutive events, and with accepting and rejecting states. RTI is based on grammatical inference algorithm called Evidence-Driven State-Merging (EDSM) in a red-blue framework (Lang, Pearlmutter, and Price 1998). After an automaton initialization phase, it generalizes the model and identifies temporal constraints with state merges and transition splits. The final automaton accepts all the positive traces and rejects all the negative ones.

The learning is realized in the red-blue framework, states of the automaton are red, blue, or uncolored:

— A red state belongs to the core of the automaton that is assumed to be correct;

— A blue state is part of the fringe of the core and is candidate for a state merge;

— An uncolored state belongs to the untouched part of the initialized automaton.

The first step of this algorithm is the creation of an automaton that is almost a transposition of the input sample. It serves as a basis for the generalization that follows. It consists of constructing a tree-shaped automaton called Augmented Prefix

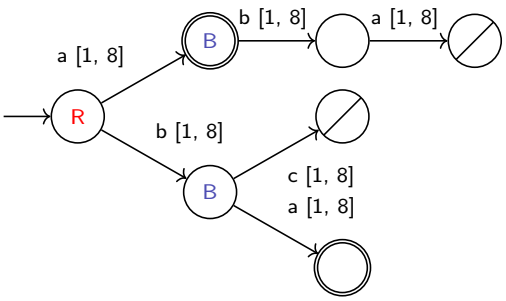
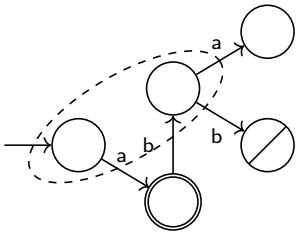


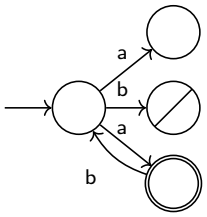
Figure 1.8 – APTA constructed from the input sample $S = S_+ = \{\langle a : 8\rangle, \langle b : 2\ a : 6\rangle\} \cup S_- = \{\langle b : 5\ c : 3\rangle, \langle a : 1\ b : 4\ a : 3\rangle\}$ in RTI. The letters R and B indicate respectively a red and a blue state

Tree Acceptor (APTA) from the input sequences, disregarding the temporal values. In this Augmented Prefix Tree Acceptor (APTA), an event sequence corresponds to a path from the initial state in the automaton, with each event of the sequence labeling a transition between two states. The state at the end of the path is either *accepting* if the sequence was from the positive sample or *rejecting* if the sequence was from the negative sample. The other states are neither accepting nor rejecting, and this is the reason this automaton is said to be "augmented". In this initial automaton, there exists only one path from the initial state to any other state, and two states share the beginning of their paths as long as the sequences of events leading to them are the same. Two sequences can be identical in their untimed version, which should imply belonging to the same set of positive or negative sequences, and corresponding to the same path. However, if they have different time values, one can be positive and the other rejecting. As a consequence, in the automaton in construction by RTI, a state can be both accepting and rejecting. At the end of the initialization, the transitions are labeled with a guard whose lower bound and upper bound are equal to the minimal and maximal time values observed in the whole input sample. Finally, states are colored to prepare the next step: the only red state is the initial state, and the destination state of its outgoing transitions (children) are blue. An example is displayed in Figure 1.8. The resulting automaton is a valid DRTA that is consistent with the input sample. However, it contains a lot of redundancy and it provides no more information than the raw input sample. To obtain a model that is more compact, generalized, and with relevant temporal information, three operations can be performed on the automaton: the merge of two states, a transition split, or a state coloring.

State merging is the main operation for the generalization. Two states can be merged



(a) Before.



(b) After.

Figure 1.9 – Merge of the two states in the dashed ellipse in RTI (example from Verwer, Weerdt, and Witteveen (2012)). The crossed out state is a rejecting state. Timed guards and state colors are not indicated.

if one is red and the other is blue, and ideally if there is no conflict in event sequences acceptance (i.e., one rejecting and the other accepting). The merge of two states consists of the accumulation of the incoming and outgoing transitions of both states into a new one, and the suppression of the old states. If one of the old states was accepting (resp. rejecting), the new state is also accepting (resp. rejecting). The new state is colored in red. A merge can cause an undeterministic transition choice from the new state and RTI must learn a deterministic automaton. Consequently, the problematic transitions and their destination states are also merged, which can recursively happen. This process is called *determinization process*, and during it, the state color requirement for the merges is suspended. When two transitions are merged, they have the same symbol but can have different timed guards. The new guard takes as lower bound and upper bound the smallest lower bound and greatest upper bound of the old ones. Finally, the uncolored children of the red states are colored in blue to maintain the blue fringe.

Transition split is the operation to extract relevant temporal information. A transition can be split if its destination is a blue state. The result of a split of transition $q \xrightarrow{[t_1,t_2]} q'$ on a time value t is two new transitions $q \xrightarrow{[t_1,t]} q_1$ and $q \xrightarrow{[t+1,t_2]} q_2$. The part of the automaton formerly accessed by q' will be divided by reconstructing the APTA from the new states q_1 and q_2. A transition split can solve the problem of a state that both rejecting and accepting.

The last operation is a state coloring and control the learning process. A blue state is colored in red if it cannot be merged with any red state.

The choice of the operation to be performed, i.e., merging, splitting, or coloring, and its location in the automaton and the value t for the split, is based on an ev-

idence value. The authors propose multiple versions of the evidence value. One of them count the number of merges of both accepting states or both rejecting states (determinization process included) induced by the operation, minus the count of merge of rejecting and accepting states.

Real-Time Identification from positive data (RTI+)

Real-Time Identification from positive data (RTI+) (Verwer, Weerdt, and Witteveen 2010) is an adaptation of the RTI algorithm to learn TA from positive data only. More precisely, it learns a subclass of TA called Probabilitic Deterministic Real-Time Automaton (PDRTA) which is similar to DRTAs, but includes probabilities for the events and the time value given the current state, and do not have final states. Learning a probabilistic model when there is no negative data to constrain the model's generalization is a classic strategy in grammatical inference. The input sample S_+ can be seen as a sample drawn from the PDRTA probability distributions. Thus, the learning task becomes finding the automaton that is the most likely to have generated the data.

The APTA creation, the state merging and the transition split operations are the same as in RTI, but without the considerations about accepting or rejecting states. The difference is that the evidence value to choose which operation to perform is replaced by a likelihood-ratio test.

Given two models H and H' (one before the operation and the other after), H being more constrained than H', the likelihood-ratio test statistic is computed as follows: $LR = \frac{\text{likelihood}(S_+,H)}{\text{likelihood}(S_+,H')}$. likelihood$(S_+, H)$ returns the maximized likelihood of the data S_+ with the model H, and depends on the probability parameters of the model. Note that the likelihood under the larger model H' will always be greater than the likelihood under H. To obtain a statistical evidence of whether the operation should be performed or not, we need to consider the number of parameters saved by the nested model H (i.e., the model obtained after the merge, or before the split), which is the goal of the likelihood-ratio test.

Timed k-Tail

Timed k-Tail (TkT) (Pastore, Micucci, and Mariani 2017) is a passive TA learning approach developed to model the behavior of software systems, to further test and

analyze them. The algorithm is specifically designed to handle potentially nested operation sequences, such as sequences of method calls where a method can call another one. Their algorithm learns a TA where an operation starts on a transition, causing the reset of a clock associated to this operation, and ends in another one, with the clock value being checked with a guard. Figure 1.10 displays an example of TA that

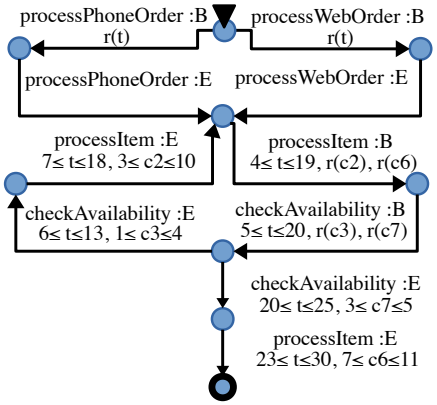


Figure 1.10 – A TA learned by Timed k-Tail (from Pastore, Micucci, and Mariani (2017)). The initial state has an incoming arrowhead with no source.

Timed k-Tail learns. It has two kinds of clocks: a global clock, t, that is only reset on the first event of a sequence and measures the time elapsed since the beginning of an execution, and local clocks c_i which are associated to pairs of transitions. Each operation has two corresponding events, the beginning indicated by B, and the end indicated by E.

Initially, a tree-shaped automaton is constructed from the input sequences where every branch that starts from the initial state is a sequence. Each pair of transitions (beginning and end of an operation) is associated to a new local clock that is reset when the operation starts and whose value is checked with a guard when the operation ends. The global clock is reset on the first transition of every sequence and checked on every transition. The clock equality constraints are set on the observed values.

The next step involves generalizing the model through state merges. The merging criterion is borrowed from a grammatical inference algorithm called k-Tail (Biermann and Feldman 1972). In the k-Tail algorithm, each state accepts a unique set of suffix strings of maximal length k called k-tail (see Figure 1.11). In Timed k-Tail, k-

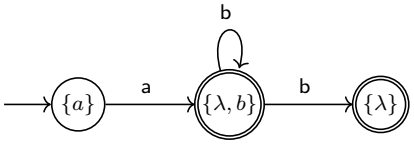


Figure 1.11 – Automaton obtained with the k-Tail algorithm for the input sample $\{a, ab, abb\}$ and $k = 1$.

tails are called k-future and correspond to sequences of events of maximal length k accepted by a state. For example, in Figure 1.10, the k-future of the initial state with $k = 2$ is the following set of sequences:

$$\{\langle \text{processPhoneOrder:B processPhoneOrder:E} \rangle,$$

$$\langle \text{processWebOrder:B processWebOrder:E} \rangle\}$$

If two states in the initialized automaton have the same k-future, they are considered to correspond to the same state and are merged together. When two states are merged, their incoming and outgoing transitions are accumulated on one state and the other state is deleted. Transitions having the same source and destination states and corresponding to the same event are also merged. Merging transitions consists of accumulating the clock constraints and resets on one transition and deleting the other(s).

The last step is the generalization of the temporal constraints. Clocks that are both reset and checked on the same transitions are considered to be the same and are renamed with the same name. The clock constraints are then summarized by keeping an interval bounded by the minimal and maximal clock equality constraint observed, potentially enlarged with a tolerance strategy.

Each clock obtained by Timed k-Tail is related to a single pair of events, it is measuring its duration. If there are no nested operations, the begin event transitions can be removed and the clocks on the end event transitions can be reduced to a delay since the last event.

GenProgTA

Tappler et al. (Tappler et al. 2019) have proposed an original approach for TA learning based on genetic programming. It is incorporated in the framework of test-

based learning enabling verification: in order to perform verification on system under test, we learn a model of it. Typically, the system is a software system on which we can execute test cases (roughly, sequences of actions) and record the system response. Therefore, they want to learn a model that produces the same outputs as the system under test in response to some given inputs. To use the formalism of TA, they need to divide the alphabet into a set of input actions and a set of output actions. The learned TA has inputs and outputs, is (input and output) deterministic, input-enabled (any input may be received in any state), and output urgent (outputs must be produced as soon as the guard is satisfied).
The outline is the following: They generate test sequences used to test the system, and get the corresponding timed traces consisting of the inputs and the outputs. Then, they use a genetic algorithm to learn a TA that should be deterministic and consistent with the set of the timed traces.

Genetic Algorithms are a kind of algorithm inspired by the biological process of natural selection (Goldberg 1989). In the general setup, a population of individuals evolves generation after generation, becoming more and more adapted to several criteria. The algorithm starts from an initial population. Every individual is evaluated on the basis of the defined criteria presented below. The better the individuals were evaluated, the greater are their chances to be selected for the reproduction step. The individuals for the next generation are created by applying crossovers and mutations to the selected individuals. A crossover consists of the recombination of two parts of two individuals. A mutation is a slight modification of the individual.

The mutations can be applied on the states (merge/split/addition), on the edges (addition/split/removing of an edge, modification of an edge destination), on the clock constraints (addition/modification/removing of a guard), on the clock resets (addition/modification/removing). The crossover of two automata consist of their product with a random selection of the guards and clock reset of one parent or the other for the edges they have in common. Finally, a simplification operation allows simplifying an automaton without changing its semantic, for example by removing unreachable states. The automata are evaluated using a fitness function that combines determinism requirement, size of the model (number of edges), and consistence with the timed traces. Each objective is assigned a weight that the user can adjust based on their preferences.

Besides the stochastic nature of the approach that can lead to different local op-

tima, the main drawback of this approach is the number of parameters. For example, the user has to specify the number of clocks, and an approximate upper bound for the clock constraints, in addition to all the classical parameters that come with a genetic algorithm (population size, number of population, operation probabilities...). However, it is the only method of this state of the art that is able to learn a TA that has multiple clock and allow clock resets without restriction.

The source code of GenProgTA, as well as the data used in the experiments, are not publicly available.

1.3.3 Conclusion on Timed Automata learning

Table 1.1 summarizes the state of the art in passive TA learning, with the algorithms advantages and limitations.

Method	**TA subclass**	**Sample**	**Approach**	**Advantages**	**Limitations**
RTI (Verwer, Weerdt, and Witteveen 2012)	DRTA	$\{S_+, S-\}$	Evidence-Driven State-Merging (EDSM) in red-blue framework	First TA passive learning algorithm	Requires negative sequences
RTI+ (Verwer, Weerdt, and Witteveen 2010)	PDRTA	$\{S_+\}$	Likelihood of the data with the model	Probabilities on the events	Loose time guards
Timed k-Tail (Pastore, Micucci, and Mariani 2017)	TA with nested operations	$\{S_+\}$	Merge of the states having the same k-future	Allows nested operations	No TA structure refinement based on time
GenProgTA (Tappler et al. 2019)	Deterministic I/O TA	$\{S_+\}$	Genetic programming	Multiple clocks and resets	Number of parameters

Table 1.1 – Summary of the passive learning approaches.

CHAPTER 2

TAG: PASSIVE LEARNING OF TIMED AUTOMATA FROM LOGS

The first contribution of this thesis is an algorithm to passively learn a Timed Automaton (TA) from event sequences, called Timed Automata Generator (TAG). TAG has a single parameter controlling the generalization level of the model. Its scalability is studied experimentally, along with a comparison with the State-of-the-Art algorithms. We also present a study case using real data and an application in data mining via model checking.

2.1 Motivation

TA is an expressive formalism to model Discrete Event System (DES). By including temporal constraints quantitatively, along with event-based transitions between states, it enables a fine description of the temporal dynamic behavior of the modeled system. Modeling a system with a TA is time-consuming, and can lead to erroneous or biased models, as it entirely depends on the expert comprehension of the system. Consequently, the automatic inference of TA has received a lot of attention in the last two decades. A first strategy is to interact with the system to iteratively construct the automaton describing the system's reaction, and is refereed as active learning. Here, we choose the passive learning strategy that only rely on observational data related to the system.

Several algorithms have been proposed to learn TAs from a sample of positive event sequences, namely Real-Time Identification from positive data (RTI+), Timed k-Tail (TkT), and GenProgTA (described in Section 1.3.2). Yet, these algorithms have drawbacks in terms of number of parameters, or in terms of 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.

2.2 General presentation

The outcome is TAG, which stands for *Timed Automata Generator*, a novel algorithm to learn Deterministic Real-Time Automata (DRTAs) from positive timestamped event sequences. It has a single parameter that controls the level of abstraction of the model. The learned automaton accepts all the event sequences of the input sample.

2.2.1 Language subclass learned

The TAs learned by TAG are DRTAs. As presented in Section 1.2, a DRTA is a TA where the temporal constraints take the form of intervals of acceptable delay since the last event. In addition, the transition triggered by the event is deterministically chosen given this delay.

During the learning process, TAG learns the DRTA structure and the delays, plus additional information about timing and transition probability that are not included in the formalism. The additional timing information relates to a *global clock* gt that measures the time elapsed since the beginning of a run of the automaton. The transition probability p corresponds to the fraction of time that a specific transition is triggered relative to the other outgoing transitions of its source state. In other words, it represents the likelihood of that transition being taken, with the sum of probabilities for all transitions from a state equaling 1. Both global clock and transition probabilities are indicative, it has no consequence on the DRTA semantic and this information is not used during the learning process. The determinism does not rely on them: only the event and the time delay since the last event are necessary to determine the next transition in a current state.

Finally, we consider that all the states of the DRTA are in the set of final states.

When displaying graphically TAG's automata (Figure 2.1), we label the states with an index assigned when the state is created. For more clarity, we also omit the constraints on the global clock and the probabilities when it is not necessary.

2.2.2 TAG's algorithm outline

TAG constructs a TA from an input sample. The input sample is composed of positive timestamped event sequences that the learned automaton must accept, meaning

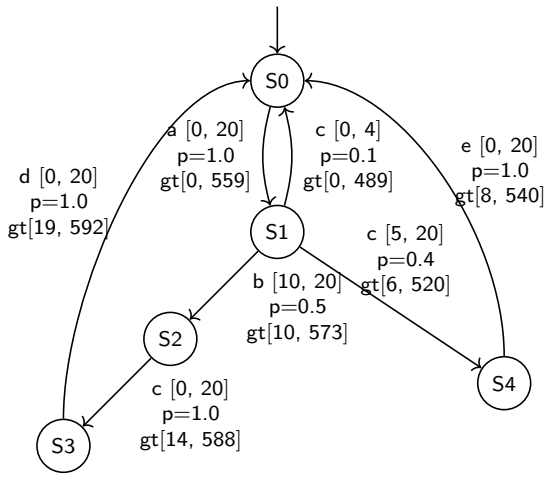


Figure 2.1 – A DRTA learned by TAG. In addition to the classical semantics, we add indicative timing constraints on the global clock indicated with a gt before the interval, and transition probabilities p. There is no particular shape for the final states because all states are final.

that there will be a path for each sequence.

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 this size reduction, it may be necessary to refine the automaton in function of the time to extract temporal determinism situations.

Those constitute the three steps of TAG:

1. Automaton initialization
2. Generalization and size reduction
3. Temporal refinement

The automaton initialization consists of transition and state creations. Then, the automaton is structurally transformed during the generalization and temporal refinement steps via two main operations: *state merging* and *transition splitting* (only on temporal refinement step).

2.3 Algorithm

Algorithm 1 summarizes the learning process. The input sample S consists of

Algorithm 1 TAG

Require: an input $S = \{S_+\} \in \mathcal{P}((\Sigma \times \mathbb{N}^0)^*)$
Return: a DRTA $\mathcal{A} = \langle Q, \Sigma, \mathcal{E}, q_0 \rangle$ consistent with S meaning that $\forall s \in S, s \in \mathcal{L}(\mathcal{A})$

1: $\mathcal{A} = \texttt{initTA}(S)$ ▷ Step 1: Automaton initialization
2: **while** some states have the same k-future **do** ▷ Step 2: Generalization and size reduction
3: $\mathcal{A} = \texttt{merge}(\mathcal{A})$
4: **end while**
5: **repeat** ▷ Step 3: Temporal refinement
6: **while** some transitions can be split **do**
7: $\mathcal{A} = \texttt{split}(\mathcal{A})$
8: **end while**
9: SAVE = $\mathcal{A}$
10: $\mathcal{A} = \texttt{merge}(\mathcal{A})$
11: **until** SAVE = $\mathcal{A}$
12: **return** $\mathcal{A}$

timed event sequences (events from an alphabet Σ, associated to delays from $\mathbb{N}^0$). After a tree-shaped automaton initialization (line 1), the second step of TAG consists in reducing the automaton size by merging all the states that can be merged together (lines 2 to 4). 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 transition splits (lines 5 to 11). 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 canceled by a split. TAG ceases when no more split or merge can be done.

We now detail the automaton initialization and the operations of state merging and transition splitting.

2.3.1 Automaton initialization

A Prefix Tree Acceptor (PTA) is a tree-shaped automaton where the states are associated with the sequence of events that label the transitions leading to it from the initial state. All the descendants of a state have the same path prefix from the initial state.

The automaton initialization consists of the construction of a PTA, a DRTA corresponding to the untimed part of the input sample, enriched with the observed time intervals. In this PTA, there is a path starting from the initial state for every sequence, and the sequences sharing a prefix share the beginning of their paths.

Algorithm 2 `initTA`

Require: an input $S = \{S_+\} \in \mathcal{P}((\Sigma \times T)^*)$
Return: a DRTA $\mathcal{A} = \langle Q, \Sigma, \mathcal{E}, q_0 \rangle$ consistent with S meaning that $\forall s \in S, s \in \mathcal{L}(\mathcal{A})$
1: $\mathcal{A} : \langle Q : q_0, \Sigma : \Sigma, \mathcal{E} : \emptyset, q_0 : q_0 \rangle$
2: **for** s in S **do**
3: $\mathcal{A}$ =`integrate`(s, $\mathcal{A}$)
4: **end for**

First, an automaton with a single state called initial state is created. Then, the initialization with the input sample is performed sequence by sequence (Algorithm 2). The order of the sequences in the sample does not change the result of the initialization operation.

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

Require: an input $s \in TS(\Sigma, T)$ and a DRTA A
Return: a DRTA consistent with $\{s\} \cup \mathcal{L}(A)$
1: $s = \langle (a_0, t_0) \dots (a_n, t_n) \rangle$
2: $q = q_0$
3: **for all** $i \in \{x | 1 \leq x \leq n\}$ **do**
4: $r = r + t_i$
5: **if** $\exists (q, a_i, g, q') \in \mathcal{E}$ **then**
6: **if not** $t_i \in g$ **then**
7: $g = [\min(g_{min}, t_i), \max(g_{max}, t_i)]$
8: **end if**
9: $q = q'$
10: **else**
11: $Q = Q \cup \{q_{new}\}$
12: $\mathcal{E} = \mathcal{E} \cup \{(q, a_i, [t_i, t_i], q_{new})\}$
13: $q = q_{new}$
14: **end if**
15: **end for**

The integration of a sequence's corresponding path in the PTA is described in Algorithm 3. For each new sequence, the integration starts from the initial state (line 2

of Algorithm 3). A sequence is composed of tuples of events and time delay with the last event. For each tuple (event a_i, time delay t_i), there are two possible cases:

— The current state has an outgoing transition labeled with the corresponding event: If needed, the guard of this transition is extended to include the new observed delay t_i (lines 7), and its destination state becomes the current state (line 9).

— The current state has no outgoing transition labeled with the corresponding event: A new transition labeled with a_i and outgoing from the current state is created, its guard is initialized with $[t_i, t_i]$ (line 12). Its destination state is also created (line 11) and become the current state (line 13).

At the end of the initialization process, the automaton is consistent with the input sample S_+, meaning that it accepts all its sequences. It accepts exactly the untimed sequences (and their prefixes, since there is no particular set of final states). However, it accepts more timed sequences than the input sample, because of the guard extensions (line 7).

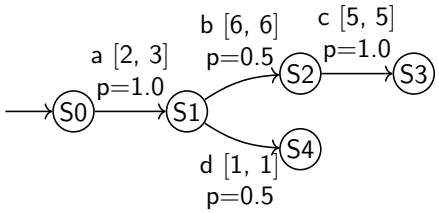


Figure 2.2 – A Timed Automaton initialized with a set of timestamped event sequences.

Example: Automaton initialization

We consider the following input sample composed of two sequences:

$$\{\langle (a,2)\,(b,6)\,(c,5)\rangle, \langle (a,3)\,(d,1)\,\rangle\}$$

Figure 2.2 presents the automaton initialized by this input sample.

First, an initial state $S0$ is created, it will be the starting state for each sequence. The first tuple of the first sequence is $(a,2)$. Starting from $S0$, there is no transition labeled with the event a, therefore, a transition is created towards a new state, $S1$, and labeled with the event a and the guard $[2,2]$. Now considering the second pair $(b,6)$, a transition must be created from $S1$ to a new state $S2$, and this transition is

labeled with the event b 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 $(a,3)$ displays an event already carried by an outgoing transition of $S0$, there is no need to create a new transition, and the time interval of the transition labeled with a is extended to $[2,3]$ to accept this new temporal value. Then, a new transition is created from $S1$ for the last pair $(d,1)$.

2.3.2 State merging

State merging is the operation that induces generalization and reduces the size of the automaton. Merge operations occur during second and third steps of TAG.

Algorithm 4 `merge`

Require: a DRTA A.
Return: a DRTA A.

1: $q_1, q_2 =$ choose_location_to_merge() ▷ based on k-futures
2: `state_merge`(q_1, q_2)
3: **while** $\mathcal{A}$ is not determinist **do** ▷ determinization process
4: e_1, e_2 = find_undeterministic_transitions()
5: $e_1 = (q_s, a, g_1, q_{d,1})$ and $e_2 = (q_s, a, g_2, q_{d,2})$
6: **if** $q_{d,1} \neq q_{d,2}$ **then** `state_merge`$(q_{d,1},\ q_{d,2})$
7: `transition_merge`$(e_1,\ e_2)$
8: **end while**
9: **return** A

The initial reason for a state merge is the similarity of to states, evaluated on the notion of k-future (described below). The states to merge are searched in a breadth-first order (line 1 in Algorithm 4). When two states with the same k-future are found, they are merged (line 2). The resulting state has a non deterministic choice of transition. Consequently, other merges not based on the notion of k-future (lines 6 and 7) must be performed to return to a deterministic automaton.

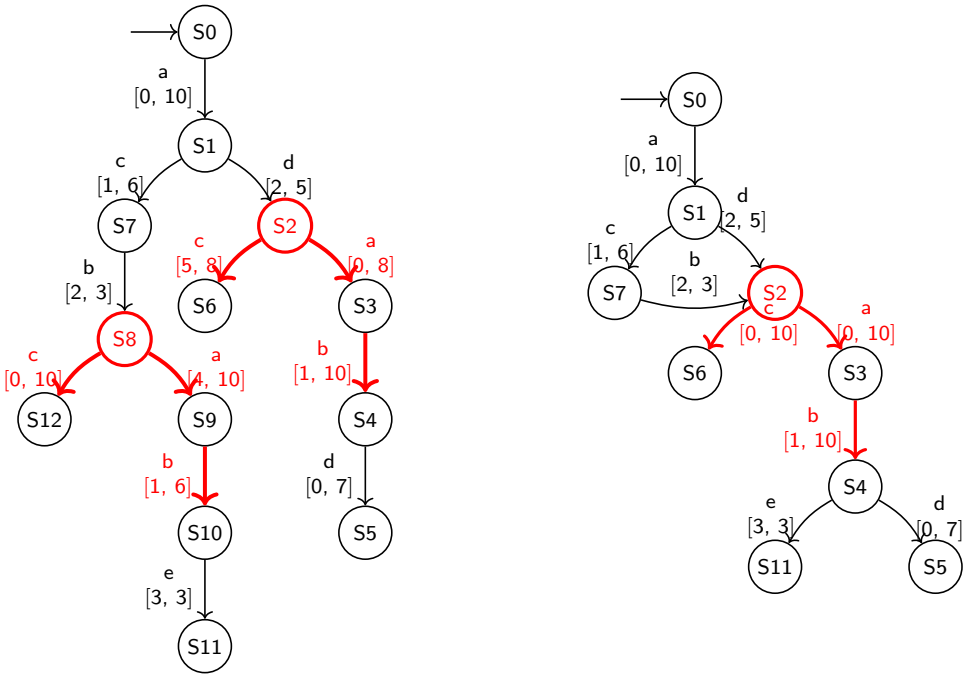
k-future

As in k-Tail and Timed k-Tail, two states are considered to correspond to the same state by TAG if their k-futures are identical, i.e., if the states offer the same paths in the short-term.

Definition: k-future of a state

The k-*future* of a state q of a DRTA $\mathcal{A}$ is the set of event sequences $\{a_1, \ldots, a_p\}$ with $p <= k$ such that there is a finite path $\langle (q, a_1, g_1, q_1), \ldots, (q_{p-1}, a_p, g_p, q_p) \rangle$ in $\mathcal{A}$.

By controlling the length of the event subsequences that are compared, the parameter k 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.



(a) Before. (b) After.

Figure 2.3 – A state merge (with determinization process).

Example: k-future

Let's consider the automaton of Figure 2.3a, and $k = 2$, the number of future events to consider per sequence. The k-future of the state $S2$ is a set of two sequences: $\{\langle a, b \rangle, \langle c \rangle\}$, with $\langle a, b \rangle$ corresponding to the path going through $S2$, $S3$ and $S4$, and $\langle c \rangle$ corresponding to the path going through $S2$ and $S6$. In this automaton, there is another state having the same k-future: $S8$. TAG will consider that these two states correspond to the same system's state and should be merged.

State merge

Two reasons can motivate a state merge: the states either have the same k-future, or the merge is required to solve a determinism issue (caused by a previous state merge).

Algorithm 5 `state_merge`

Require: a (D)RTA $\mathcal{A}$, two states q_1 and q_2
Return: a (D)RTA such that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(merge(\mathcal{A}))$.

```
for all e ∈ E_in(q2) do
    replace e = (q, a, g, q2) with (q, a, g, q1)
end for
for all e ∈ E_out(q2) do
    replace e = (q2, a, g, q) with (e1, a, g, q).
end for
Q = Q − {q2}
if q2 = q0 then q0 = q1
return A
```

Merging two states consists in accumulating their outgoing ($\mathcal{E}_{out}$) and incoming ($\mathcal{E}_{in}$) transitions on the first state and to delete the second (procedure described in Algorithm 5).

(a) Before.

(b) After.

Figure 2.4 – State merge of q_1 and q_2 (without determinization process). The states are merged because they have the same k-future (with k=1). The incoming and outgoing transitions of the two states are accumulated on one and the other is deleted. The resulting automaton is not deterministic (considering that g_1 and g_2 overlap).

As it can be seen in Figure 2.4, because of the transition accumulation, this operation results most of the time in a non-deterministic automaton where multiple transitions have the same event and the same new source state.

Determinization process

Algorithm 6 `transition_merge`

Require: a (D)RTA $\mathcal{A}$, two transtions $e_1 = (q_s, a, g_1, q_d)$ and $e_2 = (q_s, a, g_2, q_d)$
Return: a (D)RTA such that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(merge(\mathcal{A}))$.
1: let $e_1 = (q_s, a, g_1 \cup g_2, q_d)$
2: $\mathcal{E} = \mathcal{E} - \{e_2\}$
3: **return** $\mathcal{A}$

The determinization process consists in realizing all the merges necessary to determinize the automaton (lines 3 to 8 in Algorithm 4). It is necessary after a state merge based on the k-futures.

A non-determinism is caused by transitions with the same source state and same event. These transitions should be merged into one. But if the transitions have a different destination state, the merge is not directly possible because a single transition cannot lead to multiple states. Therefore, the destination states of the transitions need to be merged first. Once the transitions have the same destination state, they can be merged by extending the time interval of one of the transitions to encompass both time intervals, and removing the other transition (Algorithm 6).

Example: State merge

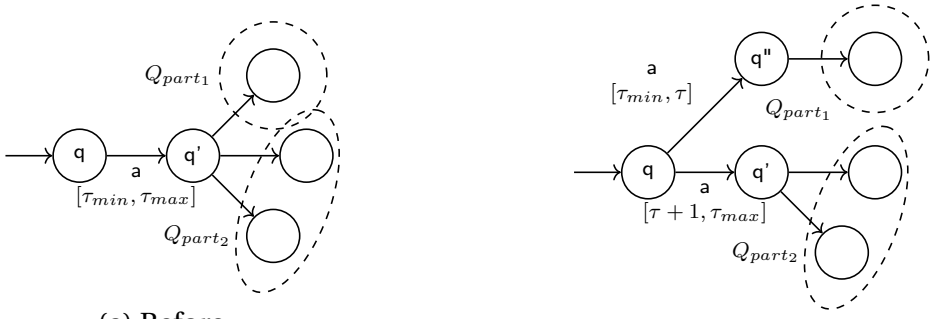
Let's consider the automaton of Figure 2.3a. S2 and S8 have the same k-future with $k = 2$ (see previous example). Figure 2.3b present the result of this merge. In addition to the merge of S2 and S8, other pairs of states and transitions have been merged during the determinization process (S6 and S12, S3 and S9, and S4 and S10).

2.3.3 Transition splitting

The transition split is the main operation of the temporal refinement. It allows distinguishing paths in the automaton where time is determinant for the behavior. It leads to the creation of an additional transition and of a new state. The split procedure relies on both automaton and input sequences analysis.

The following explanations are supported by Figure 2.5. Let's consider a transition e (Figure 2.5a):

$$e : q \xrightarrow[[\tau_{min}, \tau_{max}]]{a} q'$$



(a) Before. (b) After.

Figure 2.5 – Transition splitting.

The state q' has outgoing transitions whose destination states are either in the subsets of states Q_{part_1} or Q_{part_2} (with $Q_{part_1} \subseteq Q$, $Q_{part_2} \subseteq Q$ and $Q_{part_1} \bigcap Q_{part_2} = \emptyset$).

Let's consider the input sequences $S_e \subseteq S$ whose path goes through e.

The transition e is splittable on time value $\tau \in [\tau_{min}, \tau_{max}]$ if in $S_e \in S$:

— e is triggered with a time value $t \leq \tau$ when the path goes toward Q_{part_1},

— e is triggered with a time value $t > \tau$ when the path goes toward Q_{part_2}.

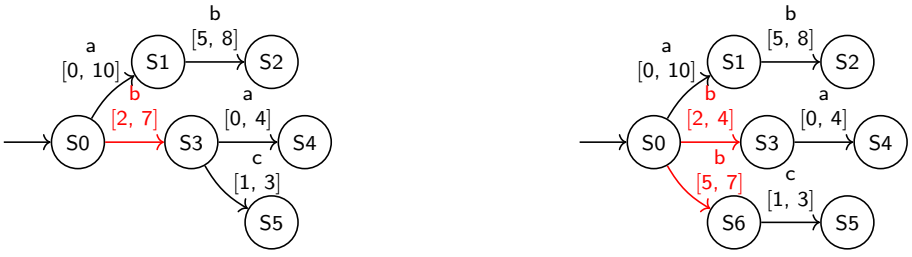
Additionally, e must be the only incoming transition of q'.

In such case, the transition e is replaced by two transitions e_1 and e_2, each one leading exclusively to Q_{part_1} or Q_{part_2} (Figure 2.5b):

$$e_1 : q \xrightarrow[[\tau_{min},\tau]]{a} q' \text{ and } e_2 : q \xrightarrow[[\tau+1,\tau_{max}]]{a} q''$$

We qualify the transitions e_1 and e_2 of *twinned transitions.*

After a split, all the sequences of the input sample are still accepted by the automaton.



(a) Before. (b) After.

Figure 2.6 – An example of transition split.

Example: Transition split

Let's consider the following event sequences:

$$\{\langle (a,2)\,(b,8)\rangle, \langle (a,10)\,(b,5)\rangle,$$
$$\langle (b,7)\,(c,1)\rangle, \langle (b,5)\,(c,3)\rangle,$$
$$\langle (b,4)\,(a,4)\rangle, \langle (b,2)\,(a,0)\rangle\}$$

They have been used to construct the automaton displayed in Figure 2.6a. From the event sequences, we can see that the transition between $S0$ and $S3$ (in red) is triggered either with a delay lower than 5, and then followed by an event a, or with a greater delay, and then followed by an event c. By knowing the time value, we already know which will be the next event. There is a temporal determinism, and the paths can be separated sooner. The result of the split is displayed in Figure 2.6b.

Algorithm 7 `split`

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

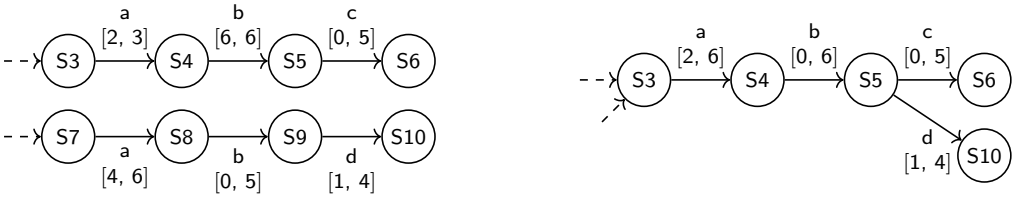
1: $(e, \tau, Q_{part_2}) = choice\_transition\_to\_split(\mathcal{A})$
2: $e = (q, a, [\tau_{min}, \tau_{max}], q')$, $\tau \in [\tau_{min}, \tau_{max}]$, and $Q_{part_2} \subseteq Q$
3: modify e guard to $[\tau_{min}, \tau]$
4: add a new state q'' to $\mathcal{Q}$
5: add a new transition $e_2 = (q, a, [\tau + 1, \tau_{max}], q'')$ to $\mathcal{E}$
6: **for** $e_{next} = (q', a, g, q_{next}) \in \mathcal{E}$ *with* $q_{next} \in Q_{part_2}$ **do**
7: change e_{next} source state to q''
8: **end for**
9: **return** $\mathcal{A}$

Algorithm 7 presents the split procedure. In line 1, a splittable transition e is identified, as well as the time value τ that create the determinism, and a subset of states Q_{part_2} that is exclusively reached when e is triggered with a time value above τ. A new transition e_2 and its destination state q'' are created (lines 4 and 5), and the guard of e is divided between e and e_2. Then the outgoing transitions of q'' are partitioned between q' and q'' according to whether their destination state is in Q_{part_2} or not (lines 6 to 8).

2.3.4 Additional considerations on the temporal refinement

During the temporal refinement step, both transition splits and state merges can be performed. Transition splits have the priority, a merge will only be considered if no split is possible. The reason for this is that a split modifies the k-future of certain states and may make a state merge possible.

During this step, an overlapping-guards requirement is added for the state merges. Besides from having the same k-future, the guards of at least one pair of transitions labeled with the same event must overlap so that it does not cancel a split. Since this requirement does not apply in step 2 (generalization step), it does not prevent the merge of states having the same k-future, unless the paths do lead to different behavior out of the scope of the k-future.



(a) Unmerged parts

(b) Merged parts

Figure 2.7 – Two parts of the automaton that would have been successively merged and then split in the absence of the guard non overlapping requirement.

Example: Overlapping-guards requirement for merging states

Figure 2.7 illustrates the need for this requirement. $S3$ and $S7$ have the same k-future with $k = 2$. In step 2, they have been merged, leading to the automaton of Figure 2.7b. In step 3, TAG identifies from the input sample that knowing the temporal value when triggering the transition from $S4$ to $S5$ allows us to know which will be the next transition. Therefore, the transition from $S4$ to $S5$ is split. The same happens to the transition from $S3$ to $S4$ and we get the automaton of Figure 2.7a.

Those splits have been applied because we already know what will be the last event (c or d) while being in $S3$ thanks to the timing information. There is a temporal determinism on the behavior, and the paths have been separated for a good reason. Without the overlapping-guards requirement, $S3$ and $S7$ would be re-merged with the determinism procedure, returning to the automaton from Figure 2.7b (and creating an infinite loop between 2.7a and 2.7b). This is the reason of the overlapping-

guard requirement in step 3, the states are not merged because the guard intersection is empty ($[0,5] \cap [6,6] = \emptyset$), and therefore it would have led to a split.

2.4 Consistency proof

Theorem. The DRTA learned by TAG is consistent with the input sample S, i.e., for a set of sequences $s \in S$, $S \subseteq \mathcal{L}(TAG(S))$ [1].

Proof. We recall that a sequence $s =< (a_1,t_1),\ldots,(a_n,t_n) >$ is consistent with an automaton $\mathcal{A}$ if there is a finite path in $\mathcal{A}$ starting in the initial state:

$$q_0 \xrightarrow[g_1]{a_1} q_1 \rightarrow \ldots \xrightarrow[g_n]{a_n} q_n$$

with $\forall i \in [1..n]\ t_i \in g_i$.

The language of $\mathcal{A}$ consists of all the timed event sequences consistent with it.

To show that S is included in $\mathcal{L}(TAG(S))$, it is sufficient to show the following three lemmas:

Lemma 1. If $s \in S$ then $s \in \mathcal{L}(Init(S))$
All sequences of S are accepted by the automaton after the initialization step.

Lemma 2. If $s \in \mathcal{L}(\mathcal{A})$ then $s \in \mathcal{L}(merge(\mathcal{A}))$
If a sequence is accepted by the automaton, then it is also accepted after a state merge in the automaton.

Lemma 3. If $s \in \mathcal{L}(\mathcal{A})$ then $s \in \mathcal{L}(split(\mathcal{A}))$
If a sequence is accepted by the automaton, then it is also accepted after a transition split in the automaton.

If these three lemmas are verified, the TA constructed by the initialization step is consistent with S, and the state merging and transition splitting operations don't change this. □

We now demonstrate the three lemmas.

1. $\mathcal{L}(TAG(S))$ is the language of the automaton learned by TAG from an input sample S.

2.4.1 Lemma 1: If $s \in S$ then $s \in \mathcal{L}(Init(S))$

Proof. The initialization consists in taking each sequence s of S and, if necessary, modifying the automaton to accept s. The modifications of the automaton can only increase the timed language of the automaton. There are three possible cases:

— There is already a path in the automaton for s: the automaton is unmodified and therefore still accepts the previously processed sequences.

— There is a path in the automaton for s if we disregard the time values: the guards are extended to accept the new observed time values, a guard g becoming $g' \subset g$, which only increase the possible time values for the event sequences.

— There is no path in the automaton for s: transitions are added from the state where the path stops, which does not suppress any existing path.

□

2.4.2 Lemma 2: If $s \in \mathcal{L}(\mathcal{A})$ then $s \in \mathcal{L}(merge(\mathcal{A}))$

Proof. A merge of two states based on identical k-future induces this first state merge, then successive state merges and transition merges during the determinization process (as detailed in Section 2.3.2). It is therefore necessary to show that the sequences $s \in S$ are still accepted after the operations of state merging and transition merging.

Merge of two states

Let's consider the states q_1 and q_2, and their sets of incoming and outgoing transitions $\mathcal{E}_{1,in}$ and $\mathcal{E}_{1,out}$, and $\mathcal{E}_{2,in}$ and $\mathcal{E}_{2,out}$, respectively. Merging two states consists in accumulating their outgoing and incoming transitions on the first state (q_1) and to delete the second (q_2).

For $e \in \mathcal{E}_{2,in}$, $e = q \xrightarrow[g]{a} q_2$ becomes $e = q \xrightarrow[g]{a} q_1$ and is added to $\mathcal{E}_{1,in}$. For $e \in \mathcal{E}_{2,out}$, $e = q_2 \xrightarrow[g]{a} q$ becomes $e = q_1 \xrightarrow[g]{a} q$ and is added to $\mathcal{E}_{1,out}$.

There are two possible cases for the sequences $s \in S$ originally going through the deleted state q_2:

— s was going through a transition $e' = \mathcal{E}_{2,in}$ then a transition $e'' \in \mathcal{E}_{2,out}$: the destination state of e' and the source state of e'' are now q_1.

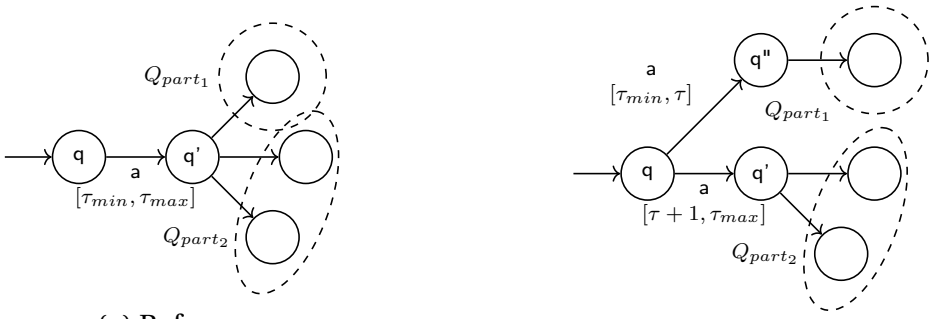
— s was going through a transition $e' \in \mathcal{E}_{2,in}$ then stopping in q_2: the destination state of e' is now q_1 and s stops in q_1.

For the sequences originally going through q_1, the original incoming and outgoing transitions remains in $\mathcal{E}_{1,in}$ and $\mathcal{E}_{1,out}$. In all cases, the paths are not discontinued. This holds for any merge of state, therefore how are selected the states to merge has no impact on the proof.

Merge of two transitions

Two transitions can be merged only if they have the same source state, the same event, and the same destination state. The result of the merge of $e_1 = q \xrightarrow[g_1]{a} q'$ and $e_2 = q \xrightarrow[g_2]{a} q'$ is a single transition $e_1 = q \xrightarrow[g_{new}]{a} q'$ with $g_{new} \subseteq g_1$ and $g_{new} \subseteq g_2$ since $g_{new} = g_1 \cup g_2$. This operation deleted no path. □

2.4.3 Lemma 3: If $s \in \mathcal{L}(\mathcal{A})$ then $s \in \mathcal{L}(split(\mathcal{A}))$



(a) Before.

(b) After.

Figure 2.5 – Transition splitting. (repeated from page 55)

Proof. As in Section 2.3.3, let's consider a transition $e = q \xrightarrow[[t_{i,min}, t_{i,max}]]{a} q'$ (Figure 2.5a). The state q' has outgoing transitions whose destination states are either in the subsets of states Q_{part_1} or Q_{part_2} from Q (with $Q_{part_1} \bigcap Q_{part_2} = \emptyset$).

In the input sequences $s \in S$, the sequence subsets σ_e and their associated path subsets π_e corresponding to the passage through e can take the two following forms:

— The sequence continues: $\sigma_e = \{...(a, t_i)(a_{i+1}t_{i+1})...\}$ and $\pi_e = ...q \xrightarrow[g_i]{a} q' \xrightarrow[g_{i+1}]{a_{i+1}} q_{i+1}\cdots$
with $g_i = [t_{i,min}, t_{i,max}]$ and either $q_{i+1} \in Q_{part_1}$ or $q_{i+1} \in Q_{part_2}$.

— The sequence stops after the transition: $\sigma_e = \{...(a, t_i)\}$ and $\pi_e = ...q \xrightarrow[g_i]{a_i} q'$
with $g_i = [t_{i,min}, t_{i,max}]$.

The transition e is splittable on time value $\tau \in [\tau_{min}, \tau_{max}]$ if for all σ_e and π_e in the sequences continuing:

— $q_{i+1} \in Q_{part_1}$ when $t_i \leq \tau$,

— $q_{i+1} \in Q_{part_2}$ when $t_i > \tau$.

For the sequences stopping after e, t_i can have any value in $[\tau_{min}, \tau_{max}]$.

The split consists in distinguishing the paths leading to Q_{part_1} from the paths leading to Q_{part_2}. A new state q'' is created, and e is replaced by two new transitions:

$$e_1 : q \xrightarrow[[\tau_{min},\tau]]{a} q' \text{ and } e_2 : q \xrightarrow[[\tau+1,\tau_{max}]]{a} q''$$

The source state of the transitions originally outgoing from q' and having a source state in Q_{part_1} is replaced by q''. When a sequence was passing through e with a delay $t_i \leq \tau$, its path now goes through e_1 and either stops or continue to Q_{part_1}, and when a sequence was passing through e with a delay $t_i > \tau$, its path now goes through e_2 and either stops or continue to Q_{part_2}. The other paths remain unchanged because q' originally only had one incoming transition, the splitted transition. Consequently, all the sequences $s \in S$ are still accepted by the automaton.

□

2.5 Experiments on synthetic data

To evaluate TAG, we addressed five questions:

Q1. How does the parameter k impact the result?

Q2. How does the state merging order impact the result?

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

Q4. How does TAG compete with the State-of-the-Art algorithms?

Q5. How does TAG scale with the complexity of the data?

This evaluation is based on synthetic data in order to have a ground truth and to control the complexity of the learning task.

2.5.1 Method

TA learning from observational data can be hard to evaluate because of the absence of ground truth about the model to obtain. As long as the automaton accepts the whole input sample, it is impossible to assess that a model is better than another

without any additional positive or negative data. In order to have a ground truth to evaluate TAG, we realized experiments on synthetic timed event sequences generated from synthetic model TAs. The goal is therefore to recover the model TA at the origin of the timed event sequences. Figure 2.6 presents the experimental workflow for each target automaton.

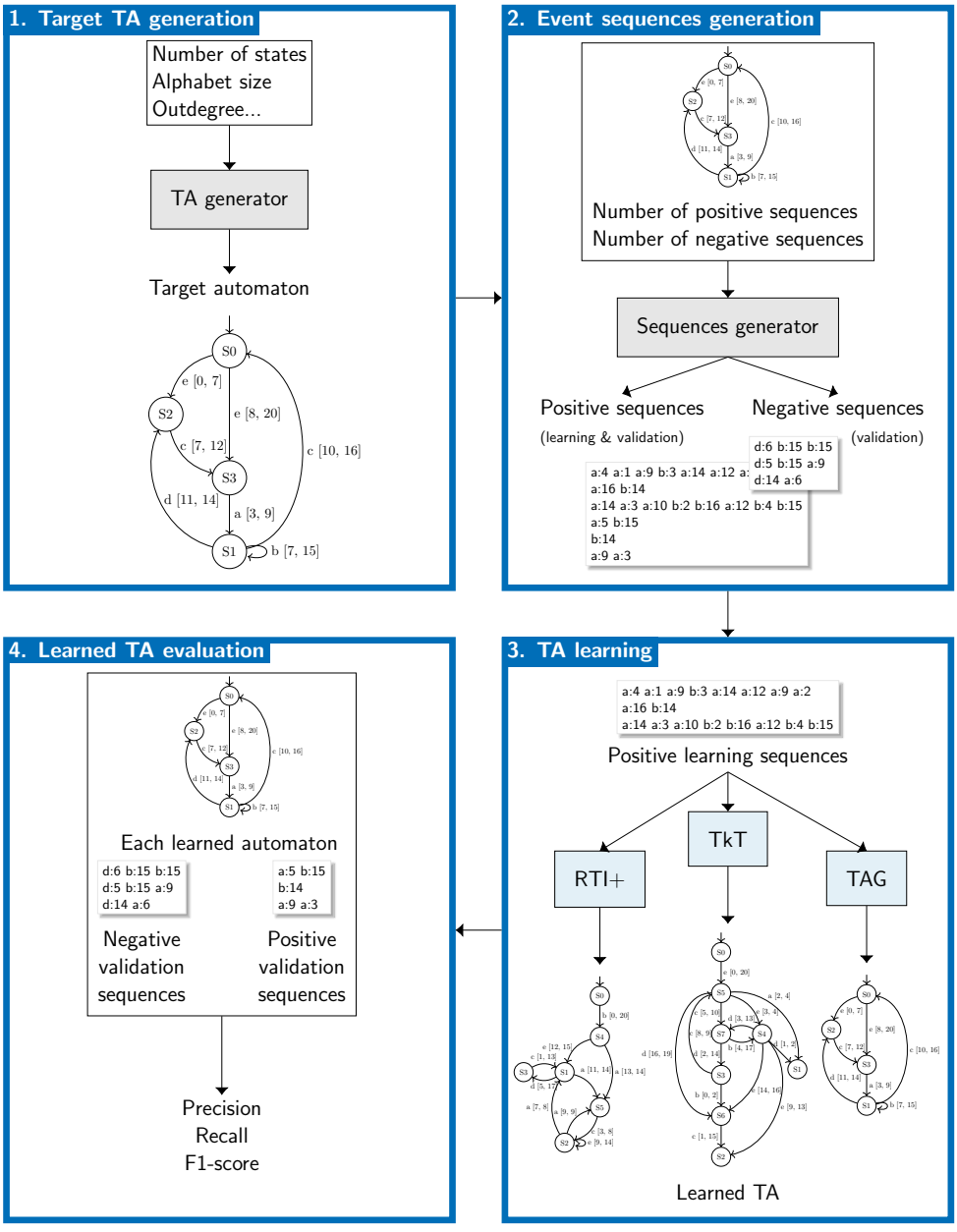


Figure 2.6 – Experimental workflow.

Timed Automata generation

The first task is to generate synthetic model TAs.

The model TAs consist in randomly generated DRTAs that are consistent with some user defined parameters. The parameters and their default value are the following:

— Alphabet size (5): number of different events,

— State number (10),

— Outdegree (1.5): average number of outgoing transitions per state,

— Twinned-transitions proportion (0.25): transitions from the same state labeled with the same event but having non-overlapping guards, these twinned-transitions create an untimed undeterminism,

— Minimal and maximal interval bounds ($[0, 20]$).

It is important to note that the alphabet size is naturally constraint by the state number and the outdegree, as well as the twinned transition proportion is constraint by the outdegree. An example of generated automaton with these default values is displayed in Figure 2.7.

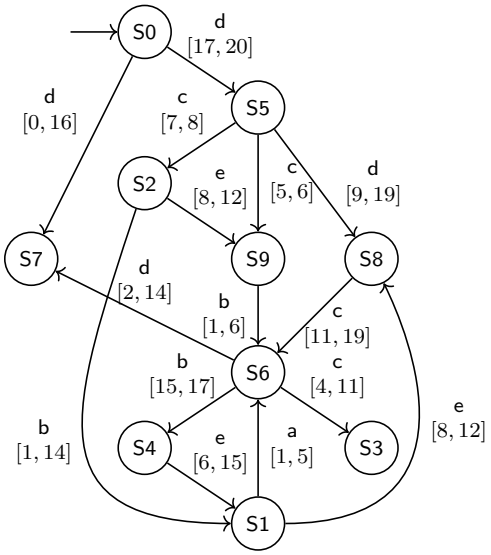


Figure 2.7 – Example of generated automaton.

The generated automaton is output in a DOT file to be easily reused by other programs.

The source code in Python is available on a public repository[2] and Appendix A presents how to use it.

Timed event sequences generation

Timed event sequences belonging or not in the language of a given TA can be generated to be used as input sample for a TA learner or to evaluate a learned model.

Positive timed event sequences

To generate positive timed event sequences, we randomly select each event from the current state's outgoing transitions, independent of the previously selected events. The timed event sequences generator can take into account the probabilities on the transitions. The delay is also chosen uniformly at random within the interval corresponding to the event's transition. If there is no designated final state, then each state has an equal probability of being the final one in the sequence. If a final state is defined, then each state is associated with a probability that the sequence will end if it passes through it. Each timed event sequence is generated independently of the others.
Other more sophisticate approaches exists such as Barbot, Basset, and Donze (2023) that proposes a sampling method that favor a better distribution over the language of the TA. It could be interesting to apply this sampling strategy for future experiments.

Negative timed event sequences

For the negative timed event sequences, the objective was to generate data that was close but not consistent with the automaton. The generation procedure is the same as for the positive timed event sequences, but with a probability to insert an error at each transition. The error can be a time value outside the guard of the selected transition, or a time value and an event corresponding to a transition outgoing from another state than the current one. There is a single error per negative timed event sequence.

2. `https://gitlab.inria.fr/lcornang/tag`

Evaluation metrics

To evaluate the learned TAs with the synthetic timed event sequences, two accuracy scores were selected. Given a set of test event sequences $tss_{test} = tss_+, tss_-$, an automaton learned using positive event sequences tss_{train} (with $tss_{train} \cap tss_{test} = \emptyset$) should accept all the positive timed event sequences tss_+ and reject all the negative timed event sequences tss_-. We define the scores using the notations of Table 2.1.

Table 2.1 – Confusion matrix.

		Consistency with learned TA	
		Accepted ($tss_{accepted}$)	Rejected ($tss_{rejected}$)
Consistency with model TA	Accepted (tss_+)	True positive ($tss_{accepted} \cap tss_+$)	False negative ($tss_{rejected} \cap tss_+$)
	Rejected (tss_-)	False positive ($tss_{accepted} \cap tss_-$)	True negative ($tss_{rejected} \cap tss_-$)

The True Positive Rate (TPR), also called recall, is the probability for an event sequence consistent with the model to be recognized by the learned automaton. It is an estimation of the part of the language of the model automaton that is well learned.

$$TPR = \frac{|tss_{accepted} \cap tss_+|}{|tss_+|}$$

The Positive Predictive Value (PPV), also called precision, is the probability for an event sequence recognized by the learned automaton to be consistent with the model automaton.

$$PPV = \frac{|tss_{accepted} \cap tss_+|}{|tss_{accepted}|}$$

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.

$$PPV = \frac{2 \times PPV \times TPR}{PPV + TPR}$$

Runtime evaluation

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. We report the median and the interquartile range, all factor's combination included.

Statistical tests

The significance of a factor effect on the scores is assessed through a variance analysis (ANOVA) with a risk $\alpha = 0.05$. The significance of a difference on the scores between two configurations is assessed through a pairwise t-test with a risk $\alpha = 0.05$. On the figures, a star indicates a statistical difference at risk below $\alpha = 0.05$.

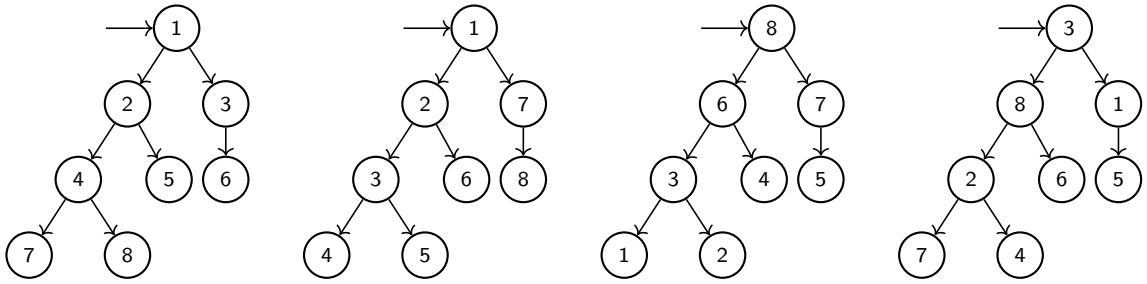
2.5.2 TAG's parameter and operations impact

These experiments were realized from 200 model TAs generated with the default values, and with 500 positive timed event sequences for each model TA for the learning plus 100 negative and 100 positive sequences for the evaluation.

Impact of the state search order for merges

The merge of two states modify the k-future of some states because of the determinization process. Consequently, depending on the order in which the states are merged, the resulting automaton may be different. To evaluate this impact and choose a state search strategy, four orders were tested for TA learning with TAG on the same synthetic data. When the situation arises in grammatical inference, in the absence of heuristic, the states are generally evaluated from the initial state of a Prefix Tree Acceptor (PTA), i.e., the states corresponding to event sequences beginning first (Oncina and García 1992; Carrasco and Oncina 1994).

Four state search orders were tested with different hypothesis. The first strategy is breadth-first search (BFS) (Figure 2.8a), in analogy to grammatical inference and with the idea that equivalent states are expected to be found at the same position in the PTA. The second strategy is depth-first search (DFS) (Figure 2.8b), which can be seen as a factorization of the automaton per timed event sequence. The last strategy is bottom-up breadth-first search (BUBFS) (Figure 2.8c), which could allow a reduction of the runtime by skipping the determinism process. Finally, as a control strategy, a random choice within the candidates for merging was also tested (Figure 2.8d).



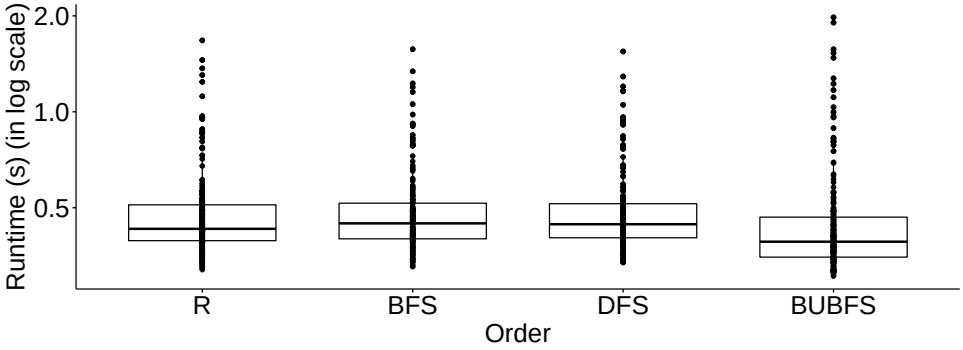
(a) Breadth-first search (BFS).

(b) Depth-first search (DFS).

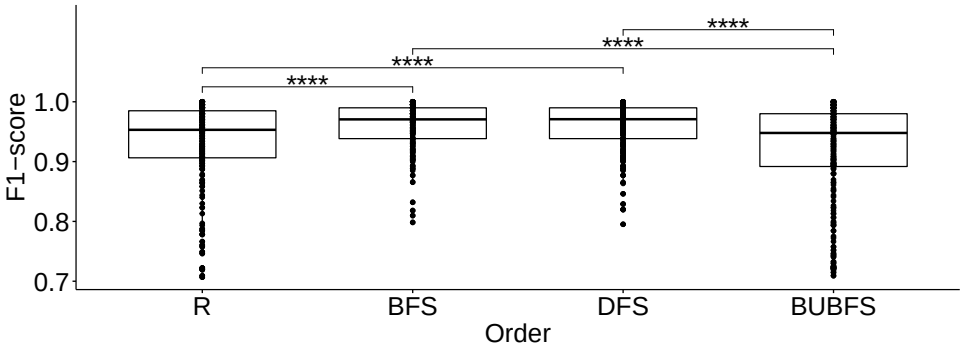
(c) Bottom-up breadth-first search (BUBFS).

(d) Random (R).

Figure 2.8 – Search order strategies for state merging candidates.



(a) Runtime.



(b) F1-score.

Figure 2.9 – F1-score and runtime in function to the search order.

The runtime (Figure 2.9a) is slightly shorter on average with BUBFS as expected, but the difference with the other search orders is not statistically significant due to the dispersion.

Regarding the recall and precision of the learned models, there is no recall differ-

ence, but the random and bottom-up breadth-first search lead to a loss of precision in comparison to BFS and DFS, resulting in a better F1-score for those strategies (Figure 2.9b).

The selected search strategy for TAG is the breadth-first search because there is no result improvement in using a depth-first search, and this strategy is more studied in the automata learning field.

Impact of parameter k

The parameter k controls the level of generalization in the learned model by tuning the degree of similarity that two states must have in order to merge.

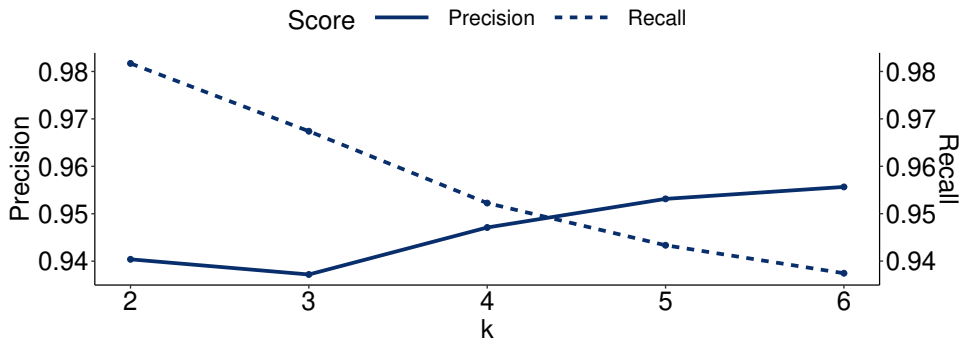


Figure 2.10 – Mean precision and recall in according to the value of parameter k.

As k increases, the precision tends to increase while the recall decreases (Figure 2.10): the model accepts less new timed event sequences and is closer to the input sample. However, the gain in precision is inferior to the loss of recall.

Besides from these experimental results, the adequate value for k is application specific. First, the dependencies between events may be more or less long. Additionally, there can be a visual impact and thus an interpretability impact if the model is meant to be human-understandable. Therefore, the default value is set to $k = 2$, letting the user freedom to tune it in function to its application and its needs.

Regarding the runtime, increasing k increases the time necessary for the learning process, probably because of the time required to compute the k-futures that tends to have more branching.

Ablation study

To confirm the importance of each steps of TAG, the learning was also realized with TAG ablated versions of the algorithm. The compared versions consist of either:

— Only the first step (tree-shaped automaton initialization),

— The two first steps (initialization and merges),

— All the steps (initialization, merges, and splits).

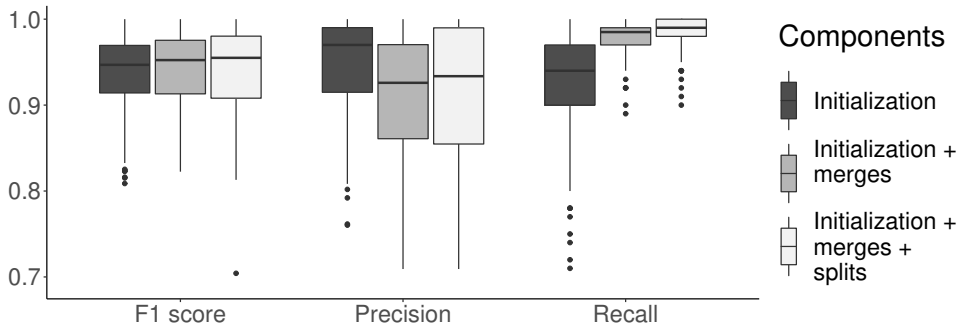


Figure 2.11 – Ablation study on recall, precision and F1-score.

Figure 2.11 compares the precision, the recall, and the F1-score obtained for the learned TA.
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 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.

2.5.3 Scalability experiment

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),

— 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 event but having non-overlapping guards), these twinned-transitions create an untimed undeterminism.

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 tss_{train} i.e., the number of timed event sequences the algorithms will have to process,

— TAG's parameter k, already studied in Section 2.5.2.

Table 2.2 – Factors order and values (default value in bold).

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 event sequences	50, 100, **500**, 1000, 2500

We defined a set of values to test for each factor (Table 2.2). To assess the effects of each factor separately, we varied them one at a time. When one factor varies, the others have a fixed value (in bold in Table 2.2). 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 estimated impact on model quality and runtime), which corresponds to the order of the factors in Table 2.2.

We generated 200 TA per factor combination to gain confidence in the results. The learning datasets are composed of runs of these model TAs (tss_{train}), each run being a timed event sequence. For the evaluation of the learned TA, we also generated 100 other timed event sequences consistent with the model and 100 event sequences inconsistent with the model ($tss_{test} = tss_+, tss_-$).

Alphabet size

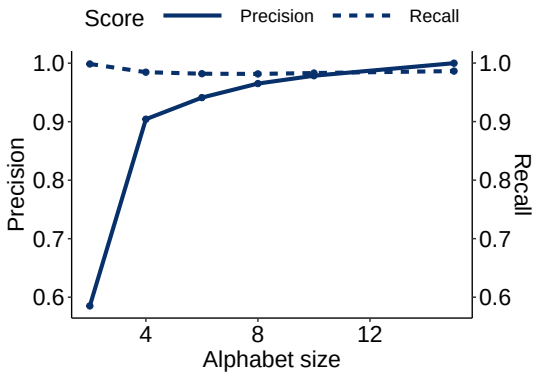


Figure 2.12 – Impact of the alphabet size of the model TA on TAG.

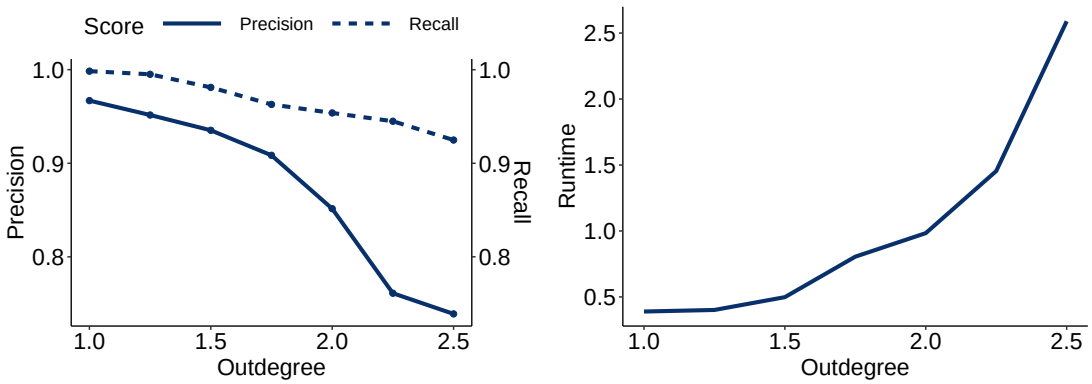
The precision and the recall are relatively stable and even tends to increase with an alphabet size ranging from 4 to 15 (Figure 2.12). With an alphabet size of 2, the recall almost reaches 1 at the cost of a low precision. This can be explained by over generalized automata because of the limited number of possible k-future and therefore the many possible merges.

Globally, there is no impact of the alphabet size on the runtime.

Outdegree

The increase of the outdegree (Figure 2.13) penalize both recall and precision, with a bigger impact on the latter. Automata with a high outdegree naturally have a larger language, which make their identification with the same number of example harder. The bigger loss of precision indicates that in addition to recognize fewer sequences of the model TA, the part of sequences wrongly recognized also increases, which may indicate that the realized merges are not the good one.

There is an important impact of the outdegree on the runtime, with an exponential evolution and a factor 5 on average between an outdegree of 1 and of 2.5. It means



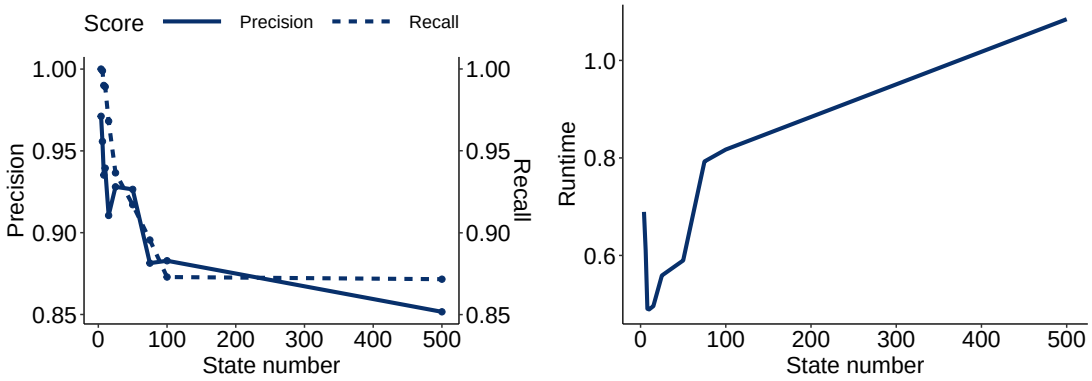
(a) Impact on scores.

(b) Impact on runtime.

Figure 2.13 – Impact of the outdegree of the model TA on TAG.

that more merges and splits were necessary for the learning process.

State number



(a) Impact on scores.

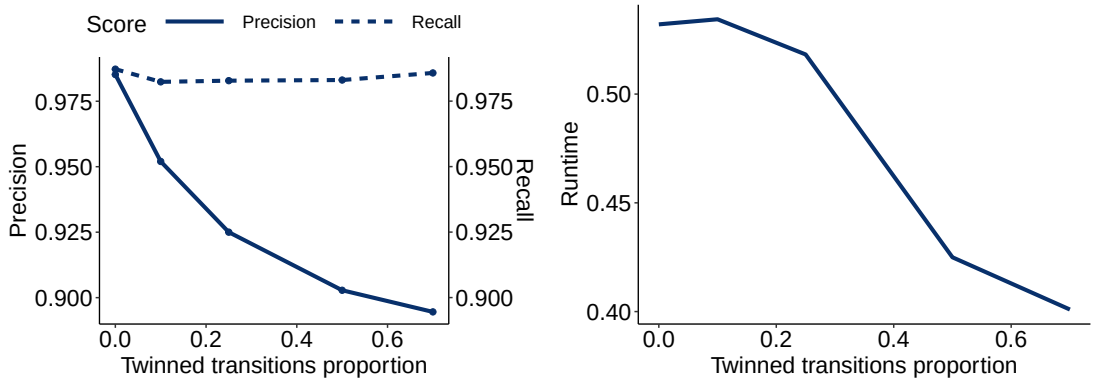
(b) Impact on runtime.

Figure 2.14 – Impact of the state number of the model TA on TAG.

All recall, precision, and runtime are affected by an increasing state number, with the same analyze as for the outdegree (Figure 2.14).

Twinned transitions proportion

The precision decreases when the twinned transition proportion grows. Missed splits leads to timed event sequences wrongly accepted. There is no statistically significant evolution of the recall, which stays high.

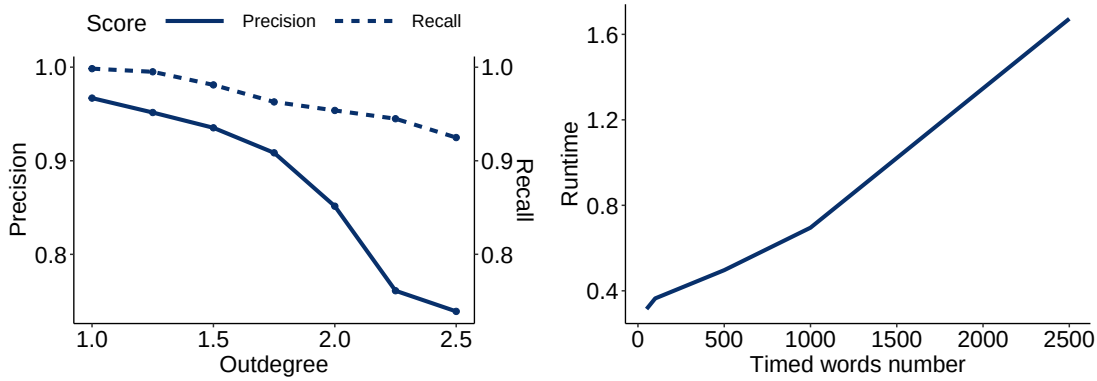


(a) Impact on scores. (b) Impact on runtime.

Figure 2.15 – Impact of the number of timed event sequences of the model TA on TAG.

Surprisingly, the runtime decreases as the twinned transition proportion increases, but in smaller proportion as for the other factors.

Number of timed event sequences



(a) Impact on scores. (b) Impact on runtime.

Figure 2.16 – Impact of the number of timed event sequences of the model TA on TAG.

The recall increases with the number of timed event sequences in the input data sample (Figure 2.16), with a stabilization of the score at 500 timed event sequences for our parameter's combination. This doesn't come along with a loss in precision, which stays close to 1. This gain in quality is due to a larger representation of the language of the model automaton in the input sample, allowing a better identification.

It is at the cost of the runtime because the PTA is naturally bigger and requires

more merges (and splits). The runtime augmentation seems to be linear with the number of timed event sequences.

2.5.4 Comparison with the State-of-the-Art algorithms

To compare TAG to the State-of-the-Art algorithms presented Section 1.3.2, we also tested RTI+ and TkT on the data of TAG's scalability experiment (Section 2.5.3). We were not able to test GenProgTA because their source code is not available, and their implementation does not allow the addition of new data. The Figures of the results obtained for each factor are available in Appendix B.

Table 2.3 – Comparison of the runtime on the scalability experiment data.

Algorithm	**Median (s)**	**IQR (s)**
TAG	0.466	0.249
TkT	0.646	0.064
RTI+	0.218	0.247

Globally, the runtime of the three algorithms is comparable, with RTI+ being faster than TkT and TAG (Table 2.3). This can at least partly be explained by the languages in which they are coded (C++ for RTI+, Java for TkT, and Python for TAG).

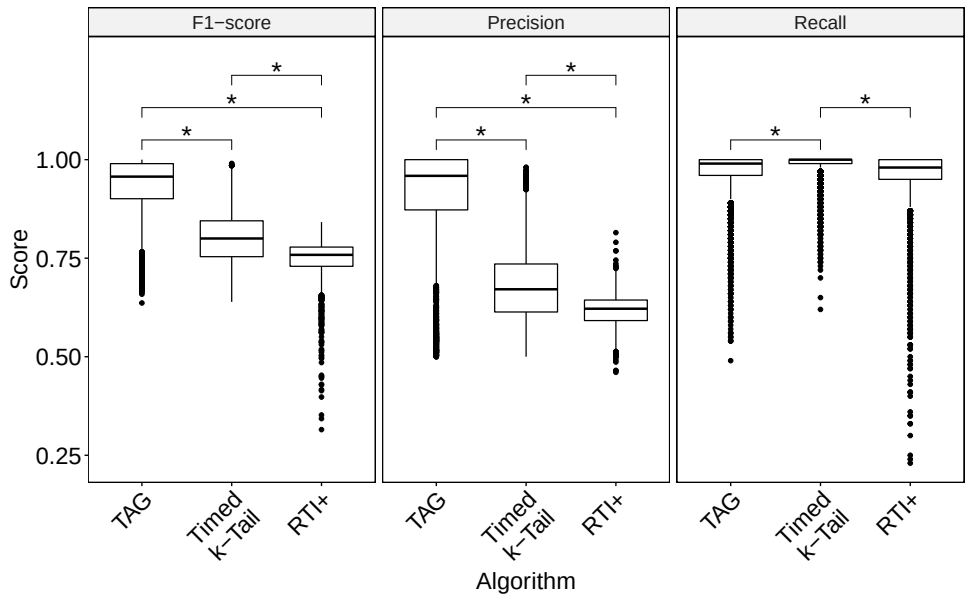


Figure 2.17 – Comparison of the quality scores on the scalability experiment data.

Regarding the scores (Figure 2.17), TAG significantly outperforms the other algorithms thanks to a precision not reached by the other algorithms while keeping a competitive recall. It is important to note that TkT does not try to learn deterministic TAs in absence of nested operation, contrarily to TAG and RTI+. Consequently, there will be multiple paths possible in TkT's automata for a single timed event sequence, increasing its chance to be accepted. Therefore, its recall naturally tends to be higher and its precision lower. RTI+'s low precision is mostly due to its bad management of the timing constraints.

There are some differences in the impact of the factors on the quality of the models of the three algorithms. TkT is less impacted by the outdegree than the other algorithms, it may be because it has no determinism requirement and thus continue to allow many behaviors even for a same event sequence (its precision does decrease as the other). TAG's precision is less impacted by the state number than the others, while its recall is more. It may generalize less in situation of state explosion because the k-future becomes more complex merge after merge.

2.5.5 Conclusion on the experiment on synthetic data

The quality of the models learned by TAG is affected by the complexity of the target model. The factors reducing the part of the language represented in the input sample such as the outdegree or the state number penalize the recall and the precision of the models. On the contrary, increasing the part of the language represented with more timed event sequences increases the quality of the TA. A high twinned transitions proportion affects the precision because a missed split is more critical than a missed merge by allowing incorrect timed event sequences. Finally, TAG can learn a TA in less than one second on average (0.7010 s), with an extremum of 16 s for 2500 timed event sequences in the input sample.

2.6 Experiment on real data: TV logs

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). The objective was to recover the model of a day of programs summarizing the information in

the logs.

We first used the data of the Friday mornings of August 2020. These sequences are expected to be similar and thus lead to a compact model. Then, we used the data of every day of July and August 2020, which corresponds to the Canadian summer school vacations. Here, the sequences are more varied, for example the programs broadcasted in the Sundays are not the same as the programs of the Fridays. Consequently, the corresponding automaton is more complex, but information can still be obtained from such model.

2.6.1 Method

Table 2.4 display an extract of the data. The logs are divided by day from 6:00 AM to 5:59 AM. Regarding the Friday morning logs, the sequences stop at 12:00 AM. The logs contain, among others, information about the class of the program (commercial message, promotion for a program...), its category (program for children, news...), its starting and duration, or its title. A timed event sequence is composed by the sequence of entries class (or category for the program class) and the delay since the last entry (i.e., the duration of the last entry).

2.6.2 Friday mornings subset

The TA learned by TAG with the TV logs of the Friday mornings (329 events) is shown in Figure 2.18. The guards are originally in seconds and have been formatted to make the figure more comprehensible. The first guard in bracket limits the delay of occurrence after the last event. The second guard in brackets 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.

This TA has 6 states, 10 transitions, and 6 distinct events. 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.

Table 2.4 – Subset of TV programs logs.

Class	Date	Start time	End time	Duration	Title	Subtitle	Category	...
Commercial	01/08/20	13:52:14	13:52:29	00:00:15	Company name	None	None	
Commercial	01/08/20	13:52:29	13:52:44	00:00:15	Company name	None	None	
Commercial	01/08/20	13:52:44	13:53:14	00:00:30	Company name	None	None	
Program	01/08/20	13:53:14	14:05:24	00:12:10	Name	Episode name	Education formal	

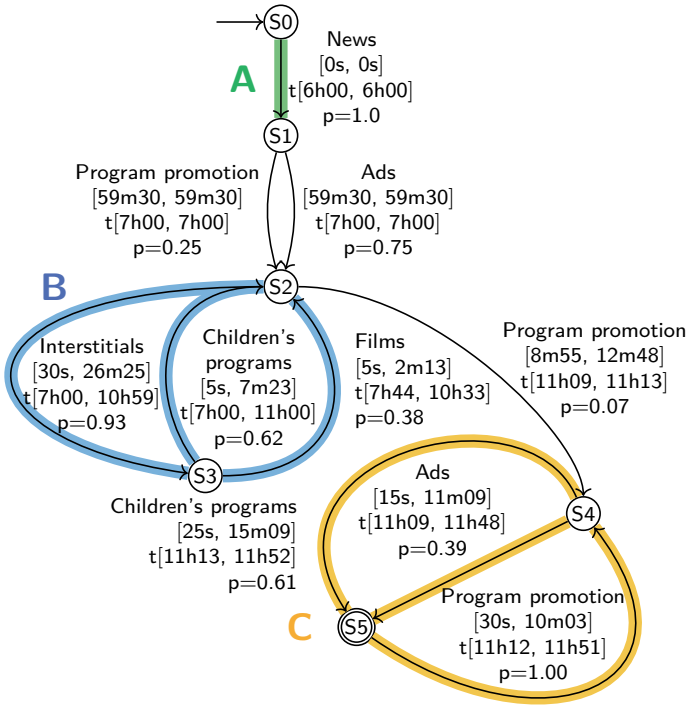


Figure 2.18 – Learned TA from the Friday morning logs of a Canadian TV channel.

2.6.3 July and August subset

Table 2.5 – Queries translated in natural language, result, and execution time.

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

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 distinct events.

To obtain information from the automaton learned from the summer data, we can take advantage of both the classical expressiveness of TAs and the probabilities associated with TAG's TA transitions. We formulated questions about the TV programs using logical queries answered by a model-checking algorithm (Section 1.2.1). We used UPPAAL SMC Bulychev et al. 2012, a probabilistic model-checking tool for TAs for TAs. 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.

Table 2.5 presents the queries submitted to the model-checker translated in natural language, the results, and the execution time. For instance, the first question in Table 2.5 was submitted to the model-checker as follows:

$$E <> Actions.S\_CHILDREN\_PROGRAMS \; and \; Automaton.d > 3600$$

In less time than a human would take to analyze even the first TV programs automaton, the responses to useful questions.

2.7 Conclusion

The main contribution of this chapter is TAG, a TA passive learning algorithm from logs of real-time systems. The subclass of TA learned by TAG is a DRTA with an additional global clock and probabilities on transitions. The algorithm is based on two operations, state merges and transition splits, that are applied on an initialized prefix tree automaton. A state merge requires identical k-future and the transition split is applied where it brings out a temporal determinism. 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.

Experiments on synthetic data have shown that it produces models that achieve a better trade-off between recall and precision than the State-of-the-Art TA passive learners. An additional experiment on real data demonstrates the interpretability of the learned models.

PART II

Timed Automata learning from numerical data

INTRODUCTION OF THE PART

Nowadays, systems' observational data often take the form of time series, thanks to the proliferation of sensors that record data, mostly numerical, over time. Consequently, most of the new machine learning and data analysis techniques are specifically designed for numerical and continuous data, and achieve good performance. However, these methods often face challenges such as the need for a significant amount of data to fine-tune their parameters, computational complexity, and a lack of explainability. Meanwhile, symbolic methods, such as TAs-based methods, which do not suffer from these limitations and have an extensive literature, have been overlooked due to their direct inapplicability to this type of data. The discretization of continuous data bridges the gap between symbolic methods and time series, by converting the time series into sequences of symbols (interpreted as events).

The second part of this thesis primarily focuses on the discretization problem to enable the learning of TAs from time series. An application of anomaly detection in time series based on TA is also presented.

We begin by considering the discretization problem in a univariate setting, and focus on identifying event sequence characteristics that are conducive to learning TAs with TAG. First, we modify an existing discretization algorithm, Persist, whose score, the persistence, is interesting for learning TAs. Afterward, we extract this score and combine it with other identified key characteristics. The result is MOODES, a multi-objective optimization-based approach for time series discretization specifically designed for TA learning. Finally, we propose an approach for detecting anomalies in time series data based on ensembles of TAs that take advantage of MOODES' ability to produce multiple discretization solutions.

Secondly, we address the multivariate problem and investigate the identification of interactions between components of a system. The aim here is to learn synchronized TA (Timed I/O Automata). We propose a discretization algorithm that identifies and preserves the synchronizations between variables.

CHAPTER 3

STATE OF THE ART AROUND TIME SERIES

Before investigating discretization methods specifically designed for TA learning, we present existing methods related to time series discretization, as well as techniques for rule discovery in time series where the problem is related to interaction identification.

Let's first recall the definition of a time series.

Definition: Time series

A (univariate) time series $\{x_t\}, t \in T$ is an ordered collection of values recorded over time domain T.

Formally, time series values can be either numerical or categorical, however, time series is commonly associated with numerical values. In this manuscript, the term "time series" is employed for numerical time series while categorical data is associated to "event sequences".

Definition: Multivariate time series

A multivariate time series $\{X_t\}, t \in T$ is an ordered collection of vectors of values recorded over time domain T.

A multivariate time series can be decomposed into multiple univariate time series.

3.1 Time series discretization

Time series discretization, also known as symbolization, is a pre-processing step used to reduce the dimensionality of the data, to reduce noise, or to enable the use of symbolic methods.

Definition: Time series discretization

The discretization of a real-valued signal consists in converting it into a sequence of discrete symbols.

Discretization techniques differ in the rules used to map the continuous data points to the discrete symbols. This implies partitioning either the value range of the variable(s) x, or the time domain T. The partition limits are referred to as cutpoints or breakpoints.

Example: Time series discretization via cutpoints in value range

Figure 3.1 displays a univariate time series that was recorded at regular time step. The blue lines represent cutpoints, which create a partition of the value range into bins or intervals that are each associated with a symbol (a, b, ...). Each data point can then be replaced by the symbol of the interval it falls in (cccdddcba...) (Algorithm 8). Furthermore, it can be summarized even more by keeping only the sequence of different symbols associated to their duration ((c, 3), (d, 3), (c, 1), (b, 1), (a, 2)...).

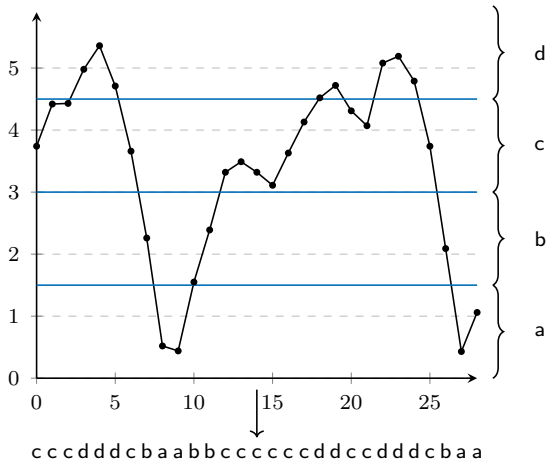


Figure 3.1 – Discretization of a time series. The blue lines correspond to cutpoints on the value range of the variable, and form intervals corresponding to symbols (a,b,...). The time series is discretized by replacing each value by the corresponding symbol (cccdddcba...).

We provide a non-exhaustive list of discretization techniques, from the simplest to methods that consider multiple dimensions (variables) of the time series.

Algorithm 8 Discretization via cutpoints in value range

Require: a set of cutpoints cps, a univariate time series ts
Return: a discretized time series $discretized\_ts$

```
discretized_ts = empty vector of the same size as ts
alphabet = vector containing the different symbols
for index = 0 to |ts| − 1 do
    symbol = 0
    while ts[index] ≥ cps[symbol] and symbol < |cps| do
        symbol = symbol + 1
    end while
    discretized_ts[index] = alphabet[symbol]
end for
return discretized_ts
```

3.1.1 Univariate discretization

The univariate discretization methods transform a time series defined in a single dimension into a sequence of symbols.

Equal Width and Equal Frequency Discretization

Partitioning the value range into n bins of equal width is the most straightforward approach to determining cutpoints. The discretization displayed in Figure 3.1 is an Equal Width Discretization (EWD).

To better account for the true value distribution of the time series, the Equal Frequency Discretization (EFD) partitions the value range into n bins of equal frequency. It is especially recommended over an EWD when the data is unevenly distributed.

Symbolic Aggregate approXimation

Symbolic Aggregate approXimation (SAX) (Lin et al. 2003) is a time series symbolic representation that is widely used in the literature. It relies on the hypothesis that time series usually have a Gaussian distribution. Figure 3.2 illustrates the approach.

First, the dimension of the data is reduced by applying a Piecewise Aggregate Approximation (PAA) on the normalized time series. The time domain is divided into w equal-sized bins. In the PAA representation, each point takes the mean value of its bin.

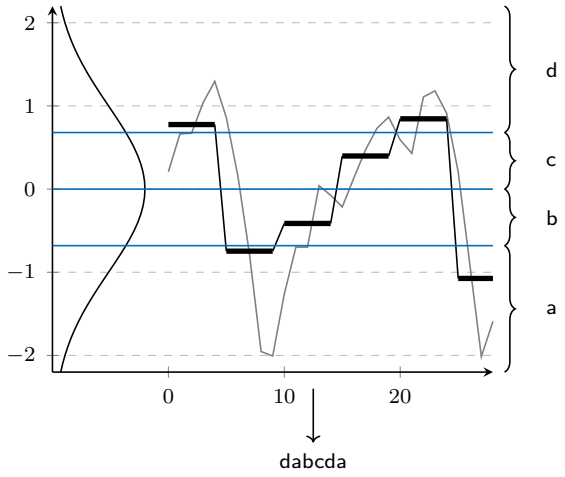


Figure 3.2 – SAX representation of a time series. The normalized time series is in gray and its Piecewise Aggregate Approximation (PAA) representation is in black and bold ($w = 6$). The cutpoints in blue were set according an EWD of the Gaussian curve ($k = 4$).

Then, under the assumption that normalized time series have a normal distribution, an equal frequency binning of the Gaussian curve is performed, giving the cutpoints. The time series data points are replaced by the symbol of the bin in which their value falls in the PAA representation.

SAX requires two parameters, the number of time intervals w for the PAA and the number of symbols k.

There exists several variations of SAX such as aSAX (Pham, Le, and Dang 2010) that overcome the Gaussian distribution assumption by using the k-means algorithm after the PAA, or 1d-SAX (Malinowski et al. 2013) that takes into account the trend of the segments in addition to the mean value.

Persist

Persist is a method proposed by Mörchen and Ultsch (2005) that produces cutpoints in the value range to discretize a univariate time series. It was employed as preprocessing step to find patterns in time series in a language called Time Series Knowledge Representation (TSKR) (Mörchen and Ultsch 2007).

Persist is based on the assumption that the time series are the reflection of an underlying process that consists of recurring persisting states, and it aims to restore these states in the form of symbols in a discretized version of the time series. Mörchen

and Ultsch state that "if there is no temporal structure in the time series, the symbols [in its discretized version] can be interpreted as independent observations of a random variable according to the marginal distribution of symbols". Thus, the idea is to look for symbols showing a persisting behavior, by creating symbols whose probability of repetition will be much higher than their probability of occurrence. In other words, the probability to have a symbol repeating itself should be much higher than the probability to have this symbol at any time step, disregarding the previous symbol.

The breakpoints are iteratively chosen in a set of candidate cutpoints according to a score called persistence score. This persistence score measures how high the repetition probability of the symbols is compared to their probability of occurrence. The set of candidates is initialized by an equal frequency binning (with a number of bins fixed to 100 by default) (line 2 in Algorithm 9). At each iteration, the function best_cp

Algorithm 9 Persist

Require: univariate time series ts, a maximal number of symbols k_{max}
Return: a set of breakpoints cps and the associated persistence scores $scores$

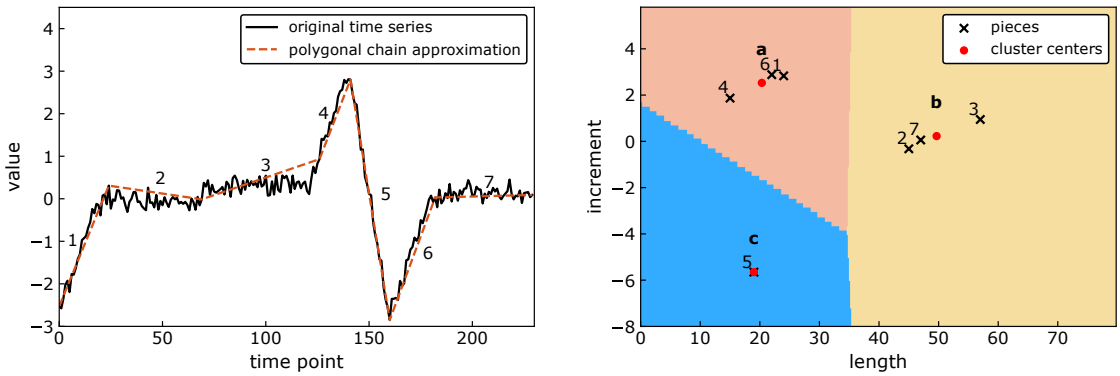
1: $cps = \emptyset$
2: $candidates =$ equal_frequency_binning$(ts, 100)$ ▷ percentiles
3: $k = 1$
4: $scores = []$
5: **while** $k < k_{max}$ **do**
6: $\quad (new\_cp, new\_score) =$ best_cp$(ts, cps, candidates)$
7: $\quad cps = cps \cup new\_cp$
8: $\quad candidates = candidates - new\_cp$
9: $\quad scores = scores \cup new\_score$
10: $\quad k = k + 1$
11: **end while**
12: **return** $cps, scores$

individually tests every candidate cutpoint added to the already selected cutpoints (cps). The candidate increasing persistence score the most is returned with its score.

Persist stops when the maximal number of symbols is reached and return the cutpoints and the scores associated to each number of symbol. The user can choose the number of cutpoints to keep by selecting the one associates to the highest persistence score, or using expert knowledge.

ABBA

ABBA (Elsworth and Güttel 2020), which stands for Adaptive Brownian Bridge-based Aggregation, is a time series discretization method in two phases: dimension reduction, and clustering.



(a) Compression.

(b) Clustering.

Figure 3.3 – ABBA steps (from Elsworth and Güttel (2020)).

It first approximates the time series via an adaptive piecewise linear continuous approximation resulting in a sequence of 2-tuples consisting of the length of each segment and its incremental value (Figure 3.3a). The incremental value of a segment is the difference of value between its start and its end.

Then, the segments are clustered based on their length and increment using the k-Means (Figure 3.3b). It is possible to give more importance to one or the other by putting a bigger weight.

The number of symbols k can be provided as parameter or set automatically by choosing the smallest k giving a cluster variance under a given tolerance.

3.1.2 Multivariate discretization

When multiple variables related to a same system are measured and recorded over a same time period, it results in multidimensional data called multivariate time series. Performing the discretization of each dimension independently is a common practice. However, it may be interesting to consider all the dimensions at the same time to take into account the interdependencies and interactions between the variables.

We now present some approaches to discretize multidimensional data. We omit here methods dedicated to time series classification that use the class of the labeled training set to guide the discretization.

Classical clustering algorithms

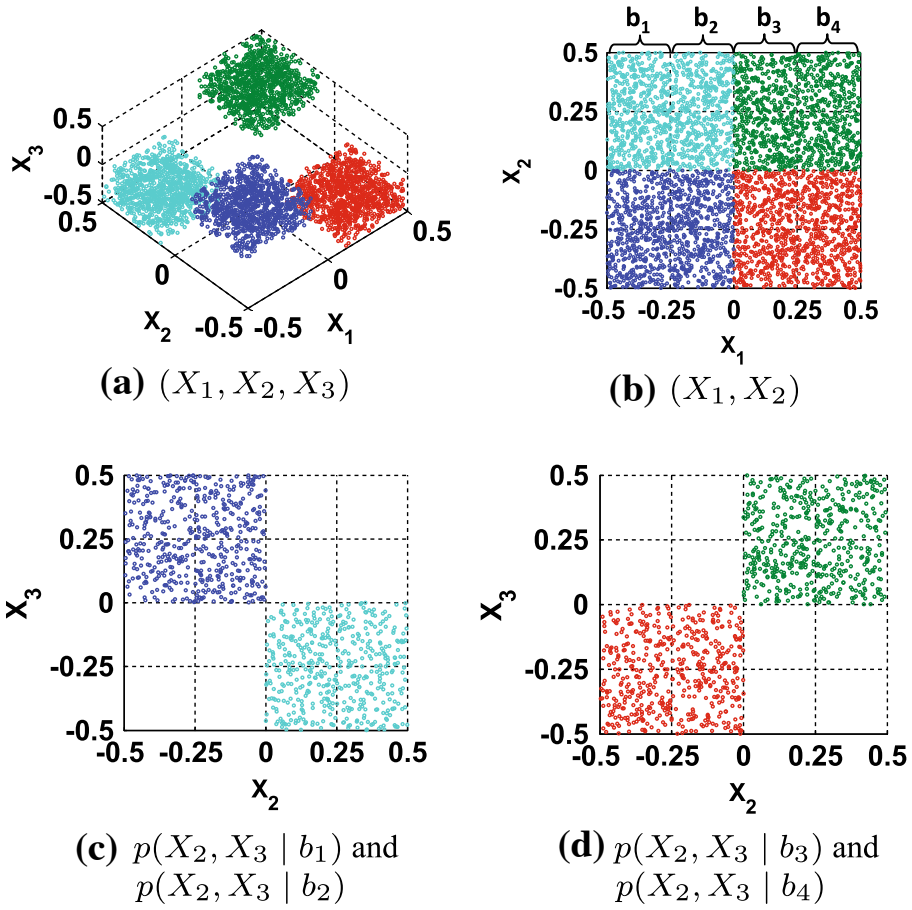
Many clustering algorithms can be used to discretize time series, the clusters corresponding to symbols. The objects clustered will typically be the value vectors at each time point, or segments (over multiple time points) of the time series. In the first case, the time domain is not taken into account. Among the clustering techniques, the k-means seems the more popular for (multivariate) time series discretization. It is also possible to perform a hierarchical clustering. Zolhavarieh, Aghabozorgi, and Teh (2014) present an overview of how the clustering algorithms can be applied to cluster time series subsequences.

Interaction-Preserving Discretization

Interaction-Preserving Discretization (IPD) (Nguyen et al. 2014) is an unsupervised data discretization method that aim to preserve the interactions between variables. It was not designed for time series, but it is still worth to mention for its interaction preservation strategy. Their discretization partition the domain of each variable X_i $[min_i, max_i]$ into k bins. The authors state that to preserve the interactions, two regions in one dimension should only be in the same bin if and only if the objects in those regions have similar multivariate joint distributions in the other dimensions.

> **Example: Interaction-Preserving Discretization**
>
> Figure 3.4 illustrates the need for such discretization. Figure 3.4.a represents data in three dimensions (X_1, X_2, X_3). In each dimension taken separately, the data is uniformly distributed from -0.5 to 0.5. A univariate discretization method would not be able to find any relevant way to partition the data. However, when all the dimensions are considered together, four evident clusters (in blue, red, green, and cyan) are visible. Data points reunited in a same cluster have similar multivariate joint distributions in the different dimensions.

First, they proceed to an equal-width segmentation in the time space (b1, b2... for X_1 in Figure 3.4). Then, they merge two consecutive regions (e.g. b1 and b2) if



(a) (X_1, X_2, X_3) **(b)** (X_1, X_2)

(c) $p(X_2, X_3 \mid b_1)$ and $p(X_2, X_3 \mid b_2)$ **(d)** $p(X_2, X_3 \mid b_3)$ and $p(X_2, X_3 \mid b_4)$

Figure 3.4 – Interaction-Preserving Discretization (from Nguyen et al. (2014)).

the multivariate distributions are statistically similar. They propose a new interaction distance and define the task of multivariate discretization in terms of the Minimum Description Length (MDL). The resulting objective scores enable to balance the interaction preservation and the information retained in the discretized time series. They propose two algorithms to select the breakpoints, one based on dynamic programming and the second on a fast greedy heuristic.

Symbolic Dynamic

In the context of dynamical systems, a system can be described by a set of variables whose values change over time according to a specific function. When the state space is continuous, it is called phase space. Dynamic filtering is the partition of

a multidimensional phase space sub-spaces associated to a symbol ($\alpha, \beta...$ in Figure 3.5). The continuous phase trajectory (in blue) is then discretized using these

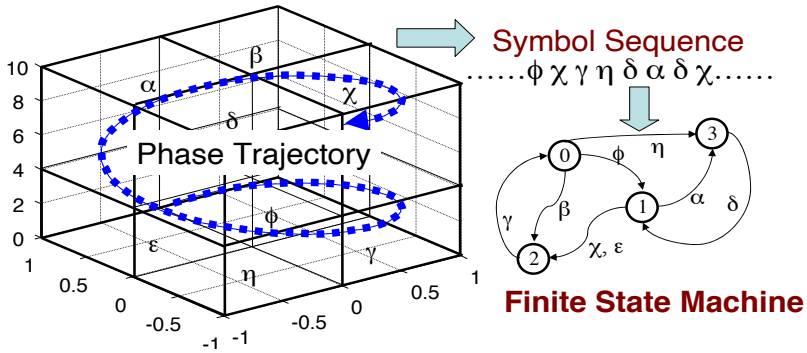


Figure 3.5 – From the continuous phase space to the Finite State Automaton (FSA) (adatpted from S. Gupta and Ray (2007)).

symbols, this practice is referred to as symbolic dynamic (S. Gupta and Ray 2007). The cutpoints can be placed such that he sub-spaces are equally sized, using threshold from expert knowledge, or such that the entropy of the resulting symbolic sequence is maximized.

S. Gupta and Ray (2007) use symbolic dynamic to model the behavior of dynamical systems from time series for early anomaly detection. They obtain a symbol sequence from the train phase trajectory, which is used to construct an (untimed) FSA. New data is also transformed into symbol sequence. Next, to assess whether the new sequence is abnormal, they compare the frequency of FSA state visits with the train sequence to the new sequence.

3.2 Rule mining in time series

The task of identifying a synchronization event between two time series is related to the task of rule mining in time series. Rule mining is the search for meaningful patterns or rules in time series. An example of rule could be: if A occurs, then B will occur. A and B can be specific shapes, trends, or else values. This task is often preceded by a discretization step to obtain the objects over which the rules will be mined.

Identifying synchronization between two components modeled by Timed I/O Automata (TIOAs) models can be viewed as searching a rule in the form if A occurs in

the time series of the first component being modeled, then B also occurs in the time series of the second component. For that reason, we now present an overview of existing methods for rule mining in time series.

3.2.1 Sequential rule mining

Das et al. (1998) introduced the rule discovery in time series problem. They want to identify patterns such that when a shape A happens in the time series, then a shape B happens within a certain delay. The mined rules are of the form $A \overset{T}{\Rightarrow} B$, meaning if A occurs, then B occurs within time T.

The time series is first segmented in the time domain with a fixed window size. The resulting subsequences are clustered using the k-Means algorithm and each cluster is associated to a symbol. The rules are formed with these symbols, and are selected in function of their frequency and confidence and can be ranked. No search method is proposed, all the possible rules have to be tested.

This work has been extended by Shokoohi-Yekta et al. (2015). They define a rule with an antecedent R_a, its consequent R_c, the maximum expected delay between the two $maxlag$, and the threshold distance used to trigger a subsequence match t. An example of rule is presented in Figure 3.6. A MDL-inspired scoring function is proposed

Figure 3.6 – Sequential rule discovery in bird vocalization (from Shokoohi-Yekta et al. (2015)). Two instances of the rule on the right are identified (antecedent in orange, consequent in green).

to evaluate the rules. A good rule, and therefore a good prediction of the consequent given an antecedent, will save many bits [1].

The time series is first discretized using an EWD on the normalized data. However, contrarily to the article of Das et al. (1998), the symbols are not used individually as antecedent and consequent. It is only performed to enable the MDL scoring function.

1. The MDL principle (Rissanen 1978) states that the best hypothesis (or model) for a given set of data is the one that results in the shortest description length when both the hypothesis and the data are encoded together.

The antecedents and consequents are subsequences of the time series, found by an iterative search.

These works can be applied to multivariate time series to obtain rules where the antecedent and the consequent are from different dimension.

3.2.2 Allen's temporal relation-based rule mining

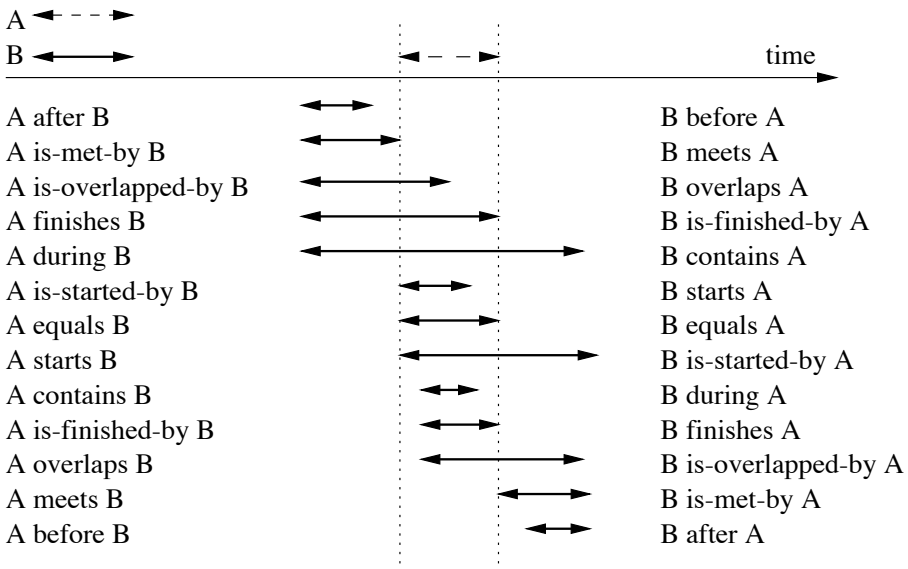


Figure 3.7 – Allen temporal relations (from Höppner (2001)).

Höppner (2001) proposed a more general vision of rule mining, yet specifically dedicated to multivariate time series. The rules are based on the Allen's temporal logic (Allen 1983) (Figure 3.7) which offers a wider way to define the temporal relationship between subsequences. In this paper, the rules are mined in multivariate time series with already labeled intervals. The temporal relations that always state in the labeled sequences are stored in a bivariate matrix in which the nature of the relation between two labels in stored.

Example: Rule mining from labeled sequences

Figure 3.8 display two labeled sequences and the temporal relations found in the matrices. The first matrix presents the temporal relations within a single sequence, and the second matrix also includes the relations between sequences. A always occurs before B (and therefore B after A). B does not always occur before A (see last

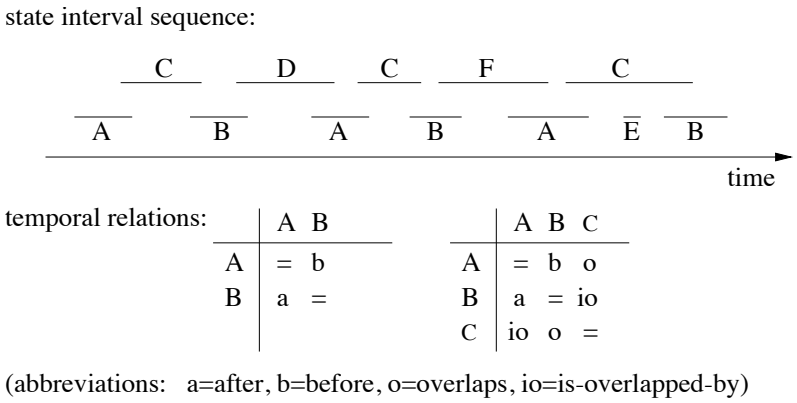


Figure 3.8 – Temporal relation extracted from labeled interval sequence (from Höppner (2001)).

occurrence of B). No rule can be identified for E or any label with a single occurrence. Finally, A always overlaps C, and C always overlaps B.

A handful of works have followed this paper to improve the search algorithm, or to handle unlabeled data by adding a discretization step before the search. For instance, Moskovitch and Shahar (2015) proposed a discretization method and improved a temporal relations enumeration algorithm, with the final objective of time series classification. The discretization is supervised, the cutpoints in the value range are created such that the symbol distributions of the different time series classes are the most differentiated. After the discovery of time interval related patterns in the now symbolic sequences, the classification is performed on the patterns.

CHAPTER 4

UNIVARIATE DISCRETIZATION FOR TIMED AUTOMATA LEARNING

In order to learn TAs from observational data that takes the form of time series, a discretization step must be performed to obtain discrete symbolic sequences that can be injected to TAG. The symbols will be interpreted as events.

The choice of a discretization method has a significant impact on the characteristics of the resulting discretized sequence. Therefore, it is crucial to consider the desired characteristics when selecting a discretization method, rather than viewing it as a pre-processing step of little importance. Here, the discretized sequences will be used to learn TAs with TAG. Often, TA learning is not the final task, the inferred model is then employed for another application such as model-checking or anomaly detection. Consequently, it becomes necessary not only to examine the characteristics required for TA learning, but also how it will impact the resulting model and its usability for the final task.

TAG has not been developed for a specific application and aims to capture the global behavior of the system being modeled. One natural application of a system's behavior model is to compare new observations with it in order to detect abnormal sequences. Anomaly detection, which extensively studies the use of untimed automata on event sequences, is specifically dedicated to this task. In this chapter, we study what is a good discretization method for learning TAs with TAG, which will be subsequently employed for online anomaly detection in time series.

To achieve this, we begin by adapting an existing algorithm for time series discretization whose hypothesis is close to our problematic, then we combine its score, the persistence, to other relevant characteristics for TA learning. Additionally, we propose an anomaly detection on time series based on ensemble of TAs.

4.1 Persist with Wasserstein distance

As presented in Section 3.1.1, Persist (Mörchen and Ultsch 2005) is an algorithm that produces cutpoints to discretize time series. It is based on the hypothesis that the data originates from a process characterized by recurring persisting states, where *persisting* indicates that a state (expressed here as symbols) tends to persist over consecutive time steps.

Persist's hypothesis is also relevant in the context of TAs learning. We assume that the behavior of the observed system exhibits regularities over time, and that the time spent in a given state is determinant for the future behavior. Moreover, when learning a TA from time series data, if inappropriate state changes arise due to data noise remaining after the discretization, the learned temporal constraints will become completely inaccurate, concealing the actual state transitions. Therefore, it can be interesting to minimize symbol changes in the discrete sequences to retain only the necessary and meaningful transitions.

With the goal of using Persist to discretize the time series for TAs learning with TAG, we examine the Persist algorithm and propose two modifications to improve its applicability and effectiveness.

4.1.1 Persistence score

Persist generates cutpoints in the value range of a time series and associates the intervals formed with symbols. The time series datapoints can then be replaced by the symbol of the interval they fall in the discretized sequence. It aims to set cutpoints such that the symbols are persisting in the discretized sequence. This notion of persistence for a symbol is evaluated by comparing its probability of occurrence (marginal probability), with its probability to repeat itself over two consecutive time steps. This comparison is based on the Kullback-Leibler (KL) divergence (Kullback and Leibler 1951). The *persistence score* summarizes the persistence of all the symbols, and is used to decide the creation of new cutpoints.

The KL divergence measures how different a probability distribution P is from another probability distribution Q. For discrete probability distributions defined on

$\mathcal{X}$, the KL divergence is defined as follows:

$$D_{KL}(P||Q) = \sum_{x \in \mathcal{X}} P(x) \log(\frac{P(x)}{Q(x)})$$

This divergence is not symmetric ($D_{KL}(P||Q) \neq D_{KL}(Q||P)$). This is why Mörchen and Ultsch use a symmetric version obtained as follows:

$$SKL(P,Q) = \frac{1}{2}(D_{KL}(P||Q) + D_{KL}(Q||P))$$

In Persist, the probability distributions P and Q are discrete two-binned probability distributions, and are based on the probability of occurrence of the symbols ($P(s)$) and their probability of repetition ($P_r(s)$):

$$P = (P(s), 1 - P(s)),$$

$$Q = (P_r(s), 1 - P_r(s)).$$

Symbol repetition is when the symbol occurring at time $t-1$ is the same as the one occurring at time t.

The persistence score for a symbol s is computed from the KL divergence between its probabilities of repetition and occurrence as follows:

$$\text{Persistence}(s) = \text{sgn}(P_r(s) - P(s))\text{SKL}(P,Q)$$

The first element of the equation ($\text{sgn}(P_r(s) - P(s))$) allows favoring only the cases where the probability of repetition is superior to the probability of occurrence, otherwise, it will contribute negatively to global persistence score. Finally, the global persistence score for the whole set of cutpoints is the mean of each symbol persistence score:

$$\text{Persistence} = \frac{1}{|\Sigma|} \sum_{s \in \Sigma}^{k} \text{Persistence}(s)$$

The symmetric KL divergence between two probability distributions with two possible outcomes as here is represented in Figure 4.1. One of the properties of the KL divergence is that it has no upper bound, a property inherited by the persistence score. The shape of the surface produced by this divergence is also particular. The symmetric KL divergence is null when the probability distributions are equal, and increases

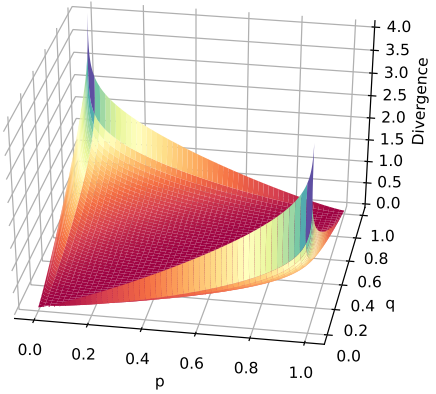


Figure 4.1 – Symmetric KL divergence between two probability distributions with two possible outcomes $P = (p, 1-p)$ and $Q = (q, 1-q)$.

non-linearly as the difference between the distribution grows. To achieve a high value of symmetric KL, p or q (i.e., $P_r(s)$ or $P(s)$) have to be close to 0 or 1. The direct consequence of these observations is that the persistence score based on the KL divergence will focus on extreme cases.

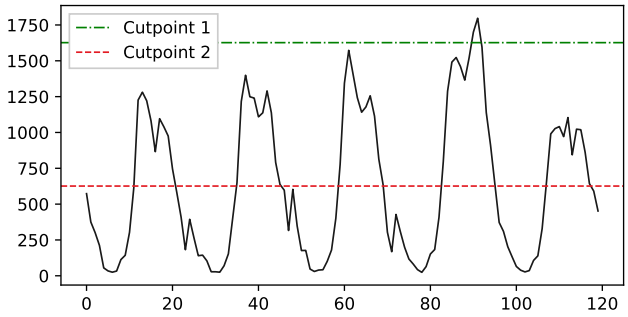


Figure 4.2 – Two candidate cutpoints, each creating two symbols (s1 and s2).

s	P(s)	Pr(s)
s1	0. 97	0.99
s2	0.03	0.62

(a) Cutpoint 1

s	P(s)	Pr(s)
s1	0.54	0.92
s2	0.47	0.94

(b) Cutpoint 2

Table 4.1 – Two candidate cutpoints, each creating two symbols (s1 and s2). The KL divergence will give a better score to cutpoint 1.

Example: Persist's cutpoints selection with the Kullback-Leibler divergence

Table 4.1 and Figure 4.2 illustrate this phenomenon. In this example, at the beginning of the algorithm, the first cutpoint will be selected to create two symbols. Two candidate cutpoints are examined. The first cutpoint (Table 4.1a) will create a first symbol that covers almost the entire discretized time series and thus has a probability of occurrence and repetition close to 1, and a second symbol that almost never occurs and doesn't show a particularly recurring behavior. The second cutpoint (Table 4.1b) will create two symbols about equally probable and with very high probabilities of repetition (greater than 0.90). Persist based on the KL divergence will choose cutpoint 1. The discretized version of the time series in Figure 4.2 will consist of the succession of about 90 "s1", then a few "s2" and again "s1" until the end, while it would have consisted of an alternation of persistent "s1" and "s2" if cutpoint 2 had been chosen.

4.1.2 Improving Persist for Timed Automata learning

Probability distribution comparison

The persistence score computed with the KL divergence tendency to favor extreme cases is not favorable to the learning of global and generalized model of behavior in the form of a TA. Based on this observation, the replacement of the KL divergence by another measure of difference between probability distribution is studied.

The Wasserstein distance (Kantorovich 1960), also called Kantorovitch distance, Kantorovitch-Rubinstein distance, or earth mover's distance, is another measure of difference between probability distributions. It corresponds to the minimal cost to transform a distribution P in another distribution Q in the same space. The Wasserstein p-distance between two probability distributions P and Q is defined by the following equation where $\Gamma(P,Q)$ are all the possible joint distributions for (X,Y) with marginal probability distributions P and Q:

$$W_p(P,Q) = \inf_{\gamma \in \Gamma(P,Q)} (\mathbb{E}_{(x,y)\sim\gamma} d(x,y)^p)^{1/p}$$

In the case of discrete probability distributions with only two possible outcomes,

the Wasserstein distance becomes a simple subtraction and is defined as follows:

$$W(P,Q) = |P(x_1) - Q(y_1)|$$

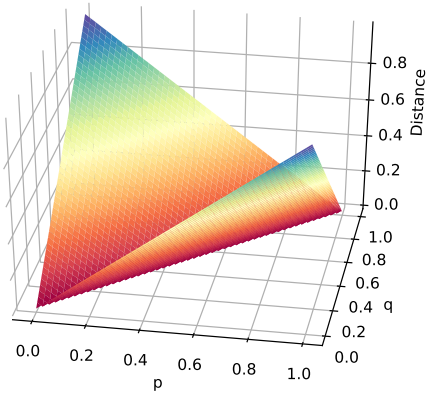


Figure 4.3 – Wasserstein distance between two probability distributions with two possible outcomes $P = (p, 1-p)$ and $Q = (q, 1-q)$.

This distance is symmetric, bounded, easier to compute in our settings than the KL divergence, and it increases linearly as the difference between the distributions grows (Figure 4.3).

Therefore, we use the Wasserstein distance to measure how the probability of occurrence of the symbols and their probability of repetition are different in the score of persistence in place of the KL divergence:

$$\text{Persistence}_W(s) = \text{sgn}(P_r(s) - P(s))\text{W}(P,Q)$$

Example: Persist's cutpoints selection with the Wasserstein distance

In front of the choice presented in table 4.1, the persistence score computed with the KL divergence will be higher for cutpoint 1 while the persistence score computed with the Wasserstein distance will be higher for cutpoint 2. The Wasserstein distance leads here to a discretized time series with more persisting symbols, better respecting the initial intuition of the persistence score.

Candidate cutpoints initialization

Finally, Persist's candidate cutpoints initialization based on an equal frequency (EF) binning allows having more possible cutpoints in high-density regions. However, some time series such as electrocardiograms have a structure that could be missed with this kind of binning. In such cases, an equal-width (EW) binning can be preferable. It is then important to let the user choose in function of the structure of its data.

We re-implemented Persist (originally coded for MATLAB) in Python with the possibility to choose between the KL divergence and the Wasserstein distance, and between an equal-frequency or equal-width binning. It is available online [1].

4.1.3 Experiment on a classification task

To assess the impact of the two alterations on the discretization quality, we conducted two experiments. Although the ultimate goal of this chapter is anomaly detection, we opted to evaluate the algorithm's performance on a time series classification task due to the abundance of recognized datasets in this domain. The first objective was to evaluate the raw discretization quality offered by Persist with and without the proposed alterations. Then, we examined its effectiveness in discretizing time series for TAs learning.

Method

Raw classification performance

We first want to evaluate the amount of information retained in the data after discretization. For a quantitative evaluation, we choose a classification task, as higher classification performance indicates better retention of information in the discretized time series. We conducted this evaluation on 111 datasets of the Time Series Classification Repository [2] (univariate datasets only).

For each dataset, we utilized Persist to generate a set of cutpoints based on the training subset. The train and test time series were then discretized using those cutpoints for the classification task. We trained a Random Forest classifier (arbitrary

1. Link to the repository of Persist re-implementation in Python: `https://gitlab.inria.fr/lcornang/persist_discretization`

2. Anthony Bagnall, Jason Lines, William Vickers and Eamonn Keogh, The UEA & UCR Time Series Classification Repository, www.timeseriesclassification.com

choice) with 100 trees with the discretized train subset. The classification was then performed on the discretized test subset. The accuracy, which is the rate of correct classifications, was used to measure the classification performance.

We tested Persist using either the KL divergence or the Wasserstein distance, and either an equal frequency binning or an equal width binning for the candidate cut-points initialization. Additionally, we compared the results with the well-known SAX method (presented in Section 3.1.1) to have a performance baseline. Unlike Persist, a number of symbols must be given for SAX. We used a number of symbols ranging from 2 to 10, a time interval width of 2, and we report all these results.

Classification performance with Timed Automata

We are interested in Persist to enhance the learning of TA for the underlying system behind the time series. To evaluate its improvement, we need to evaluate the learned models. Figure 4.4 illustrates the experimental setup.

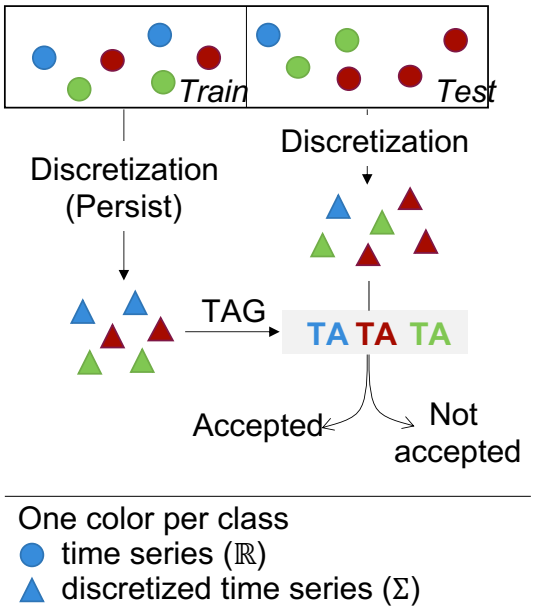


Figure 4.4 – Time series classification using TAs learned after a discretization step.

As in the first experiment, Persist and SAX were used to discretize the time series. However, instead of using the discretized data to train a classifier, we used it to learn one discrete event model per class. For each class, the corresponding discretized train time series were provided to TAG (see Section 2), which produced a TA. Then the dis-

cretized test time series were injected into the TAs. More precisely, each automaton received the discretized test time series of its own class, and an equal number of discretized test time series from other classes. The objective was to ensure that each automaton correctly accepts the data of its own class while rejecting the others. The accuracy was computed as the rate of successful acceptance or rejection by the automaton.

The Time Series Classification repository gathers time series of various types (motion, sensor, traffic, image, spectrographs...). The image and spectrographs types differ from the others as they consist of shapes converted into pseudo time series. As this type of data is not relevant to the problem of modeling dynamical systems, these datasets were excluded from the experiment.

Results

Raw classification performance

Figure 4.5 displays the accuracy achieved according to the discretization method. The classification performance is increased by the replacement of the KL divergence

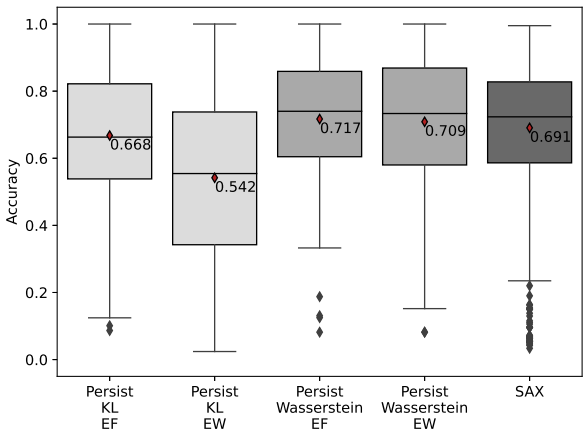


Figure 4.5 – Classification accuracy with random forest for the different discretization strategies. The diamond indicates the mean value. EF: equal-frequency, EW: equal-width.

by the Wasserstein distance. The initialization of the candidate breakpoints by an equal frequency binning leads generally to a better performance, however, it depends on the dataset which confirms our hypothesis. Thanks to the Wasserstein distance,

using Persist globally leads to better results than using SAX in this setup. This indicates that Persist using the Wasserstein distance allows a good information retention in the discretized data.

Classification performance with Timed Automata

Using automata to perform a classification task is unusual and not optimal. Indeed, each automaton is meant to represent a normal global behavior. There is no emphasis for the modeling on what makes the data of the different classes singular. For this reason, we cannot expect as good performances as while using a real classifier. Nevertheless, it is interesting to compare the classification performance according to the discretization method. If the discretization method is pertinent for discrete event modeling, a good part of the information contained in the time series would be retained in the models and therefore leading to good classification performance.

Figure 4.6 displays the classification accuracy using TAs. When using SAX, the

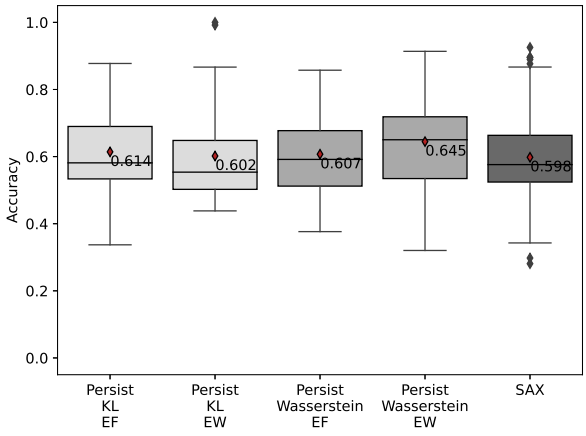
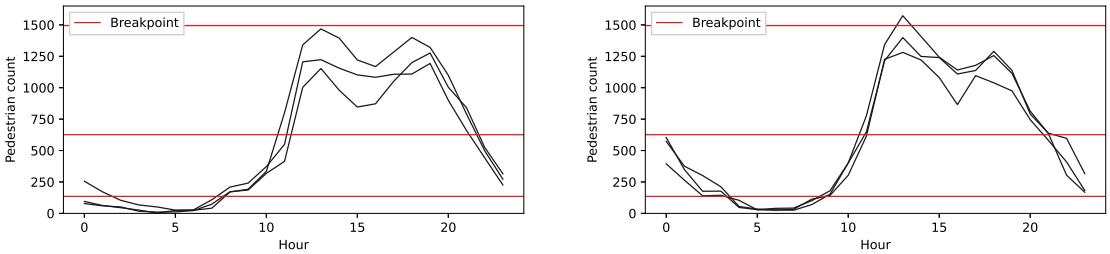


Figure 4.6 – Classification accuracy with timed automata for the different discretization strategies.

classification performance suffers the most from the use of TAs. Persist, in particular while using the Wasserstein distance and an equal-width binning, preserves a better classification accuracy. This confirms the interest in using an improved version of Persist to preprocess time series for TAs learning.

To provide an insight of this experiment, we show the discretization and the discrete event models obtained for one dataset (Chinatown dataset). It consists of the pedestrian traffic along the day in a street of Melbourne. The goal is to classify the

days between weekend and weekday. Figure 4.7 shows instances of time series from this dataset. The best accuracy using TAs for the classification was obtained using the



(a) Weekdays.

(b) Weekends.

Figure 4.7 – Instances of time series from the Chinatown dataset and breakpoints selected by Persist (on the whole train set) with the Wasserstein distance and an equal-width binning.

breakpoints from Persist with the Wasserstein distance and an equal-width binning (accuracy results in Table 4.2). These breakpoints are shown in Figure 4.7 and the intervals they create can be associated with a symbol (very low to very high). Figure 4.8 displays the TA learned for each class. One can note that the activity in the street in generally higher during the night (until 3 or 4 a.m.) on weekends than on weekdays. The street also shows a more pronounced affluence during the weekend than in the weekdays afternoons. On weekdays, the end of the day is either calm, or more animated than during weekend days (with a lower probability, so probably one specific day of the week).

Discretization method	**Accuracy**
Persist (KL, EF)	0.686
Persist (Wasserstein, EF)	0.687
Persist (KL, EW)	0.780
Persist (Wasserstein, EW)	0.819
SAX	0.652

Table 4.2 – Classification accuracy with timed automata for the Chinatown dataset. EF: equal-frequency, EW: equal-width.

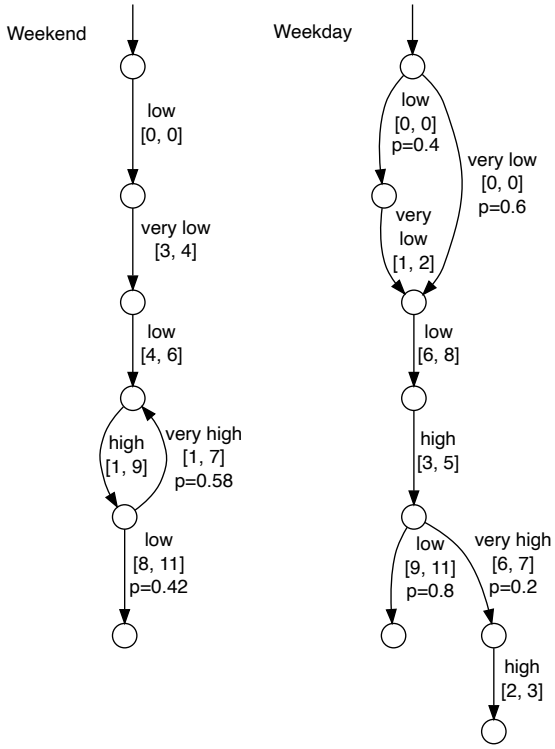


Figure 4.8 – Discrete event model learned for each class of the Chinatown dataset.

4.2 Multi-objective optimization-based discretization for Discrete-Event Systems

In the previous section, we altered an existing discretization method called Persist (Mörchen and Ultsch 2005) by replacing the probability distribution divergence used in its score, the persistence. In order to go further in the development of a specific time series discretization method for TA learning, we want to combine this score with other criteria that are desirable for the resulting discrete event sequences. As a result, we formulate a Multi-Objective Optimization Problem (MOOP) with conflicting objectives. This section presents this MOOP, called Multi-Objective Optimization discretization for Discrete-Event Systems (MOODES), and the discretization strategy developed around it.

4.2.1 Multi-Objective Optimization Problem

Objectives

Three criteria were identified as desirable qualities for discretized time series used to learn a discrete-event timed model of normal behavior.

1. First, having a small number of distinct symbols is preferable in order to keep an interpretable, generalizing, and not too complex model. It will also ease the learning process.
2. However, too few distinct symbols induces an important loss of information between the real-valued data and the discretized data. Minimizing the dispersion of real values corresponding to a same symbol allows counterbalancing such loss of information.
3. Finally, the information contained in a time series not only lies in each individual value but in their arrangement, i.e., in its structure. The persistence score is based on the assumption that the temporal structure of a time series can be preserved if the discretized TS has persisting symbols.

The criteria to optimize are therefore the number of distinct symbols (f_1), the intra-symbol value dispersion (f_2), and the symbols' persistence (f_3).

Given a time series $x = (x_1, ..., x_n)$ defined on $\mathbb{R}$ and its discretized version $y = (y_1, ..., y_n)$ where each value is replaced by a categorical value from an alphabet $s \in \Sigma$, these three objectives are formalized as follows:

$$
\begin{aligned}
\min \quad & f_1(x, y) = |\Sigma| \\
\min \quad & f_2(x, y) = \sum_{s \in \Sigma} \sum_{x_j \in \mathcal{X}_s} \| \, x_j - \bar{\mathcal{X}}_s \, \|^2 \\
& \text{given } \mathcal{X}_s = \{x_i | y_i = s\}_{i \in [1..n]} \\
\max \quad & f_3(x, y) = \frac{1}{|\Sigma|} \sum_{s \in \Sigma} \text{Persistence}_W(s)
\end{aligned}
$$

In this multi-objective optimization problem (MOOP), the objectives are conflicting, it is impossible to optimize all the objectives at the same time. There exists multiple way to map the time series data points to symbols, each exhibiting distinct trade-

offs between the objectives. To identify the best subset of solutions, we use the concept of dominance. A solution x_1 is said to dominate another solution x_2 if it is at least as good for all the objectives and strictly better for at least one. A solution that is not dominated by any other solution is said to be Pareto optimal and the set of Pareto optimal solutions is called the Pareto-optimal set (Sawaragi, Nakayama, and Tanino 1985) (Figure 4.9). One way to find those solutions making a compromise between

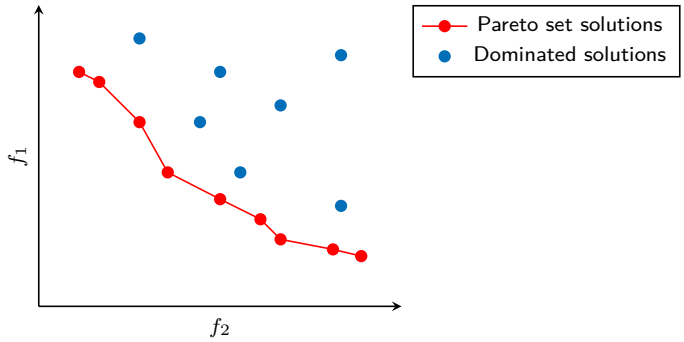


Figure 4.9 – Objective function space for two objective functions to minimize.

the conflicting objectives is to use a Genetic Algorithm (GA).

Solutions

The criteria are defined on both the time series x and its discretized version y. Therefore, the solutions are the mapping between x and y. To limit the optimization process, we need to set a kind of mapping. The time series values will be mapped to symbols using cutpoints in the value range of x. The solutions are therefore sets of cutpoints represented by an ordered vector of real values (Figure 4.10). The transformation of x into y is described in Algorithm 8.

4.2.2 Multi-Objective Optimization Problem solving

Genetic Algorithms are a kind of algorithm inspired by the biological process of natural selection (Goldberg 1989). In the general setup, a set of solutions, called population of individuals, evolves generation after generation, becoming more and more adapted to several criteria. The algorithm starts from an initial population. Every individual is evaluated on the basis of the defined criteria. The better the individuals were evaluated, the greater are their chances to be selected for the reproduction step.

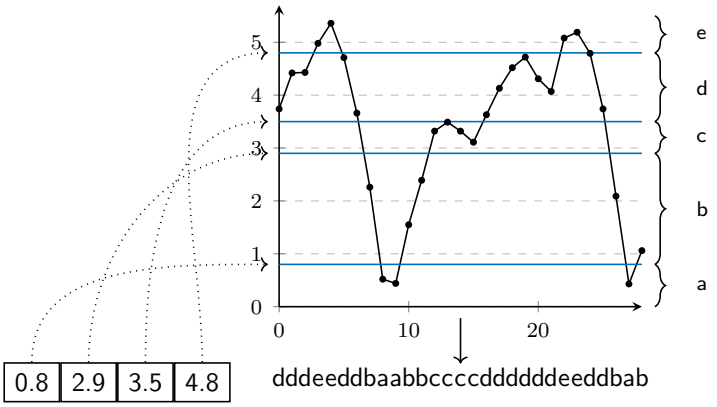


Figure 4.10 – On the left, a set of cutpoints that creates intervals of values ([0,0.8],[0.8,2.9]...). Each interval corresponds to a symbol (a,b,...). The time series is discretized by replacing each value by the corresponding symbol (dddeeddb...).

Algorithm 8 `discretization` (repeated from page 85)

Require: a set of cutpoints cps, a univariate time series ts
Return: a discretized time series $discretized\_ts$

```
discretized_ts = empty vector of the same size as ts
alphabet = vector containing the different symbols
for index = 0 to |ts| − 1 do
    symbol = 0
    while ts[index] ≥ cps[symbol] and symbol < |cps| do
        symbol = symbol + 1
    end while
    discretized_ts[index] = alphabet[symbol]
end for
return discretized_ts
```

During the reproduction step, the individuals for the next generation are created by applying crossovers and mutations to the selected individuals. A crossover is the recombination of two parts of two individuals. A mutation is a slight modification of the individual. The exact mechanisms for selecting individuals or the convergence criterion will depend on the genetic algorithm chosen (for example, NSGA-III (Deb and Jain 2014) is a well-known GA). The choice of algorithm does not impact the result if the GA converges well.

MOODES is not a new genetic algorithm, it should be seen as a configuration: individual encoding, objective functions, mutations, crossover.

A generic pseudo-code for MOODES is presented in Algorithm 10. The exact pro-

cedure will depends of the chosen genetic algorithm. The input is a time series or a

Algorithm 10 MOODES

Require: a univariate time series ts
Return: an ensemble E of sets of cutpoints cps
1: E = population_initialization(ts)
2: **while** not converged **do**
3: **for** each cps in E **do**
4: $discretized\_ts$ = `discretization`(ts, cps)
5: evaluation($discretized\_ts$, ts)
6: **end for**
7: individual_selection(E)
8: mutations(E)
9: crossovers(E)
10: converged = no more evolution in Pareto set solution space
11: **end while**
12: E = keep non-dominated solutions in E
13: **return** E

set of time series corresponding to a same variable. For simplicity, we consider here a single time series ts. The result will be a set of individuals, the sets of cutpoints, making tradeoff between the conflicting objectives. The first population of individuals is initialized using the optimal solutions obtained by the k-means on the time series (minimization of the value dispersion per symbol), plus random solutions (line 1). The individuals are evaluated according to the three previously defined criteria (lines 3 to 6). This step requires the discretization of the time series with each solution. The results for the different criteria do not have to be combined since we use the concept of solution dominance. The selection (line 7) does not only keep the best solutions that are non-dominated (the Pareto set), it also keeps other good solutions to avoid local optima and preserve the diversity of the population. After the selection, three kinds of mutations can be applied to the individuals (line 8):

— The addition of a cutpoint of random value,

— The suppression of a cutpoint,

— Or a small shift of the value of one of the cutpoints using a Laplace noise.

As crossover method (line 9), we apply the simulated binary crossover, which is used in optimization problems having continuous variables to simulate the classical single point for binary-coded problems (Deb and Agrawal 1995). When there is no more

improvement for the different objectives among the individuals of the Pareto set, MOODES terminates and outputs the non-dominated solutions.

4.2.3 Solution selection

Usually, a single solution is selected from the Pareto set. We first considered finding the best solution among the solutions obtained with the genetic algorithm. In the absence of preference among the criteria, the solution can be chosen randomly, or using a solution selection method. We present two solution selection methods.

TOPSIS

TOPSIS (Hwang and Yoon 1981) is based on the concepts of fictive best and worst solutions. The Positive Ideal Solution (PIS) (resp. Negative Ideal Solution (NIS)) is the combination of the best (resp. worst) observed values for each normalized objective. The selected solution should have the shortest Euclidean distance from the PIS and the greatest Euclidean distance from the NIS. When there is a preference among the objectives, each criterion has a weight and an impact that is positive or negative. Given the distance between a solution i and the PIS d_i^*, and the distance between i and the NIS d_i^-, the similarity to the PIS is computed as follows:

$$C_i^* = \frac{d_i^-}{d_i^* + d_i^-}$$

The solution ranking is realized in decreasing order of similarity to the Positive Ideal Solution C_i^*.

Trade-off ranking

Jaini and Utyuzhnikov (2017) proposed to select the solution making the most compromises among the others. They measure the trade-off degree (DT) of a solution A_k as the sum of its distance with all the other solutions.

$$\mathrm{DT}_k = \sum_{i=1}^{q} d(A_k, A_i)$$

with k ranging from 1 to the number of solutions q

$$\text{and } d(A_1, A_2) = (\sum_{j=1}^{m}(A_j^1 - A_j^2)^2)^{\frac{1}{2}}.$$

In function of the shape of the Pareto front, this method won't select the same individual than TOPSIS. A limitation is that this requires a well-distributed Pareto front.

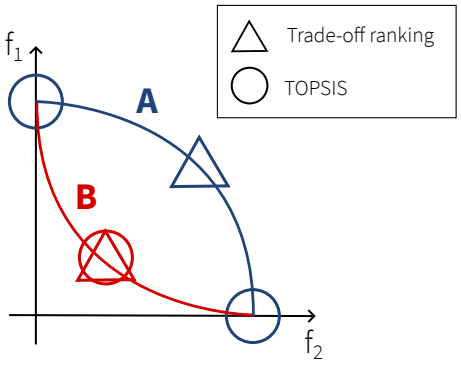


Figure 4.11 – Difference between TOPSIS and trade-off ranking (adapted from Jaini and Utyuzhnikov (2017)).

Example: Difference between TOPSIS and trade-off ranking (example from Jaini and Utyuzhnikov (2017))

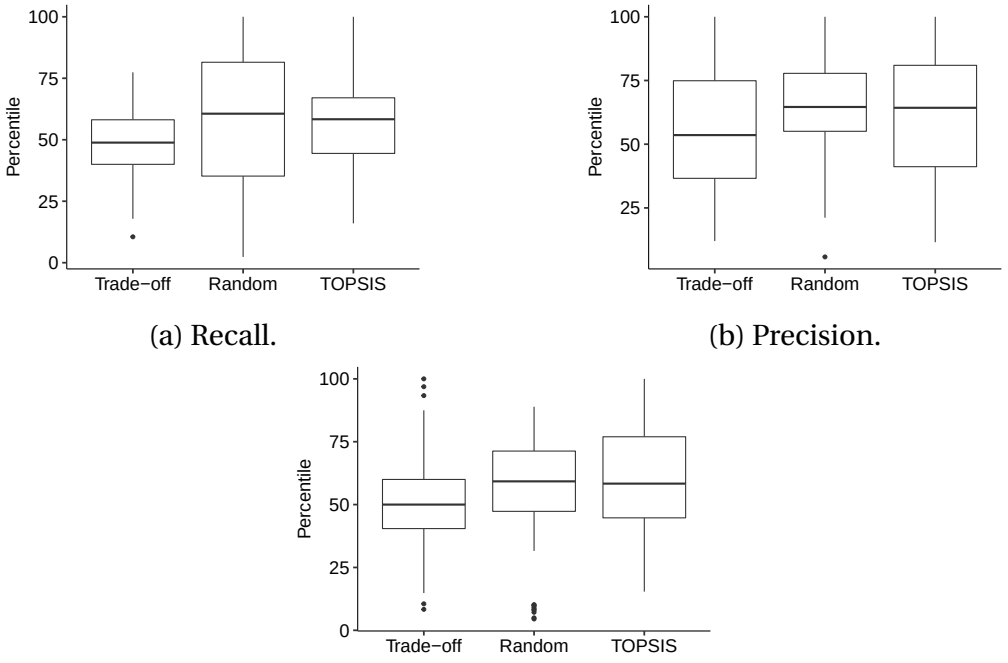
Figure 4.11 illustrates a case where two objective functions f_1 and f_2 are to be minimized, the ideal solution is at the intersection of the two axes. Two possible Pareto front (A and B) are represented. The circles and triangles are solutions of the Pareto set. In the case of the Pareto front A, TOPSIS will select one of the blue circle solutions which are extreme solutions. Trade-off ranking will select a solution that is more balanced under the assumption that the Pareto front is well distributed.

4.2.4 Evaluation of the solution selection techniques

To evaluate if the solution selection technique has an impact on the quality of the resulting automata, an experiment on a classification task was realized. MOODES has been applied to 68 univariate classification datasets [3] from the Time Series Classification Repository. For each dataset, the time series have been discretized using the different solutions of the Pareto set, and a TA has been learned for each class and each

3. The experiment was not performed on all the datasets after analysis of the first results.

solution (see Section 4.1.3 for more detail about discretized time series based classification based on TA). Then, the classification performance of the automata corresponding to each solution of the MOOP were evaluated.



(a) Recall.

(b) Precision.

(c) F1-score.

Figure 4.12 – Score percentiles distribution for each solution technique over all datasets.

The classification performance was evaluated with the precision, the recall, and the F1-score for all the solutions. The percentile achieved by TOPSIS and trade-off ranking's solutions within the whole Pareto set was computed. A Pareto set solution was also chosen randomly for each dataset as control, and its percentile computed as well. Figure 4.12 shows this percentile distribution on all datasets. The classification performance of the automata corresponding to solutions selected by TOPSIS or trade-off ranking are globally at the bottom of the upper half. Therefore, neither of them allow selecting a solution that will lead to the best TA-based classification performance. Yet, they still seem to be a better option than a selection at random that should have led to a uniform percentile distribution from 0 to 1 with a median at 0.5 with more samples.

4.2.5 Conclusion on MOODES

MOODES is a discretization approach based on a Multi-Objective Optimization Problem (MOOP) for system's behavior model learning in the form of TAs. The objectives are the minimization of alphabet size of the future model, the maximization of the symbols' persistence, and the minimization of the dispersion of values from the original time series corresponding to a same symbol. The MOOP is solved using a genetic algorithm that will produce multiple solutions that are non-dominated and make different tradeoff between the objectives.

Due to the absence of positive results during the study of the solution selection methods within the Pareto set, and the chosen final task which is anomaly detection, the strategy is to keep all the solutions. Consequently, for a single variable, multiple discretization cutpoints are obtained. The TAs learned with them will correspond to the system viewed from different perspectives. Thus, the behavior model obtained will be an ensemble of TAs. This should improve the anomaly detection performance by capturing different aspects of the behavior, thus increasing the capacity to detect the anomalies in the time series and increasing the resistance to noise.

MOODES evaluation for TA-based anomaly detection on time series compared to other discretization methods is presented in the next section, after a description of the anomaly detection method from ensemble of TA.

4.3 Timed Automata-based anomaly detection on time series

In this section, we propose a novel approach for anomaly detection on time series based on ensembles of TA. This method takes advantage of both TAG (presented in Section 2) and MOODES (presented in Section 4.2). It is evaluated on synthetic data and on realistic data from a water distribution system challenge.

4.3.1 Motivation

In a Cyber-Physical System (CPS), physical elements are monitored thanks to computation processes based on networked sensor data. Examples of CPS include industrial control systems, autonomous cars and health monitoring systems. They enable

the automatic control of complex and critical systems in real-time. The counterpart of their automation is that is makes malfunctions harder to detect by the users and their networked operation increases their vulnerability to attacks. Consequently, it is important to have an anomaly detection strategy along their functioning. The challenge is not only to detect the anomalies as fast as possible but also to locate the faulty component in real-time and to explain to end-users what happened on the system. Sensor data, which often takes the form of time series, is a precious source of information about the system's state to detect the anomalies.

Anomaly detection in time series is an active field of research. An extensive overview of anomaly detection techniques for Time Series (TS) and other temporal data has been carried out by M. Gupta et al. (2014). Currently, the approaches based on statistical and deep learning techniques are the most widely used given the good accuracy scores they offer on the benchmark datasets and most of the recent publications are exploring autoencoders (Audibert et al. 2020) and transformers (J. Xu et al. 2021) solutions. The drawback of these approaches is that it is difficult to get an interpretable justification for their decision, which can complicate the human decision, leading to distrust and concerns about the corrective action to be taken. In addition, these algorithms often require a considerable amount of data for their training.

Here, we propose an alternative approach based on behavior model of the system, enabling to locate and explain anomalies in real-time. The behavior models take the form of TAs automatically inferred by TAG, and the gap between continuous data and symbolic semantic is bridged by MOODES discretization method.

4.3.2 Anomaly detection with ensemble of Timed Automata

Figure 4.13 presents the overview of the TA-based anomaly detection method. We first describe the left part, which relate to the learning of the behavior models realized offline. Then, we delve into the online anomaly detection part on the right.

Models learning

This anomaly detection method is based on models of normal behavior of the system. Any new data that deviates from these models will be considered anomalous. The models take the form of an ensemble of TA for each component of the system.

The learning part is realized offline from historical time series data related to the

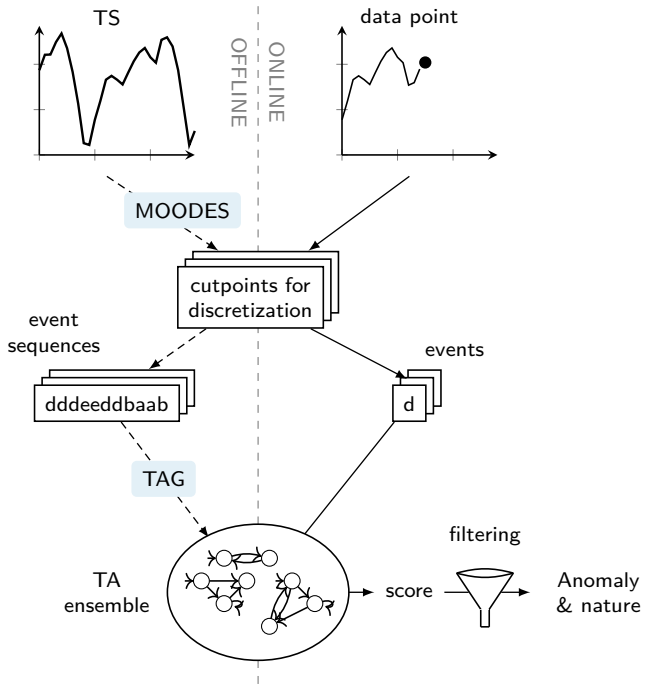


Figure 4.13 – Approach overview.

system, which must be free of anomalies. We assume that each component has its own time series data. For each component, its respective time series data is provided to MOODES, which generates multiple discretization solutions. All solutions are retained, resulting in multiple event sequences for each time series. These event sequences are used independently by TAG to produce a Deterministic Real-Time Automaton (DRTA). This results in an ensemble of DRTAs representing different aspects of the component's normal behavior.

Overview of the anomaly detection

Anomaly detection is performed in an online setting, meaning that new observations arrive continuously. The data must be classified as normal or abnormal at each time step (right part of Figure 4.13). Algorithm 11 outlines the online anomaly detection procedure for all components of the system.

For now, we omit the calibration step (lines 1 to 3). At each time step, new observations are received (line 4). It consists of one data point per component or variable. These new observations are discretized using the same sets of cutpoints that were used to discretize the historical data for the TAs learning step (line 10). The consis-

Algorithm 11 Anomaly detection (online)

Require: new streaming multivariate time series obs, a sample of historical data without anomaly $calib\_mts$, ensemble E of pairs of TA and associated cutpoints (ta, cps) for each variable

1: **for** each ts in $calib\_mts$ **do**
2: τ = threshold_calibration(ts, E) ▷ different for each component
3: **end for**
4: **while** new incoming multivariate observation obs **do**
5: $t = t + 1$
6: **for** each variable observation pt in obs **do**
7: E is the ensemble corresponding to the variable
8: $AS_{E,t} = 0$
9: **for** each (ta, cps) in E **do**
10: $symbol$ = `discretization`(pt, cps)
11: $a_{ta,t}$ = is_inconsistent($symbol$, ta) ▷ return 0 if consistent else 1
12: $AS_{E,t} = AS_{E,t} + a_{ta,t}$
13: **end for**
14: $AS_{E,t} = \frac{AS_{E,t}}{|E|}$
15: $CuDAS_{E,t} = \sum_{i=0}^{w_{size}} \frac{AS_{E,t-i}}{i+1}$
16: **if** $CuDAS_{E,t} > \tau$ **then** an anomaly is detected
17: **end for**
18: **end while**

tency with each TA is checked (line 11), and the proportion of inconsistencies within the ensemble of TA gives an anomaly score (line 14). Since high peaks of anomaly scores are as alarming as smaller but recurring peaks, we compute an "augmented" score that takes into account the historical scores for the same component (lines 15 to 18) Finally, an alert is raised if the augmented anomaly score is over a threshold (line 16), fixed during the calibration step (line 2).

In addition to the anomaly alert, we also offer as explanations the components where the anomalies were found, as well as the nature of the inconsistencies within the models.

We now detail each step.

TA-based anomaly detection

The consistency between the new observations with the TAs is checked in line 11 of Algorithm 11.

The TAs are monitored continuously, i.e. their current state and clock value, which depends on the history of events, is known and actualized in real-time. When the anomaly detection begin and the first data arrives, the TAs are in their initial state. When a new event occurs, if the current state has an outgoing transition labeled with that event and a timing constraint concurring with the observed delay, the transition is triggered and the current state is now the destination state of the transition.

An inconsistency can arise for three reasons:

— Alphabet inconsistency: If the event isn't in the automaton alphabet, it means that the value doesn't correspond to the usual values of the component.

— Symbolic inconsistency: If the event belongs to the alphabet, but there is no transition with this symbol from the current state, the value is considered as unusual given the previous observations.

— Temporal inconsistency: If there is a transition with the required event, but the delay with the last event doesn't meet the timing constraint of the transition, it means that the duration of the event is unusual.

When an inconsistency occurs, the automaton returns to its initial state.

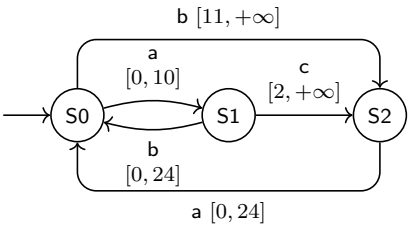


Figure 4.14 – Example of Timed Automaton.

Example: Anomaly detection with a TA

Let's consider the TA presented in Figure 4.14.

Initially, the current state is $S0$, and the value of the clock that measures the delay between two events is 0. The first observation is a: there is a transition from the current state labeled with a and whose guard is respected, therefore the new current state is $S1$. There is no new event for 10 time steps, so the current state does not change either. After a c event, the current state is updated to $S2$. Then, an event a happens after 30 time steps. This is a temporal inconsistency: there is a transition with that event from the current state, but the guard $[0, 24]$ is not respected. Consequently, an anomaly is raised, the current state is reset to $S0$, and the clock value

reset to 0.

In the same manner, from $S2$, an event b in place of a would also be a symbolic anomaly, and an event d (or any other unknown event) would be an alphabetic inconsistency.

Anomaly score

In a component's ensemble, some automata may have detected an anomaly, while some others not. There are multiple ways to combine the predictions of the models in an ensemble, such as taking the maximal score value or using a majority voting approach (Zhao 2017). We apply the simplest combination method, which is an unweighted average of the score of all models.

For each automata ensemble E (i.e., component), the anomaly score at time t noted $AS_{E,t}$ corresponds to the ratio of automata where an inconsistency was found, giving a score between 0 and 1:

$$AS_{E,t} = \frac{1}{|E|} \sum_{A \in E} a_{A,t}$$

$$\text{where } a_{A,t} = \begin{cases} 1, & \text{if inconsistency,} \\ 0, & \text{otherwise.} \end{cases}$$

This anomaly score is computed from line 8 to line 14 of Algorithm 11.

A high peak of anomaly score at a given time is alarming because it means that the observation is not consistent with many behavior models. The deviation from normal behavior is clear, so an alert should be raised. However, the observation of smaller but recurring spikes should also be considered as alarming because it means that over a short period of time, many behavior models have found unexpected data. Those multiple small deviations should also result in raising an alert. To take this into consideration, we don't directly place a threshold on the anomaly score, but rather compute an augmented anomaly score that enhances the anomaly score at a given time based on the anomaly scores previously observed. Unlike smoothing measures such as the Exponential Moving Average (EMA), our measure can only increase the value in function of the past values, and not decrease it. Indeed, when an anomaly starts, the fact that no anomaly was detected just before is normal, and the anomaly score should not be reduced because of it. This Cumulative Decayed Anomaly Score

(CuDAS) is computed as follows:

$$\text{CuDAS}_{E,t} = \sum_{i=0}^{w_{size}} \frac{AS_{E,t-i}}{i+1}$$

In Algorithm 11, CuDAS is computed from line 15 to line 18.

The value of the window size w_{size} that controls the number of past anomaly score values to take into account has actually little importance, since the contribution of values far from the current timestamp quickly becomes negligible.

Filtering

Model-based anomaly detection techniques are particularly sensitive to false alarms because a slight deviation from the learned normal behavior may lead to raising an alarm. Such alarm may not be an anomaly, but just a normal behavior that has not been captured in the learning phase. Because we use multiple models for each CPS component, this further increases the risk of false alarms, leading to non-zero anomaly scores. To mitigate this, we want to filter the anomalies that will be actually raised according to the CuDAS. Zohrevand and Glässer (2020) have realized on overview about automatic false alarm mitigation in anomaly detection. Many methods rely on the data distribution, which require a large amount of data to get a reliable fit. Here, this filtering must be performed online, to be able to raise the alerts as soon as suspicious data arrives. We realize this filtering based on a threshold fixed after a calibration step that learns a basal error noise for each component.

We want to learn the basal error noise of our ensembles of models to set a proper threshold for each component. To do so, we need a calibration step which is performed at line 2 in Algorithm 11. During a certain amount of time, new observations are injected into the models and the anomaly scores are computed. Under the assumption that no real anomaly occurred during that time period, the maximal value of cumulative anomaly score obtained or a high percentile can be used as threshold. An anomaly score under this threshold will be considered as ordinary noise. It is necessary to use data that wasn't used to learn the models, otherwise those observations would all be consistent with them.

Output

When an alert is raised at a given time step, an explanation is provided. A log reporting a single anomaly will look like this: "Anomaly at time t on automaton a of component c: maximal time delay exceeded by x time units for transition from state s_1 to state s_2 with event e". When considering all reported anomalies, automata ensembles with an anomaly score above the threshold are listed in decreasing order of score. The ratio of automata with inconsistency in their ensembles is given, as well as the nature of the inconsistencies given in the form of the ratio of alphabet, symbolic, and timing anomalies. Furthermore, the user can have access to the automata individually and can display them to see the problematic paths. With this information, the system supervisor(s) can diagnose the situation as described in the experimentation section and implement corrective actions.

4.3.3 Experiment on synthetic data

To validate the approach, we conducted an experiment on simple synthetic time series. The anomaly detection is performed on data generated randomly and in which anomalies were injected.

Method

We generated two datasets: one with noise and one without, each consisting of 5 variables. The design principle for each time series (variable) is a pattern that is repeated with or without noise (Figure 4.15). These patterns were generated using a Gaussian random walk with a random standard deviation, and have the same length. The noise is drawn from a Laplacian distribution.

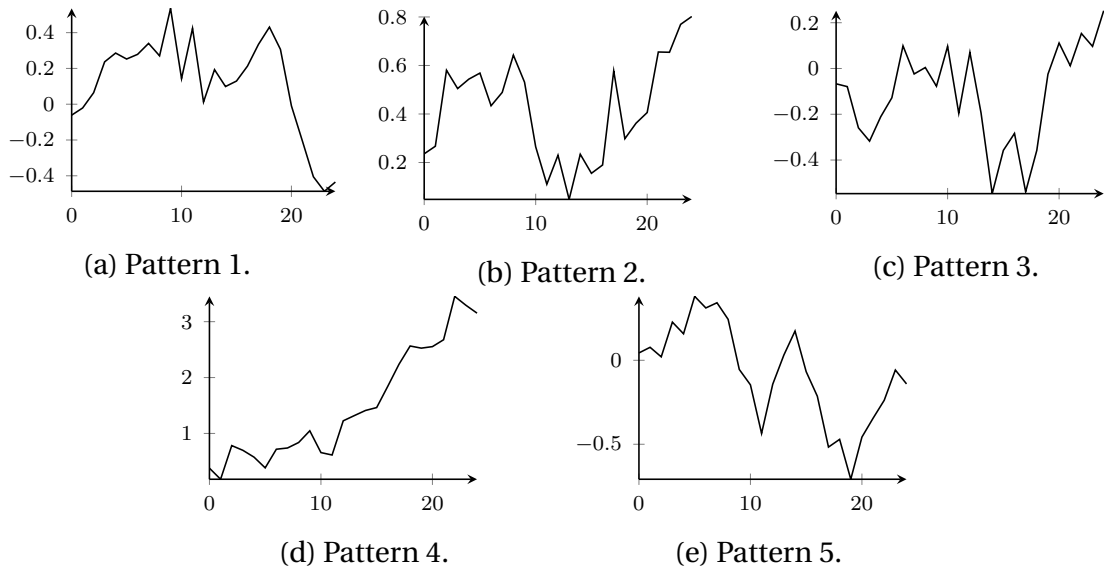
Each pattern was repeated for 1000, 500 and 750 time units, resulting in the training, calibration, and test time series. We then injected anomalies into the test time series (one anomaly per time series at a distinct time step, plus one time series free of anomaly). The anomalies last 50 time units and consist of the following:

— Random values;

— Values multiplied by a factor 1.5;

— Slowed pattern with a factor 1.5 (Figure 4.16);

— Reversed pattern.



(a) Pattern 1.

(b) Pattern 2.

(c) Pattern 3.

(d) Pattern 4.

(e) Pattern 5.

Figure 4.15 – Time series patterns.

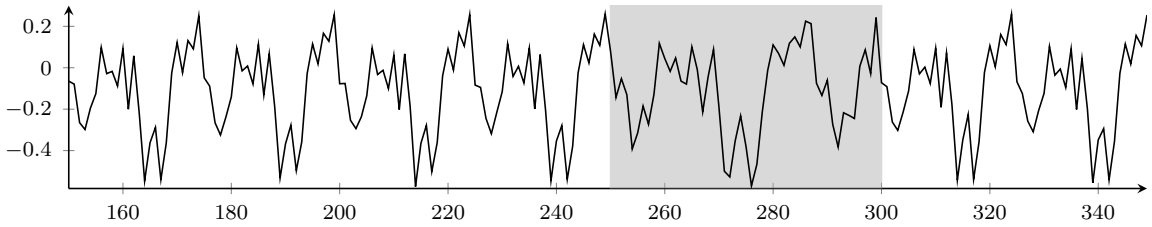


Figure 4.16 – Subset of the test time series for variable 3. The pattern is slowed from $x = 250$ to $x = 300$.

The learning time series are given as input to MOODES that produces an ensemble of discretization solutions for each variable. The learning, calibration and test time series of each variable can then be discretized, and TAG is used to learn an ensemble of TA per variable. The calibration data is used for the calibration of the anomaly detector, and the anomaly detection is performed on the test data.

Results

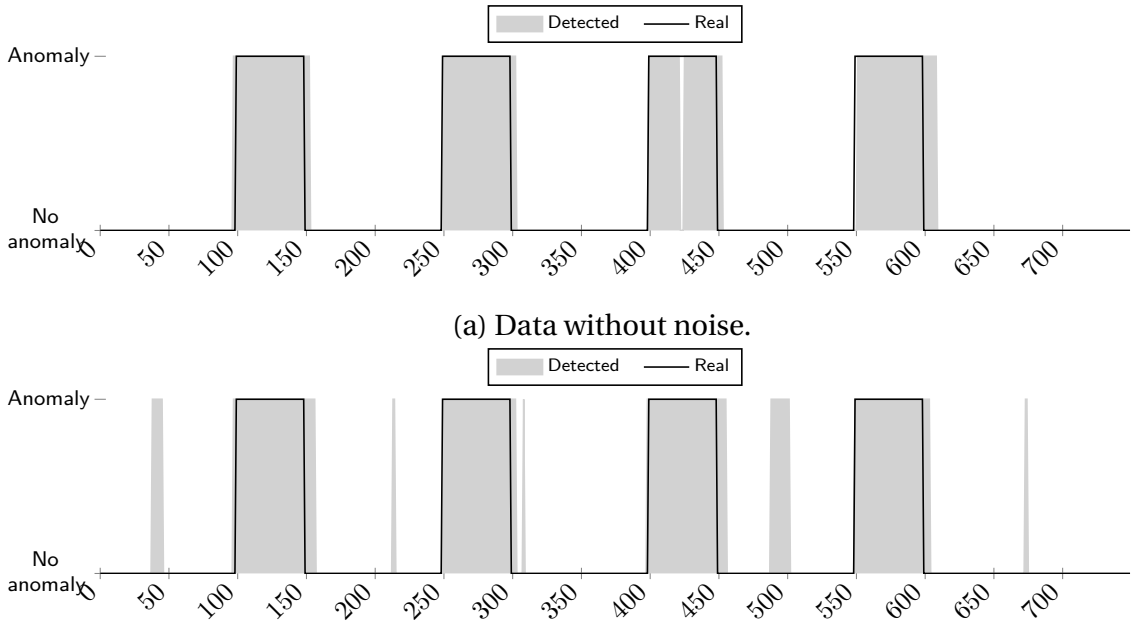
Table 4.3 presents the values for the recall, precision, and F1-score.

In the case of data without noise, the calibration thresholds were set to 0 during the calibration since the pattern was exactly repeated and the calibration data perfectly matched the learning data. In the test data, the pattern was also exactly repeated (excluding during anomalies). Except for a small period where the injected

Table 4.3 – Anomaly detection performances on synthetic data.

Noise	TPR	PPV	F1
No	0.97	0.89	0.928
Yes	0.995	0.786	0.878

anomaly resulted in normal data, the automata successfully detected the anomalies, leading to a positive anomaly score (which automatically is over the threshold). However, due to the cumulative effect of the CuDAS, an anomaly flag continued to be raised during the cumulative window size after the anomaly ended. Consequently, the precision is not equal to 1. This effect is generally minimized with a non-zero calibration threshold.



(a) Data without noise.

(b) Data with noise.

Figure 4.17 – Detected and real anomaly period.

In the case of data with noise, the anomalies were well detected, but there were more false alarms due to the presence of noise (Figure 4.17).

In addition to these global results, one may look at the generated automata to get an explanation of the anomalies. For instance, the timed automaton shown in Figure 4.18 detected an anomaly while in state S3. It received an event discretized as "2" after a duration of 10 time units. The transition guard indicates that this duration

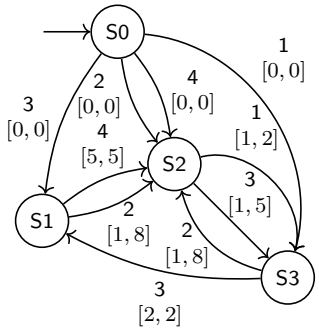


Figure 4.18 – Timed automaton for variable 3.

should have been at most 8, so an anomaly was raised. In the data, it can be found that it corresponds to a slowed instance of the input pattern in the automaton variable, confirming the correct detection, and detecting it due to a time delay exceeding the guard was consistent with the nature of this anomaly.

4.3.4 Experiment on BATADAL challenge

The approach have then be tested on a reference anomaly detection dataset called BATADAL. The BATtle of the Attack Detection ALgorithms (BATADAL) (Taormina et al. 2018) is a challenge that was organized within the Water Distribution Systems Analysis Symposium in 2016. Participants were asked to propose online anomaly detection systems for CPS. For this, they were given simulated data from sensors on physical assets in a Water Distribution System (WDS). This experiment aims to evaluate the approach on more complex and realistic data, and to compare its quantitative and interpretability performances with the other techniques for anomaly detection in time series.

Data

The water distribution system, represented in Figure 4.19, consists of 7 water tanks fed by 11 pumps and 5 valves (actuators) via pipes from a single reservoir. These physical assets are monitored and controlled by network devices such as Programmable Logic Controllers (PLCs) and a Supervisory Control and Data Acquisition (SCADA) system. PLCs control the pumps and valves according to the tank's water level and record the water flow in the actuators, the inlet and outlet pump pressure, and the water flow passing through the pumps. Especially, the pumps are activated when the

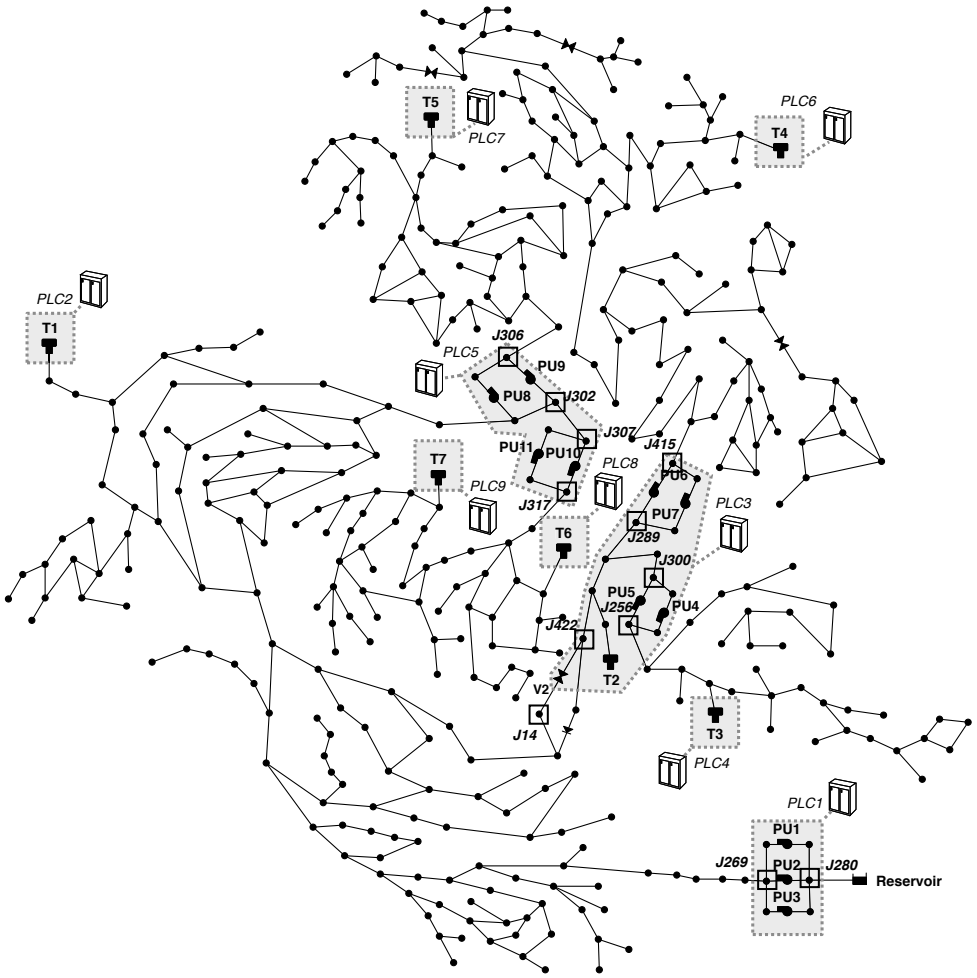


Figure 4.19 – Aerial view of BATADAL water distribution system. The nodes correspond to the junctions and the edges to the pipes. (Adapted from Taormina et al. 2018)

water level falls below a threshold and deactivated when the water level exceeds another threshold. The SCADA system stores the readings provided by the PLCs and coordinates the operations in the WDS.

The data consists of the hourly SCADA readings for the 43 variables (tank levels, actuator status, flow in actuators, actuator inlet and outlet pressure measured at junctions). A first dataset contains 365 days of data without any attack. A second dataset contains 174 days with attacks partially labeled. The third dataset (test) is the one on which the anomaly detection methods are evaluated and contains 87 days of data. Additionally, the normal settings for the (de)activation of the actuators depending on tank levels are known.

Attack description

There are seven attacks to detect in the test dataset, ranging in duration from 30 hours to 100 hours. During the attacks, the attacker gains control of PLCs to change actuators settings, directly (de)activate actuators, or alter readings arriving to PLCs. At the same time, they partially replace actual traffic information between the PLCs and the SCADA system with previously recorded data (replay attacks) to hide effects of the attack.

Existing approaches

Several works have tackled this challenge. Most are presented in the main publication of the challenge (Taormina et al. 2018). The method that achieves the best results (Housh and Ohar 2018) is specific to WDS. Water demand in the network was estimated with a Mixed Integer Linear Programming (MILP) optimization, using a software for WDS modeling to generate the corresponding data. The error between the simulated data and the actual BATADAL data was used to raise the anomalies. Other approaches are based on Principal Component Analysis (PCA), to separate the normal data from the abnormal data spaces (Abokifa et al. 2017; Giacomoni, Gatsis, and Taha 2017; Quiñones-Grueiro et al. 2019). The use of Variational Auto-Encoder (VAE) has also been widely investigated (Chandy et al. 2017; Gjorgiev and Gievska 2020; Stojanović et al. 2022). Other deep learning techniques have been applied to BATADAL such as Recurrent Neural Networks (RNN) to predict tank levels based on the other variables (Brentan et al. 2017). Finally, classification techniques have been tested, using Random Forests or by checking if the normal actuator activation rules were respected (Aghashahi et al. 2017; Pasha, Kc, and Somasundaram 2017).

If these methods can, for some, reach high performances in anomaly detection, they are either application-specific, require negative data, or their output lacks explainability. In order to intervene on the system when an alert is raised, end users should have the localization of the component that presents an abnormal behavior and explanations about why it is considered abnormal.

Method

We adapted our approach to the BATADAL challenge.

Discretization and model learning

To learn the CPS behavior models, we used the first dataset, which contains benign data only. We discretized the time series of the pressure values and the flows in actuators using the different solutions produced by MOODES. For the level of the tanks, we took advantage of the expert knowledge and directly fixed the activator activation threshold as discretization cutpoints because this allows us to check the compliance with the settings (threshold). Although it is a specificity of this challenge, this allows us to make a fairer comparison with the other approaches since it was done by almost all of them. From the discretized data, we used TAG to learn the behavior models, getting an ensemble of TAs for each pressure and actuator flow sensor, and a unique TA for each tank and actuator status to consider jointly. For MOODES, we used as genetic algorithm NSGA-III (Deb and Jain 2014). We initialized the first population of individuals with solutions having a minimal intra-symbol dispersion for a number of symbols between 2 and 20 using the k-Means algorithm, and with random individuals. We set the population size to 500. For the generalization parameter of TAG, we choose the default value ($k = 2$). Automata guard corresponds to the minimal and maximal delay observed with a 3σ tolerance during consistency check.

Anomaly detection

We performed the anomaly detection on the third dataset, which contains malicious attacks. To fix the threshold to filter the anomalies, we used the 1000 first observations of the second dataset because we know that no attack was performed during this period and they were not used to train the models. Since the actuator status and flow correspond to the same component in the physical system, we consider their results jointly at the time to filter the anomalies. Finally, most attacks in BATADAL are at least partially concealed with replay attacks (replacement of the real readings by previously recorded data). The received data corresponds to true data and therefore has no reason to be considered as anomalous by the TA. Therefore, we also checked for each TA of pressure and flow if the event sequence was not repeating itself in terms of event and duration. We kept in memory the last 50 events and raised a replay anomaly if a repetition of at least 10 events is observed.

Evaluation of the discretization method impact

To confirm the interest in a discretization method specifically dedicated to our purpose, we also performed the experiment using other discretization methods. In place of MOODES and keeping the same experimental setup for the rest, we used Persist (the algorithm from which our persistence score in MOODES comes) and SAX (both presented in Section 3.1.1). While Persist finds an optimal number of symbols by itself, SAX requires the number of symbols as a parameter. We produced the discretization representation for a number of symbols varying from 2 to 20, using the same number for all the components. We only present the best results (three symbols).

Furthermore, to evaluate the benefit of using an ensemble of models, we also tested MOODES with a method that selects only one unique individual in the Pareto set. Let us assume that the ideal individual is a fictitious individual whose score for each objective is the best-observed value among true individuals. To select the unique individual, we used TOPSIS, presented in Section 4.2.3. As with Persist and SAX, the anomaly detection was performed using a unique automaton per component.

Evaluation of the anomaly detection performance

To evaluate our method's anomaly detection performance, we computed recall, precision, and F1-score. Recall or True Positive Rate (TPR) is the ratio of abnormal periods actually labeled as abnormal. Precision or Positive Predictive Value (PPV) is the ratio of timestamps labeled as abnormal when there was indeed an anomaly. F1-score is the harmonic mean of these two measures. To evaluate the interpretability of our output, we also make reports of the detected anomalies during the attacks. We compute the total CuDAS (Cumulative Decayed Anomaly Score) which is the sum of the CuDAS of the components where an anomaly was detected over the attack period. We rank the components in order of their contribution to this total CuDAS and present the distribution of the nature of their anomalies.

Results

We present the results obtained in terms of quantitative performance and interpretability of the output.

Anomaly detection performance

Table 4.4 presents our approach's anomaly detection performance using TA ensembles learned from data discretized by MOODES (our discretization method), and using a single TA learned from data discretized by Persist, by SAX, or by MOODES but with solution selection by TOPSIS. What stands out is the benefit of using MOODES,

Table 4.4 – Anomaly detection performance of our TA-based approach in function of the discretization method.

Discretization method	TPR	PPV	F1
MOODES (TA ensemble)	*0.826*	*0.724*	*0.772*
MOODES + TOPSIS (single TA)	0.455	0.678	0.544
Persist	0.388	0.681	0.495
SAX	0.420	0.435	0.427

which is specially designed for discrete-event model learning. The interest in the ensemble of models is also clear in comparison with the results using a single model per component (MOODES vs. MOODES + TOPSIS). The significant gain in recall is even joined by an improvement in precision, which confirms the robustness to noise brought by the ensembles. The mapping between the real attack periods and the de-

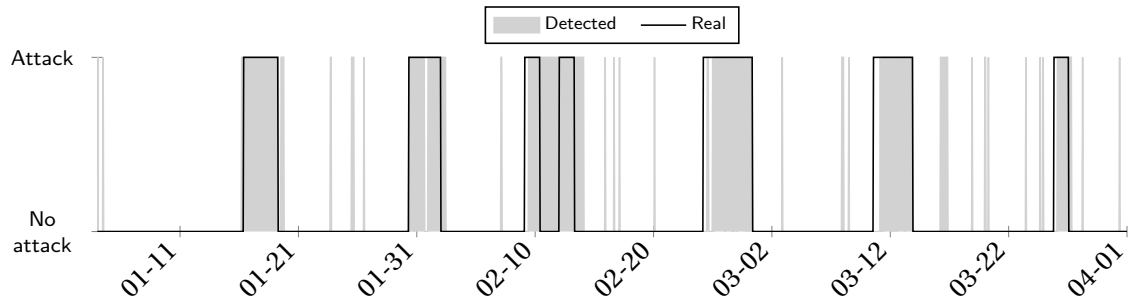


Figure 4.20 – Detected and real attack period.

tected periods is shown in Figure 4.20. The real attacks are well detected, and there are few false positives immediately before or after them: the start and stop of the real anomaly period are well detected by our approach. Without surprise, our method doesn't reach the performance of the WDS-specific method (Housh and Ohar 2018) which achieves a F1-score of 0.970. However, we achieve a much higher performance

than the state-of-the-art most interpretable method (Pasha, Kc, and Somasundaram 2017) which is based on rules and obtained a F1-score of 0.483, while the explanatory potential of our method (shown in next section) is not reached by the aforementioned approaches. We also outperform State-of-the-Art methods based on large deep neural network models such as Anomaly Transformer (J. Xu et al. 2021) or USAD (Audibert et al. 2020) (with a classical evaluation protocol instead of the *point-adjust* method). Table 4.5 summarizes those results.

Table 4.5 – Comparison of the anomaly detection performance of different approaches.

Discretization method	**TPR**	**PPV**	**F1**
WDS-specific method (Housh and Ohar 2018)	0.953	0.987	0.970
MOODES (TA ensemble)	0.826	0.724	0.772
USAD (Audibert et al. 2020)	0.561	0.312	0.401
Rule-based method (Pasha, Kc, and Somasundaram 2017)	0.329	0.905	0.483
Anomaly Transformer (J. Xu et al. 2021)	0.059	0.250	0.095

Ablation study on anomaly detection

As explained above, we made some adjustments to the general approach to fit to the challenge, which are a simple check for repeated event sequences (i.e., replay attack) and the use of tanks and actuators automata to check the compliance with the given activation threshold. To complete the ablation study on the discretization method and to provide transparent results, we also present the results to the challenge using only the ensembles of automata learned after a discretization by MOODES (Table 4.6). Although these adaptations to the challenge increase the performance (in particular for checking replay attacks), the performance is delivered for the most part by our new approach.

Table 4.6 – Ablation study on the anomaly detection part.

Experimental settings	**TPR**	**PPV**	**F1**
All checks	0.826	0.724	0.772
No replay check	0.597	0.670	0.631
No actuator status and tank level coherence check	0.816	0.706	0.757

Table 4.7 – Contribution to the sum of CuDAS all components included (%) and anomaly type distribution during attacks 4 and 1 for the components whose part of total CuDAS is over 10%.

	%	Alphabetic		Symbolic	Temporal		Replay
		low	high		short	long	
PU1	0.61	0.97	0	0.01	0.01	0	0
PU2	0.27	0	0	0	0	1	0

(a) Attack 4

	%	Alphabetic		Symbolic	Temporal		Replay
		low	high		short	long	
J256	0.56	0	0	0.51	0.04	0.43	0.02
J302	0.18	0	0	0.01	0.25	0.74	0
PU4	0.11	0	0	0	0	0	1

(b) Attack 1

Anomaly explanation

Modeling the behavior of each component allows to locate the anomalies. Using TA is also useful to understand the anomalies.

Let's take the example of the fourth attack, which consists in the malicious activation of a pump (PU3). Because it has never been activated in the dataset containing normal data only, no behavior model has been learned for it. However, we still managed to detect the anomaly. Table 4.7a presents a report for the period of this attack. Pump PU1 was the main component affected by the anomalies, followed by pump PU2. Most of the anomalies for PU1 were alphabetic (unknown symbol corresponding to low flow value), while they were temporal for PU2 (too long delays between transitions). Those two pumps are next to pump PU3 and supply water in the same area. Because PU3 was maliciously activated in a continuous manner, it wasn't necessary to activate PU1 causing its flow to be abnormally low (alphabetic anomaly), and PU2 was kept activated for less time than usual (temporal anomaly).

During the first attack, the level thresholds of tank T3 which controls the activation of two pumps (PU3 and PU4) were maliciously modified. In addition, replay attacks were launched on the tank level and the pump flow data to mask the attack. Table 4.7b presents a report for the period of this attack. The component responsi-

ble for the most anomalies is the downstream junction of pump PU4 (J256) where its outlet pressure is measured. Anomalies are mostly temporal (irregular delay between transitions) and symbolic the rest of the time (non-existing transitions). Changing the thresholds caused PU4 over activation and deactivation, which was detected through its outlet pressure in spite of the replay attack (which also was detected for PU4). Thanks to these reports, corrective actions could be carried out on the affected components.

4.3.5 Conclusion on the anomaly detection approach

This section described a new approach for time series anomaly detection based on behavior models in the form of TAs, and which takes advantage of MOODES' ability to generate multiple discretization solutions. The great advantage of using TAs to describe system behavior is their ability to provide explanations about the nature of anomalies and their location (component) in the system. Experimental results emphasize the significant role of the discretization method and confirm the interest in a specific one. The robustness to noise has been provided by the ensembles of timed automata that reinforce the modeling of the possible behaviors for each component of the system. Our performance in anomaly detection is effective and outperforms other comparable methods based on symbolic discretization (SAX, Persist) or providing interpretable results on the BATADAL challenge. Thanks to TAs, useful and detailed explanations are provided to qualify and locate the anomaly.

CHAPTER 5

SYNCHRONIZATION-PRESERVING DISCRETIZATION

In the previous chapters, our analysis primarily focused on individual systems and their internal behavior, overlooking the interactions between systems or components within a system. In this chapter, we shift our attention toward the identification of a specific type of interaction: the synchronization. Based on the formalism of Timed I/O Automata, we propose to identify, given a multivariate time series, events from different variables whose occurrences are synchronized. The result of this ongoing work is a synchronization-preserving discretization algorithm for multivariate time series. The method is validated on synthetic data.

5.1 Problematic

In a system composed of multiple components, each component has its own internal dynamic, plus a global dynamic resulting of the interactions with the other components. These interactions can take the form of synchronization, i.e., the coordination of operations from different processes or components.

Example: Synchronization between components

Let's consider a production line in a factory with two robotic arms performing different operations on a same product (Figure 5.1). Each arm as has its own set of operations, which can be executed simultaneously and independently on the product. However, certain operations require synchronization between the two arms to ensure proper sequencing.

In particular, the first arm must lift the product after completing its operations so that the second arm can perform a final operation on it (Figure 5.1). The first arm cannot lift the product immediately after completing its own operations, it

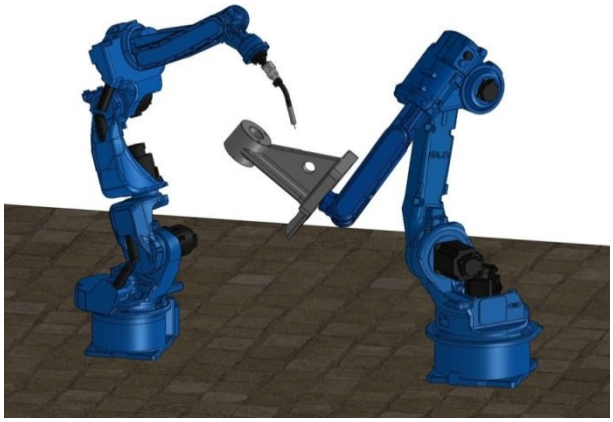


Figure 5.1 – Two robotic arms cooperating (from Gan, Dai, and Li (2013)).

must wait for the second arm to complete its operations. Similarly, the second arm cannot perform its final operation until the first arm has lifted the product.

Some operations of the arms are synchronized. Consequently, the behavior of each arm cannot be fully captured without considering the other arm.

To model such systems, the formalism of Timed I/O Automata (TIOA) (Kaynar et al. 2006) (Figure 1.7), which was presented in Section 1.2.2, is particularly adapted. A

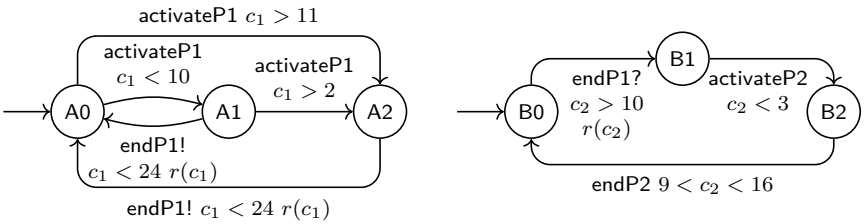


Figure 1.7 – Graphical representation of two parallel Timed I/O Automata (TIOAs). Pairs of input and outputs events are followed respectively by ? and !. (repeated from page 33)

TIOA is a Timed Automaton whose events are partitioned into a set of *input events*, a set of *output event*, and a set of *internal events*. Input and output events, generally indicated by a *?* and *!* and simply referred to as inputs and outputs, respectively, are used for the synchronization between multiple TIOAs. Input event labeled transitions can be triggered only if a transition with the corresponding output event is triggered in a parallel automaton. In Figure 1.7, the transition from state $B0$ to $B1$, labeled with input event $endP1?$, is possible only if the transition from $A1$ to $A0$ or the transition

from $A2$ to $A0$, both labeled with output event $endP1!$, is triggered in the other automaton. The other transitions can be triggered independently. In this example, this synchronization prevents the activation of a process (P2) before the end of a process (P1) in another component.

Given two event sequences with identified inputs and outputs, each sequence corresponding to a component of a system and recorded at the same time, it is possible to learn TIOAs (one per component) with any classical passive TA learner. Each component will be modeled individually and the input and output events will be handled as classical events. The synchronization will then be ensured by the program used to monitor the TIOAs.

If the events are not distinguished between internal, input, and output in the event sequences, and input and output of a pair do not have the same name, the pairs of input and output can be identified using for example a pattern mining algorithm (Mannila, Toivonen, and Inkeri Verkamo 1997) that will identify the events occurring simultaneously within the sequences.

However, learning TIOAs from raw multivariate time series is much more challenging. Let's first clarify the meaning of *synchronization* in the context of time series. In a multivariate time series, a synchronization is a pattern of the form: if i occurs in one variable, then o is occurring in another variable, where i and o are particular shapes, trends, or values, and correspond to the input and output, respectively.

In rule mining (in time series), the i and o used for the rule are typically mined from event sequences obtained by a discretization step that has assigned the same symbol or event to time series segments having a similar shape, trend, or value. Here, we argue that a bad discretization could hide the synchronizations. If a segment involved in a synchronization is associated with the wrong discrete event due to noise or an inadequate discretization method, the synchronization will not be detected in the discretized sequences. Hence, the challenge is to discretize the time series without hiding the synchronizations.

Example: Hidden synchronization after a bad discretization

Let's examine the time series represented in Figure 5.2, where a and b are two variables. We observe that when the value of a is around 0.5, the value of b reaches a value around 1 shortly thereafter. This is the result of synchronization between the components corresponding to each variable behind the observed time series. If the discretization associates the yellow segments with the same event in each variable,

let say o for variable a and i for variable b, it will be possible to find the synchronization in the resulting event sequences because when i occurs in one variable, then o is systematically occurring.

However, even though this time series is simple, it contains noise. For instance, the last segment of a may be associated with a different event than the two other yellow segments, let say h, because its mean value is lower than the others. In this case, i will not only occur when o occurs, but also when h occurs. As a result, the rule is not verified, and the synchronization is hidden.

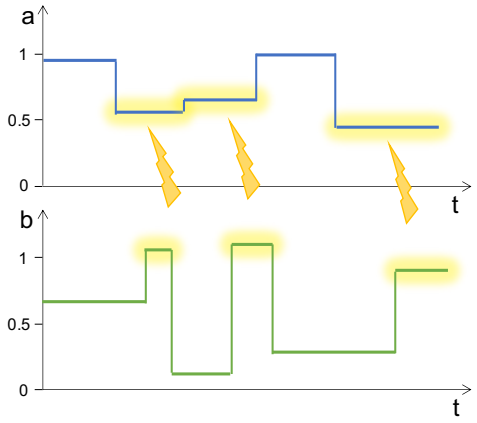


Figure 5.2 – Two time series with hidden synchronization (indicated in yellow).

Starting from a segmented multivariate time series, we propose to perform the discretization and the synchronization search jointly, by constraining the segments labeling (i.e., the discretization) inside each variable with the co-occurrences across the variables. Discretizing a time series consists in associating similar segments with a same discrete event. We will address this problem using a clustering algorithm. Meanwhile, searching for synchronized events in two sequences is searching for events that occur simultaneously. Our method combines both thanks to a new data structure and an algorithm to identify synchronized events in time series. At present, this method can only be applied to multivariate time series with two variables, i.e., of dimension 2, and with the hypothesis that if there are several pairs of synchronizing events, one of the variables has the inputs and the other has the outputs.

5.2 Synchronization discovery in time series

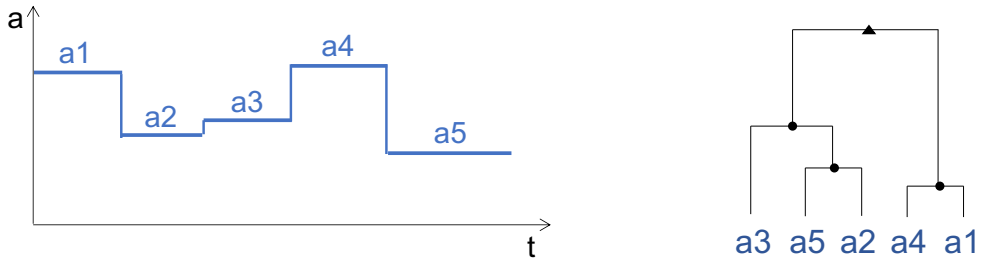
Before describing our synchronization-preserving discretization algorithm, we describe individually the time series discretization strategy to obtain an event sequence, and the synchronization discovery task given two event sequences. Then, we present how to combine these two tasks with the introduction of a new data structure dedicated to the discovery of synchronization across time series.

5.2.1 Discretization via agglomerative hierarchical clustering

We first present a discretization method based on the clustering of the segments of a univariate time series (or a single variable of a multivariate time series), where all the segments of a same cluster will be associated with a same event. It is important to note that, by definition, an event is instantaneous, while a segment can cover multiple timestamps. Therefore, technically, the start of the segment corresponds to the event's occurrence, and no new event happens during the segment period.

Agglomerative hierarchical clustering (Hastie, Tibshirani, and Friedman 2009) refers to the clustering of objects based on a similarity measure, that starts by assigning each object to an individual cluster and then iteratively merges the clusters based on their similarity. In the context of time series discretization, the objects can be the datapoints or the segments of a segmented time series, and each cluster is associated with a symbol or event. Here, we will consider segments [1] ($a1$, $a2$ ... in Figure 5.3a). Starting from a configuration where each segment constitutes an individual cluster (and therefore a distinct event), the clusters are progressively merged by pair according to their similarity. At the upper level, all the segments are in the same cluster. To determine which individual clusters to merge, any relevant object similarity measure can be utilized. When there are more than one object in the clusters, we must use a *linkage measure* that evaluates the similarity between the clusters. Examples of linkage measures include average linkage, which corresponds to the average distance between the objects in the first cluster and the objects in the second cluster, and complete linkage, which corresponds to the maximal pairwise distance between objects of the first cluster and of the second cluster. The hierarchical agglomeration of clusters can be represented using a dendrogram (Figure 5.3b), where the individual clus-

1. Please note that the segmentation problem is not addressed here, a review of the rich literature on the subject has been done by Mörchen (2006).



(a) Time series. (b) Dendrogram.

Figure 5.3 – A segmented time series and the agglomerative hierarchical clustering dendrogram. In the dendrogram, the root is indicated by the triangle, the internal nodes by the circles, and the leaves correspond to the segments of the time series (a1, a2...).

ters are the leaves, the (non-individual) clusters are internal nodes, and the cluster grouping all the objects is the root. Many combinations of clusters can be obtained from the agglomerative hierarchical clustering. The final clusters can be obtained by cutting the dendrogram at a fixed height or with a pre-fixed number of clusters. Back to the context of time series discretization, all the segments in a same cluster will be labeled with the same symbol or event.

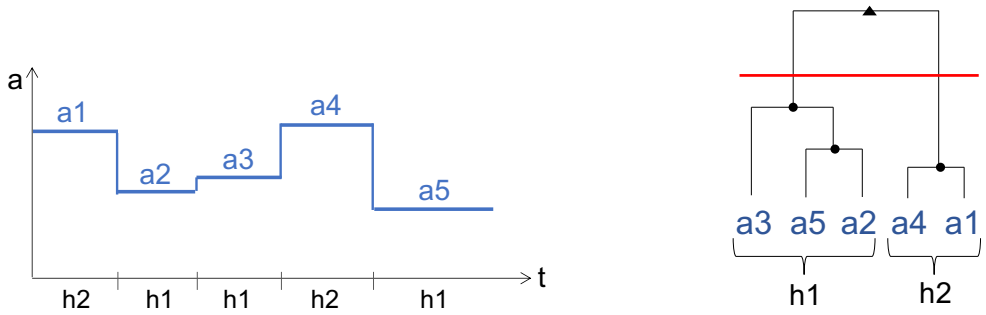
Example: Time series discretization using agglomerative hierarchical clustering

Figure 5.4b shows the result of the discretization case presented in Figure 5.3b. The dendrogram was cut following the red line, which create two clusters. If we associate the cluster composed of segments $a3$, $a5$ and $a2$ with event $h1$, and the cluster composed of segments $a4$ and $a1$ with event $h2$, we can discretize the time series as illustrated in Figure 5.4a.

5.2.2 Bivariate synchronization discovery

We now present a method to mine synchronized pairs of events given two event sequences. In this section, we will be considering discrete event sequences, the approach will be applied to time series later.

In TIOA formalism, the input and the output of a pair typically have the same name, associated with a *?* and a *!*, respectively. Yet, in real data, they may have a totally different name. It is then necessary to search which events form a pair of synchro-



(a) Time series.

(b) Dendrogram.

Figure 5.4 – Result of the agglomerative hierarchical clustering dendrogram. According to the red cutline, there are two resulting clusters ($h1$ and $h2$). The resulting event sequence is displayed below the time series.

nizing input and output. Given two discrete event sequences, identifying the events corresponding to pairs of input and output consists of finding the events from each event sequence that co-occur.

Let's first define what is meant by event co-occurrence. According to the formalism of TIOAs, the synchronization is instantaneous, there is no delay between the output and the input, therefore the input and output events should occur at the same time. In real settings, there can be a delay between the output event issue, the reception, and the input event reaction. Consequently, we accept a (positive) delay between the output event and the input event occurrences, as long as there are no intermediate events occurring between them. We still do not permit input to happen prior to output. Consequently, we need to distinguish two notions of event co-occurrence based on which event may happen with a delay.

Definition: Events co-occurrence

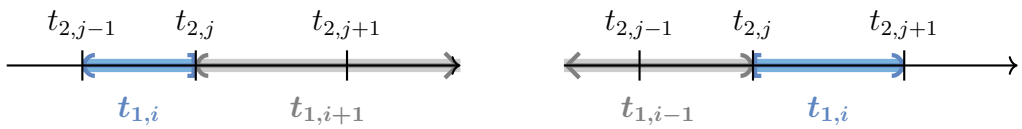
Given two sequences composed of tuple of events e and their timestamp of occurrence t:

$$s_1 = \langle (e_{1,i}, t_{1,i}) | \forall i \in [1, m] \rangle$$

$$s_2 = \langle (e_{2,j}, t_{2,j}) | \forall j \in [1, n] \rangle$$

An event $e_{1,i}$ *leading co-occurs* with an event $e_{2,j}$ if: $t_{2,j-1} < t_{1,i} \leq t_{2,j} \wedge t_{1,i+1} > t_{2,j}$
An event $e_{1,i}$ *trailing co-occurs* with an event $e_{2,j}$ if: $t_{2,j} \leq t_{1,i} < t_{2,j+1} \wedge t_{1,i-1} < t_{2,j}$

Figure 5.5 illustrates the definitions of leading and trailing co-occurrence. An event



(a) Leading co-occurs. (b) Trailing co-occurs.

Figure 5.5 – Intervals of possible values if $e_{1,i}$ leading or trailing co-occurs with $e_{2,j}$.

leading co-occurring with another happens at the same time or precedes it (without intermediate event), while an event trailing co-occurring with another happens at the same time or after (without intermediate event). During a synchronization, the input event trailing co-occurs with the output event and the output event leading co-occurs with the input event. By not allowing intermediate events between the occurrence of two co-occurring events, an event can leading co-occurs with at most one event and can trailing co-occurs with at most one event.

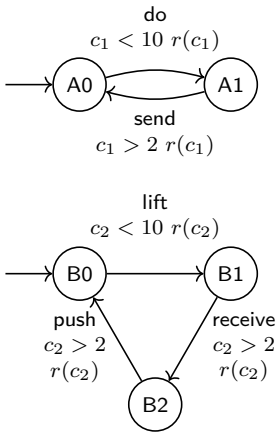
Example: Timed I/O Automata synchronization with different names

Let's consider the automata displayed in Figure 5.6a corresponding to two variables A and B. Let's assume that the events "send" and "receive" correspond to an output and an input, respectively. As a consequence, the transition from $B1$ to $B2$ (labeled with input "receive") is possible only if the transition from $A1$ to $A0$ (labeled with output "send") is triggered in the other automaton. We now consider that we don't know that "send" and "receive" form a pair of synchronizing events.

Figure 5.6b presents two event sequences (one for each automaton or variable). At 10:00:00, the events "do" (for automaton A) and "lift" (for automaton B) have occurred, followed by "send" and "receive" 5 seconds later, then "do" for automaton A without new event for B, etc. From such sequences, it is possible to identify which events correspond to pairs of input and output by searching the events that only occurs when a specific event in the other variable also occurs. Here, it is the case for the event "receive" that only occurs if "send" is the last event occurring in the other variable.

It can be noted that at 10:00:45, we observe the event "send" without observing "receive" in the other variable. The reason is that "send" is an output while "receive" is an input. A transition labeled by an output may be triggered at any time, while a transition labeled by an input may be triggered only if a transition labeled by the corresponding output is triggered. Intuitively, it is possible to send something

without the recipient being ready to receive it. On the contrary, it is impossible to receive something that was never sent. In TIOAs, "the recipient [not] being ready to receive it" translates as "the automaton having the input is currently not in a state with a transition labeled with the input".



(a) Timed Automata (A on the top, B below).

timestamp	A	B
10:00:00	do	lift
10:00:05	send	receive
10:00:14	do	-
10:00:16	-	push
10:00:19	-	lift
10:00:23	send	-
10:00:24	-	receive
10:00:26	do	push
10:00:30	-	lift
10:00:35	send	receive
10:00:41	-	push
10:00:43	do	-
10:00:45	send	-
10:00:50	-	lift
10:00:52	do	-
10:00:55	send	receive

(b) Event sequences.

	10:00:05 send	10:00:23 send	10:00:35 send	10:00:45 send	10:00:55 send
10:00:05 receive	1	0	0	0	0
10:00:24 receive	0	1	0	0	0
10:00:35 receive	0	0	1	0	0
10:00:55 receive	0	0	0	0	1

(c) Co-occurrence matrix.

Figure 5.6 – Bivariate synchronization discovery in event sequences.

We propose an alternative method to classical pattern mining algorithms for identifying co-occurring pairs of events in event sequences. While it may look less efficient at this point, it will be useful when discretization and synchronization search will be combined in the next section. To identify the co-occurring events, we propose to create a *co-occurrence matrix* for each candidate pair of input and output events (i.e., two matrices for each possible pair of events). In this matrix, the rows correspond to the occurrences of the candidate input event, and the columns to the

occurrences of the candidate output event. The matrix cells are filled with a "1" if the row event trailing co-occurred with the column event, and a "0" otherwise. If all the rows in the matrix contain a "1", the events can be considered as a pair of synchronizing events, with the event from the rows being the input, and the event from the columns the output.

Example: Co-occurence matrix

We still consider the automata displayed in Figure 5.6a, the output "send" and the input "receive", and the execution sequences shown in Figure 5.6b. The co-occurence matrix for candidate output event "send" and input event "receive" is displayed in Figure 5.6. Each time "receive" occurred, it trailing co-occurred with "send", hence, there is a "1" in every row. Sometimes there was a delay, e.g. between "10:00:23 send" and "10:00:24 receive", but no intermediate event. "10:00:45 send" is not considered as leading co-occurring with "10:00:55 receive" because there were intermediate events. In the context of TIOAs, it means that automaton B was not in a state with an outgoing transition labeled by the input and another transition was needed to reach such state.

5.2.3 Combining discretization and synchronization discovery

Given two event sequences, an input event is an event that always trailing co-occurs with an event of the other variable, the output event. Given a multivariate segmented time series whose segments were associated to events according to their similarity, this entails a particular alignment for the segments associated to an input or an output event. We recall that when a segment is associated to an event, the start of the segment corresponds to the event's occurrence, and no new event happens during the segment period. Contrarily to events, segments have a duration, we can use the Allen's interval relations (Allen 1983), presented in Figure 3.7, to characterize this particular alignment. The segments $s \in S_i$ associated to an input should respect one of the following relation with one of the segments $s' \in S_o$ associated with the corresponding output, without intermediate segment:

— s is-overlapped-by s';

— s finishes s';

— s during s';

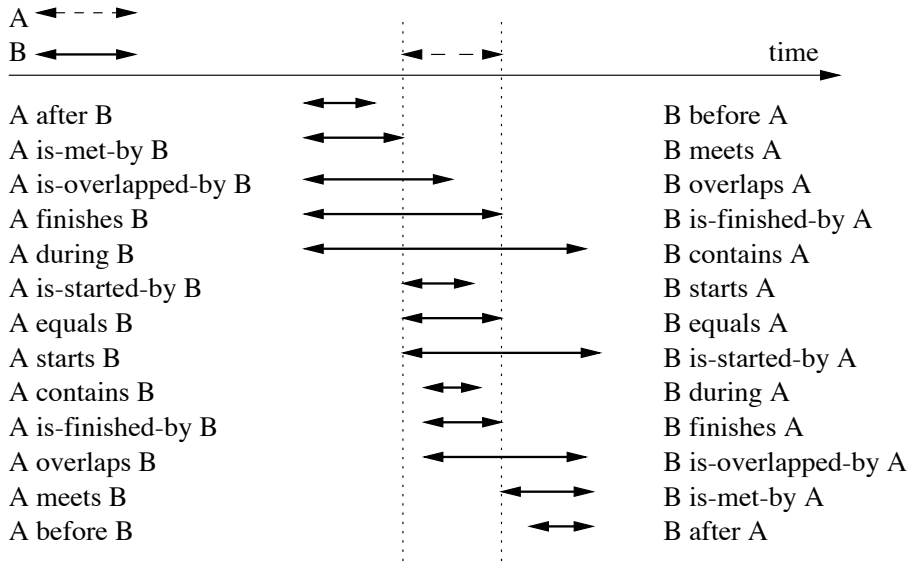


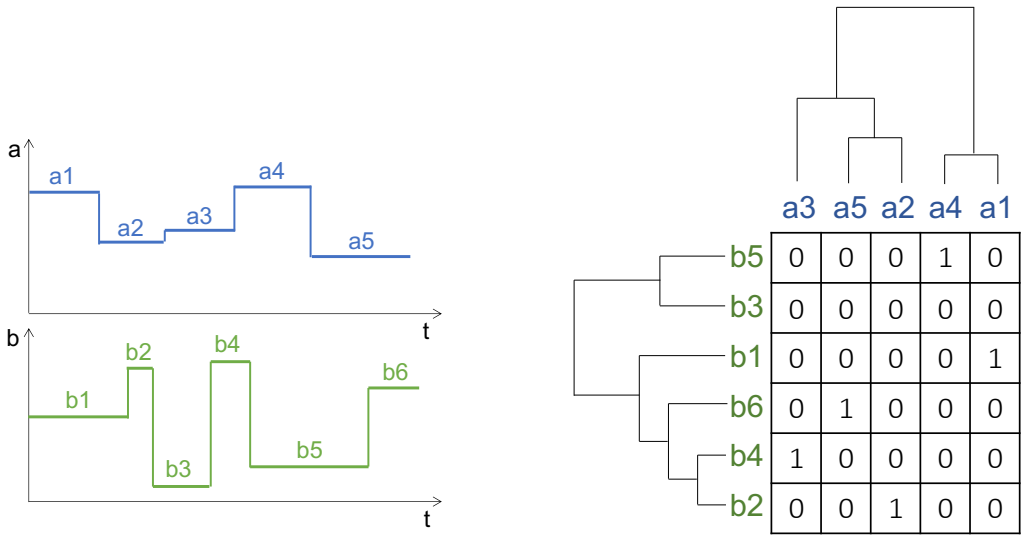
Figure 3.7 – Allen temporal relations (from Höppner (2001)). (repeated from page 93)

— s is-started-by s' (equivalent to s starts s');

— s equals s'.

To associate segments of a segmented multivariate time series to input and output events, we need to identify subsets of similar segments for which this specific alignment is respected. The segments' similarity within a variable can be obtained via agglomerative hierarchical clustering, which will give the possible clusters in a dendrogram. To investigate the segment alignment across variables, we can associate all the segments to a distinct event and construct a global co-occurrence matrix where the rows correspond to the segments of one variable, which will potentially contain input events, and the columns to the segments of the other variable, which will potentially contain corresponding output events. To determine whether to place a variable in rows or columns, we can use background knowledge or test both options. Note that this matrix will initially typically look like a diagonal matrix if the segments are put in order of occurrence, the first segment of the first variable starting at the same time as the first segment of the second variable and so on.

We propose a data structure that combines both similarity and co-occurrence information (Figure 5.7b). In this figure, $a1$, $a2$ etc. are the segments of the first variable and $b1$, $b2$ etc. are the segments of the second variable (Figure 5.7a). The clustering



(a) Time series. (b) Data structure.

Figure 5.7 – Data structure for synchronization discovery in time series.

dendrograms are placed alongside the global co-occurrence matrix, and the leaves, which correspond to segments, are mapped to the corresponding row or column. This involves reordering matrix rows and columns as the order of the leaves is constrained by segment similarity in the dendrograms. This data structure will allow us to look for the subsets of similar segments that respect the temporal alignment condition by examining subparts of the matrix.

It is important to note that despite the visual similarity with biclustering matrix representations, the problem and the encoded information are different. Biclustering is the identification of subsets of rows which exhibit similar behavior across subsets of columns, or vice versa, without additional information than the matrix content. The dendrograms that are often represented alongside the matrix represent the similarity between the matrix rows and column. Here, the information contained in the matrix and in the dendrograms is different and the identification of subsets of rows and columns is performed from the matrix content constrained by the dendrograms.

5.2.4 Identifying synchronizations in the data structure

In the data structure previously defined, we can identify pairs of input and output by finding the submatrices with one “1” per row and whose rows and columns form a cluster in the dendrogram. For instance, in Figure 5.8, the submatrix in yellow has

one "1" per row, meaning that the row segments always respected one of the Allen's relation selected in the previous section with one of the column segments, and the set of segments $\{b2, b4, b6\}$ and $\{a2, a5, a3\}$ both form a cluster, meaning that they are similar. Hence, if we associate these segments with an input i and an output o, respectively, i will always trailing co-occurs with o while the segments associated to a same event will be similar.

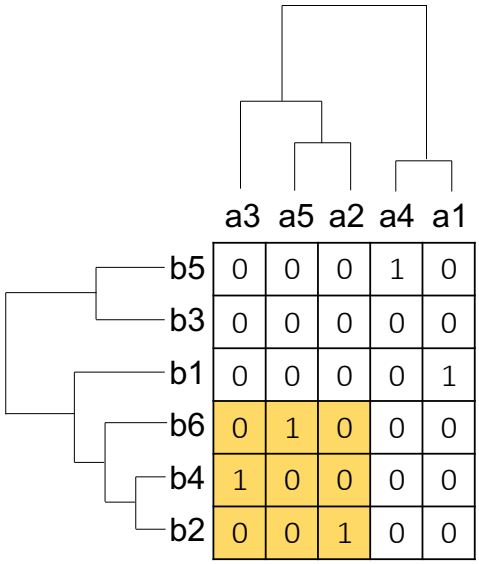


Figure 5.8 – Maximal p-diagonal c-submatrix (in yellow).

We now formalize the conditions a submatrix must satisfy for its row and column segments to be associated with an input and an output. First of all, these submatrices are characterized by the configuration in the dendrograms.

Definition: Cluster-submatrix

A *cluster-submatrix* (c-submatrix) is a submatrix whose rows and columns form a cluster in the dendrograms.

There exist many c-submatrices, from 1×1 submatrices to the entire initial matrix, these matrices being nested within each other, i.e. contained in a larger c-submatrix.

Example: c-submatrices

In Figure 5.8, the submatrix formed by rows $\{b2, b4\}$ and by columns $\{a1, a4\}$ is a c-submatrix, as well as the submatrix formed by rows $\{b2, b4, b6\}$ and by columns $\{a1, a4\}$. The submatrix formed by rows $\{b2, b4\}$ and by columns $\{a1, a4, a2\}$ is not a

c-submatrix, because $\{a1, a4, a2\}$ is not a cluster.

These submatrices are also characterized by their cell content.

Definition: p-diagonal c-submatrix

A *permuted-diagonal c-submatrix* (p-diagonal c-submatrix) is a c-submatrix with one "1" per row and "0" in the other cells. If the c-submatrix is squared, the identity matrix can be retrieved by permutation of the rows and columns.

Example: p-diagonal c-submatrix

In Figure 5.8, the submatrix in yellow formed by rows $\{b2, b4, b6\}$ and by columns $\{a2, a5, a3\}$ is a p-diagonal c-submatrix, there is exactly one "1" per row and its rows and columns form clusters. If rows $b4$ and $b6$ are permuted, its diagonal is filled with "1"s while the other cells are filled with "0"s: it is an identity matrix.

Finally, we want the p-diagonal c-submatrices to be as large as possible and not to be 1×1 submatrices. However, it is important to ensure that they do not contain all the rows or columns, because otherwise, for any time series with the same segmentation applied to both variables, all segments would be associated with a unique input or output.

Definition: Maximal p-diagonal c-submatrix

A *maximal p-diagonal c-submatrix* is the largest p-diagonal c-submatrix in a nested set of c-submatrices that have $1 < r < m$ rows and $1 < c < n$ columns with $m \times n$ the size of the full matrix.

Example: Maximal p-diagonal c-submatrix

In Figure 5.8, the submatrix in yellow formed by rows $\{b2, b4, b6\}$ and columns $\{a2, a5, a3\}$ is a maximal p-diagonal c-submatrix. None of the submatrices in which it is nested, namely those formed by rows $\{b2, b4, b6, b1\}$ and columns $\{a2, a5, a3\}$, rows $\{b2, b4, b6, b1, b3, b5\}$ and columns $\{a2, a5, a3\}$, rows $\{b2, b4, b6, b1\}$ and columns $\{a1, a4, a2, a5, a3\}$, is a p-diagonal c-submatrix. It is nested in the c-submatrix formed by rows $\{b2, b4, b6\}$ and columns $\{a1, a4, a2, a5, a3\}$ which also have one "1" per row, but this matrix is formed by all the columns of the full matrix, so the condition is not respected.

Note that it is possible to have multiple (not nested) maximal p-diagonal c-submatrices

in the global co-occurrence matrix. These will constitute distinct pairs of input and output events.

5.3 Algorithm

To automatically identify the maximal p-diagonal c-submatrices, we propose an enumeration algorithm with pruning strategies that aim to reduce the enumeration time.

5.3.1 Enumeration problem

The c-submatrices nested within a p-diagonal c-submatrices are not necessarily p-diagonal. For instance, the c-submatrix formed by $\{b2, b4\}$ and $\{a2, a5\}$ in Figure 5.8 is nested in the maximal p-diagonal c-submatrix, but is not p-diagonal. There is no reason to expect that the segments co-occurring with the most similar segments of the input cluster are also the most similar within the output cluster. Therefore, the immediate idea of starting from the smallest p-diagonal c-submatrices and enlarging them until they are maximal is not possible. We can still exploit the property that the c-submatrices are nested due to the hierarchical clustering at their origin. Since we want the maximal p-diagonal c-submatrices, the enumeration of the c-submatrices starts from the full matrix. Then, the exploration of the nested c-submatrices is realized in a lexicographic order (row clusters, column clusters), the row and column clusters being ordered from the root to the leaves. In other words, the enumeration of the submatrices is realized by following the dendrograms, starting from the root, and exploring the column dendrogram for each node of the row dendrogram.

To avoid enumerating all c-submatrices, we can use pruning strategies, i.e. stop exploring a branch if some criterion is true. Two criteria allow pruning a branch:

(1) Fewer columns than row in the current submatrix ($r < c$): It will be impossible to get one "1" per row because there is at most one "1" per column, hence we can stop exploring this branch of the column dendrogram, as it will only lead to submatrices with a decreasing number of columns;

(2) A p-diagonal c-submatrix was already found for the rows of the current submatrix: There can be only one "1" per row, hence we can stop exploring this branch of the row dendrogram.

Algorithms 12, 13, and 14 describe the enumeration with the pruning.

Algorithm 12 Maximal p-diagonal c-submatrices identification

Require: A co-occurence matrice C, two dendrograms roots n_r and n_c
Return: The maximal p-diagonal c-submatrices contained in C
1: $S = \emptyset$
2: S = `explore_row_submatrices`(C, n_r, n_c, S)
3: **return** S

Algorithm 13 `explore_row_submatrices`

Require: A co-occurence matrice C, two dendrograms nodes n_r and n_c, a set of submatrices S
Return: The maximal p-diagonal c-submatrices previously found and the one contained in the submatrix formed by n_r and n_c
1: **if** n_r or n_c are leaves **then return** S
2: S = `explore_col_submatrices`(C, n_r, n_c, S)
3: **if** n_r form a submatrix $s \in S$ **then return** S ▷ pruning 2
4: $n_{r,left}$, $n_{r,right}$ = get_children(n_r)
5: S = `explore_row_submatrices`(C, $n_{r,left}$, n_c, S)
6: S = `explore_row_submatrices`(C, $n_{r,right}$, n_c, S)
7: **return** S

The algorithm starts from the root of each dendrogram, then explores the submatrices via a depth-first search (DFS) in the row dendrogram, with a nested depth-first search (DFS) in the column dendrogram. For a better comprehension, let's focus on the DFS in the row dendrogram. When the enumeration reaches a node of the row dendrogram n_c (entering in Algorithm 13), the column dendrogram is explored from the root (line 2 in Algorithm 13, which corresponds to Algorithm 14) to evaluate the submatrices formed by rows (leaves) of n_c and the different combination of columns. If no p-diagonal c-submatrix was found, the left children of n_c (a cluster formed by a subset of n_c leaves) becomes the current node (line 5 of Algorithm 13) and we enter back in Algorithm 13. Once $n_{c,left}$ and its children have been explored, the cluster formed by the other subset of n_c leaves, $n_{c,right}$, is explored in the same manner (line 6).

Algorithm 14 `explore_col_submatrices`

Require: A co-occurence matrice C, two dendrograms nodes n_r and n_c, a set of submatrices S
Return: The maximal p-diagonal c-submatrices previously found and the one contained in the submatrix formed by n_r and n_c

1: **if** n_r or n_c are leaves **then return** S
2: $candidate$ = the submatrix formed by n_r and n_c
3: **if** $candidate$ has more rows than columns **then return** S ▷ pruning 1
4: **if** $candidate$ is a p-diagonal c-submatrix **then**
5: $S = S \cup candidate$
6: **return** S
7: **end if**
8: $n_{c,left}$, $n_{c,right}$ = get_children(n_c)
9: S = `explore_col_submatrices`(C, n_r, $n_{c,left}$, S)
10: **if** n_r form a submatrix $s \in S$ **then return** S ▷ pruning 2
11: S = `explore_col_submatrices`(C, n_r, $n_{c,right}$, S)
12: **return** S

5.3.2 Handling noise

Due to the noise, or due to the imperfection of the chosen similarity measure or segmentation method, a segment corresponding to a synchronizing event could be considered closer to another event cluster, as well as the contrary. Without strategy to handle the noise, the algorithm can miss whole p-diagonal c-submatrices.

Ideally, the maximal p-diagonal c-submatrices must be square and with one "1" per row (and therefore per column). We propose five tolerance policies, which we describe below:

(1) No tolerance;

(2) The submatrix can be not square and have more columns than rows, has one "1" per row, but there is a limit on the size ratio;

(3) The submatrix can be not square and have more columns than rows as long as it has and one "1" per row (default);

(4) The submatrix must be square, with a limited allowed percentage of row without "1";

(5) The submatrix can be not square with either more columns or more rows, but with a limit on the size ratio.

The first policy requires that the output systematically co-occurs with the input, which is more strict than required by the TIOA formalism. The second and third policies ensure that the input systematically co-occurs with the output, and some outputs can occur without the corresponding input, as in the TIOA formalism. The amount of output without co-occurring input is limited in the second policy. The fourth and fifth policies allow the occurrence of inputs without output, which is not allowed in the TIOA formalism. These strategies prevent entire p-diagonal c-submatrices from being unrecognized due to misclustered segments in noisy data.

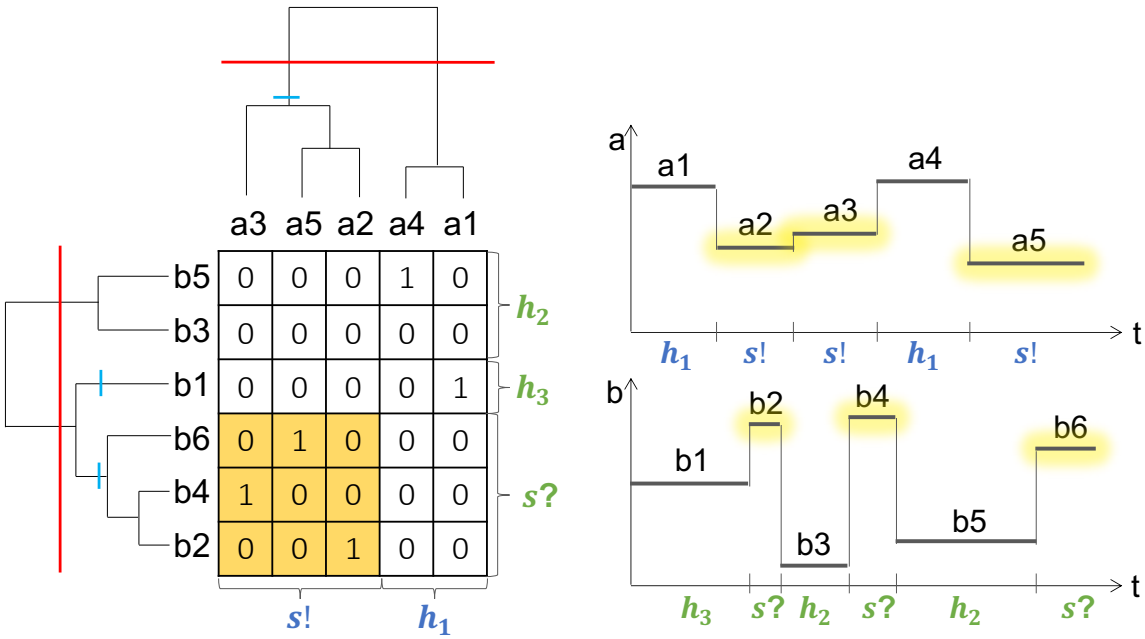
5.3.3 Synchronization-preserving discretization

The previously presented algorithm enables the identification of segments in a bivariate time series that correspond to input or output events. We recall that the goal is to learn interacting TIOAs, which requires event sequences. The remainder of the time series segments still have to be associated to an (internal) event in the discretized version of the time series. These events are not synchronizing, there is no need to consider co-occurrence across variables. We only consider the segment similarity within variables, which is given in the dendrograms. However, the cluster should be formed while respecting the clusters formed by the previously found synchronizing events.

We can use any classical dendrogram cut method, which will return a cut level (red lines in Figure 5.9). Then, the dendrogram branches can be cut according to this cut level, as long as none of their leaves form a maximal p-diagonal c-submatrix in the co-occurrence matrix. Otherwise, the branch is cut above the node corresponding to the maximal p-diagonal c-submatrix (blue lines). This ensures that the discretization does not hide the synchronizations between the variables. Finally, each of these clusters is associated with internal events to obtain the discrete event sequences.

Example: Synchronization-preserving discretization

In Figure 5.9a, a p-diagonal c-submatrix was found (in yellow). Consequently, segments $b6$, $b4$, $b2$ (set S_i) should stay in the same cluster associated to an input event, $s?$, and segments $a3$, $a5$, $a2$ (set S_i) should stay in the same cluster associated to an output event, $s!$. To cluster the remaining segments, we utilize a conventional agglomerative hierarchical clustering method, cutting here at 0.7 of the maximum cluster distance. This method returns the cut levels indicated by the red lines. If we



(a) Co-occurrence matrix.

(b) Time series.

Figure 5.9 – The red lines over the dendrograms correspond to the cut level proposed by the classical clustering algorithm. The blue lines correspond to cuts necessary to preserve the synchronizations, obtained by the synchronization-preserving discretization algorithm. The clusters are associated to an internal event (h_1, h_2, h_3), or to an input $(s?)$ or output $(s!)$.

respect this cut level for the rows, the segments from S_i would be associated to $b1$, and $b3$, $b5$ would form a second cluster. However, since S_i is included in the leaves of the first branch, we don't follow the red cut level for the branch: we need to distinguish synchronizing events and internal events event if they appear in the same cluster. Instead, we cut the branch above the node of the cluster formed by S_i (blue lines), which separates $b1$ from the segments of S_i. The second branch can be cut using the red cut level. This forms three clusters, the previously identified cluster associated to $s?$, and two new clusters associated to internal events, $h2$ and $h3$. The same logic is applied for the columns, forming two clusters: $s!$ and $h1$.

Then, the time series can be discretized by replacing the segments by the events (Figure 5.9b).

5.3.4 Timed I/O Automata learning

Once the time series are discretized using the synchronization-preserving method, any classical passive TA learner can be used. The model of each component is learned individually, and the input and output events will be considered as normal events by the learner. The TIOAs can then be monitored by tools such as UPPAAL that will handle the synchronizations.

5.4 Experiment on synthetic data

To validate the synchronization-preserving discretization method, we conducted an experiment on synthetic data. Starting from simulated time series with synchronized segments, the objective was to identify and preserve the synchronizations in the discretized versions of the time series. To assess the gain in jointly performing the discretization and the synchronization search, we also performed them sequentially.

5.4.1 Data generation

The first step was to generate two event sequences of length $n = 250$, with two pairs of synchronization input and output events ($synchro_1[?/!]$ and $synchro_2[?/!]$) within their events. The first generated sequence, a, contains the outputs $synchro_1!$ and $synchro_2!$, and two internal events $event_{a,1}$ and $event_{a,2}$. Each event had a fixed probability to be chosen (p), and probability to not be replaced (p_r) over the next time step (Table 5.1a). Then, we generated the second variable, b, containing the input

Table 5.1 – Generation parameters.

(a) Variable a.

Event	**Value**	p	p_r
$event_{a,1}$	0	0.40	0.80
$synchro_1!$	1	0.25	0.89
$event_{a,2}$	1.5	0.10	0.90
$synchro_2!$	2	0.25	0.84

(b) Variable b.

Event	**Value**	p	p_r
$event_{b,1}$	0	0.5	0.73
$synchro_2?$	0.25	0	0.80
$event_{b,2}$	1	0.5	0.76
$synchro_1?$	2	0	0.75

events $synchro_1?$ and $synchro_2?$, and two internal events $event_{b,1}$ and $event_{b,2}$, while taking into account the first one. One can observe in Table 5.1b that the probabilities

of occurrence for the input events ($synchro_1?$ and $synchro_2?$) are null, as their occurrence is constrained by the occurrence of the corresponding output. Therefore, event sequence for variable b was generated based on p and p_r, except for the data point where the first variable had an output generated and consequently the corresponding input was automatically selected.

Next, the event sequences were converted into time series, by replacing each event by a fixed value (value column in Table 5.1). For instance, from the first sequence:

$$\langle event_{a,1}, synchro_2!, ..., event_{a,1}, ..., synchro_1!... \rangle$$

We obtain the following numerical data:

$$\langle 0, 2, ..., 0, ..., 1... \rangle$$

Finally, a random normal noise (mean of 0, standard deviation of 0.1) was applied to the time series. We selected a small standard deviation because two events are associated with a close value ($event_{b,1}$ corresponds to 0 and $synchro_2?$ to 0.25) to make them difficult to separate, but we do not want their resulting data points in the time series to have an overlapping range of values since we are not using any noise handling strategy in this first experiment.

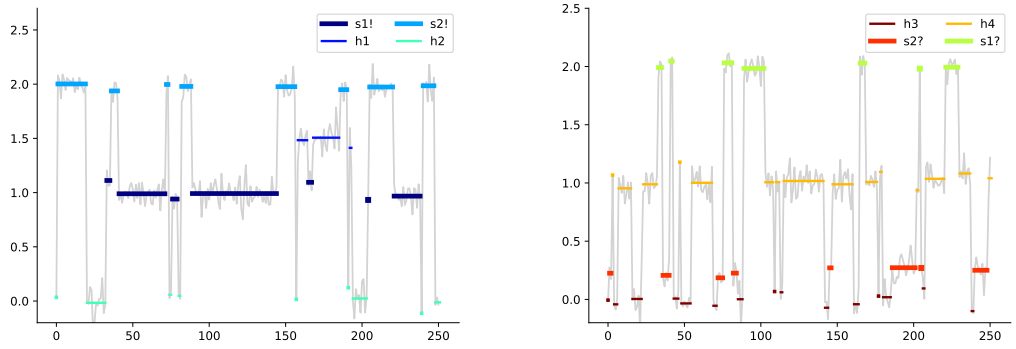
5.4.2 Discretization

To only evaluate the proposed method and have no influence from the time series segmentation method, the segments were directly created from the event changes. For the clustering, we chose the Euclidean distance between the segments mean values as similarity measure, since the data were generated using a mean value per segment. The cluster similarity was evaluated using an average linkage. The dendrograms default cut level was set to 0.6 of the maximal cluster distance. For the synchronization discovery part, the default tolerance policy was applied.

5.4.3 Results

Using the proposed synchronization-preserving discretization approach, i.e., by performing jointly the segment clustering and the synchronization search, we identified two pairs of synchronizing events ($s_1!$ and $s_1?$, and $s_2!$ and $s_2?$), two internal

events in variable a (a_1, and a_2), and two internal events in variable b (b_1, and b_2). Figure 5.10 displays the result of the discretization. The colored lines correspond to the

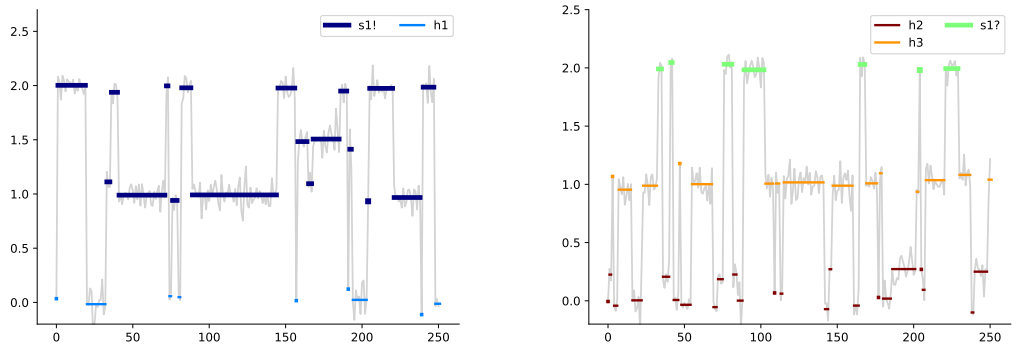


(a) Variable a. (b) Variable b.

Figure 5.10 – Result of the discretization using the synchronization-preserving discretization method. The original time series are in gray. Each color corresponds to an event obtained via the discretization.

mean value of the time series segment and the color indicates the cluster and thus the event.

Figure 5.11 presents the discretization obtained using the same elements of the synchronization-preserving method, but sequentially. Firstly, the time series segments



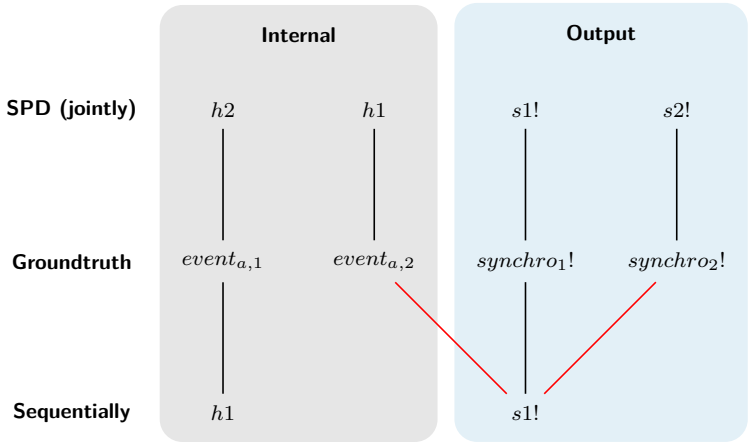
(a) Variable a. (b) Variable b.

Figure 5.11 – Result of the discretization and the synchronization discovery performed sequentially. The original time series are in gray. Each color corresponds to an event obtained via the discretization.

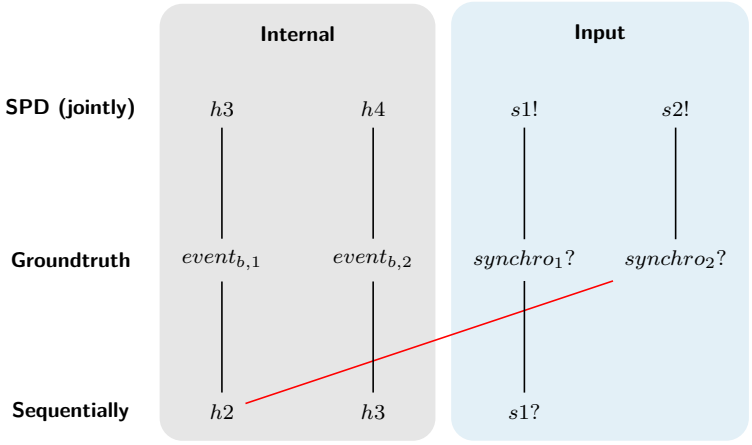
were clustered with the agglomerative hierarchical clustering using the Euclidean

distance on segment mean values, the dendrograms were cut using the same criterion (without specific cut for branches containing inputs or outputs), and finally, synchronizing pairs of input and output were searched in the resulting event sequences. Two events were created for variable a, and three for variable b. A synchronization was discovered, where the output occurs frequently without co-occurring with the input (which is allowed by the default tolerance policy but may not be ideal).

Figure 5.12 presents how the events found by the two methods match the true events. A line indicates that some (or all) of the segments labeled by the event in



(a) Variable a.



(b) Variable b.

Figure 5.12 – Input, output, and internal event matching with the groundtruth (SPD: Synchronization-Preserving Discretization).

the resulting discretized version correspond to the linked groundtruth event. There is a perfect match between the result of the synchronization-preserving discretiza-

tion method and the ground truth. When performing the tasks sequentially, the segments from a corresponding to the two outputs were associated altogether with the segments of an internal event (resulting event: $s1!$). In variable b, the segments corresponding to the input $synchro_1?$ were associated to a single event ($s1?$). A synchronization was found between these two events $s1!$ and $s1?$ because when $s1?$ (the input) occurs, $s1!$ (the output) always leading co-occurs. More precisely, $s1?$ trailing co-occurs with $s1!$ when it corresponds to the true output event $synchro_1!$. While a synchronization was found, it is not as fine as one could hope. The second synchronization was not found at all.

These results demonstrate the relevance of the method proposed in this chapter, since using the same clustering method and the same default dendrogram cutting criterion, all the synchronizations were preserved by our method.

5.5 Conclusion

In this chapter, we introduced the problem of identifying synchronizations in multivariate time series in order to learn interacting Timed I/O Automata (TIOAs), a synchronization being a rule of the following form: if i occurs in one variable, then o is occurring in another variable, where i and o are particular shapes, trends, or values, and correspond to the input and output, respectively. We proposed a discretization method that identifies and preserves potential pairs of input and output events. To the best of our knowledge, our approach is the first to tackle this problem. In this work, we make several assumptions. We consider that the synchronizing events are shared between two variables only. Furthermore, we assume that all the input events are contained exclusively in one variable, and all the output events are exclusively in the second variable. Finally, we assume that the time series segmentation and the segment similarity measures were appropriate. Further work will be devoted to reducing of these assumptions. Additionally, our early experiment highlighted the need to explore different tolerance strategies and to investigate how to properly evaluate the results of discretization and synchronization identification.

CONCLUSION AND PERSPECTIVES

Summary of the contributions

This thesis addressed the problem of learning a behavior model of a system from observational data, in the form of TA. We had identified three main challenges related to this subject. The first one was to learn a TA from event sequences that was representing the learning data without overfitting, in order to also represent the system at the origin of the data. The second challenge was to enable the inference of TAs, which are labeled with discrete events, from numerical time series. The last challenge was to learn interacting TAs with synchronized events from multivariate time series without further information about the synchronizations.

We addressed the first challenge by proposing a new TA learning algorithm, TAG, to passively learn TAs from event sequences. TAG is the first contribution of this thesis. More precisely, it learns a TA subclass called DRTA, which means that the automaton has a single clock reset at each transition, and that there is a single state sequence for a given timed event sequence. The learned DRTA accepts all the input sequences. Additionally, TAG incorporates a parameter, k, that controls the generalization level of the learned model. TAG starts with a model that is an almost exact representation of the input sequences, and then has a state merging strategy based on the short-term possible behaviors (event sequences) that each state offers. The parameter k corresponds to the length of the paths used to assess whether two states should be merged. The larger k is, the more closely the model will match the learning data. TAG also has a transition splitting procedure that distinguishes paths for a same event from the state when the temporal value is determining for the future behavior. Experiments on synthetic data have shown that this new algorithm globally achieves a significantly higher precision in terms of accepted sequences compared to the state-of-the-art algorithms without loss of recall, resulting in a better overall balance. Finally, an experiment on real data has demonstrated that even when the learned model is too complex to be readable by a human, it is possible to query the model to obtain useful information thanks to the existing tools dedicated to TAs.

For the second challenge, we investigated which attributes of discretized event sequences are useful for learning TAs. Initially, we proposed a first contribution where we studied and adapted an existing discretization algorithm, Persist, which aims to create persisting symbols or events, i.e., more likely to repeat themselves than to appear. This criterion seemed interesting to us, since it tends to limit the event changes, and thus leads to transitions with meaningful timed guards. However, we also considered it crucial to have a small dispersion of values in the time series corresponding to a same event. Finally, we also wanted to limit the number of different events to reduce the complexity of the TA learning process and of the learned model. To combine all these criteria, we proposed the use of a genetic algorithm to simultaneously optimize the conflicting objectives. This approach is named MOODES and is the second main contribution of this thesis. Since the genetic algorithm produces an ensemble of discretization solutions that make different trade-offs between the objectives, we also suggested keeping all the solutions and to learn a TA for each of them. Therefore, the behavior model takes the form of an ensemble of TAs, each one modeling the behavior at a different scale and perspective. We applied this discretization technique, along with TAG, to a dataset for detecting anomalies in time series called BATADAL. By using TA ensembles as normal behavior models, we were able to detect and explain the attacks in the time series of the test dataset.

The third challenge was also related to time series, but included the additional task of identifying synchronizing events. As this problem is, to the author's best knowledge, being raised for the first time, we revert to a simpler discretization method, and combine it with a synchronization preserving strategy. In this fourth contribution, we identify input and output synchronizing events in segmented bivariate time series along the discretization. We implement a hierarchical clustering of the segments within the variables, where the cluster corresponds to events, and constrain the cluster formation with a matrix containing the segment co-occurrence information across variables. The resulting events are either classical or internal events, or input and output events, which correspond to similar segments of a variable (input) that always co-occur with similar segments of another variable (output). This algorithm was tested using small synthetic data. This contribution is an ongoing work and has still to be tested on real data.

Perspectives

The various contributions of this thesis offer multiple perspectives.

There are several potential developments for TAG. It currently learns DRTAs, which are TAs with a single clock reset at each transition. Investigating strategies to improve the comprehension of the timing dynamics of the underlying system constitutes an interesting and substantial work, e.g., by resetting the clock only at certain transitions or by identifying additional clocks. A danger of this potential evolution is to obtain a model that is too complex to be well interpreted by a human user. Another possible evolution for the formalism learned by TAG would be to consider the probabilities. At this time, they are purely indicative in order to use a classical and well-studied formalism. Yet, the probabilities could be very useful both during the learning process, and for further uses of the learned model. For instance, real-world observational data often contain noise that can affect the accuracy of the learned models. Using the transition probabilities to filter the outlying ones during the learning process may be of great interest.

The other possible change for TAG is its learning setting. It is a passive algorithm, meaning that it learns a model from timed event sequences only, without interaction with the system. Further work could be devoted to making TAG an active algorithm. In this setting, it would be possible to provide sequences from the learned model (or the learned model directly) to an oracle (possibly by testing the system itself), and this oracle would confirm the correctness or provide a counterexample. In such case, TAG should also be able to process negative sequences, i.e., sequences that should not be accepted by the learned automaton. TAG should also be able to cancel a merge or a split without going back to the initialized automaton. Another research direction related to the training data is to accept expert knowledge in the input sample in addition to the timed event sequences. This could involve expert rules, constraints, or prior (sub-)models to guide the learning algorithm.

Regarding MOODES, the discretization method specifically dedicated to TA learning from time series, the main perspective is to further investigate desirable qualities for the resulting discrete sequences. Although the genetic algorithm has the interesting characteristic of producing multiple solutions, its outcome is not deterministic and the search space to be explored can be too large to ensure good convergence. Further work is needed to identify a more efficient optimization strategy.

The anomaly detection approach based on TA ensemble may be improved in two ways. When an anomaly is raised because there is no transition corresponding to the occurring event, the true anomaly may have actually occurred earlier. To illustrate this, let's consider the TA in Figure 5.13. The event a occurs with a delay of 3, followed

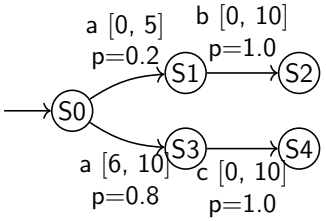


Figure 5.13 – A Timed Automaton.

by c with a delay of 5. This sequence is not accepted by the automaton, an anomaly would be raised while being on S1 when c occurs. However, the true anomaly could be the duration of a, which would make the following of the event sequence correct. Investigating this problem could improve the anomaly explanations. On another note, the transition probabilities learned by TAG are not yet leveraged for the anomaly detection, although they are a rich source of information. Computing the probability of the paths is a promising approach to enhance the anomaly detection performance.

The last contribution of this thesis, related to the synchronizations, is an ongoing work, therefore, the perspectives are numerous. In the first place, experiments on real data need to be conducted, particularly to evaluate the noise handling strategies and the scalability of the approach. Then, the algorithm has several points for improvement. Currently, the direction of the synchronizations, i.e., which variable contains the inputs and which variable contains the outputs, must be known and constant for all the events. The algorithm should be able to identify the synchronization direction for each pair of synchronized events. Furthermore, to date, the algorithm can only handle bivariate time series. The method should be extended to more variables. Finally, the synchronization-preserving discretization algorithm is based on the clustering of similar segments and is totally disconnected from the work on discretization specifically adapted for TA learning. It could be interesting to study how to combine the synchronization-preserving aspect with the identified objectives.

BIBLIOGRAPHY

Abokifa, Ahmed A., Kelsey Haddad, Cynthia S. Lo, and Pratim Biswas (May 2017), « Detection of Cyber Physical Attacks on Water Distribution Systems via Principal Component Analysis and Artificial Neural Networks », en, *in*: *World Environmental and Water Resources Congress 2017*, Sacramento, California: American Society of Civil Engineers, pp. 676–691, ISBN: 978-0-7844-8062-5, DOI: 10.1061/9780784480625.063.

Aghashahi, Mohsen, Raanju Sundararajan, Mohsen Pourahmadi, and M. Katherine Banks (May 2017), « Water Distribution Systems Analysis Symposium–Battle of the Attack Detection Algorithms (BATADAL) », *in*: American Society of Civil Engineers, DOI: 10.1061/9780784480625.062.

Allen, James F. (Nov. 1983), « Maintaining knowledge about temporal intervals », en, *in*: *Communications of the ACM* 26.*11*, pp. 832–843, ISSN: 0001-0782, 1557-7317, DOI: 10.1145/182.358434.

Alur, Rajeev (1999), « Timed Automata », en, *in*: *Computer Aided Verification*, ed. by Nicolas Halbwachs and Doron Peled, vol. 1633, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 8–22, ISBN: 978-3-540-66202-0 978-3-540-48683-1, DOI: 10.1007/3-540-48683-6_3.

Alur, Rajeev, Costas Courcoubetis, and David Dill (May 1993), « Model-Checking in Dense Real-Time », en, *in*: *Information and Computation* 104.*1*, pp. 2–34, ISSN: 08905401, DOI: 10.1006/inco.1993.1024.

Alur, Rajeev and David Dill (Apr. 1994), « A theory of timed automata », en, *in*: *Theoretical Computer Science* 126.*2*, pp. 183–235, ISSN: 03043975, DOI: 10.1016/0304-3975(94)90010-8.

Alur, Rajeev, Limor Fix, and Thomas A. Henzinger (Jan. 1999), « Event-clock automata: a determinizable class of timed automata », en, *in*: *Theoretical Computer Science* 211.*1-2*, pp. 253–273, ISSN: 03043975, DOI: 10.1016/S0304-3975(97)00173-4.

Angluin, Dana (Nov. 1987), « Learning regular sets from queries and counterexamples », en, *in*: *Information and Computation* 75.*2*, pp. 87–106, ISSN: 08905401, DOI: 10.1016/0890-5401(87)90052-6.

Asarin, Eugene, Oded Maler, Amir Pnueli, and Joseph Sifakis (July 1998), « Controller Synthesis for Timed Automata », *in*: *5th IFAC Conference on System Structure and Control 1998 (SSC'98), Nantes, France, 8-10 July* 31.*18*, pp. 447–452, ISSN: 1474-6670, DOI: 10.1016/S1474-6670(17)42032-5.

Audibert, Julien, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga (Aug. 2020), « USAD: UnSupervised Anomaly Detection on Multivariate Time Series », en, *in*: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Virtual Event CA USA: ACM, pp. 3395–3404, ISBN: 978-1-4503-7998-4, DOI: 10.1145/3394486.3403392.

Badouel, Eric, Luca Bernardinello, and Philippe Darondeau (2015), *Petri net synthesis*, eng, Texts in theoretical computer science, an EATCS series, Heidelberg New York Dordrecht London: Springer, ISBN: 978-3-662-47966-7.

Barbot, Benoit, Nicolas Basset, and Alexandre Donze (May 2023), « Wordgen : a Timed word Generation Tool », en, *in*: *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control*, San Antonio TX USA: ACM, pp. 1–7, ISBN: 9798400700330, DOI: 10.1145/3575870.3587116.

Ben Abdallah, Emna, Tony Ribeiro, Morgan Magnin, Olivier Roux, and Katsumi Inoue (Jan. 2017), « Learning Delays in Biological Regulatory Networks from Time Series Data », *in*: *Genomics and Computational Biology* 3.*2*, p. 43, ISSN: 2365-7154, DOI: 10.18547/gcb.2017.vol3.iss2.e43.

Bengtsson, Johan, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi (1996), « UPPAAL — a tool suite for automatic verification of real-time systems », *in*: *Hybrid Systems III*, ed. by G. Goos, J. Hartmanis, J. van Leeuwen, Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, vol. 1066, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 232–243, ISBN: 978-3-540-61155-4 978-3-540-68334-6, DOI: 10.1007/BFb0020949.

Bérard, Beatrice, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux (2005), « Comparison of the Expressiveness of Timed Automata and Time Petri Nets », *in*: *Formal Modeling and Analysis of Timed Systems*, ed. by David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Paul Pettersson, and Wang Yi, vol. 3829, Series Title: Lecture Notes in Computer Science,

Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 211–225, ISBN: 978-3-540-30946-8 978-3-540-31616-9, DOI: 10.1007/11603009_17.

Biermann, Alan W. and Jerome Feldman (June 1972), « On the Synthesis of Finite-State Machines from Samples of Their Behavior », *in*: *IEEE Transactions on Computers* C-21.*6*, pp. 592–597, ISSN: 0018-9340, DOI: 10.1109/TC.1972.5009015.

Bowden, F.D.J (May 2000), « A brief survey and synthesis of the roles of time in petri nets », en, *in*: *Mathematical and Computer Modelling* 31.*10-12*, pp. 55–68, ISSN: 08957177, DOI: 10.1016/S0895-7177(00)00072-8.

Bozga, Marius, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine (1998), « Kronos: A model-checking tool for real-time systems: Tool-presentation for FTRTFT '98 », en, *in*: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, ed. by Gerhard Goos, Juris Hartmanis, Jan Van Leeuwen, Anders P. Ravn, and Hans Rischel, vol. 1486, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 298–302, ISBN: 978-3-540-65003-4 978-3-540-49792-9, DOI: 10.1007/BFb0055357.

Brentan, Bruno Melo, Enrique Campbell, Gustavo Meirelles, Daniel Manzi, David Ayala-Cabrera, Manuel Herrera, Idel Montalvo, Joaquín Izquierdo, and Edevar Luvizotto Jr. (May 2017), « On-Line Cyber Attack Detection in Water Networks through State Forecasting and Control by Pattern Recognition », *in*: American Society of Civil Engineers, DOI: 10.1061/9780784480625.062.

Bulychev, Peter, Alexandre David, Kim Gulstrand Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang (July 2012), « UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata », en, *in*: *Electronic Proceedings in Theoretical Computer Science* 85, pp. 1–16, ISSN: 2075-2180, DOI: 10.4204/EPTCS.85.1.

Burch, Jerry R, Edmund Clarke, Kenneth L. McMillan, David Dill, and L.J. Hwang (June 1992), « Symbolic model checking: 1020 States and beyond », en, *in*: *Information and Computation* 98.*2*, pp. 142–170, ISSN: 08905401, DOI: 10.1016/0890-5401(92)90017-A.

Caldwell, Ben, Rachel Cardell-Oliver, and Tim French (2016), « Learning Time Delay Mealy Machines From Programmable Logic Controllers », *in*: *IEEE Transactions on Automation Science and Engineering* 13.*2*, pp. 1155–1164, DOI: 10.1109/TASE.2015.2496242.

Canadas, Nuno, José Machado, Filomena Soares, Carlos Barros, and Leonilde Varela (Oct. 2018), « Simulation of cyber physical systems behaviour using timed plant models », en, *in*: *Mechatronics* 54, pp. 175–185, ISSN: 09574158, DOI: 10.1016/j.mechatronics.2017.10.009.

Canadian Radio-television and Telecommunications Commission (2015), *Television program logs*.

Carrasco, Rafael C. and Jose Oncina (1994), « Learning stochastic regular grammars by means of a state merging method », *in*: *Grammatical Inference and Applications*, ed. by Jaime G. Carbonell, Jörg Siekmann, G. Goos, J. Hartmanis, J. Leeuwen, Rafael C. Carrasco, and Jose Oncina, vol. 862, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 139–152, ISBN: 978-3-540-58473-5 978-3-540-48985-6, DOI: 10.1007/3-540-58473-0_144.

Cassandras, Christos G. and Stéphane Lafortune, eds. (2008), *Introduction to Discrete Event Systems*, en, Boston, MA: Springer US, ISBN: 978-0-387-33332-8, DOI: 10.1007/978-0-387-68612-7.

Chandy, Sarin E., Amin Rasekh, Zachary A. Barker, Bruce Campbell, and M. Ehsan Shafiee (May 2017), « Detection of Cyber-Attacks to Water Systems through Machine-Learning-Based Anomaly Detection in SCADA Data », en, *in*: *World Environmental and Water Resources Congress 2017*, Sacramento, California: American Society of Civil Engineers, pp. 611–616, ISBN: 978-0-7844-8062-5, DOI: 10.1061/9780784480625.057.

Cornanguer, Lénaïg (May 2021), « Passive learning of Timed Automata from logs (Student Abstract) », *in*: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.*18*, Section: AAAI Student Abstract and Poster Program, pp. 15773–15774.

Cornanguer, Lénaïg, Christine Largouët, Laurence Rozé, and Alexandre Termier (June 2022), « TAG: Learning Timed Automata from Logs », *in*: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.*4*, pp. 3949–3958, ISSN: 2374-3468, 2159-5399, DOI: 10.1609/aaai.v36i4.20311.

— (Feb. 2023), « Persistence-Based Discretization for Learning Discrete Event Systems from Time Series », *in*: *MLmDS 2023 - AAAI Workshop When Machine Learning meets Dynamical Systems: Theory and Applications*, Washington (DC), United States, pp. 1–6.

Das, Gautam, King-Ip Lin, Heikki Mannila, Gopal Renganathan, and Padhraic Smyth (July 1998), « Rule Discovery From Time Series », *in*: *KDD Proc* 98.

David, Alexandre, Kim G. Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen (Aug. 2015), « Uppaal SMC tutorial », en, *in*: *International Journal on Software Tools for Technology Transfer* 17.*4*, pp. 397–415, ISSN: 1433-2779, 1433-2787, DOI: 10.1007/s10009-014-0361-y.

De la Higuera, Colin (2010), *Grammatical inference: learning automata and grammars*, eng, Cambridge: Cambridge University Press, ISBN: 978-0-521-76316-5.

Deb, Kalyanmoy and Ram Bhushan Agrawal (1995), « Simulated binary crossover for continuous search space », *in*: *Complex systems* 9.*2*, Publisher: Citeseer, pp. 115–148.

Deb, Kalyanmoy and Himanshu Jain (Aug. 2014), « An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints », *in*: *IEEE Transactions on Evolutionary Computation* 18.*4*, pp. 577–601, ISSN: 1089-778X, 1089-778X, 1941-0026, DOI: 10.1109/TEVC.2013.2281535.

Dima, Catalin (2001), « Real-time automata », *in*: *Journal of Automata, Languages and Combinatorics* 6.*1*, pp. 3–24.

Elsworth, Steven and Stefan Güttel (July 2020), « ABBA: adaptive Brownian bridge-based symbolic aggregation of time series », en, *in*: *Data Mining and Knowledge Discovery* 34.*4*, pp. 1175–1200, ISSN: 1384-5810, 1573-756X, DOI: 10.1007/s10618-020-00689-6.

Furia, Carlo A., Dino Mandrioli, Angelo Morzenti, and Matteo Rossi (Feb. 2010), « Modeling time in computing: A taxonomy and a comparative survey », en, *in*: *ACM Computing Surveys* 42.*2*, pp. 1–59, ISSN: 0360-0300, 1557-7341, DOI: 10.1145/1667062.1667063.

Gan, Yahui, Xianzhong Dai, and Dongwei Li (July 2013), « Off-Line Programming Techniques for Multirobot Cooperation System », en, *in*: *International Journal of Advanced Robotic Systems* 10.*7*, p. 282, ISSN: 1729-8814, 1729-8814, DOI: 10.5772/56506.

Giacomoni, Marcio, Nikolaos Gatsis, and Ahmad Taha (May 2017), « Identification of Cyber Attacks on Water Distribution Systems by Unveiling Low-Dimensionality in the Sensory Data », *in*: American Society of Civil Engineers, DOI: 10.1061/9780784480625.062.

Gjorgiev, Laze and Sonja Gievska (2020), « Time Series Anomaly Detection with Variational Autoencoder Using Mahalanobis Distance », *in*: *ICT Innovations 2020. Ma-*

chine Learning and Applications, ed. by Vesna Dimitrova and Ivica Dimitrovski, Cham: Springer International Publishing, pp. 42–55, ISBN: 978-3-030-62098-1.

Goldberg, David E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, New York: Addison-Wesley.

Grinchtein, Olga (2008), « Learning of Timed Systems », PhD Thesis, Uppsala University, Sweden.

Gupta, Manish, Jing Gao, Charu C. Aggarwal, and Jiawei Han (2014), *Outlier detection for temporal data*, eng, OCLC: 877885434, Cham, Switzerland: Springer, ISBN: 978-1-62705-376-1.

Gupta, Shalabh and Asok Ray (Jan. 2007), « Symbolic dynamic filtering for data-driven pattern recognition », *in*: *Pattern Recognition: Theory and Application*, Erwin A. Zoeller, pp. 17–71, ISBN: 978-1-60021-717-3.

Hastie, Trevor, Robert Tibshirani, and J. H. Friedman (2009), « Hierarchical clustering », *in*: *The elements of statistical learning: data mining, inference, and prediction*, 2nd ed., Springer series in statistics, New York, NY: Springer, pp. 520–8, ISBN: 978-0-387-84857-0.

Henry, Léo, Thierry Jéron, and Nicolas Markey (2020), « Active Learning of Timed Automata with Unobservable Resets », en, *in*: *Formal Modeling and Analysis of Timed Systems*, ed. by Nathalie Bertrand and Nils Jansen, vol. 12288, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 144–160, ISBN: 978-3-030-57627-1 978-3-030-57628-8, DOI: 10.1007/978-3-030-57628-8_9.

Höppner, Frank (Jan. 2001), « Learning temporal rules from state sequences », *in*.

Housh, Mashor and Ziv Ohar (Aug. 2018), « Model-based approach for cyber-physical attack detection in water distribution systems », en, *in*: *Water Research* 139, pp. 132–143, ISSN: 00431354, DOI: 10.1016/j.watres.2018.03.039.

Hwang, Ching-Lai and Kwangsun Yoon (1981), *Multiple Attribute Decision Making*, ed. by M. Beckmann and H. P. Künzi, vol. 186, Lecture Notes in Economics and Mathematical Systems, Berlin, Heidelberg: Springer Berlin Heidelberg, ISBN: 978-3-540-10558-9 978-3-642-48318-9, DOI: 10.1007/978-3-642-48318-9.

Jaini, Nor and Sergey Utyuzhnikov (May 2017), « Trade-off ranking method for multi-criteria decision analysis: Trade-off ranking method for multi-criteria decision analysis », en, *in*: *Journal of Multi-Criteria Decision Analysis* 24.*3-4*, e1600, ISSN: 10579214, DOI: 10.1002/mcda.1600.

Kantorovich, Leonid Vitalievitch (July 1960), « Mathematical Methods of Organizing and Planning Production », *in*: *Management Science* 6.*4*, Publisher: Institute for Operations Research and the Management Sciences (INFORMS), pp. 366–422, DOI: 10.1287/mnsc.6.4.366.

Kaynar, Dilsun K., Nancy Lynch, Roberto Segala, and Frits Vaandrager (2006), *The Theory of Timed I/O Automata*, eng, Synthesis lectures on computer science 1, San Rafael, Calif: Morgan & Claypool, ISBN: 978-1-59829-010-3.

Kullback, Solomon and Richard Leibler (1951), « On information and sufficiency », English, *in*: *Annals of Mathematical Statistics* 22, pp. 79–86, ISSN: 0003-4851, DOI: 10.1214/aoms/1177729694.

Lang, Kevin J., Barak A. Pearlmutter, and Rodney A. Price (1998), « Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm », *in*: *Grammatical Inference*, ed. by Jaime G. Carbonell, Jörg Siekmann, G. Goos, J. Hartmanis, J. van Leeuwen, Vasant Honavar, and Giora Slutzki, vol. 1433, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–12, ISBN: 978-3-540-64776-8 978-3-540-68707-8, DOI: 10.1007/BFb0054059.

Laroussinie, François, Nicolas Markey, and Philippe Schnoebelen (2004), « Model Checking Timed Automata with One or Two Clocks », *in*: *CONCUR 2004 - Concurrency Theory*, ed. by Philippa Gardner and Nobuko Yoshida, vol. 3170, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 387–401, ISBN: 978-3-540-22940-7 978-3-540-28644-8, DOI: 10.1007/978-3-540-28644-8_25.

Larsen, Kim G., Florian Lorber, and Brian Nielsen (2018), « 20 Years of UPPAAL Enabled Industrial Model-Based Validation and Beyond », en, *in*: *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice*, ed. by Tiziana Margaria and Bernhard Steffen, vol. 11247, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 212–229, ISBN: 978-3-030-03426-9 978-3-030-03427-6, DOI: 10.1007/978-3-030-03427-6_18.

Lin, Jessica, Eamonn Keogh, Stefano Lonardi, and Bill Chiu (2003), « A symbolic representation of time series, with implications for streaming algorithms », en, *in*: *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery - DMKD '03*, San Diego, California: ACM Press, p. 2, DOI: 10.1145/882082.882086.

Malinowski, Simon, Thomas Guyet, René Quiniou, and Romain Tavenard (2013), « 1d-SAX: A Novel Symbolic Representation for Time Series », *in*: *Advances in Intelligent Data Analysis XII*, ed. by David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Allan Tucker, Frank Höppner, Arno Siebes, and Stephen Swift, vol. 8207, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 273–284, ISBN: 978-3-642-41397-1 978-3-642-41398-8, DOI: 10.1007/978-3-642-41398-8_24.

Mannila, Heikki, Hannu Toivonen, and A. Inkeri Verkamo (1997), « Discovery of Frequent Episodes in Event Sequences », *in*: *Data Mining and Knowledge Discovery* 1.*3*, pp. 259–289, ISSN: 13845810, DOI: 10.1023/A:1009748302351.

Merlin, Philip Meir (1974), « A study of the recoverability of computer systems », PhD thesis, UC Irvine: Donald Bren School of Information and Computer Sciences.

Mörchen, Fabian (2006), « Time series knowlegde mining », PhD thesis, Marburg: Philipps-Universität Marburg.

Mörchen, Fabian and Alfred Ultsch (2005), « Optimizing time series discretization for knowledge discovery », *in*: *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, ed. by Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett, ACM, pp. 660–665, DOI: 10.1145/1081870.1081953.

— (Aug. 2007), « Efficient mining of understandable patterns from multivariate interval time series », en, *in*: *Data Mining and Knowledge Discovery* 15.*2*, pp. 181–215, ISSN: 1384-5810, 1573-756X, DOI: 10.1007/s10618-007-0070-1.

Moskovitch, Robert and Yuval Shahar (July 2015), « Classification-driven temporal discretization of multivariate time series », en, *in*: *Data Mining and Knowledge Discovery* 29.*4*, pp. 871–913, ISSN: 1384-5810, 1573-756X, DOI: 10.1007/s10618-014-0380-z.

Nguyen, Hoang-Vu, Emmanuel Müller, Jilles Vreeken, and Klemens Böhm (Sept. 2014), « Unsupervised interaction-preserving discretization of multivariate data », en, *in*: *Data Mining and Knowledge Discovery* 28.*5-6*, pp. 1366–1397, ISSN: 1384-5810, 1573-756X, DOI: 10.1007/s10618-014-0350-5.

Oncina, Jose and Pedro García (Jan. 1992), « Inferring regular languages in polynomial updates time », en, *in*: *Series in Machine Perception and Artificial Intelligence*,

vol. 1, WORLD SCIENTIFIC, pp. 49–61, ISBN: 978-981-02-0881-3 978-981-279-790-2, DOI: 10.1142/9789812797902_0004.

Pasha, M. Fayzul K., Bijay Kc, and Saravanakumar Lakshmanan Somasundaram (May 2017), « An Approach to Detect the Cyber-Physical Attack on Water Distribution System », *in*: American Society of Civil Engineers, DOI: 10.1061/9780784480625.062.

Pastore, Fabrizio, Daniela Micucci, and Leonardo Mariani (May 2017), *Timed k-Tail: Automatic Inference of Timed Automata*, arXiv:1705.08399 [cs].

Peterson, James L. (Sept. 1977), « Petri Nets », en, *in*: *ACM Computing Surveys* 9.*3*, pp. 223–252, ISSN: 0360-0300, 1557-7341, DOI: 10.1145/356698.356702.

Petri, Carl (1962), « Kommunikation mit Automaten », PhD Thesis, TU Darmstadt.

Pham, Ninh D., Quang Loc Le, and Tran Khanh Dang (Apr. 2010), « Two Novel Adaptive Symbolic Representations for Similarity Search in Time Series Databases », *in*: *2010 12th International Asia-Pacific Web Conference*, Busan, Korea (South): IEEE, pp. 181–187, ISBN: 978-1-4244-6599-6, DOI: 10.1109/APWeb.2010.23.

Quiñones-Grueiro, Marcos, Alberto Prieto-Moreno, Cristina Verde, and Orestes Llanes-Santiago (2019), « Decision Support System for Cyber Attack Diagnosis in Smart Water Networks », *in*: *IFAC-PapersOnLine* 51.*34*, pp. 329–334, ISSN: 2405-8963, DOI: https://doi.org/10.1016/j.ifacol.2019.01.024.

Ramchandani, Chander (1973), « Analysis of asynchronous concurrent systems by timed petri nets », PhD thesis, Massachusetts Institute of Technology. Department of Electrical Engineering.

Ravn, Anders P., Jiří Srba, and Saleem Vighio (2010), « A Formal Analysis of the Web Services Atomic Transaction Protocol with UPPAAL », *in*: *Leveraging Applications of Formal Methods, Verification, and Validation*, ed. by Tiziana Margaria and Bernhard Steffen, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 579–593, ISBN: 978-3-642-16558-0.

Rissanen, Jorma (Sept. 1978), « Modeling by shortest data description », en, *in*: *Automatica* 14.*5*, pp. 465–471, ISSN: 00051098, DOI: 10.1016/0005-1098(78)90005-5.

Sawaragi, Yoshikazu, Hirotaka Nakayama, and Tetsuzo Tanino (1985), *Theory of multiobjective optimization*, Mathematics in science and engineering v. 176, Orlando: Academic Press, ISBN: 978-0-12-620370-7.

Shen, Wei, Jie An, Bohua Zhan, Miaomiao Zhang, Bai Xue, and Naijun Zhan (2020), « PAC Learning of Deterministic One-Clock Timed Automata », en, *in*: *Formal Meth-*

ods and Software Engineering, ed. by Shang-Wei Lin, Zhe Hou, and Brendan Mahony, vol. 12531, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 129–146, ISBN: 978-3-030-63405-6 978-3-030-63406-3, DOI: 10.1007/978-3-030-63406-3_8.

Shokoohi-Yekta, Mohammad, Yanping Chen, Bilson Campana, Bing Hu, Jesin Zakaria, and Eamonn Keogh (Aug. 2015), « Discovery of Meaningful Rules in Time Series », en, *in*: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Sydney NSW Australia: ACM, pp. 1085–1094, ISBN: 978-1-4503-3664-2, DOI: 10.1145/2783258.2783306.

Sommerville, Ian (2016), *Software engineering*, Tenth edition, OCLC: ocn907941065, Boston: Pearson, ISBN: 978-0-13-394303-0.

Stojanović, Branka, Helmut Neuschmied, Martin Winter, and Ulrike Kleb (Aug. 2022), « Enhanced Anomaly Detection for Cyber-Attack Detection in Smart Water Distribution Systems », en, *in*: *Proceedings of the 17th International Conference on Availability, Reliability and Security*, Vienna Austria: ACM, ISBN: 978-1-4503-9670-7.

Tang, Xiaochen, Wei Shen, Miaomiao Zhang, Jie An, Bohua Zhan, and Naijun Zhan (2022), « Learning Deterministic One-Clock Timed Automata via Mutation Testing », en, *in*: *Automated Technology for Verification and Analysis*, ed. by Ahmed Bouajjani, Lukáš Holík, and Zhilin Wu, vol. 13505, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 233–248, ISBN: 978-3-031-19991-2 978-3-031-19992-9, DOI: 10.1007/978-3-031-19992-9_15.

Taormina, Riccardo, Stefano Galelli, Nils Ole Tippenhauer, Elad Salomons, Avi Ostfeld, Demetrios G. Eliades, Mohsen Aghashahi, Raanju Sundararajan, Mohsen Pourahmadi, M. Katherine Banks, B. M. Brentan, M. Herrera, Amin Rasekh, Enrique Campbell, I. Montalvo, G. Lima, J. Izquierdo, Kelsey Haddad, Nikolaos Gatsis, Ahmad Taha, Saravanakumar Lakshmanan Somasundaram, D. Ayala-Cabrera, Sarin E. Chandy, Bruce Campbell, Pratim Biswas, Cynthia S. Lo, D. Manzi, E. Luvizotto Jr, Zachary A. Barker, Marcio Giacomoni, M. Fayzul K. Pasha, M. Ehsan Shafiee, Ahmed A. Abokifa, Mashor Housh, Bijay Kc, and Ziv Ohar (Aug. 2018), « The Battle Of The Attack Detection Algorithms: Disclosing Cyber Attacks On Water Distribution Networks », *in*: *Journal of Water Resources Planning and Management* 144.*8*, Publisher: ASCE, p. 04018048, DOI: 10.1061/(ASCE)WR.1943-5452.0000969.

Tappler, Martin, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber (Feb. 2019), « Learning Timed Automata via Genetic Programming », *in*: *arXiv: 1808.07744 [cs]*, arXiv: 1808.07744.

Vaandrager, Frits and Bengt Jonsson (2018), *Learning Mealy machines with timers*.

Vain, Juri, G. Kanter, and A. Anier (2019), « Learning Timed Automata from Interaction Traces », en, *in*: *IFAC-PapersOnLine* 52.*19*, pp. 205–210, ISSN: 24058963, DOI: 10.1016/j.ifacol.2019.12.097.

Vain, Juri, Fujio Miyawaki, Sven Nõmm, T. Totskaya, and A. Anier (Sept. 2009), « Human-robot interaction learning using timed automata », *in*: *ICCAS-SICE 2009 - ICROS-SICE International Joint Conference 2009, Proceedings*, pp. 2037–2042.

Valk, Rüdiger and Guy Vidal-Naquet (1981), « Petri Nets and Regular Languages », *in*: *Journal of Computer and System Sciences* 23.*3*, pp. 229–325.

Verwer, Sicco, Mathijs de Weerdt, and Cees Witteveen (2010), « A Likelihood-Ratio Test for Identifying Probabilistic Deterministic Real-Time Automata from Positive Data », *in*: *Grammatical Inference: Theoretical Results and Applications*, ed. by David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, José M. Sempere, and Pedro García, vol. 6339, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 203–216, ISBN: 978-3-642-15487-4 978-3-642-15488-1, DOI: 10.1007/978-3-642-15488-1_17.

— (Mar. 2012), « Efficiently identifying deterministic real-time automata from labeled data », en, *in*: *Machine Learning* 86.*3*, pp. 295–333, ISSN: 0885-6125, 1573-0565, DOI: 10.1007/s10994-011-5265-4.

Walter, Bernd (1983), « Timed Petri Nets for Modelling and Analyzing Protocols with Real-time Characteristics », *in*: *Protocol Specification, Testing and Verification* III, pp. 149–159.

Xu, Jiehui, Haixu Wu, Jianmin Wang, and Mingsheng Long (2021), « Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy », *in*: Publisher: arXiv Version Number: 5, DOI: 10.48550/ARXIV.2110.02642.

Xu, Runqing, Jie An, and Bohua Zhan (2022), « Active Learning of One-Clock Timed Automata Using Constraint Solving », en, *in*: *Automated Technology for Verification and Analysis*, ed. by Ahmed Bouajjani, Lukáš Holík, and Zhilin Wu, vol. 13505,

Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 249–265, ISBN: 978-3-031-19991-2 978-3-031-19992-9, DOI: 10.1007/978-3-031-19992-9_16.

Yang, Yixiao, Yu Jiang, Ming Gu, and Jiaguang Sun (2016), « Verifying Simulink Stateflow model: Timed automata approach », *in*: *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 852–857.

Zhao, Zhiruo (2017), « Ensemble methods for anomaly detection », PhD thesis, Syracuse University.

Zhou, Yuchen, Dipankar Maity, and John S. Baras (June 2016), « Timed automata approach for motion planning using metric interval temporal logic », *in*: *2016 European Control Conference (ECC)*, Aalborg, Denmark: IEEE, pp. 690–695, ISBN: 978-1-5090-2591-6, DOI: 10.1109/ECC.2016.7810369.

Zohrevand, Zahra and Uwe Glässer (Aug. 2020), *Should I Raise The Red Flag? A comprehensive survey of anomaly scoring methods toward mitigating false alarms*, arXiv: 1904.06646 [cs, stat].

Zolhavarieh, Seyedjamal, Saeed Aghabozorgi, and Ying Wah Teh (July 2014), « A Review of Subsequence Time Series Clustering », *in*: *The Scientific World Journal* 2014, ed. by Jesus A. Gonzalez, Publisher: Hindawi Publishing Corporation, p. 312521, ISSN: 2356-6140, DOI: 10.1155/2014/312521.

LIST OF ABBREVIATIONS

APTA Augmented Prefix Tree Acceptor. 36–39

BFS breadth-first search. 66, 68

BUBFS bottom-up breadth-first search. 66, 67

CPS Cyber-Physical System. 114, 120, 124

CuDAS Cumulative Decayed Anomaly Score. 119, 120, 123, 128, 131

DES Discrete Event System. 4, 5, 7, 10, 11, 14–18, 21, 23, 28, 31, 45

DFA Deterministic Finite Automaton. 19

DFS depth-first search. 66, 68, 148

DRTA Deterministic Real-Time Automaton. 33, 36, 37, 39, 44, 46, 47, 49, 58, 63, 79, 116, 157, 159, 176

EDSM Evidence-Driven State-Merging. 36, 44

EFD Equal Frequency Discretization. 85

ERA Event-Recording Automaton. 32

EWD Equal Width Discretization. 85, 86, 92

FSA Finite State Automaton. 17–19, 21–23, 25–27, 91

KL Kullback-Leibler. 96–103

MDL Minimum Description Length. 90, 92

MOODES Multi-Objective Optimization discretization for Discrete-Event Systems. 6, 106, 109, 111, 114–116, 122, 127–130, 132, 158, 159

MOOP Multi-Objective Optimization Problem. 106, 113, 114

NIS Negative Ideal Solution. 111

PAA Piecewise Aggregate Approximation. 85, 86

PAC Probably Approximately Correct. 36

PDRTA Probabilitic Deterministic Real-Time Automaton. 33, 39, 44

PIS Positive Ideal Solution. 111

PLC Programmable Logic Controller. 124–126

PPV Positive Predictive Value. 128–130

PTA Prefix Tree Acceptor. 48, 49, 66, 73

RTA Real-Time Automaton. 32

RTI Real-Time Identification. 36–39

RTI+ Real-Time Identification from positive data. 39, 45, 74, 75

SAX Symbolic Aggregate approXimation. 85, 86, 102, 104, 128, 129

SCADA Supervisory Control and Data Acquisition. 124–126

TA Timed Automaton. 1–8, 10, 11, 26–36, 39–47, 58, 61–66, 69, 71–73, 75, 79, 82, 83, 95, 96, 99, 101–106, 112–118, 122, 127, 129, 132, 135, 141, 152, 157–160, 194

TAG Timed Automata Generator. 5, 7, 10, 45–48, 51, 57, 58, 61, 62, 66, 69, 71–75, 79, 82, 95, 96, 102, 114–116, 122, 127, 157–160, 175

TCTL Temporal Computation Tree Logic. 30

TIOA Timed I/O Automata. vii, 33, 34, 82, 91, 134, 135, 138, 139, 141, 142, 150, 152, 156

TkT Timed k-Tail. 39–41, 45, 51, 74, 75

TPR True Positive Rate. 128–130

TS Time Series. 115

WDS Water Distribution System. 124–126, 129, 130

APPENDIX A

TAG IMPLEMENTATION IN PYTHON

An implementation of TAG in Python is available here: `https://gitlab.inria.fr/lcornang/tag/`. The version described here is correspond to commit *a8990c52*.

A.1 Input sample syntax

The input sequences must be provide in a text file with one sequence per line, the tuples event delay separated by a space and the event and the corresponding delay separated by a colon as follows:

```
event:delay event:delay ... event:delay
...
event:delay event:delay ... event:delay
```

The events can be letters, words, or numbers. The delays must be integers.

A.2 Usage

A.2.1 Example

```
from TAG.TALearner import TALearner

learner = TALearner(filename)
learner.ta.show()
learner.ta.print()
```

A.2.2 Arguments

— tss_path (str): path of the file containing the timed event sequences

— k (int, default=2): number of transitions to consider for the merging process, 2 by default

— res_path (str, optional): if an export is required, path to the file where to export the learned automaton

— $debug$ (boolean, default=False): true for verbose mode

— $splits$ (boolean, default=True): true if splits are allowed

— $merges$ (boolean, default=True): true if merges are allowed

— $order$ (str, default=breadth-first): ordering method for the operations (breadth-first/depth-first/random/bottom-up)

A.3 Output

A DRTA in DOT format, which can be render online [1] or in a terminal with Graphviz [2].

A.4 Requirements

TAG is implemented in Python (version: 3.8.3).
It requires the following modules for a graphical representation of the automata:

— graphviz (0.16)

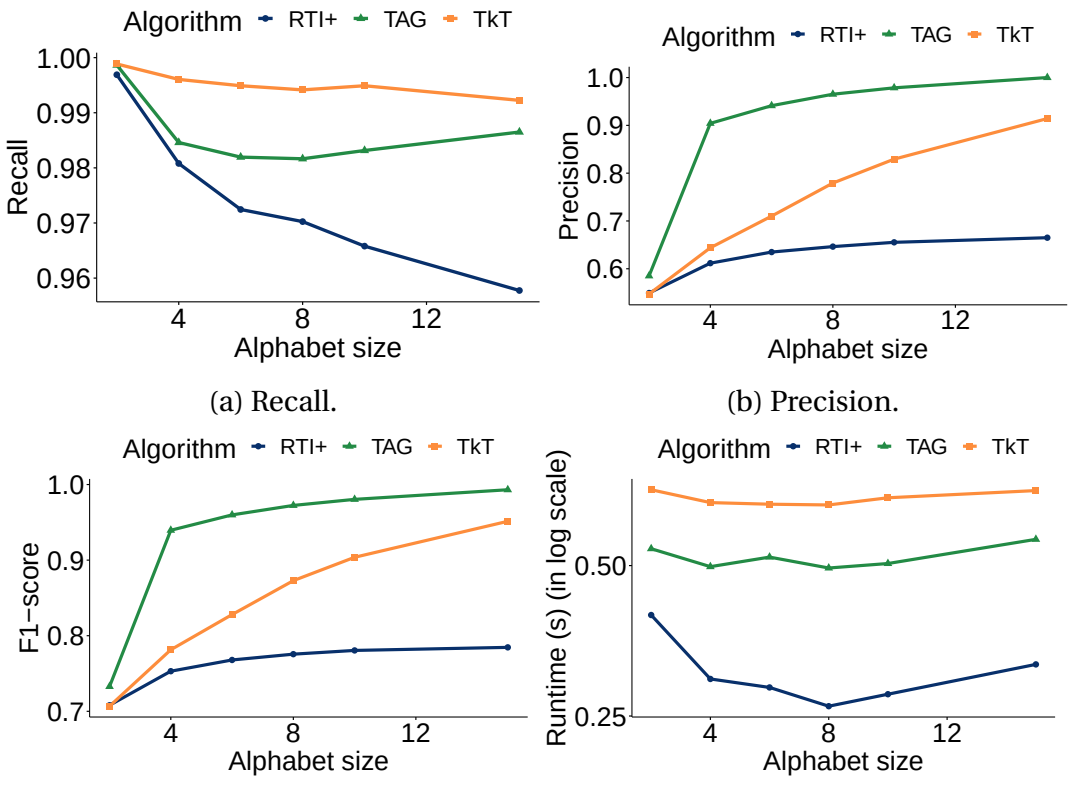
— IPython (7.21.0)

1. `https://dreampuf.github.io/GraphvizOnline`
2. `https://graphviz.org`

APPENDIX B

COMPARISON OF THE TAS LEARNING ALGORITHMS ON THE SCALABILITY EXPERIMENT

B.1 Alphabet size



(a) Recall.

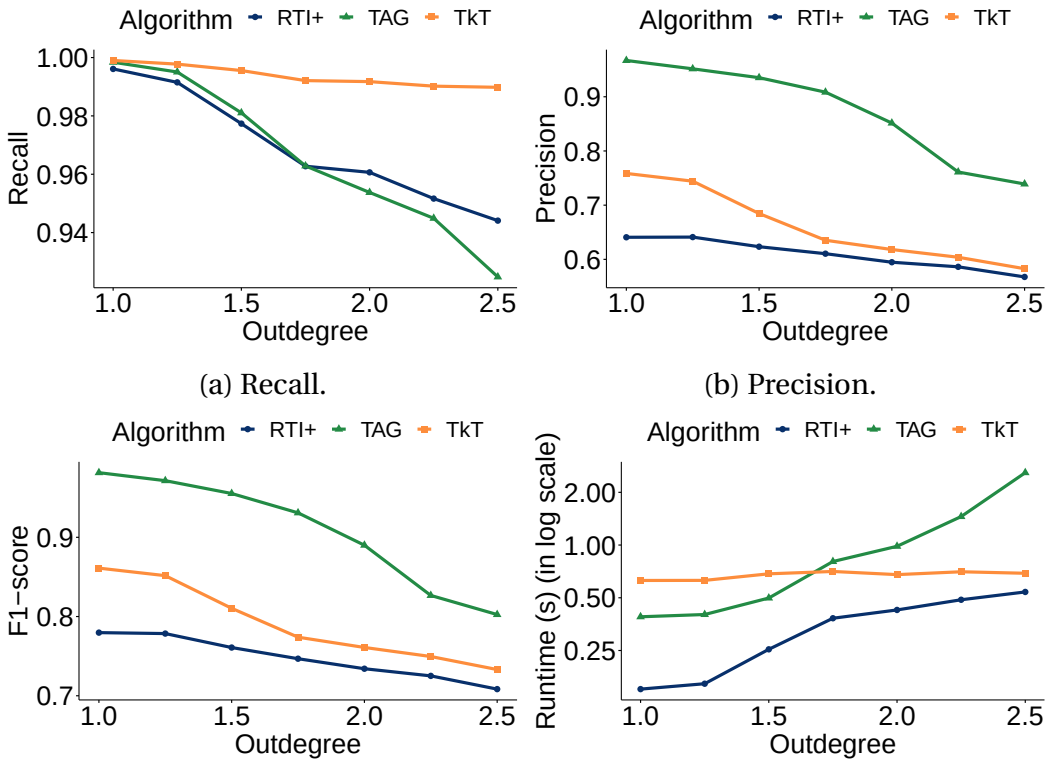
(b) Precision.

(c) F1-score.

(d) Runtime.

Figure B.1 – Impact of the alphabet size.

B.2 Outdegree



(a) Recall.

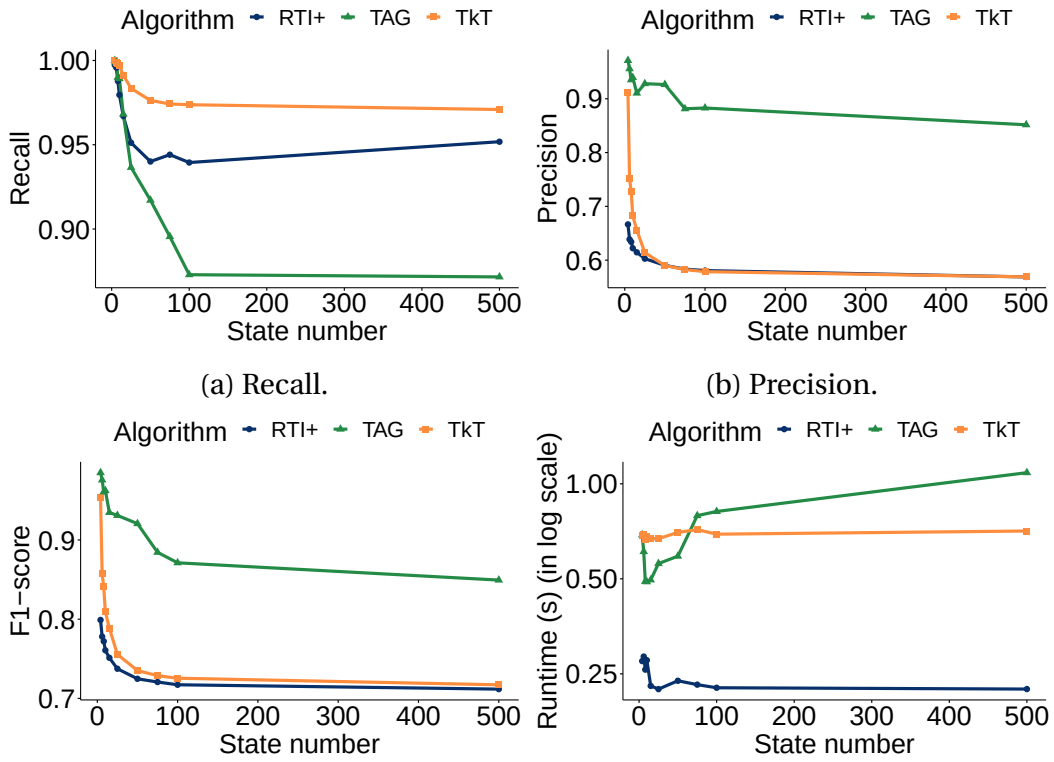
(b) Precision.

(c) F1-score.

(d) Runtime.

Figure B.2 – Impact of the outdegree.

B.3 State number



(a) Recall.

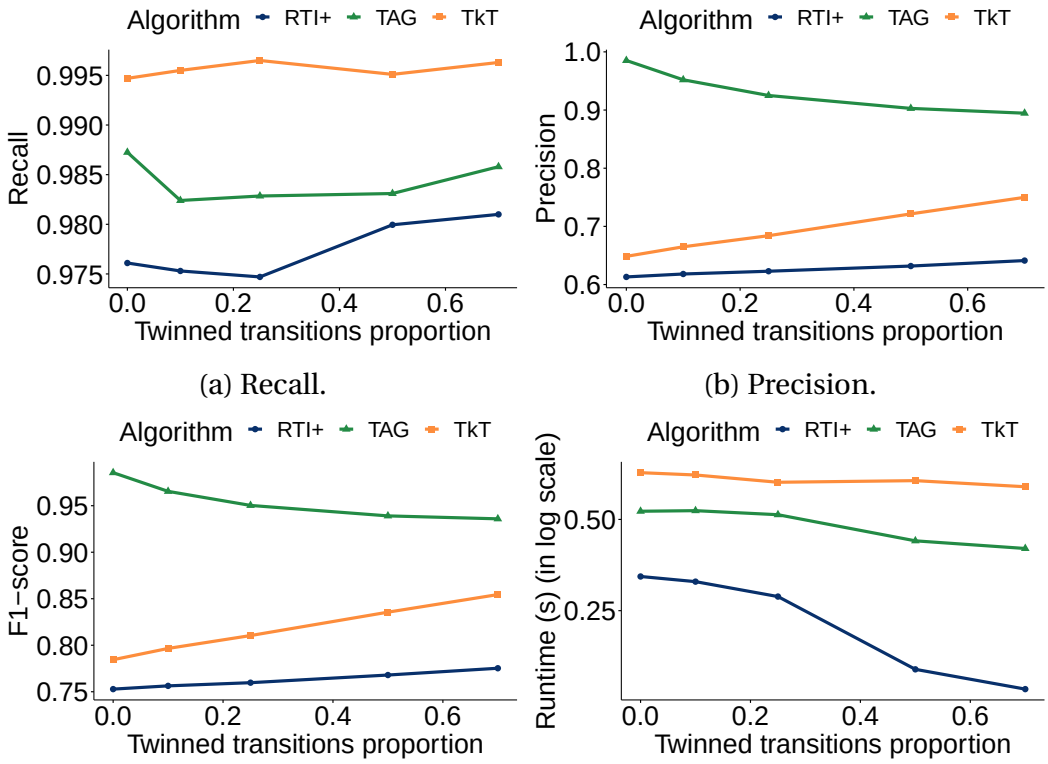
(b) Precision.

(c) F1-score.

(d) Runtime.

Figure B.3 – Impact of the state number.

B.4 Twinned transitions proportion



(a) Recall.

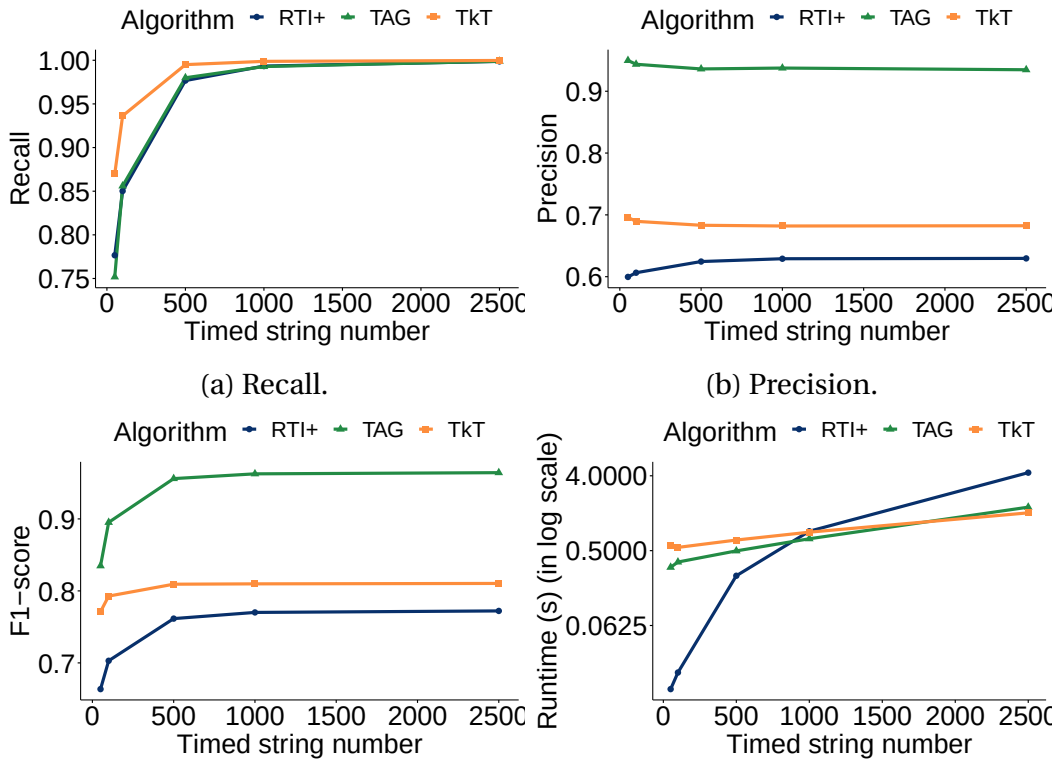
(b) Precision.

(c) F1-score.

(d) Runtime.

Figure B.4 – Impact of the proportion of twinned transitions.

B.5 Timed event sequences number



(a) Recall.

(b) Precision.

(c) F1-score.

(d) Runtime.

Figure B.5 – Impact of the number of timed event sequences.

RÉSUMÉ EN FRANÇAIS

Contexte et motivation

Cette thèse traite de l'apprentissage automatique de modèles de comportement de systèmes à partir de données d'observations, et plus particulièrement à partir de séries temporelles. Le formalisme choisi pour le modèle de comportement est l'automate temporisé (ALUR et DILL 1994), un automate fini doté de variables mesurant le temps qui passe, nommées horloges. La Figure 1 présente un automate temporisé modélisant le fonctionnement d'une lampe à intensité lumineuse variable. Un auto-

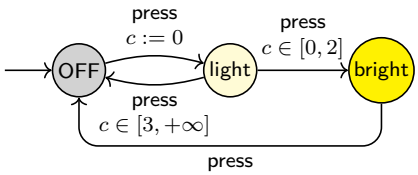


FIGURE 1 – Un automate temporisé représentant une lampe à intensité lumineuse variable.

mate temporisé a un ensemble d'états pouvant correspondre aux état d'un système, ici *OFF* pour «éteinte», *light* pour «allumée à intensité moyenne», et *bright* pour «allumée à forte intensité». La transition d'un état à un autre est provoquée par l'occurrence d'un évènement, ici *press* signifiant que l'interrupteur est pressé, satisfaisant des contraintes temporelles sur les valeurs des horloges, ici c. Par exemple, la transition de l'état *light* à l'état *bright* ne peut s'effectuer si la valeur de l'horloge c est comprise entre 0 et 2. Cette horloge mesure le temps écoulé depuis sa dernière réinitialisation effectuée sur la transition de l'état *OFF* à l'état *light* ($c := 0$). La contrainte temporelle sur la transition s'interprète donc comme un délai de 2 secondes au maximum entre les deux évènements *press*. Au-delà, l'évènement *press* provoquera l'extinction de la lampe par la transition entre *light* et *OFF*.

Par sa modélisation explicite de l'impact du temps sur les changements d'états, l'automate temporisé est particulièrement adapté pour modéliser des systèmes dans lesquels le temps est critique. En outre, ce formalisme compte parmi ses avantages

la possibilité de le représenter graphiquement, une riche littérature théorique explorant ses propriétés, quelques algorithmes permettant son apprentissage automatique à partir de séquences d'évènements, et la possibilité de décomposer des systèmes complexes en considérant plusieurs modèles en interaction.

De plus en plus, les données disponibles relatives aux systèmes d'intérêt proviennent de capteurs et prennent la forme de séries de valeur numériques mesurées au cours du temps, appelées séries temporelles. Dans leur forme brute, ces données ne sont pas utilisables pour la modélisation de système en s'appuyant sur un formalisme à la sémantique discrète tel que l'automate temporisé. Il existe de nombreuses méthodes numériques développées ces deux dernières décennies permettant de les exploiter pour une tâche précise telle que la prédiction ou la détection d'anomalie. Cependant, ces méthodes ont tendance à nécessiter beaucoup de données d'entraînement ou encore à manquer d'interprétabilité concernant leur fonctionnement ou le résultat de la tâche.

Dans cette thèse, l'objectif est d'utiliser ces données d'observation numériques pour apprendre automatiquement un modèle de comportement du système d'intérêt en utilisant le formalisme de l'automate temporisé. Pour ce faire, une première partie est dédiée à l'amélioration de l'état de l'art en matière d'apprentissage automatique d'automates temporisés à partir de données discrètes. La suite de ce travail est axée sur la discrétisation de séries temporelles, ce qui consiste en la transformation d'une série temporelle quantitative en une séquence de points discrets (Figure 2a). La discrétisation d'une série temporelle permet de la rendre utilisable pour la création d'un automate temporisé (Figure 2b) ou autre modèle de système à évènement discret, les valeurs discrètes étant alors considérées comme des évènements. Nous souhaitons particulièrement insister sur l'importance de cette étape de discrétisation qui peut trop souvent être vue comme une simple étape de préparation des données sans grand impact sur la suite. Dans notre contexte, la méthode de discrétisation choisie aura un très grand impact sur l'automate résultant. Pour cette raison, la seconde partie de ce travail est dédiée à la recherche de caractéristiques des séquences discrétisées favorisant l'apprentissage d'un automate interprétable et exploitable pour d'autres applications, menant à une nouvelle méthode de discrétisation spécialement développée pour ce but.

L'objectif de la troisième partie de cette thèse est toujours d'apprendre un modèle de comportement d'un système à partir de séries temporelles, mais avec la contrainte

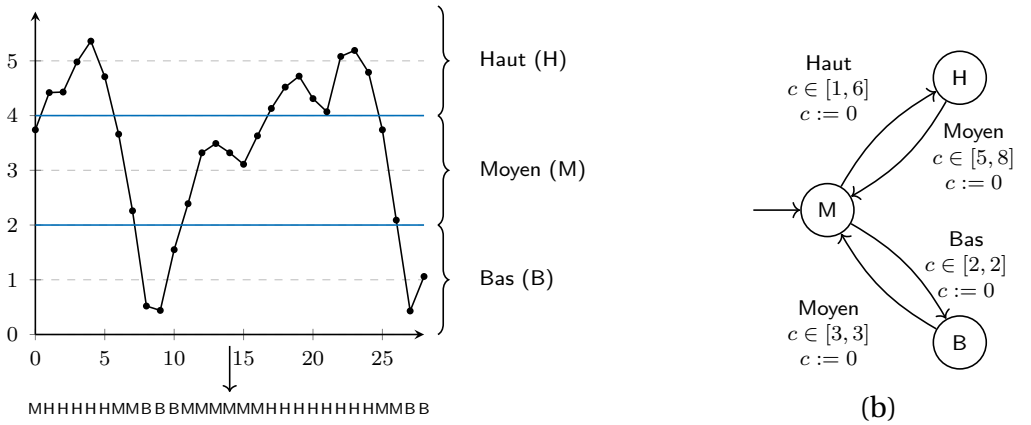


FIGURE 2 – De la série temporelle à l'automate temporisé. Ici, la série temporelle est discrétisée en partitionnant sa plage de valeurs. Les valeurs discrètes résultantes (Bas (B), Moyen (M) and Haut (H)) sont interprétées comme des évènements pour la construction de l'automate.

supplémentaire sur le système qui est alors constitué d'un ensemble de composants en interaction. Le formalisme de l'automate temporisé permet de synchroniser les transitions de plusieurs automates grâce à des évènements partagés. Étant donné

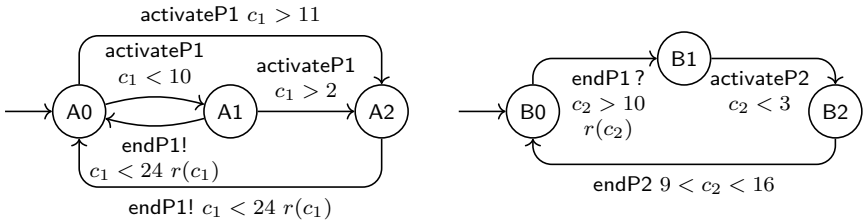


FIGURE 3 – Deux automates temporisés en interaction synchronisés par l'évènement *endP1[?/!]*.

une série temporelle multivariée où chaque variable correspond à un composant du système, l'identification de ces évènements partagés est un défi supplémentaire puisqu'une mauvaise discrétisation de chaque variable pourrait les cacher. Il est donc nécessaire de considérer les autres variables lors de la discrétisation de chacune.

Résumé des contributions

TAG : Apprentissage d'automates temporisés à partir de séquences d'évènements (Chapitre 2)

La première contribution de cette thèse est TAG (CORNANGUER 2021 ; CORNANGUER, LARGOUËT et al. 2022), un nouvel algorithme permettant d'apprendre un automate temporisé à partir de séquences d'évènements correspondant à des exécutions du système modélisé. C'est un algorithme d'apprentissage passif, signifiant qu'il ne nécessite pas d'interaction avec le système. L'apport de cet algorithme relativement à l'état de l'art est un meilleur compromis entre généralisation et précision en termes de séquences d'évènements reconnues non issues de l'échantillon d'apprentissage, mais venant du même système. La qualité de ce compromis a été démontrée par une vaste comparaison expérimentale sur données synthétiques.

L'automate produit par TAG appartient à une sous-classe des automates temporisés : l'automate déterministe temps-réel. Cet automate est déterministe, ce qui signifie qu'il n'existe qu'un seul chemin dans l'automate pour une séquence d'évènements donnée. Par ailleurs, il ne possède d'une seule horloge qui est réinitialisée à chaque transition, elle mesure donc le délai entre deux évènements. Cette restriction du nombre d'horloges est à ce jour encore nécessaire en apprentissage passif, l'identification des paramètres temporels étant la plus grande difficulté.

TAG a un unique paramètre, k, contrôlant le niveau de généralisation du modèle vis-à-vis des données, avec une valeur par défaut suggérée. Lors de son déroulement, TAG commence par construire un automate correspondant presque exactement aux séquences d'apprentissage (avec une légère généralisation au niveau des contraintes d'horloge). Ensuite, la généralisation est permise par la fusion des états offrant à court terme les mêmes séquences d'évènement possibles. Ces ensembles de séquences sont nommés le k-futur d'un état. Le paramètre k contrôle la longueur des séquences à considérer. Par exemple, le k-futur de l'état A0 de la Figure 3 avec $k = 2$ est l'ensemble suivant :

$$\{\langle activateP1, endP1!\rangle, \langle activateP1, activateP1\rangle\}$$

Plus k est grand, plus les séquences à considérer seront longues et moins l'auto-

mate sera factorisé. En conséquence, l'automate final sera plus proche des données d'apprentissage. Cette opération de fusion d'états ne considère pas le temps. TAG a donc une opération supplémentaire, la division de transition, qui a pour but d'identifier des transitions où le temps est déterminant pour le comportement du système. Cette opération créée un déterminisme temporel, il est alors nécessaire de regarder la valeur temporelle associée à l'évènement pour choisir une transition unique parmi toutes celles partant de l'état courant et portant le même évènement.

MOODES : Discrétisation de séries temporelles par optimisation mulit-objectifs pour l'apprentissage d'automates temporisés (Chapitre 4)

La seconde contribution de cette thèse est MOODES, une méthode de discrétisation de séries temporelle spécialement dédiée à l'apprentissage d'automates temporisés. Cette méthode repose sur l'optimisation de trois critères que nous avons identifiés comme étant favorable pour des séquences discrétisées destinées à apprendre un automate temporisé.

Le premier critère retenu est le nombre d'évènements différents créés dans la séquence. Un grand nombre d'évènements mènera à un automate plus grand et plus complexe, ce qui peut nuire à l'interprétabilité du modèle. On cherchera donc à minimiser ce nombre d'évènements. Le risque à trop le réduire est cependant d'avoir un modèle trop simple, avec le cas extrême où tous les points de la série temporelle sont remplacés par un même évènement. Le second critère vise donc à contrebalancer le premier, il s'agit de minimiser la dispersion de valeurs dans la série temporelle correspondant à un même évènement dans la séquence discrétisée. Le dernier critère est la persistance des événements dans la séquence. Un évènement persistant a une probabilité de se répéter sur deux pas de temps consécutifs plus élevée que sa probabilité d'occurrence à n'importe quel pas de temps. Cette notion et un moyen de l'évaluer ont été proposés par MÖRCHEN et ULTSCH (2005), et nous avons amélioré la manière de calculer ce score (CORNANGUER, LARGOUËT et al. 2023).

L'optimisation simultanée de ces trois critères contradictoires est réalisée au moyen d'un algorithme génétique. L'objectif est d'associer des intervalles de valeurs dans la série temporelle à des évènements. Les meilleures limites pour ces intervalles sont celles qui optimisent les trois objectifs. Le résultat de l'algorithme génétique est un ensemble des meilleures solutions faisant des compromis différents entre les objec-

tifs établis. Chaque solution va mener à une séquence différente et donc à un automate temporisé différent. Il est possible de choisir une unique solution, aléatoirement ou en fixant une préférence sur l'un des critères. Cependant, nous suggérons de conserver toutes les solutions et d'apprendre un ensemble d'automates temporisés au lieu d'un unique automate. Le modèle de comportement du système prend alors la forme d'un ensemble de modèles représentant le système sous différentes perspectives et à différentes échelles.

Suite à la présentation de cette méthode de discrétisation, nous proposons une application en détection d'anomalies dans les séries temporelles sur un jeu de données de système cyberphysique soumis à des attaques. Les données d'apprentissage sans attaque sont discrétisées avec MOODES et des ensembles d'automates sont appris avec TAG pour chaque composant du système. Ensuite, les données de test contenant des attaques sont discrétisées et injectées dans les automates. Cette méthode de détection d'anomalie n'atteint pas les meilleures performances observées à ce jour, notamment obtenues en utilisant une méthode spécifique à ce problème qui utilise des notions hydroliques (HOUSH et OHAR 2018) avec un score F1 de 0.970 contre 0.772 pour notre approche (Figure 4). Cependant, l'avantage des automates temporisés est qu'ils permettent d'expliquer la nature de l'incohérence entre le modèle et la donnée lorsqu'il y en a une, et notre approche est celle permettant d'obtenir le meilleur score F1 parmi les méthodes interprétables.

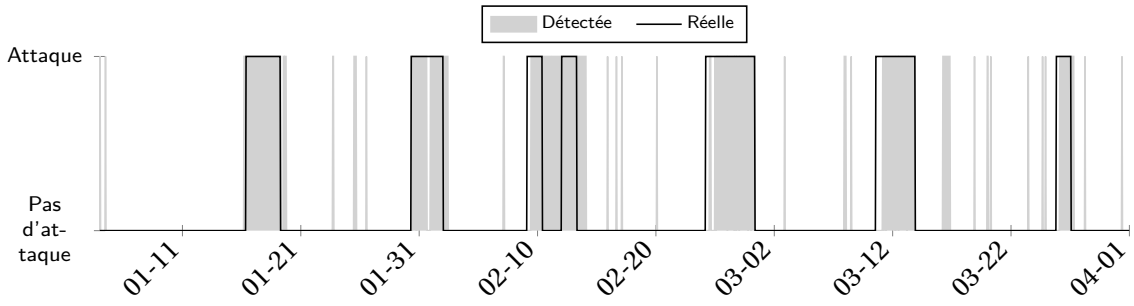


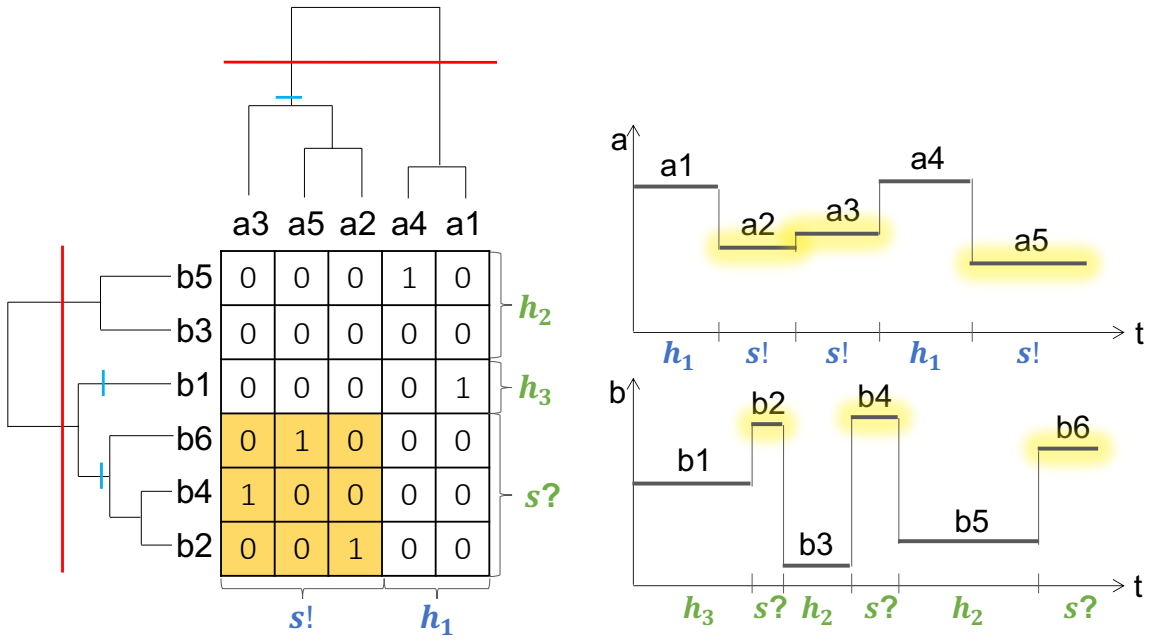
FIGURE 4 – Périodes d'attaque réelles et détectées par notre approche.

Discrétisation de séries temporelles multivariées pour l'apprentissage d'automates synchronisés (Chapitre 5)

La dernière contribution de cette thèse est une méthode de discrétisation de séries temporelles multivariées préservant les synchronisations. Ici, la prise en compte des autres variables lors de la discrétisation d'une variable est primordiale, car certains évènements dits synchronisés seront partagés par les automates de chaque variable.

La recherche d'évènements de synchronisation dans une série temporelle multivariée étant un nouveau problème, nous l'avons abordé par une méthode de discrétisation plus classique. Notre méthode repose sur un partitionnement hiérarchique agglomératif des segments de la série temporelle, basé sur une mesure de similarité laissée au choix de l'utilisateur. Ce type de partitionnement est souvent visualisé sous la forme d'un dendrogramme qui peut ensuite être découpé pour obtenir les groupes définitifs. Ce partitionnement est effectué indépendamment pour chaque variable, et il sera ainsi possible d'associer chaque groupe de segments similaires à un évènement qui va remplacer les segments dans la version discrétisée. Cependant, ici, nous souhaitons contraindre la création des groupes de segments en fonction du résultat du partitionnement pour les autres variables.

Un évènement de synchronisation entre deux variables est un évènement qui survient simultanément dans les variables. Chercher des évènements synchronisés dans une série temporelle multivariée revient à chercher les groupes de segments similaires dans chaque variable qui surviennent simultanément. Il faut donc étudier la cooccurrence des segments des différentes variables, tout en regardant la similarité des segments au sein de chaque variable. Nous proposons une structure de données permettant d'étudier ces deux facteurs simultanément (Figure 5a). Étant donné une série temporelle multivariée segmentée (segments a1, a2, ... sur la Figure 5b), la structure de données rassemble les informations sur la cooccurrence des segments entre deux variables dans une matrice (1 pour cooccurrence, 0 sinon). La similarité entre les segments d'une même variable est donnée dans les dendrogrammes dont les feuilles sont associées aux colonnes et lignes de la matrice et donc aux segments. L'exploration de cette structure de données permet l'identification d'évènement de synchronisation (en jaune), et permet par la suite la création des groupes de segments en tant qu'évènement classique ou de synchronisation.



(a) Matrice de co-occurrence.

(b) Série temporelle multivariée.

FIGURE 5 – Les lignes rouges au-dessus des dendrogrammes correspondent à un niveau de coupe proposé par un algorithme de partitionnement classique. Les lignes bleues correspondent à un niveau de coupe nécessaire pour préserver les synchronisations, obtenues par notre algorithme. Les groupes de segments sont ensuite associés à un évènement classique (h_1, h_2, h_3) ou a un évènement de synchronisation $(s[?/!])$.

Les premières expériences sur données synthétiques montrent que l'approche permet bien de retrouver des synchronisations, y compris dans des données bruitées.

Références

ALUR, Rajeev et David DILL (avr. 1994), « A theory of timed automata », en, in : *Theoretical Computer Science* 126.*2*, p. 183-235, ISSN : 03043975, DOI : 10.1016/0304-3975(94)90010-8.

CORNANGUER, Lénaïg (mai 2021), « Passive learning of Timed Automata from logs (Student Abstract) », in : *Proceedings of the AAAI Conference on Artificial Intelligence* 35.*18*, Section : AAAI Student Abstract and Poster Program, p. 15773-15774.

CORNANGUER, Lénaïg, Christine LARGOUËT, Laurence ROZÉ et Alexandre TERMIER (juin 2022), « TAG : Learning Timed Automata from Logs », in : *Proceedings of the AAAI Conference on Artificial Intelligence* 36.*4*, p. 3949-3958, ISSN : 2374-3468, 2159-5399, DOI : 10.1609/aaai.v36i4.20311.

— (fév. 2023), « Persistence-Based Discretization for Learning Discrete Event Systems from Time Series », in : *MLmDS 2023 - AAAI Workshop When Machine Learning meets Dynamical Systems : Theory and Applications*, Washington (DC), United States, p. 1-6.

HOUSH, Mashor et Ziv OHAR (août 2018), « Model-based approach for cyber-physical attack detection in water distribution systems », en, in : *Water Research* 139, p. 132-143, ISSN : 00431354, DOI : 10.1016/j.watres.2018.03.039.

MÖRCHEN, Fabian et Alfred ULTSCH (2005), « Optimizing time series discretization for knowledge discovery », in : *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, sous la dir. de Robert GROSSMAN, Roberto J. BAYARDO et Kristin P. BENNETT, ACM, p. 660-665, DOI : 10.1145/1081870.1081953.

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Apprentissage d'automates temporisés à partir de séries temporelles

Mot clés : apprentissage d'automates, automates temporisés, système à évènements discrets, séries temporelles, discrétisation, détection d'anomalies

Résumé : Cette thèse explore le développement de nouvelles techniques pour exploiter les données d'observation de système afin d'apprendre automatiquement des modèles de comportement. Nous utilisons le formalisme de l'automate temporisé, un automate fini dont les changements d'état sont déclenchés par des événements et contraints par le temps. Trois enjeux principaux sont abordés : (1) l'apprentissage d'automates temporisés à partir de séquences d'événements ; (2) la discrétisation de séries temporelles pour permettre l'apprentissage d'automates temporisés à partir de données numériques ; et (3) l'apprentissage d'automates temporisés en interaction à partir de séries temporelles. Tout d'abord, nous présentons TAG, un algorithme d'apprentissage d'automates temporisés. Une experience sur des données synthétiques démontre qu'il atteint un meilleur équilibre entre sur-ajustement et généralisation que les algorithmes de l'état de l'art. Ensuite, nous proposons MOODES, une méthode de discrétisation de séries temporelles basée sur l'optimisation de critères favorables à l'apprentissage d'automates temporisés. TAG et MOODES sont appliqués sur un défi de détection d'anomalies dans les séries temporelles avec une approche basée sur des ensembles d'automates. Enfin, nous présentons une nouvelle approche de discrétisation pour séries temporelles multivariées qui identifie et préserve les synchronisations afin d'apprendre des automates en interaction.

Title: Timed automata learning from time series

Keywords: automata learning, timed automata, discrete event systems, time series, discretization, anomaly detection

Abstract: This thesis explores the development of new techniques for using the observational data of a system to automatically infer behavior models. We adopt the Timed Automata formalism, a finite-state machine in which state changes are triggered by events and are constrained by time. Three main challenges are addressed: (1) learning a Timed Automaton from event sequences; (2) discretizing time series to enable the learning of Timed Automata from numerical data; and (3) learning interacting Timed Automata from time series. First, we introduce TAG, a Timed Automata learning algorithm. Experiment on synthetic data demonstrates that it achieves a better balance between overfitting and generalization than the State-of-the-Art algorithms. Then, we propose MOODES, a discretization method based on optimizing criteria favorable for learning Timed Automata, which produces multiple solutions. TAG and MOODES are applied to a time series anomaly detection challenge where we use ensembles of Timed Automata and provide explanations for the anomalies. Finally, we present discretization approach for multivariate time series that identifies and preserves the synchronizations with the aim of learning interacting automata.